Министерство образования Российской Федерации

Нижегородский государственный университет
им. Н.И. Лобачевского

Высокопрозводительные параллельные вычисления на кластерных системах

Материалы второго Международного
научно-практического семинара

26–29 ноября 2002 г.

Издательство Нижегородского госуниверситета
Нижний Новгород
2002

УДК 681.3.012:51
ББК 32.973.26–018.2:22
В 93

В93 **Высокопроизводительные параллельные вычисления на кластерных системах.** Материалы второго Международного научно-практического семинара / Под ред. проф. **Р.Г. Стронгина.** Нижний Новгород: Изд-во Нижегородского госуниверситета, 2002. 351 с.

Сборник сформирован по итогам научного семинара, посвященного теоретической и практической проблематике параллельных вычислений, ориентированных на использование современных многопроцессорных архитектур кластерного типа. Семинар проходил в Нижнем Новгороде 26-29 ноября 2002 года.

Вошедшие в сборник материалы семинара представляют интерес для преподавателей и научных сотрудников высших учебных заведений, а также для инженеров, аспирантов и студентов вузов.

Отв. за выпуск к.ф.-м.н., доцент **В.А. Гришагин**

ББК 32.973.26–018.2:22

Поддержка семинара

РФФИ — Российский фонд фунтаментальных исследований

intel. — Компания Intel Technologies

NSTLab — Нижегородская лаборатория программых технологий

Министерство образования Российской Федерации (ФЦП "Интеграция" и НП "Университеты России")

FASIE — Фонд содействия развитию малых форм предприятий в научно-технической сфере

26–29 ноября 2002 года Вычислительный Центр РАН, Институт математического моделирования РАН, Нижегородский государственный университет им. Н.И. Лобачевского провели в Нижнем Новгороде второй Международный научно-практический семинар и Всероссийскую молодежную школу «Высокопроизводительные параллельные вычисления на кластерных системах».

Главная направленность семинара и школы состояла в обсуждении основных аспектов организации высокопроизводительных вычислений в кластерных в кластерных компьютерных системах, активизации научно-практической деятельности исследователей в этой перспективной области развития современных средств вычислительной техники, обмене опытом учебно-образовательной деятельности при подготовке специалистов в области параллельных вычислений.

Проблематика семинара нацелена на рассмотрение следующих вопросов параллельных вычислений:

- принципы построения кластерных вычислительных систем;
- методы управления параллельных вычислениями в кластерных системах;
- параллельные алгоритмы решения сложных вычислительных задач;
- программные среды и средства для разработки параллельных программ;
- прикладные программные системы параллельных вычислений;
- методы анализа и оценки эффективности параллельных программ;
- проблемы подготовки специалистов в области параллельных вычислений.

В данный сборник включены материалы сообщений, которые представлены как в виде статей, так и в виде кратких тезисов. Материалы сборника упорядочены в алфавитном порядке по фамилии первого автора.

ОРГКОМИТЕТ СЕМИНАРА

Стронгин Р.Г.	председатель оргкомитета, Первый проректор ННГУ, профессор, д.ф.-м.н.
Гергель В.П.	заместитель председателя Оргкомитета, профессор, д.т.н., ННГУ
Швецов В.И.	заместитель председателя Оргкомитета, директор Регионального центра НИТ, д.т.н., ННГУ
Батищев Д.И.	профессор, д.т.н., ННГУ
Евтушенко Ю.Г.	директор Вычислительного центра РАН, чл.-корр. РАН
Нестеренко Л.В.	директор по стратегии и технологиям Нижегородской лаборатории Intel (INNL), к.ф.-м.н.
Сергеев Я.Д.	профессор, д.ф.-м.н., ННГУ, Калабрийский университет (Италия)
Четверушкин Б.Н.	директор Института математического моделирования РАН, чл.-корр. РАН
Гришагин В.А.	ученый секретарь семинара, к.ф.-м.н., ННГУ

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ПОСТРОЕНИЯ ОСТОВА МНОГОГРАННОГО КОНУСА

Е.А. Агафонов, Е.Л. Земскова, Н.Ю. Золотых

Нижегородский государственный университет им. Н.И. Лобачевского

Целью данной работы является разработка и программная реализация параллельного алгоритма построения остова многогранного конуса. С помощью разработанной программы найдены условия совместности трехиндексных транспортных задач с параметрами 4×4×3 и 4×4×4.

Необходимые определения и обозначения

Приведенные в этом пункте сведения можно найти, например, в [1, 2].

(Многогранным) конусом называется множество $C(A)$ решений системы линейных неравенств $Ax \geq 0$, то есть

$$C(A) = \{x \in F^n : Ax \geq 0\},$$

где $A \in F^{mxn}$. Конической оболочкой $\mathrm{Cone}\{r_1,\ldots,r_s\}$ векторов $r_1, \ldots, r_s \in F^n$ называется множество их всех неотрицательных линейных комбинаций, то есть

$$\mathrm{Cone}\{r_1, \ldots, r_s\} = \{\alpha_1 r_1 + \ldots + \alpha_s r_s : \alpha_1 \geq 0, \ldots, \alpha_s \geq 0\}.$$

Говорят, что система $r_1,\ldots,r_s$ порождает конус C, если $C = \mathrm{Cone}\{r_1,\ldots,r_s\}$. Система $r_1,\ldots,r_s$, порождающая конус C называется остовом конуса, если никакая собственная подсистема системы $r_1,\ldots,r_s$ не порождает C.

Постановка задачи

Алгоритм Моцкина–Бургера позволяет находить описание (2) по описанию (1) и обратно в силу теоремы Вейля (см. [1]).

Алгоритм Моцкина–Бургера

Алгоритм Моцкина–Бургера (см., например, [2]) позволяет по заданной системе однородных линейных неравенств построить остов многогранного конуса. Следующая модификация этого алгоритма

описана в [3].

Пусть острый конус K задан системой $Ay \geq 0$, $A \in Z^{m\times n}$, rank $A = n$. Алгоритм Моцкина–Бургера строит остов $B = \{b_1, \ldots, b_s\}$ конуса K. Систему B удобно рассматривать как матрицу из $Z^{s\times n}$, строки которой суть $b_1, \ldots, b_s$.

Так как rank $A = n$, то в A существует невырожденная квадратная подматрица $A' \in Z^{m\times n}$. Пусть $A = (A'|A'')$, тогда $K = \{y: A'y \geq 0; A''y \geq \geq 0\}$.

На предварительном шаге алгоритмом Гаусса найдем невырожденную подматрицу A' матрицы A (применяя алгоритм Гаусса к A^T), а также матрицы $B^0 \in Z^{n\times m}$, $F^0 \in Z^{n\times m}$, $Q^0 \in Z^{n\times(m-n)}$ такие, что F^0 – диагональная матрица с положительными диагональными элементами и $B^0 \circ A^T = (F^0|Q^0)$, причем $НОД\{b_{i1}^0, b_{i2}^0, \ldots, f_{ii}^0\} = 1 (i = \overline{1, 2})$, то есть, $b_{ij}^0 \big/ f_{ii}^0$ (i, j)-ый элемент матрицы, обратной к $A' (i, j = \overline{1, n})$. Заметим, что проделанные операции соответствуют замене координат $y' = A'y$.

Рассмотрим последовательность $\{A^l\}$ $(l = 0, 1, \ldots, m - n)$ подсистем матрицы A, образованную по следующему правилу: $A^0 = A'$ и для каждого $l = 1, \ldots, m - n$ матрица A^l получается из A^{l-1} приписыванием к последней произвольной строчки из A'', еще не вошедшего в A^l; не нарушая общности будем считать, что это l-ая строчка матрицы A''. Остов конуса, соответствующего подсистеме неравенств $A^l y \geq 0$, обозначим через B^l.

Общий шаг алгоритма представляет собой преобразование остова B^l в остов B^{l+1}, которое опишем как преобразование таблицы $(B^l|F^l|Q^l)$.

Пусть для некоторого l таблица $(B^l|\ F^l|\ Q^l)$ известна. Если $l = = m - n$, то B^l – искомый остов конуса $C(A)$.СТОП. В противном случае найдется строчка матрицы A'', которая еще не вошла в A^i ни для какого $i = n, \ldots, l - 1$. Для простоты последующего описания введем следующие обозначения: $B = B^l$, $F = F^l$, $Q = Q^l$. Пусть матрица T имеет размеры $s_l \times (n + m)$. Через $b_{ij}, f_{ij}, q_{ij}, t_{ij}$ будем обозначать элементы матриц B, F, Q, T. Основной шаг алгоритма (переход $l \to \to l + 1$) складывается из следующих действий:

1. Формируем множества

$$J_-, J_+ : J_- = \{r : q_{r,l+1} < 0, r = 1, 2, ,..., s_l),$$

$$J_+ = \{r : q_{r,l+1} > 0, r = 1, 2, ,..., s_l)\}.$$

2. Если $|J_-| = s_l$, то СТОП – система $Ay \geq 0$ не имеет решения, отличного от нулевого.
3. Если $|J_-| = 0,$, то переход $l \to l + 1$ завершен.
4. Формируем множество $P = \{(i, j)\colon i \in J_-, j \in J_+\}$.
5. Если $P = \varnothing$, то переходим к 11.
6. Произвольным образом выбираем $(i, j) \in P$, исключив его из P.
7. Строим множество $I_{ij} = I_i \cap I_j$, где

$$I_r = \{\mu : \{1, 2, ,..., n + l\} : t_{r,\mu} = 0\}.$$

8. Если $|J_{ij}| < n - 2,$ то переходим к 5.
9. Если существует $r \in \{1, 2, ,..., s_l\} \setminus \{i, j\},$ для которого $I_{ij}^l \subseteq I_r^l,$ то возвращаемся к 5.
10. Приписываем к T вектор-строку $t = q_{j,l+1}t_i + q_{i,l+1}t_j$. Сокращаем компоненты t на НОД. Возвращаемся на 5.
11. Вычеркиваем из T строки с номерами из J_-.

Полученную после завершения операций таблицу обозначим через T^{l+1} – шаг алгоритма $l \to l + 1$ завершен.

Если на шаге 3 алгоритма $|J_-| = 0,$ это означает, что текущее неравенство системы $Ay \geq 0$ является следствием из предыдущих неравенств системы. Поэтому оно может быть сразу удалено из системы.

Алгоритм незначительно изменится для случая неострого конуса, то есть rank $A = r < n$. Так как rank $A = r < n$, то в A существует невырожденная подматрица $A' \in Z^{r\times r}$. Пусть $A = (A'|A'')$, тогда $K = \{y : A'y \geq 0; A''y \geq 0\}$.

На предварительном шаге алгоритмом Гаусса найдем невырожденную подматрицу A' матрицы A, а также матрицы $B^0 \in Z^{r\times m}$, $F^0 \in \in Z^{r\times r}$, $Q^0 \in Z^{r\times(m-r)}$, определенные как и в предыдущем случае. Также

будем искать базис подпространства (строчки матрицы E), содержащегося в неостром конусе. Для этого, если при применении алгоритма Гаусса мы получим нулевую строку в матрице A^T, то добавляем соответствующую строку из B^0 в E и удаляем ее из B^0, F^0, Q^0. Таким образом, получим базис подпространства, описанный строками матрицы E.

Общий шаг выполняется аналогично (где n заменяется на r).

В случае неострого конуса его остов будет состоять из строк матриц B, E и вектора $e_0 = -\sum_{j=1}^{p} e_j$, где $e_j (j = \overline{1, p})$ – строки матрицы E.

Программа, реализующая данный алгоритм, описана в [4].

Параллельный алгоритм

В пункте 4 основного шага алгоритма формируется множество пар $P = \{(i, j) : i \in J_-,\ j \in J_+\}$. Далее, мы сравниваем пары строчек с такими номерами, считаем количество общих нулей. Если выполняются дальнейшие условия, то вычисляем новую вектор-строку.

Предлагается разделить множество P на части, число которых соответствует количеству параллельных процессов. Поиск общих нулей для пары строчек и вычисление новых векторов-строк будут выполняться в соответствующем процессе.

Созданная программа, реализующая данный алгоритм, использует технологию MPI [5].

Эксперимент

Для исследования рабочих характеристик написанной программы был проведен следующий вычислительный эксперимент: находились условия совместности трехиндексной транспортной задачи

$$\begin{cases} \sum_{i=1}^{n} x_{ijk} = a_{jk} (j = \overline{1, m};\ i = \overline{1, l}), \\ \sum_{j=1}^{m} x_{ijk} = b_{ik} (i = \overline{1, n};\ k = \overline{1, l}), \\ \sum_{k=1}^{l} x_{ijk} = c_{ij} (i = \overline{1, n};\ j = \overline{1, m}), \\ x_{ijk} \geq 0,\ i = \overline{1, n};\ j = \overline{1, m};\ k = \overline{1, l}. \end{cases}$$

Задача такого типа совместна тогда и только тогда, когда вектор

правой части принадлежит конусу, порожденному векторами столбцов матрицы левой части.

В общей постановке эта проблема решена только для задач, у которых один из параметров (n, m или l) равен 2, и для задач с параметрами $n \times m \times l = 3 \times 3 \times 3$.

Эксперимент проводился на вычислительном кластере ННГУ. Использовалось до 8 рабочих станций на базе процессора Intel Pentium 4 (1300 МГц), 10/100 Fast Ethernet card.

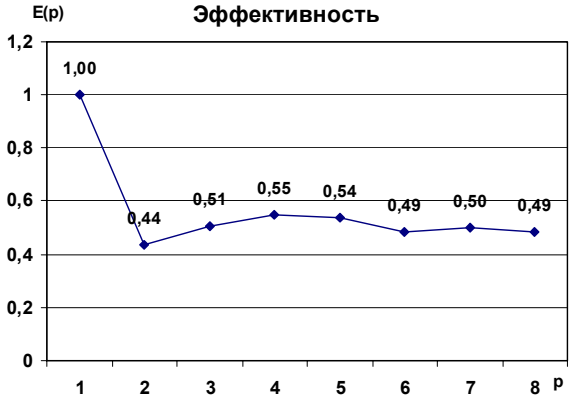
Результаты эксперимента

На вход подавались матрицы, составленные из коэффициентов правых частей трехиндексных транспортных задач размерностей 4×3×3, 4×4×3, 4×4×4.

В 1995 г. Золотых Н.Ю. были получены результаты для задачи размерности 4×3×3. Они также были получены в ходе данного вычислительного эксперимента и полностью подтверждены ранее полученными данными. Полученные условия состоят из 9 условий баланса и 717 однородных неравенств, которым должны удовлетворять коэффициенты a_{ij}, b_{ij}, c_{ij}.

Для случая 4×4×3 результаты работы программы сведены в таблицы и графики. Время работы t приведено в секундах, p – число задействованных процессоров. Определение используемых терминов см., например, в [6].

p	T, с
1	35
2	40
3	23
4	16
5	13
6	12
7	10
8	9



Полученные условия состоят из 10 условий баланса и 4948 однородных неравенств, которым должны удовлетворять коэффициенты a_{ij}, b_{ij}, c_{ij}.

Для случая размерности 4×4×4 были проведены эксперименты для 1 и 6 процессоров.

p	*t*
1	38580 с. = 10 ч. 43 м.
6	7344 с. = 2ч. 04 м.

$S(6) = 5{,}25,$
$E(6) = 0{,}878.$

Литература

1. *Шевченко В.Н.* Линейное и целочисленное линейное программирование. Горький: Изд-во ГГУ, 1976.
2. *Черников С.Н.* Линейные неравенства. М.: Наука, 1968.
3. *Золотых Н.Ю.* Расшифровка пороговых и близких к ним функций многозначной логики. Дисс. … кандидата физ.-мат. наук. Нижний Новгород. 1998.

4. *Золотых Н.Ю.* Программная реализация алгоритма Моцкина–Бургера построения остова многогранного конуса // Труды Второй международной конференции «Математические алгоритмы». Под ред. М.А. Антонца, В.Е. Алексеева В.Н. Шевченко. – Н. Новгород: Изд-во Нижегородского университета, 1997. С. 72–74.
5. *Group W.*, *Lusk E.*, *Skjellum A.* Using MPI. Portable Parallel Programming with the Message-Passing Interface. The MIT Press, 1994.
6. *Гергель В.П.*, *Стронгин Р.Г.* Основы параллельных вычислений для многопроцессоцных вычислительных систем. Н.Новгород: Изд-во ННГУ, 2001.

ОРГАНИЗАЦИЯ УДАЛЕННОГО ДОСТУПА К КЛАСТЕРНЫМ УСТАНОВКАМ

К.Е. Афанасьев, А.В. Демидов, С.В. Стуколов

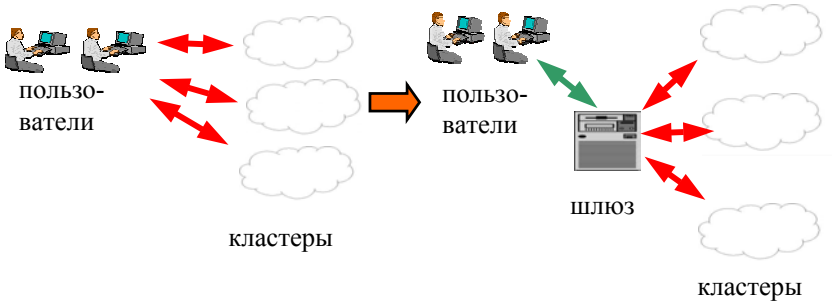
Кемеровский Государственный университет

С каждым годом во всем мире растет популярность параллельных вычислений. Однако если в Европе и Америке на массивно-параллельных компьютерах уже давно снимают художественные фильмы, то в России тема параллелизма актуальна пока что только в наиболее продвинутой в этом отношении академической среде. Конечно, купить многопроцессорную (порядка нескольких десятков процессоров) систему могут лишь немногие институты и ВУЗы, однако собрать кластер (или по крайней мере прототип) на базе серийных ПК под силу не то что ВУЗу, но и практически любому человеку.

Наиболее дешевым вариантом является создание Linux-кластера с использованием одного из свободно распространяемых средств распараллеливания [1]. При этом возникает одна проблема: ОС Linux, которая славится своей надежностью и безопасностью, никогда не была особо дружественна к конечному пользователю. Для работы на кластере программисту приходится изучать основы работы с командным интерпретатором shell или осваивать одну из графических оболочек (GNOME, KDE). К тому же, если на нескольких кластерах установлены разные системы (или разные версии), то для работы на каждом из них требуется отдельная подготовка.

Решить эту проблему призван программный комплекс «Вычислительный портал» (далее ВП), разрабатываемый в настоящее время в КемГУ. ВП по замыслу разработчиков должен предоставлять дружественный пользователю интерфейс удаленного доступа к кластерным установкам, причем этот интерфейс не изменяется при работе с различными кластерами.

Это реализуется посредством добавления еще одного звена в цепочку «Пользователь – Кластер» таким образом, что схема взаимодействия Пользователя с вычислительными ресурсами принимает вид:



Исходя из логики работы приложения и некоторых других критериев, для реализации ВП была выбрана архитектура «клиент-сервер». В данном случае клиентская часть по действиям пользователя генерирует команды и посылает их серверу. Сервер интерпретирует команды и посылает их на кластер, после чего информирует клиента об успешном или неуспешном завершении операции и переходит в режим ожидания новой команды.

Унификация доступа к различным кластерам обеспечивается за счет того, что клиентская часть общается всегда лишь с сервером, а сгенерировать специфические для выбранного кластера команды – уже задача сервера.

Серверная часть представляет собой демон ОС UNIX, который обеспечивает принятие запросов на сетевое соединение, принимает от пользователей команды в виде запросов, исполняет их и возвращает результаты пользователю. Серверное ПО содержит подсистему для работы с сетевым соединением, процедуры для работы с ОС, логику для хранения пользовательских объектов на диске и в памяти, логику для синтаксического разбора запросов и формирования ответов.

Клиентская часть предоставляет интерфейс пользователя. Она отображает сессию пользователя, позволяет ему делать запросы и наблюдать результаты, приходящие в виде ответов сервера. Содержит в себе визуальный интерфейс, подсистему работы с сетевым соединением и логику для визуального представления пользовательских настроек, его программ и экземпляров программ. Клиентская часть инвариантна и в базовом варианте представляет собой Win32-приложение, требующее установки на машине пользователя. В будущем перспективными видятся Web и E-mail интерфейсы.

Для взаимодействия клиента и сервера между собой разрабатывается собственный протокол прикладного уровня, работающий поверх протокола TCP/IP. Новый протокол сделан несколько похожим на протокол работы с электронной почтой POP3. Пользователь соединяется с сервером, получает информацию о версии сервера и протокола и отдает команды серверу. Команды состоят из ключевого слова и параметров команды, завершаемых символом «перевод строки» на каждую команду сервер сообщает общий результат ее выполнения и краткое состояние сессии пользователя и переходит в режим ожидания новой команды. Использование на транспортном уровне протоколов стека TCP/IP позволяет использовать ВП как в локальных, так и глобальных сетях, тем самым привлекая удаленных пользователей.

Однако работа удаленных пользователей требует изменения правил работы ВП, что связано с усилением безопасности как сервера, так и кластерных установок, к которым открывается доступ. Защита сервера ВП заключается в правильной с точки зрения безопасности настройке операционной системы, установке firewall'ов и других программных средств предотвращения внешних атак. Защита пользователя состоит в шифровании передаваемого между клиентом и сервером. Это может быть реализовано, например, заменой интерфейса сокетов на интерфейс SSL (Secure Socket Layer). И, наконец, защита вычислительных ресурсов подразумевает замену стандартных системных библиотек ввода/вывода на «интеллектуальные» библиотеки, которые не позволяют пользовательскому процессу выходить за рамки определенного физического и логического дискового пространства.

К настоящему моменту проведен анализ предметной области, составлены требования пользователей к системе, построены структурная и функциональная модели [2]. Также имеется реализованный прототип системы, включающий клиентскую и серверную часть.

Литература

1. *Афанасьев К.Е.*, *Гудов А.М.*, *Стуколов С.В.* Вопросы развития высокопроизводительных вычислений в Кемеровском государственном университете. Теоретические и прикладные вопросы современных информационных технологий: Материалы всероссийской научно-технической конференции. Улан-Уде: Изд-во ВСГТУ, 2001, т.1, , С. 35-40.
2. *Вендров А.М.* CASE-технологии. Современные методы и средства проектирования информационных систем. http://www.citforum.ru/ case.shtml

РАСЧЕТЫ РАЗВИТИЯ НЕУСТОЙЧИВОСТИ НА ГРАНИЦЕ РАЗДЕЛА ГАЗОВ ПО МЕТОДИКЕ «МЕДУЗА» С ВЫДЕЛЕНИЕМ КОНТАКТНОЙ ЛИНИИ В СМЕШАННЫХ ЯЧЕЙКАХ

Р.А. Барабанов, О.И. Бутнев, С.Г. Волков, Б.М. Жогов

РФЯЦ–ВНИИЭФ, г. Саров

Численное моделирование развития неустойчивости Рихтмайера–Мешкова привлекает большое внимание прикладных математиков на протяжении последних 30 лет [1–9]. Связано это в первую очередь с практической значимостью понимания данного явления, особенно в связи с изучением начальной стадии развития турбулентного перемешивания. Кроме того, моделирование нелинейной стадии развития неустойчивости является хорошим тестом для любой методики с точки зрения ее возможностей по описанию сильных струйных и вихревых течений. Развитие вычислительной техники и технологии параллельных вычислений позволило в последние годы перейти к более детальному исследованию неустойчивости Рихтмайера–Мешкова.

Нерегулярная явная свободно лагранжева методика МЕДУЗА [10, 11], разработанная в математическом отделении РФЯЦ–ВНИИЭФ, хорошо зарекомендовала себя при решении газодинамических задач с геометрически сложными начальными и граничными условиями и с сильными сдвиговыми течениями. Основными особенностями этой методики являются: определение всех сеточных величин (скалярных и векторных) в центре ячейки, переменный разностный шаблон для чис-

ленного интегрирования дифференциальных уравнений и возможности изменения топологии сетки в процессе решения. При использовании однообластной модели решения задачи на границах сред возникают смешанные ячейки, решение в которых ищется на основании многокомпонентного подхода без явного выделения контактной границы. В процессе решения задачи осуществляется переопределение соседства на основании известного принципа Дирихле (принцип метрической близости), выброс или добавление новых точек, а также пересчет сеточных функций со старой сетки на новую сетку путем их наложения.

Для моделирования контактных границ в методике используются смешанные ячейки, то есть ячейки, состоящие из нескольких (обычно двух) веществ. Алгоритм расчета смешанных ячеек приведен в [13, 14]. В докладе изложен алгоритм построения выделенных контактных линий в смешанной многоугольной невыпуклой ячейке, содержащей произвольное количество смесей, на основе поля объемной концентрации каждого из веществ.

Изложены результаты тестовых расчетов на сходимость задачи о падении плоской ударной волны на заполненную тяжелым газом прямоугольную область [12]. Полученные результаты сравниваются с результатами эксперимента и с результатами расчета без выделения контактной линии в смешанных ячейках. Кроме того, приводится сравнение с результатами расчетов по регулярной лагранжевой методике Д [15] с автоматическим вызовом приказа ИСПВЕЛ глобального исправления фрагмента сетки [16] с применением метода концентраций [17]. Результаты расчетов хорошо согласуются между собой, а также с результатами эксперимента.

Распараллеливание методики осуществляется по следующим принципам: геометрическая декомпозиция с частичным перекрытием данных, использование библиотеки обменов стандарта MPI, минимизация количества обменов, выделение множеств точек, прилежащих к межпроцессорным границам (расчет в первую очередь этих точек позволяет совмещать счет с обменами путем использования асинхронных обменов). В докладе приведены результаты измерений эффективности распараллеливания расчета задачи о развитии неустойчивости Рихтмайера–Мешкова с числом процессоров до 32. Для всех вариантов расчетов эффективность распараллеливанияния составила не менее 85%, что позволяет сделать вывод о масштабируемости расчетов при достаточной вычислительной нагрузке процессоров (количество счетных точек на процессоре более 10000).

Литература

1. *Lord Kelvin*. Mathematical and Phisical Papers, IV, Hyddrodynamics and General Dynamics, Cambridge, England, 1910.
2. *Taylor J.* The instability of liquid surfaces when accelerated in a direction perpendicular to their planes, I. Proc. Roy Soc., 1950, J. London. Ser. A, vol. 201, No. 1065, p.192.
3. *Richtmyer R.D.* Taylor Instability in Shock Acceleration of Compressible Fluids. Communications on Pure and Applied Mathematics, vol. XIII, 1960. P. 297–319.
4. *Мешков Е.Е.* Неустойчивость границы раздела двух газов, ускоряемой ударной волной. Изв. АН СССР, Механика жидкости и газа, №5, 1969. С. 151–158.
5. *Meyer K.A., Blewett P.J.* Numerical Investigation of the Stability of the Shock-Accelerated Interface between Two Fluids. The Physics of Fluids, vol.15, number 5, May 1972, p. 753–759.
6. *Youngs D.L.* Time dependent multi-material flow with large distortion // in Numerical Methods for Fluid Dynamics (K.W. Morton and J.H. Baines, eds.), Academic Press (1982).
7. *Youngs D.L.* Numerical simulation of turbulent mixing by Rayleigh–Taylor instability. Physica D 12, 1984, p. 32.
8. *Howell B.P.*, *Ball G.J.* Damping of mesh-induced errors in Free-Lagrange simulations of Richtmyer–Meshkov instability. Shock Waves, 10, 2000, p. 253–264.
9. *Kotelnikov A.D.*, *Ray J.*, *Zabusky N.J.* Vortex morphologies on reaccelerated interfaces: Visualization, quantification and modeling of one- and two-mode compressible and incompressible environments. Physics of fluids, vol.12, N. 12, 2000, p. 3245–3263.
10. *Глаголева Ю.П.*, *Жогов Б.М.*, *Кирьянов Ю.Ф. и др.* Основы методики МЕДУЗА численного расчета двумерных нестационарных задач газодинамики // Численные методы механики сплошной среды. Новосибирск, 1972, Т.3, № 2. С. 18–55.
11. *Sofronov I.D.*, *Rasskazova V.V.*, *Nesterenko L.V.* The use of nonregular nets for solving two-dimensional nonstationary problems in gas dynamics //Numerical Methods in Fluid Dynamics Ed.by N.N. Yanenko, Ju.I. Shokin. M.:Mir Publishers, 1984. P. 82–121.
12. *Жогов Б.М.*, *Клопов Б.А.*, *Мешков Е.Е.*, *Пастернак В.М. Толшмяков А.И.* Численный расчет и сравнение с экспериментом задачи о прохождении плоской ударной волны через «тяжелый» угол // Вопро-

сы атомной науки и техники. Сер. Математическое моделирование физических процессов. 1992. Вып. 3. С. 59–65.
13. *Барабанов Р.А.*, *Бутнев О.И.*, *Волков С.Г.*, *Воронин Б.Л.*, *Жогов Б.М.*, *Пронин В.А.*, *Софронов И.Д.* Распараллеливание счета двумерной задачи газовой динамики на неструктурированных сетках на ВС МП-3 с распределенной памятью // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2000. Вып.2. С. 3–9.
14. *Barabanov R.A.*, *Butnev O.I.*, *Pronin V.A.*, *Sofronov I.D.*, *Volkov S.G.*, *Voronin B.L.*, *Zhogov B.M.* (RFNC–VNIIEF). 2D Gas Dynamics Problem Computation Parallelization On Unstructured Grids On Distributed-Memory Computer., PDPTA-2000, Las-Vegas, CSREA Press, p.2899–2906.
15. *Софронов И.Д.*, *Дмитриев Н.А.*, *Дмитриева Л.В.*, Малиновская Е.В. Методика расчета двумерных нестационарных задач газодинамики в переменных Лагранжа. ИПМ АН СССР, препринт №59, 1976.
16. *Будников В.И.*, *Делов В.И.*, *Логинова О.К.* Методика и программа глобального автоматического исправления двумерной сетки внутри математических областей в комплексе Д // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2001. Вып.3. М. 49–59.
17. *Будников В.И.*, *Вершинин В.Б.*, *Делов В.И.*, *Садчиков В.В.*, *Хитева Е.С.*, *Чернышев Ю.Д.* Расчет смешанных ячеек в двумерном комплексе программ Д // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2001. Вып.3. С. 3–13.

АЛГОРИТМИЧЕСКАЯ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МЕТОДОВ ПРИВЕДЕННЫХ НАПРАВЛЕНИЙ ДЛЯ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ СИСТЕМ

О.В. Бастракова

Марийский государственный университет

Введение

Для решения задач нелинейного программирования с ограничениями-неравенствами ранее разработана единая схема построения численных методов на основе понятия «приведенного направления» [1]. В рамках единой схемы были теоретически обоснованы как известные, так и новые алгоритмы, относящиеся к следующим широко распространенным группам методов оптимизации – методы точных, дифференцируемых и барьерных штрафных функций, методы центров, основанные на внешних и внутренних функциях расстояния, методы возможных направлений. Для данных групп методов реализованы алгоритмы первого и второго порядков, использующие соответственно линейные и криволинейные траектории движения к новой итерационной точке [2], мультистадийные алгоритмы, использующие методы разных групп и порядков для построения итерационных приближений на разных стадиях процесса решения [3].

Возникающие в настоящее время задачи нелинейной условной оптимизации большой размерности, требуют высокой скорости решения. Организация методов приведенных направлений по единой схеме позволяет эффективно реализовать алгоритмы различных групп методов оптимизации для высокопроизводительных вычислительных систем.

Единая схема методов приведенных направлений

Рассматривается задача

$$\min_{x\in\Omega} f_0(x); \quad \Omega = \{x \in E^n \,\big|\, f_j(x) \le 0, j \in J\}, \tag{1}$$

где E^n есть n-мерное евклидово пространство, J –- конечное множество индексов, $f_j(x),\ j \in J \cup \{0\}$ – гладкие функции.

Для решения задачи (1) строится итерационный процесс $x_{k+1} = x_k + s^{(i)}(t_k), \quad i = 1,2$, где $s^{(i)}(t_k)$ – траектория движения из текущей итерационной точки x_k, $t_k \ge 0$ – шаг вдоль траектории.

Для построения приведенного направления в текущей итерационной точке используются различные способы построения множества активных ограничений $J(x,\varepsilon), \varepsilon \ge 0$, использующие значения функций ограничений задачи $f_j(x),\ j \in J$.

Направление движения к следующей итерационной точке определяется следующим образом:

$$s = -P(x)\,z - R(x)(v + f(x)). \tag{3}$$

Вектора $z \in E^{n-r}$, $v \in E^r$, $r = |J(x,\varepsilon)| \le n$ – параметры направления, $f(x) = (f_j(x))_{j \in J(x,\varepsilon)}$. Матрицы $P(x)$ и $R(x)$, определяются следующими условиями:

$$A^T P(x) = 0, \quad A^T R(x) = I_r, \quad A = (f_j(x))_{j \in J(x,\varepsilon)}, \tag{4}$$

Для построения матриц $P(x)$, $R(x)$, удовлетворяющих условиям (4) используется LQ-разложение матрицы A^T.

$$P(x) = HQ_2^T, \quad R(x) = HQ_1^T L^{-1}, \quad A^T H = (L;0)Q, \quad Q^T = (Q_1^T, Q_2^T).$$

Здесь $G = HH^T$ – симметричная положительно определенная $n \times n$ матрица, H – нижняя треугольная, L – левая треугольная $r \times r$ матрица, Q – ортогональная $n \times n$ матрица.

Траектория движения определяется из:

$s^{(1)}(t) = t\,s$ – для линейной траектории;

$s^{(2)}(t) = t\,s - \frac{1}{2}t^2 R(x)\,q,$ – для криволинейной

$q = (< s, f_j'' s >)_{j \in J(x,\varepsilon)}$ траектории движения.

Следующие формы векторов z, v обеспечивают убывание целевой функции при движении вдоль траектории $s^{(i)}(t), i = 1, 2$:

$$z = -g(x),$$

$$z = -(P(x)^T G\, P(x))^{-1} P(x)^T f_0'(x), \quad G = \left(f_0(x) + \sum_{j \in J(x,\varepsilon)} u^j f_j(x) \right)''_{xx},$$

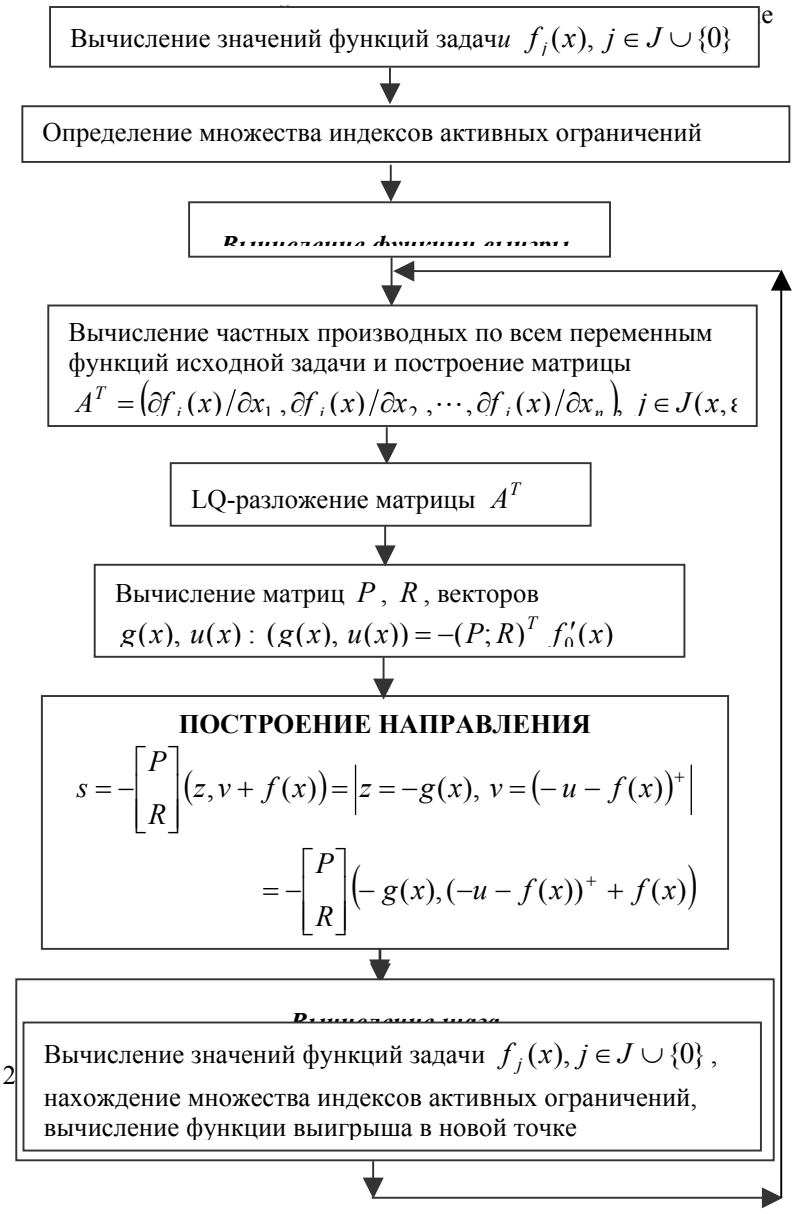
$$v = [-u(x) - f(x)]^+,$$

$$g(x) = -P(x)^T f_0'(x), \quad u(x) = -R(x)^T f_0'(x).$$

Для вычисления шага $t \ge 0$ строится убывающая последовательность $\left\{1, \frac{1}{2}, \frac{1}{4}, \ldots\right\}$ до тех пор, пока не будет достигнуто убывание функции выигрыша, являющейся средством оценки качества приближений.

Методы приведенных направлений для параллельных вычислительных систем

Параллельная реализация единого вычислительного алгоритма методов приведенных направлений выполняется на основе следующих принципов [4]:



– вычислительный алгоритм методов приведенных направлений работает с функциями задачи, значения которым могут вычисляться параллельно;
– единая схема методов приведенных направлений базируется на матрично-векторных умножениях, которые поддаются эффективному распараллеливанию.

В качестве средства реализации выбрана библиотека MPI для языка программирования C++. Средства MPI выбраны в силу своей распространенности, доступности и универсальности.

Процесс решения задачи (1) по единой схеме методов приведенных направлений можно представить кратко в виде блок-схемы (см. рис.)

Для вычисления значений функций задачи и их частных производных при построении множества активных ограничений, и вычислении функции выигрыша в данной системе предлагается использовать вычисление значения алгебраического выражения по двоичному дереву [5]:

– алгебраическое выражение функции считывается в виде строки;
– проводится синтаксический анализ строки, и строится двоичное дерево выражения. Если выражение было введено неправильно, выдается сообщение об ошибке и номер позиции, в которой обнаружена ошибка;
– на основе построенного дерева для алгебраического выражения, строятся деревья частных производных по заданным переменным;
– по дереву вычисляется значение введенного выражения.

Использование деревьев для выполнения задач вычислительного характера значительно упрощает проектирование системы в целом, полностью исключается перекомпиляция и необходимость иметь дополнительные файлы для ее организации.

Как следствие, увеличивается мобильность системы, значительно уменьшается требуемое для работы системы дисковое пространство. Для изменения задачи пользователю необходимо ввести лишь строковое выражение, автоматическое вычисление производных избавляет пользователя от аналитического дифференцирования и ввода большого количества информации.

Пусть p – количество процессов в системе. Обозначим $d_f = p \operatorname{div} m,\ m_f = p \bmod m,\ d_x = p \operatorname{div} n,\ m_x = p \bmod m.$

Функции и переменные распределяются между процессами блочным образом равномерно. Каждый процесс отвечает за свой набор переменных и функций, указанных в таблицах 1–2 В таблицах приведены индексы соответствующих процессу функций и переменных.

Таблица 1

Распределение функций по процессам

k	Индексы приписанных процессу k функций		
$0...m_f$	$k(d_f+1)$	···	$k(d_f+1)+d_f$
$m_f+1...p-1$	kd_f+m_f+1	···	$(k+1)d_f+m_f$

Таблица 2

Распределение переменных по процессам

k	Индексы приписанных процессу k переменных		
$0...m_x-1$	$k(d_x+1)+1$	···	$(k+1)(d_x+1)$
$m_x...p-1$	kd_x+m_x+1	···	$(k+1)d_x+m_x$

После загрузки задачи строки-выражения для функций исходной задачи рассылаются посредством коллективного обмена по всем процессам. Каждый процесс выполняет следующие действия:

1. Преобразование строки, содержащей алгебраическое выражение, в дерево.
2. По построенному дереву берутся частные производные каждой функции по заданному набору переменных.
3. Если функция не входит набор функций, определенных для данного процесса, то после взятия частных производных, дерево удаляется.

В результате, в каждом процессе хранится набор деревьев определенных для данного процесса функций и деревья производных всех ограничений исходной задачи по заданному набору переменных. Следует отметить, что вся процедура построения деревьев выполняется только один раз, после загрузки новой задачи, построение деревьев выполняется параллельно каждым процессом по мере поступления строк-выражений функций, а, следовательно, на построение деревьев затрачивается достаточно малое количество временных ресурсов до начала вычислительного процесса. Кроме того, распределенное хранение деревьев позволяет экономить пространство оперативной памяти.

Перед началом итеративного перехода параллельно каждым процессом вычисляются значения функций по построенным деревьям, результаты значений хранятся в локальных массивах каждого процесса и используются только им. В результате описанного выше способа хранения значений функций и их производных, матрица A^T автоматически распределена между процессами по строкам: после того как уже получен набор индексов активных ограничений, каждый процесс выбирает среди производных всех функций только те, которые входят в активный набор.

Как видно из блок-схемы, параллельный алгоритм построен таким образом, что практически на каждом этапе вычислений производятся операции в n-мерном пространстве. Такой подход позволяет реализовать равномерное блочное распределение матриц и векторов по процессам, которое не зависит от количества элементов в активном множестве, а зависит только от размерности задачи, не меняющейся в процессе вычислений. Все матрично-векторные умножения, необходимые для нахождения направления движения, проводятся каждым процессом для своего, определенного на предварительном этапе, блока данных.

Таким образом, при выполнении матрично-векторных операций для нахождения направления движения удается избежать множественных пересылок данных, что увеличивает производительность системы.

Литература

1. *Izhutkin V.S.*, *Petropavlovskii M.V.* The Hybrid Methods of the Reduced Direction with Different Cost Functions for Solving Nonlinear Extremal Problems // Proc. 17th International Conference «Mathematical Methods in Economics», Jindrichuv Hradec, Czech Republic, 1999. P. 127–132.
2. *Ижуткин В.С.*, *Бастракова О.В.* Мультистадийный метод приведенных направлений для задачи нелинейного программирования// Сборник докладов Международной конференции «Распределенные системы: оптимизация и приложения в экономике и науках об окружающей среде» (DSO'2000), Екатеринбург, 2000. С. 139–141.
3. *Izhutkin V.S.*, *Bastrakova O.V.* Parallel Algorithm of Barrier Cost Function Method in the Uniform Scheme of Methods of Reduced Directions// Abstracts of International Conference on Operations Research (OR2000), Dresden, 2000. P. 26–27.

4. *Ортега Дж.* Введение в параллельные и векторные методы решения линейных систем. М.: Мир, 1991.
5. *Кнут Д.* Искусство программирования для ЭВМ. Т. 1: Основные алгоритмы. М.: Мир, 1976.

РАСЧЕТЫ НИЗКОСКОРОСТНОГО РЕЖИМА РАЗВИТИЯ ДЕТОНАЦИИ ВВ

С.М. Бахрах, Н.А. Володина, И.В. Кузьмицкий, М.Н. Леонтьев, К.В. Циберев

РФЯЦ–ВНИИЭФ, г.Саров

В работе приведены результаты двумерных расчетов по инициированию октогенового взрывчатого вещества посредством удара стальным сферическим осколком, а также сравнение расчетов с экспериментами, проведёнными в Институте Физики Взрыва.

Расчеты проводились на многопроцессорной ЭВМ с распределенной памятью в рамках кинетической модели «Ignition And Growth» реализованной в программном комплексе ЛЭГАК. Эта кинетика была разработана Тарвером и сотрудниками Ливерморской национальной лаборатории США на основе модели Зельдовича - Неймана – Дюринга. В этой модели в качестве механизма энерговыделения принимается модель «горящих пятен», а также предполагается, что первоначальное энерговыделение в лагранжевой точке после прихода в нее ударной волны начинается в небольших локальных образованиях, связанных с порами в гетерогенном твердом ВВ. Рост таких пятен происходит до тех пор, пока они не охватят всю массу лагранжевой точки. Выражение для скорости реакции имеют следующий вид:

$$\frac{dF}{dt} = \underbrace{I(1-F)^b\left(\frac{\rho}{\rho_0}-1-a\right)^x}_{0\le F\le F_{ig\max}} + \underbrace{G_1(1-F)^c F^d P^y}_{0\le F\le F_{G1\max}} + \underbrace{G_2(1-F)^e F^g P^z}_{F_{G1\max}\le F\le 1}$$

Видно, что каждый член в этом выражении действует в своем диапазоне изменения массовой концентрации продуктов взрыва (F). Первый член описывает инициирование части ВВ при рождении «горящих пятен» из-за схлопывания пор в материале, то есть первый член – член

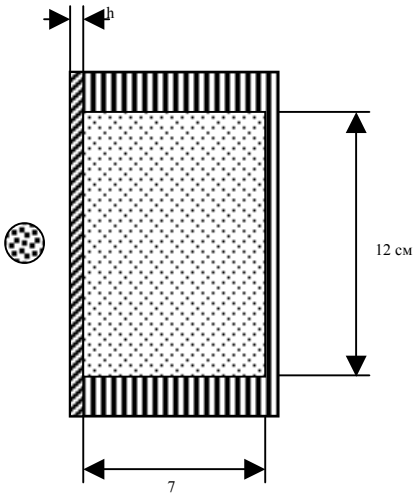
инициирования («*ignition*»). Для сильных ударных волн количество «горящих пятен» приблизительно равно исходному количеству пор. Второй член описывает возрастание скорости за счет увеличения «горящих пятен» в оставшемся не прореагировавшем ВВ. Третий член описывает быстрый переход к детонации при слиянии «горящих пятен» в оставшемся не прореагировавшем ВВ, что обеспечивает высокую скорость реагирования и генерацию пика высокого давления, который догоняет фронт волны и тогда устанавливается стационарная детонация.

Для описания свойств ВВ и ПВ использовалось уравнение состояние JWL (по первым буквам авторов Jones–Wilkins–Lee). Во ВНИИЭФ этот УРС одним из первых применил И.В. Кузьмицкий

$$\varepsilon = \frac{P}{\omega \cdot \rho} - \left(A \cdot \left(\frac{V}{\omega} - \frac{1}{R_1} \right) \cdot e^{-R_1 \cdot V} + B \cdot \left(\frac{V}{\omega} - \frac{1}{R_2} \right) \cdot e^{-R_2 \cdot V} \right) \cdot \frac{1}{\rho_0}$$

Для проверки работоспособности реализованной в комплексе программ ЛЭГАК кинетики были проведены серии одномерных расчетов в лагранжевой постановке по ударно-волновому инициированию ВВ. Результаты сравнивались с точными решением и решением, полученным по методике ИЗУМРУД. Полученные результаты хорошо согласуются. Стоит также отметить, что результаты расчетов, проведенных на эйлеровой сетке, согласуются с результатами на лагранжевой сетке.

Взрывчатое вещество диаметром 12 см, толщиной 7 см помещалось в стальной корпус. Нагружение ВВ осуществлялось стальным шариком диаметром 1.43 см, разгоняемым до фиксированной скорости 1.6 км/с. Интенсивность воздействия на ВВ варьировалась за счет изменения толщины *h* прикрывающей ВВ крышки – пластины из алюминия (рис.)



Расчеты проводились с пластинами толщиной 0,13 см, 0,14 см, 0,15 см, 0,2 см, 0,3 см и 2,0 см. на эйлеровой сетке с пространственным шагом

0,01 см. Таким образом, размерность задачи равнялась 1,1~1,2 млн. счетных ячеек. Расчеты проводились на многопроцессорной ЭВМ с распределенной памятью. Декомпозиция задачи проводилась по группам столбцов с использованием 60 процессоров. Эффективность распараллеливания составила порядка 50%.

В ходе расчетов получен следующий порог инициирования ВВ – толщина прикрывающей ВВ Al-пластины 0,14 см.

Следующим этапом расчетов было определение средней скорости отлёта индикаторной пластины на базе 0,2…0,7 см. Использовалась правая точка поверхности пластины на оси симметрии системы. В случае толщины прикрывающей ВВ пластины 0,2 см средняя скорость отлёта пластины составила величину ~ 0,36 км/с, что достаточно хорошо согласуется с результатами эксперимента: 0,40 км/с; при толщине прикрывающей пластины 0,3 см ~ 0,35 км/с, что также достаточно хорошо согласуется с результатами эксперимента: 0,39 км/с. При расчёте отлёта пластины-индикатора в случае толщины защитного экрана 2 см получена скорость ~ 0,17-0,19 км/с, что примерно в 2 раза меньше экспериментально определённой скорости отлёта индикаторной пластины 0,38 км/с.

Мы можем предположить, что различие скорости отлета индикаторной пластины в случае расчёта задачи и проведения эксперимента с Al-пластиной толщиной 2,0 см объясняется следующим обстоятельством: на картину развития детонации могут существенно влиять упругопластические свойства ВВ (напомню, что расчеты проводились в гидродинамическом приближении). Например, одномерные расчеты с учетом упругопластических свойств ВВ показали существенное увеличение нагружающего индикаторную пластину давления. В настоящий момент проводятся дополнительные исследования данного предположения. В целом же можно сказать, что в наших расчётах опытов в рамках кинетической модели «Ignition And Growth» воспроизведена картина постоянства средней на базе 0,2…0,7 см скорости отлёта индикаторной пластины, полученная в ходе её регистрации в эксперименте в области давлений, где с хорошей точностью работает гидродинамическое приближение.

СТРУКТУРНАЯ ОРГАНИЗАЦИЯ ОС ДЛЯ КЛАСТЕРНЫХ ВЫЧИСЛЕНИЙ

М.О. Бахтерев

Уральский государственный университет, Екатеринбург

Работа посвящена разработке и реализации операционной системы для поддержки вычислений на кластерах. В работе предпринимается попытка не просто слепо следовать уже существующим принципам построения операционных систем, а выдвигать и реализовывать некие разумные предложения, которые могли бы повысить эффективность работы операционной системы. Кроме этого, также предпринимается попытка наделить операционную систему двумя особенностями, необходимыми при вычислениях на кластерах (особенно больших), которые, по-видимому, до сих пор никем для малобюджетных систем удовлетворительно не реализованы. Первая из них – это возможность удобно отлаживать параллельные программы, вторая – это возможность периодически сохранять состояние приложения (набора процессов на разных узлах кластера) так, чтобы было можно продолжить вычисления с момента этого сохранения, когда произойдет сбой на одном из узлов кластера, где выполняется приложение (обычно, системы с таким свойством называют устойчивыми).

К настоящему моменту разработаны две подсистемы из тех, без которых ни одна операционная система не обходится – это система управления задачами и система управления памятью. Именно о них хотелось бы сделать доклад, и после него выслушать критику и пожелания участников семинара.

Опишем основные, на наш взгляд, отличия предлагаемых подходов от подходов, реализованных в широко распространенных ОС.

Во-первых, предлагается разбить ядро операционной системы на задачи, выполняемы по правилу FIFO. В принципе, такой метод организации работы применяется в планировщике Linux 2.X.X, однако, также предлагается выполнять задачи с учетом их приоритета, что, теоретически, позволит повысить эффективность работы операционной системы с устройствами ввода/вывода.

Во-вторых, предлагается внести изменения в логическую организацию виртуального адресного пространства процессов. Самое главное из них заключается в том, чтобы организовать такую трансляцию страниц, при которой ядро в любой момент времени может получить

доступ к области данных процесса по тем же адресам, что и сам процесс. Это, теоретически, сильно упростит реализацию некоторых функций ядра (в том числе и тех, что связаны с обеспечением устойчивости) и сэкономит память, которая в широко распространённых системах тратится на то, чтобы создать структуры для отображения некоторой области виртуальной памяти ядра на всю физическую память компьютера.

И напоследок несколько слов о реализации. Сейчас мы пишем планировщик, который нацелен на работу на IA-32 PC, мы выбрали целевой именно эту платформу, во-первых, из-за доступности ее самой, и, во-вторых, из-за наличия свободно распространяемых ее эмуляторов.

FORTRAN OPENMP/DVM – ЯЗЫК ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ ДЛЯ КЛАСТЕРОВ

В.А. Бахтин, Н.А. Коновалов, В.А. Крюков, Н.В. Поддерюгина

Институт прикладной математики им. М.В. Келдыша РАН, Москва

Кластеры. В настоящее время наиболее распространенной системой массового параллелизма становится SMP-кластер. Кластер состоит из вычислительных узлов, объединенных некоторой сетью. Каждый узел является многопроцессорной системой (мультипроцессором) с общей памятью. Кроме этого существует тенденция объединения нескольких кластеров в сеть. Объединение кластеров отличается неоднородностью производительности процессоров и неоднородностью сети (скорость передачи данных). В идеале система параллельного программирования для этого типа кластеров должна объединять модели параллелизма с распределенной и общей памятью. Важнейшими компонентами такой системы являются средства балансировки загрузки неоднородных процессоров и способы сглаживания неоднородности сети с целью повышения производительности.

OpenMP-модель является стандартом параллельного программирования для мультипроцессоров. Эта модель программирования базируется на комбинации директив для компилятора, вызовов функций системы поддержки и системных переменных. Директивы оформля-

ются в последовательной программе в виде специальных комментариев. Директивы указывают участки программы, которые должны выполняться последовательно или параллельно.

Основным достоинством модели является простота (относительно модели нитей) и мобильность среди архитектур с общей памятью.

Отметим основные недостатки данной модели:

- OpenMP-программы могут выполняться только на мультипроцессорах;
- требуется явная синхронизация доступа к общим данным;
- недетерминированность выполнения нитей.

Первый недостаток обусловлен сущностью модели с общей памятью. Выполнение нитей и синхронизации производится в непредсказуемом порядке, что затрудняет анализ, прогноз и оптимизацию производительности.

DVM-модель является высокоуровневой моделью спецификации параллелизма без явной ориентации на модель с общей или распределенной памятью. Модель программирования базируется на директивах спецификации параллелизма. Директивы оформлены в последовательной программе в виде специальных комментариев. В DVM-модели существуют два уровня параллелизма: задачи и параллельные циклы. Кроме этого определены директивы в языке и механизмы в системе поддержки для управления балансировкой загрузки и производительностью параллельного выполнения.

Модель DVM не требует явной синхронизации при обработке общих данных. DVM-программы могут выполняться как на системах с распределенной памятью, так и на системах с общей памятью. Первая возможность обеспечивается системой поддержки DVM, которая на нижнем уровне базируется на MPI. Вторая возможность обеспечивается как через MPI, так и с помощью компилятора Fortran-DVM→OpenMP Fortran. Несмотря на высокий уровень абстракции модели DVM показала достаточную эффективность параллельного выполнения (сравнимую с MPI) как на тестах (NAS Benchmarks), так и на производственных задачах.

В данном докладе рассматривается расширение OpenMP Fortran моделью DVM. При разработке проекта преследовались следующие основные цели:

- расширение сферы применения модели DVM (язык OpenMP Fortran);

– расширение сферы применения OpenMP Fortran (однородные и неоднородные кластеры с распределенной памятью).

Программы на языке OpenMP Fortran, специфицированные директивами Fortran-DVM могут выполняться в двух режимах

– в среде OpenMP, т.к. DVM-директивы «невидимы» для компиляторов OpenMP Fortran,
– в среде DVM, так как компилятор Fortran OpenMP/DVM обрабатывает и DVM-директивы, и OpenMP-директивы.

На наш взгляд переход программиста от модели OpenMP к модели DVM будет происходить относительно легко, так как эти модели близки друг к другу. Отметим только основные факторы «близости» моделей:

– высокий уровень абстракции моделей (уровень DVM выше уровня OpenMP);
– одинаковые технологические принципы реализации моделей (неизменяемость последовательной программы, директивы параллелизма в виде спецкомментариев и т.п.).

В докладе также рассматриваются средства управления балансировкой загрузки относительно неоднородного кластера.

ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ ИТЕРАЦИОННЫХ МЕТОДОВ РЕШЕНИЯ УРАВНЕНИЯ ПУАССОНА

Н.Н. Богословский, А.О. Есаулов

Томский государственный университет

Введение

Применение высокопроизводительных вычислительных систем является одним из наиболее важных направлений развития вычислительной техники. Это вызвано не только тем, что производительные мощности обычных последовательных систем ограничены, но и постоянной необходимостью решения задач, требующих больших вычислительных возможностей чем те, которые может обеспечить существующая серийная вычислительная техника. Так современные проблемы моделирования климата, генной инженерии, создания лекарств, задачи экологии и др. – требуют для своего решения ЭВМ с произво-

дительностью на 2-3 порядка выше, чем производительность обычных компьютеров. В наше время создана вычислительная техника, которая позволяет решать данные задачи. Но необходимо разрабатывать алгоритмы и программы, способные решать задачи такого рода.

Математическая постановка задачи

Рассмотрим уравнение в частных производных эллиптического типа, называемое уравнением Лапласа

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \qquad (1)$$

вместе с граничными условиями:

$$x = 0 : u = 1; \quad x = L_x : u = 1; \quad y = 0 : u = 1; \quad y = L_y : u = 1. \qquad (2)$$

Решение данного уравнения будем находить на прямоугольной области, которая показана на рис. 1. На границе области зададим граничные условия первого рода, имеющие вид (2).

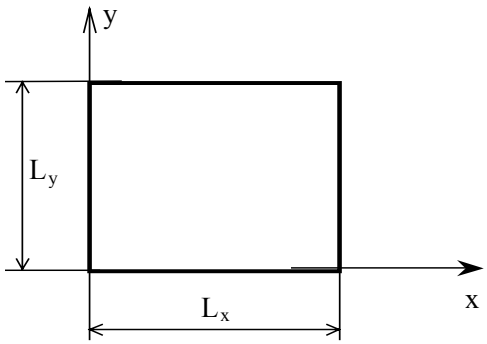


Рис. 1. Расчетная область

Методы расчета

Задачу (1)–(2) будем решать численно, методом конечных разностей. Для этого первоначально покроем область исследования равномерной сеткой

$$\overline{\omega}_h = \{(x_i, y_j), x_i = i\,h_x, y_j = j\,h_y,\ i = \overline{0, N}_x,\ j = \overline{0, N}_y,$$
$$h_x = L_x / N_x,\ \ h_y = L_y / N_y\}.$$

Дифференциальные операторы в уравнении Лапласа аппроксимируем конечно разностными формулами со вторым порядком погрешности аппроксимации.

Тогда дифференциальной задаче можно поставить в соответствие разностную задачу следующего вида:

$$\begin{cases} \dfrac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h_x^2}+\dfrac{u_{i,j+1}-2u_{i,j}+u_{i,j-1}}{h_y^2}=0, & i=\overline{1,N_x-1},\\ & j=\overline{1,N_y-1},\\ u_{0,j}=1, \quad j=\overline{0,N_y}, \\ u_{N_x,j}=1, \quad j=\overline{0,N_y}, \\ u_{i,0}=1, \quad i=\overline{0,N_x}, \\ u_{i,N_y}=1, \quad i=\overline{0,N_x}. \end{cases} \tag{3}$$

Разностная задача (3) представляет собой систему линейных алгебраических уравнений относительно неизвестных $u_{i,j}$ ($i=\overline{0,N_x}$; $j=\overline{0,N_y}$). Число уравнений равно числу неизвестных. Далее для удобства положим $h_x=h_y, N_x=N_y=N$. Для решения СЛАУ (3) использовались итерационные метод Якоби, явный метод Булеева и метод верхней релаксации SOR.

Метод Якоби. [1] Заметим, что матрица системы (3) является примером диагонально разреженной матрицы и содержит только пять ненулевых диагоналей независимо от величины N. И формула итерационного метода Якоби для решения уравнений (3) имеет следующий вид:

$$u_{i,j}^{k+1}=\frac{1}{4}(u_{i+1,j}^{k}+u_{i-1,j}^{k}+u_{i,j+1}^{k}+u_{i,j-1}^{k}) \tag{4}$$
$$i,j=1,\ldots,N-1;\ \ k=0,1,\ldots.$$

где k – номер итерации.

Метод SOR. [1] Определим

$$u_{i,j}^{k+1}=u_{i,j}^{k}+\omega(\hat{u}_{i,j}^{k+1}-u_{i,j}^{k}). \tag{5}$$

Параметр ω называется параметром релаксации, где $\hat{u}_{i,j}^{k+1}$ вычисляется по методу Гаусса–Зейделя.

Формула (5) определяет итерационный метод последовательной верхней релаксации SOR (successive overrelaxation). Возвращаясь к решению нашей задачи получаем

$$u_{ij}^{k+1} = 0{,}25\,\omega(u_{i+1,j}^{k} + u_{i-1,j}^{k+1} + u_{i,j+1}^{k} + u_{i,j-1}^{k+1}) + (1-\omega)u_{ij}^{k};$$
$$i = 1,\ldots,N-1; \quad j = 1,\ldots,N-1; \quad k = 0,1,2,\ldots \tag{6}$$

Явный метод Булеева (*ЯМБ).* [2] Будем искать решение системы пятиточечных уравнений

$$ap_{i,j}u_{i,j} = ae_{i,j}u_{i+1,j} + aw_{i,j}u_{i-1,j} + an_{i,j}u_{i,j+1} + as_{i,j}u_{i,j-1} + b_{i,j}$$

в виде

$$u_{i,j} = P_{i,j}u_{i+1,j} + Q_{i,j}u_{i,j+1} + R_{i,j} \tag{7}$$

с неопределенными пока коэффициентами $P_{i,j}$, $Q_{i,j}$, $R_{i,j}$. Выражая с помощью этого рекуррентного соотношения величины $u_{i-1,j}, u_{i,j-1}$, подставляя их в исходное уравнение, добавляя величину $-\theta_{i,j}(aw_{i,j}Q_{i-1,j} + as_{i,j}P_{i,j})u_{i,j}$ в правую и левую части уравнения, преобразуя и сравнивая полученный результат с (7), имеем:

$$R_{i,j}^{k} = \frac{b_{i,j} + aw_{i,j}R_{i-1,j}^{k-1} + as_{i,j}R_{i,j-1}^{k-1} + aw_{i,j}Q_{i-1.j}(u_{i-1.j+1}^{k-1} - \theta_{i,j}u_{i,j}^{k-1})}{g_{i,j}} +$$
$$+ \frac{as_{i,j}P_{i,j-1}(u_{i+1,j-1}^{k-1} - \theta_{i,j}u_{i,j}^{k-1})}{g_{i,j}},$$

$$P_{i,j} = \frac{ae_{i,j}}{g_{i,j}}; \quad Q_{i,j} = \frac{an_{i,j}}{g_{i,j}} \quad i = 1,\ldots,N_x - 1, \quad j = 1,\ldots,N_y - 1,$$

где $g_{i,j} = ap_{i,j} - aw_{i.j}P_{i-1,j} - as_{i,j}Q_{i,j-1} - aw_{i,j}Q_{i-1,j}\theta_{i,j} - as_{i,j}P_{i,j-1}\theta_{i,j}$

$$u_{i,j}^{k} = P_{i,j}u_{i+1,j}^{k} + Q_{i,j}u_{i,j+1}^{k} + R_{i,j}.$$

Таким образом, каждая итерация состоит из двух этапов: прямого и обратного хода при выбранной упорядоченности узлов.

Параллельная реализация

Распараллеливание по данным заключается в том, что производится разрезание области данных на части и в каждой подобласти вычисления производятся одновременно. Для двумерной области возможны два варианта разбиения: одномерное и двумерное. Рассмотрим одномерное разбиение области. Наша исходная область представляет из себя матрицу, в ячейках которой располагаются узлы сетки. Матрица имеет размерность $N \times N$. Одномерное разбиение производим по столбцам, то есть каждому процессу выделяется определенное количество столбцов, исходной матрицы (*j*-х элементов $u_{i,j}$), для которых он будет проводить вычисления. На рис. 2 в левой части схематично показано разбиение области между процессорами. Нижняя часть выделенного круга представляет область, рассчитываемую 0 – процессором, а верхняя – 1-м. Поскольку для вычисления по формулам, представленным выше, для расчетов на 0 – процессоре нужны данные с первого столбца 1 – процессора, то после каждого шага итерационного процесса необходимо осуществлять пересылку граничных значений между процессорами, то есть, необходима пересылка последнего столбца 0 – процесса 1 – процессу и пересылка первого столбца 1 – процесса 0 – процессу. В остальных процессах рассматривается аналогичная ситуация и пересылки граничных столбцов проводится для соседних процессов.

Результаты

Были реализованы программы для одномерного разбиения, в которых используется метод Якоби, SOR и явный метод Булеева и двумерное разбиение для метода Якоби. Так же написаны программы, реализующие синхронный и асинхронный обмен. Вычисления прово-

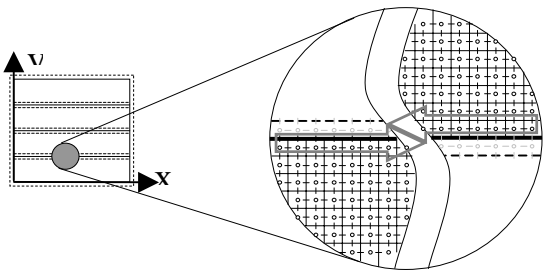


Рис. 2. Одномерная декомпозиция

дились на кластере ТГУ (9 двухпроцессорных узлов с топологией «звезда»).

Для анализа полученных данных использовалась характеристика ускорения, получаемая из времени выполнения основного цикла программы. Ускорение – это отношение времени выполнения программы на одном процессоре ко времени выполнения на p процессорах. По графику ускорения можно судить о том, насколько эффективна параллельная реализация.

На рис.3 приведены графики ускорения для метода Якоби и SOR полученные с использованием различных методов обмена граничными значениями между процессами (синхронный и асинхронный). Анализируя график, видно, что при использовании асинхронного метода обмена данными ускорение возрастает в 2 раза. Что говорит о том, что время выполнения программы уменьшается. Но при реализации асинхронного обмена возникают трудности, связанные с тем, что необходимо выполнять синхронизацию процессов после каждой итерации. Наибольшее ускорение, достигается на 8 процессорах и равно 6.3 на сетке 150x150.

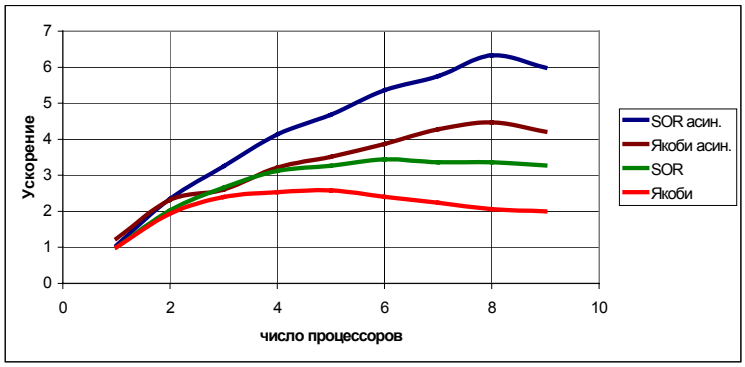


Рис. 3. Ускорение (сетка 150×150)

Обратимся к рис. 4, на котором представлены графики ускорения для всех трех методов. Для оценки ускорения бралось наилучшее время выполнение последовательных программ, реализующих эти три метода, и делилось на время выполнения программы на p процессорах. В данном случае, как видно из графиков на рис. 5, наилучшее время –

это время выполнения программы реализующий явный метод Булеева. Как видно из графиков наилучшее ускорение дает явный метод Булеева. Значение ускорения меньшее 1 (метод Якоби рис. 4), означает что время выполнения на p процессорах больше времени выполнения на 1 процессоре программы для ЯМБ.

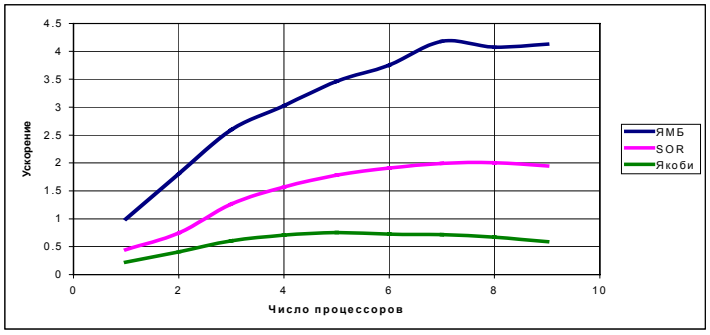


Рис. 4. Ускорение (сетка 200×200)

На рис. 5 приведены графики зависимости времени выполнения программы от числа процессоров. Наименьшее время расчета имеет

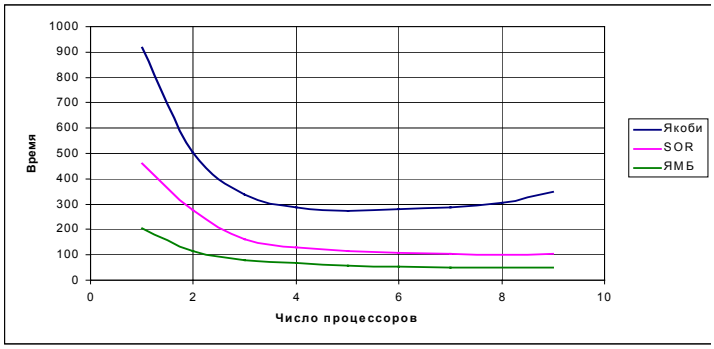


Рис. 5 Время выполнения программы в зависимости от числа процессоров (сетка 200x200).

явный метод Булеева ($\theta = 0{,}75$). Это объясняется тем, что ЯМБ имеет лучшую сходимость. В этом можно убедится, обратившись к рис.6 На этом рисунке графики показывают зависимость порядка невязки

($\lg(r_N)$) от номера итерации. Быстрее всего сходится ЯМБ, поэтому и время расчета меньше.

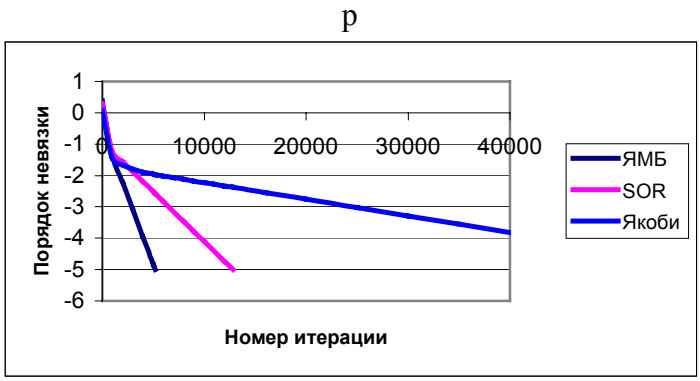


Рис.6. Невязка (число процессоров 8, сетка 200x200)

Заключение

Представлены результаты численного решения задачи Дирихле для уравнения Лапласа в прямоугольнике с помощью итерационных методов на многопроцессорных вычислительной технике. Расчеты показали преимущество применения асинхронного способа обмена информации между процессами перед синхронным. При сравнении разных итерационных методов получено, что метод Булеева с компенсацией значительно опережает метод Якоби и SOR по быстродействию, причем с ростом числа используемых процессоров это отличие возрастает.

Данная работа выполняется при поддержке ФЦП «Интеграция» проект Ф0039

Литература

1. *Ортега Дж.* Введение в параллельные и векторные методы решения линейных систем. М.: Мир, 1991.
2. *Ильин В.П.* Методы не полной факторизации для решения алгебраических систем. М.: Наука. Физматлит, 1995.

РАСПАРАЛЛЕЛИВАНИЕ ПО НАПРАВЛЕНИЯМ ПРИ РЕШЕНИИ ДВУМЕРНОГО УРАВНЕНИЯ ПЕРЕНОСА В КОМПЛЕКСЕ САТУРН-3 С ИСПОЛЬЗОВАНИЕМ ИНТЕРФЕЙСА OPENMP

А.И. Бочков, В.А. Шумилин

РФЯЦ-ВНИИЭФ, г. Саров

Комплекс программ САТУРН [1–2] предназначен для численного решения на ЭВМ широкого класса стационарных и нестационарных многомерных задач переноса. Ввиду большой размерности эти задачи требуют большого времени счета. С целью сокращения времени счета в комплексе программ САТУРН реализованы различные алгоритмы распараллеливания.

Для двумерного нестационарного уравнения переноса нейтронов на ЭВМ с распределенной памятью с использованием программного интерфейса MPI реализованы:

- распараллеливание по энергетическим группам;
- распараллеливание по математическим областям;
- мелкоблочное распараллеливание для задач с регулярной пространственной сеткой.

Для двумерного стационарного уравнения переноса нейтронов - распараллеливание по энергетическим группам как с использованием программного интерфейса MPI, так и с использованием интерфейса OpenMP.

Интерфейс OpenMP представляет собой стандарт для программирования на масштабируемых симметричных мультипроцессорных системах (SMP-системы) в модели общей памяти. В стандарт OpenMP входят спецификации набора директив компилятора, процедур и переменных среды.

Интерфейс OpenMP идеально подходит для разработчиков, желающих быстро распараллелить свои вычислительные программы с большими параллельными циклами. Разработчик не создает новую параллельную программу, а просто добавляет в текст последовательной программы OpenMP-директивы.

В докладе приводится реализованный в рамках комплекса САТУРН метод распараллеливания по направлениям при решении двумерного нестационарного уравнения переноса нейтронов с использованием интерфейса OpenMP.

Постановка двумерной нестационарной задачи включает в себя систему групповых уравнений переноса нейтронов, записанных в цилиндрической системе координат:

$$\frac{\partial}{\partial t}\left(\frac{N_i}{\upsilon_i}\right)+\frac{\partial}{r\cdot\partial r}\left(r\cdot\sqrt{1-\mu^2}\cdot\cos\varphi\cdot N_i\right)+\frac{\partial}{\partial z}\left(\mu\cdot N_i\right)-$$
$$-\frac{\partial}{\partial\varphi}\left(\frac{\sqrt{1-\mu^2}\cdot\sin\varphi\cdot N_i}{r}\right)+\alpha_i\cdot N_i=F_i, \qquad (1.1)$$

где:

$$F_i=\frac{1}{2\cdot\pi}\cdot\sum_j\beta_{ji}\cdot n_j^{(0)}+\frac{1}{2\cdot\pi}\cdot Q_i \qquad (1.2)$$

Здесь: t – время; $z(t)$, $r(t)$ – цилиндрические координаты положения частицы; μ, φ – угловые переменные; $\upsilon(t, z, r)$, $\alpha(t, z, r)$, $\beta(t, z, r)$, $Q(t,z,r)$ – характеристики среды; $N(t,z,r,\mu,\varphi,\upsilon)$ – поток частиц;

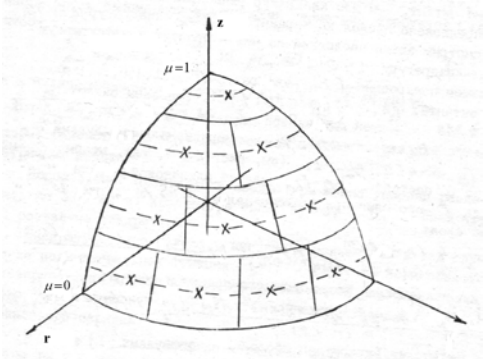
$n^{(0)}=\int_{-1}^{1}d\mu\int_{0}^{\pi}Nd\varphi$.

Уравнения (1.1), (1.2) требуется решить в области:

$$D=\left\{(z,r)\in L,-1\le\mu\le1,0\le\varphi\le\pi\right\}.$$

Область L – половина сечения тела вращения плоскостью, проходящей через ось $\vec{z}$ и образующую l.

Для дискретизации уравнения по угловым переменным применяются S_n-квадратуры (рис.)



Так как в уравнении переноса (1.1) нет производной по переменной μ, а для аппроксимации уравнения по угловым переменным используются S_n-квадратуры, то это позволяет применить при решении уравнения переноса метод распараллеливания по угловой переменной μ. Переменная μ – косинус угла между направлением полёта частиц и осью симметрии z. В программе решения двумерного уравнения переноса нейтронов было проведено распараллеливание цикла по угловой переменной μ. Данный метод не зависит от числа счётных областей и типа пространственной сетки, что позволяет его использовать на различных производственных задачах.

В качестве характеристик эффективности распараллеливания использовались следующие функции:

$S_p = t_1 / t_p$ – ускорение;

$E_p = t_1 / (p \cdot t_p) \cdot 100\%$ – эффективность распараллеливания.

В зависимости от постановки тестовых задач и числа задействованных процессоров эффективность распараллеливания решения двумерного нестационарного уравнения переноса нейтронов составила 50–100%.

Литература

1. *Шагалиев Р.М.*, *Шумилин В.А.*, *Беляков И.М. и др.* Организация комплекса САТУРН // ВАНТ. Сер.: Математическое моделирование физических процессов. 1992, вып. 3. С. 49–51.
2. *Шагалиев Р.М.*, *Шумилин В.А.*, *Алексеев А.В. и др.* Математические модели и методики решения многомерных задач переноса частиц и энергии, реализованные в комплексе САТУРН-3 // ВАНТ. Сер.: Математическое моделирование физических процессов. 1999, вып. 4. С. 20–26.

РАЗРАБОТКА МЕТАСИСТЕМЫ ПОДДЕРЖКИ РАСПАРАЛЛЕЛИВАНИЯ ПРОГРАММ НА БАЗЕ МНОГОЦЕЛЕВОЙ СИСТЕМЕ ТРАНСФОРМАЦИЙ ПРОГРАММ

А.А. Букатов

Ростовский государственный университет, Ростов-на-Дону

При переносе существующих программ на многопроцессорные вычислительные системы (МВС) возникает проблема преобразования этих программ к виду, обеспечивающему их эффективное выполнение на параллельных МВС. Эта же проблема актуальна и при необходимости разработки прикладных программ МВС программистами, не владеющими методами параллельного программирования. Указанная проблема до определенной степени решена для векторно-конвейерных МВС и МВС с архитектурой SMP, то есть для МВС с общей (разделяемой) памятью [1]. Производители ряда таких систем включают в состав их программного обеспечения автоматически распараллеливающие трансляторы, генерирующие параллельный исполнимый код по исходной последовательной программе. Однако для МВС с распределенной памятью и многокомпьютерных кластеров задача создания промышленных распараллеливающих трансляторов и по сей день не решена. Научные исследования в области создания средств автоматического и/или автоматизированного распараллеливателя последовательных программ для МВС указанных классов сохраняют свою актуальность. Результатом таких исследований, проводимых научными коллективами из различных стран, являются «материализованные» в виде экспериментальных систем распараллеливания программ (СРП) методы распараллеливания программ для МВС с распределенной памятью. Указанные методы реализуются в СРП в процедурном виде. Это означает, что развитие СРП при усовершенствовании какого либо из используемых методов или при заимствовании метода, разработанного в другом научном коллективе, потребует внесения изменений в реализацию СРП. Ввиду того, что реализация различных распараллеливающих преобразований зачастую «пронизывает» программный код всей СРП, развитие таких СРП и, тем более, заимствование находок коллег, работающих в той же области, становится весьма непростой задачей. Решение этой задачи может быть значительно упрощено, если реализация СРП основана на явном непроцедурном описании распа-

раллеливающих преобразований программ, «интерпретируемом» универсальным ядром СРП. Разработка методов непроцедурного описания распараллеливаюших преобразований программ и методов построения универсальной системы выполнения описанных в непроцедурной форме распараллеливающих преобразований является предметом настоящей работы.

В докладе рассматриваются общая организация специализированной оболочки экспертной системы, обеспечивающей создание систем автоматизации распараллеливания последовательных программ путем заполнения базы знаний экспертной системы о распараллеливающих преобразованиях программ. Эта экспертная система строится на базе создаваемой под руководством автора универсальной системе трансформаций программ. Знания о методах распараллеливания программ представляются в виде правил трансформации программ, задающих в наглядной непроцедурной форме параметризованные схемы распараллеливающих преобразований программ. Наглядная форма представления этих знаний позволяет, в частности, относительно просто выполнять развитие базы знаний о методах распараллеливания программ и, таким образом, обеспечивает возможность адаптации системы к новым и/или проблемно-ориентированным методам распараллеливания программ и к новым целевым параллельным платформам.

Создаваемая экспертная система должна обеспечивать возможность автоматического или полуавтоматического применения правил трансформации, описывающих различные распараллеливающие преобразования, к распараллеливаемой программе. При полуавтоматическом выполнении распараллеливающих преобразований эксперту предоставляется возможность определять порядок применения различных преобразований. Кроме того, при необходимости экспертная система может интерактивно обращаться к эксперту с запросами на оценку истинности некоторых неформализованных условий, влияющих на применимость тех или иных преобразований к конкретным конструкциям распараллеливаемой программы.

Остановимся подробнее на способе непроцедурного описания распараллеливающих преобразований в системе трансформаций программ и на организации программных средств, обеспечивающих выполнение преобразований программ, описываемых в виде правил трансформации. В работах [2, 3] автором предложен язык непроцедурного описания преобразований программ в виде схематических правил трансформации программ. Каждое из таких правил задает отображе-

ние некоторой схемы исходной программной структуры (семантической конструкции) в соответствующую ей схему распараллеленной программной структуры. Описание правила трансформации включает схему преобразуемой конструкции, называемую *входным образцом*, схему результата преобразования, называемую *выходным образцом*, и, возможно, *условие применимости правила*, зависящее от значений параметров входного образца. Правила трансформации могут иметь внешние параметры, значения которых указываются при применении правила.

В простейшем случае схема программной конструкции – это параметризованный по некоторым вложенным компонентам синтаксический шаблон программной конструкции. В более сложном случае схема программной структуры может включать несколько шаблонов взаимосвязанных (по параметрам или посредством указания семантических отношений) синтаксических конструкций, а также функциональные термы, вычисляющие значения вспомогательных параметров по значениям параметров схемы. Для каждого из синтаксических шаблонов, входящих в схему программной структуры, указывается синтаксический тип соответствующей конструкции, для каждого из параметров синтаксических шаблонов также указывается их синтаксический тип. Это позволяет повысить эффективность процесса поиска программных конструкций, сопоставимых с синтаксическим шаблоном, и унификации (определения значений) его параметров.

Условие применимости правила является логическим выражением, включающим значения параметров входного образца, а также встроенные и запрограммированные экспертом предикаты (функции, возвращающие в качестве результата логическое значение), фактическими параметрами которых являются значения параметров входного образца. В число встроенных предикатов входит интерактивный предикат, используемый для интерактивного запроса к проводящему распараллеливание эксперту для оценки истинности выводимого на терминал условия применимости с подставленными в него значениями параметров (компонентов) преобразуемой конструкции.

Таким образом, правило трансформации имеет следующий вид:

rule *rule-name* (*rule-parameters-list*)
    [var *variable-declarations-list* ;]
    compound-input-pattern && *enabling-condition*
    => *compound-output-pattern*
end

Здесь и далее имена нетерминальных конструкций выделены *курсивом*

compound-input-pattern ::= *pop-term* { & *pop-term* } □

pop-term:: = pattern-term | *procedural- term* □

pattern-term:: = identifying-expression : *schema* □

identifying-expression =::

<*s-type-name*>{.*relation-name*} [(*alias-name*)] |

alias-name{.*relation-name*} [(*alias-name*)] |

$*parameter-name*{.*relation-name*}[(*alias-name*)□

Здесь символ «□» является признаком конца синтаксического правила, символ «::=» – разделителем правой и левой частей правила, символ «|» – разделителем альтернатив правила, а квадратные и фигурные скобки используются для выделения необязательных или кратных конструкций соответственно.

Применение правила трансформации к преобразуемой программе включает следующие действия. Вначале выполняется поиск в тексте преобразуемой программы конструкции, сопоставимой с входным образцом и определение значений параметров входного образца (их значениями являются компоненты конструкции, сопоставленные параметрам образца). Затем выполняется проверка условий применимости правила, зависящих от параметров входного образца. В случае истинности этого условия выполняется замена исходной программной конструкции на выходной образец с подставленными в него значениями параметров.

Порядок применения правил может либо определяться экспертом в интерактивном режиме, либо описываться в виде сценария трансформации, определяющего последовательность применения правил на специальном языке. Базовыми операторами этого языка являются: оператор однократного применения правила к указанной программе или программной конструкции и оператор итеративного применения правила или группы правил до тех пор, пока применимо хотя бы одно правило. При этом взаимный порядок применения правил из одной итеративно применяемой группы правил недетерминирован. При взаимной независимости правил группы результат итеративного применения правил группы будет, тем не менее, детерминирован. Предусматриваются также средства явного «вызова» правил трансформации в выходных образцах других правил, что обеспечивает эффективную реализацию «многоступенчатых» преобразований

Отметим, что предложенный в работах [2, 3] язык трансформаций,

допускающий использование многокомпонентных входных и выходных образцов в отличие от языков других систем трансформаций программ [4] предоставляет средства описания нелокальных трансформаций. Тем самым обеспечивается возможность описания схем распараллеливания достаточно сложных программных структур, включающих несколько взаимосвязанных фрагментов, расположенных в различных местах исходной программы. Типичным примером такой структуры является, например, структура, включающая оператор цикла и операторы описания и инициализации переменных. Кроме того, язык обеспечивает возможность применения процедурных преобразований (в виде функциональных термов образцов) в случае, если требуемое преобразование данных выполняется сложными алгоритмами, наиболее выразительной формой описания которых является программа на обычном языке программирования. Типичным примером такого процедурного преобразования является решение системы диофантовых уравнений для определения параметров преобразуемого гнезда циклов.

Остановимся подробнее на организации системы поддержки трансформации программ. Система включает в свой состав средства (подсистемы), обеспечивающие:

- выполнение преобразования исходных текстов программ во внутреннее представление этих программ в БД и обратного преобразования;
- поддержку создания и модификации базы знаний экспертной системы (наборов правил и сценариев трансформации);
- автоматизированное выполнение трансформаций программ.

Внутренней формой представления трансформируемых программ является атрибутированное дерево абстрактного синтаксиса программы, дополненное семантическими связями (отношениями) и вспомогательными представлениями, такими как граф потока управления и граф зависимости по данным. Эти вспомогательные представления используются для эффективной реализации предикатов, требуемых для описания условий применимости распараллеливающих преобразований. Средства преобразования исходных текстов во внутреннее представление и обратно допускают настройку на синтаксис используемых языков программирования, вид атрибутированных деревьев абстрактного синтаксиса и на состав и способ создания семантических связей между синтаксическими объектами. Синтаксис языка задается L-атрибутной грамматикой, представленной в БНФ-подобном виде.

При настойке рассматриваемых средств на требуемый язык программирования правила грамматики компилируются в подпрограммы (функции) на языке Си, выполняющие разбор и порождение соответствующих конструкций. Это позволяет использовать для описания атрибутов нетерминальных символов, правил вычисления атрибутов и правил порождения семантических связей и вспомогательных технологических объектов соответствующие средства языка Си. Рассмотренные правила грамматики компилируются также в другие программные структуры, используемые в других подсистемах системы преобразования программ. На основе этого базового внутреннего представления программы конструируется ряд вспомогательных внутренних представлений программы, включающих, в частности, граф потока управления и граф зависимости по данным. Как отмечалось выше, указанные производные представления используются для эффективной реализации ряда базовых предикатов. Методы эффективной модификации указанных представлений при выполнении трансформаций основного внутреннего представления программы рассмотрены в работе [5]. В качестве примера приведем правило (несколько упрощенное), описывающее преобразование развертки циклов (см., например [1])

```
rule loop_unfold:
    var <name> $i, $n, $j;  <stmt> $loop_body, $loop_body_j;
    <loop_stmt>(L): for $i=1 to $n do  $loop_body
    && few_iterations(L) & no_jumps_in(L)
    =>
    instead_of(L):
    forall $j in (1:$n)
    $loop_body_j = apply(($i => $j), $loop_body)
end
```

В этом примере выходной образец генерирует последовательность тел в цикла, в которых выполняется замена (путем вызова вложенного правила трансформации) всех вхождений переменной цикла на константы 1, 2 и т.д.

Средства поддержки создания правил и сценариев трансформации включают в свой состав интерактивный конструктор образцов правил трансформации, базовые библиотеки предикатов и функций, предназначенных для использования в составе условий применимости правил и выходных образцов, и средства преобразования правил трансформации программ в исполнимую форму. Преобразование правил транс-

формации программ в исполнимую форму осуществляется путем преобразования каждого из правил в выполняющую соответствующую трансформацию процедуру на языке Си и последующей компиляции этой процедуры. Такой двухступенчатый подход обеспечивает возможность использования соответствующих конструкций языка Си для представления логических условий и функциональных термов, входящих соответственно, в условие применимости правила и во входной и выходной образцы. Это позволяет существенно упростить реализацию языка трансформаций программ за счет возложения на компилятор языка Си задач трансляции функциональных термов и логических выражений, входящих в описание правил трансформации. Более подробное рассмотрение этих средств выходит за рамки настоящей работы.

Средства автоматизированного выполнения трансформаций предоставляют программный и интерактивный интерфейсы для выполнения (применения) правил и сценариев трансформации. Интерактивный интерфейс обеспечивает возможность интерактивного применения правил трансформации экспертом, выполняющим распараллеливание программы. Наличие программных интерфейсов позволяет использовать в качестве языка написания сценариев трансформации универсальные языки программирования. В создаваемой версии системы трансформации программ в качестве такого языка используется язык Си.

Предложенные средства описания распараллеливающих преобразований программ были практически применены автором и его учениками (см., например, [6]) при разработке правил трансформаций для классических методов распараллеливания. Опыт применения указанных средств продемонстрировали достаточно высокую их выразительность. В настоящее время ведутся работы по макетной реализации рассмотренной системы трансформаций программ и по созданию на ее основе экспертной системы поддержки распараллеливания последовательных программ.

Литература

1. *Евстигнеев В.А.*, *Спрогис С.В.* Векторизация программ (обзор). // В сб. Векторизация программ: теория, методы, реализация. М., Мир, 1991. С. 246–271.
2. Букатов А.А. Разработка средств непроцедурной реализации распараллеливающих преобразований программ // Труды Всероссийской научной конференции «Фундаментальные и прикладные

аспекты разработки больших распределенных программных комплексов», Абрау-Дюрсо, 1998. С. 109–116.
3. *Букатов А.А.* Разработка средств построения систем преобразования исходных текстов программ // Информационные технологии, № 2, 1999. С. 22–25.
4. *Partch H.*, *Steinbruggen R.* Program transformation systems // ACM Computer Survey, 1983, v. 15, №3, P. 199–236.
5. *Луговой В.В.* Разработка механизмов модификации внутреннего представления программ в CASE-системе распараллеливания программ // Материалы конференции «Высокопроизводительные вычисления и их приложения», Черноголовка, 2000, с.133-136.
6. Жегуло О.А. Представление знаний о методах распараллеливания в экспертной системе поддержки распараллеливания программ // Искусственный интеллект, № 3/2001, Донецк: «Наука і освіта», 2001. С. 323–330.

ЦЕНТР ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ КОЛЛЕКТИВНОГО ПОЛЬЗОВАНИЯ РОСТОВСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

А.А. Букатов, В.Н. Дацюк, Л.А. Крукиер

Ростовский государственный университет, Ростов-на-Дону

Центр высокопроизводительных вычислений (ЦВВ) Ростовского государственного университета (РГУ) был создан в 1997 году на базе многопроцессорной вычислительной системы (МВС) nCUBE 2S. В 1999-2000 г. в рамках работ по проектам ФЦП «Интеграция» было выполнено развитие вычислительной базы ЦВВ РГУ. В настоящее время ЦВВ РГУ располагает следующими вычислительными и вспомогательными ресурсами.

1. МВС nCUBE 2S – классическая MPP (Massively Parallel Processing) система из 64-х вычислительных узлов, каждый из которых состоит из процессорного модуля производительностью около 2 Mflops и модуля оперативной памяти объемом 32 Мб. Таким образом, пиковая производительность системы 128 Mflops (на тесте LINPACK получена производительность 96 Mflops). Сбалансированная архитектура дан-

ной MPP системы (в смысле соотношения производительности процессора к скорости передачи данных) идеально подходит для многопроцессорных вычислений, однако из-за низкой производительности в настоящее время используется в основном для отладки программ и обучения студентов РГУ и сотрудников других научно-образовательных организаций технологиям параллельного программирования [1].

2. МВС Compaq Alpha DS20E – 2-х процессорная SMP (Symmetric Multiple Processing) система с общей памятью объемом 1 Гб и объемом дискового пространства 2x18 Гб. Пиковая производительность одного процессора 1 Gflops. На однопроцессорном тесте LINPACK достигается производительность 190 Mflops без использования специальных библиотек и 800 Mflops при использовании оптимизированной библиотеки CXML. На параллельной MPI-версии теста LINPACK получена производительность 1,5 Gflops. на двух процессорах. Используется главным образом для объемных научных расчетов.

3. МВС SUN Ultra 60 – 2-х процессорная SMP система с общей памятью объемом 1 Гб и объемом дискового пространства 18 Гб. Пиковая производительность одного процессора 800 Mflops. На однопроцессорном тесте LINPACK достигается производительность 85 Mflops без использования специальных библиотек и 470 Mflops при использовании оптимизированных библиотек LAPACK+ATLAS (больше чем при использовании фирменной SUN Performance Library – 390 Mflops). На параллельной MPI-версии теста LINPACK получена производительность 0,8 Gflops на двух процессорах. Компьютер выполняет также функции сетевой информационной службы (NIS) для авторизации пользователей и файлового сервера (NFS), экспортирующего домашние директории пользователей на все другие вычислительные системы (nCUBE, Linux-кластер, DS20E). Используется также как host-компьютер для работы с nCUBE2.

4. Linux-кластер – вычислительная система из 10 узлов, соединенных сетью Fast Ethernet через коммутатор Cisco Catalyst 2924. Каждый из узлов представляет собой компьютер Pentium III 500 Мгц, с 256 Мб оперативной памяти и 10 Гб жестким диском. Возможно исполнение как однопроцессорных, так и параллельных программ. В качестве основного средства параллельного программирования используется библиотека MPI. Пиковая производительность каждого процессора 500 Mflops. На однопроцессорном тесте LINPACK достигается производительность 44 Mflops без использования специальных библиотек и 300

Mflops при использовании оптимизированных библиотек MKL+ATLAS. На параллельной MPI-версии теста LINPACK получена производительность 2.5 Gflops для всего кластера в целом.

5. SUN Ultra 10 – однопроцессорная рабочая станция с оперативной памятью 256 Мб и объемом дискового пространства 20 Гб. Используется главным образом в учебном процессе для обучения UNIX технологиям и параллельному программированию (в качестве host-компьютера nCUBE2). Выполняет также множество вспомогательных функций (WWW сервера, FTP сервера, NFS сервера, Mail сервера). Управляет источником бесперебойного питания и выключением компьютеров в аварийных ситуациях.

Прикладное программное обеспечение всех вычислительных систем в максимально возможной степени унифицировано для обеспечения переносимости программ на уровне исходных текстов. Это было достигнуто установкой на всех системах коммуникационной библиотеки MPI, системы компиляции программ с языка HPF, базовой библиотеки линейной алгебры (ATLAS), однопроцессорной и параллельной версий библиотеки LAPACK, параллельной версии библиотеки для решения систем линейных алгебраический уравнений с разреженными матрицами (Aztec). Такая унификация позволила пользователям одни и тот же тексты программ компилировать на любой из вычислительных систем без какой-либо их модификации. Для облегчения процесса компиляции были написаны универсальные Makefile, в которых пользователям для перекомпиляции программы достаточно изменить название архитектуры, для которой должен быть создан исполнимый файл.

Из приведенных характеристик вычислительных систем, очевидно, что основу вычислительных ресурсов на сегодняшний день составляют системы Alpha DS20E, SUN Ultra 60 и Linux-кластер.

Опыт эксплуатации этих систем на протяжении 2000-2001 г.г. показал, что практически не возможно обеспечить их эффективное использование без автоматизированной системы управления заданиями. Для решения этой проблемы было изучено и опробовано несколько бесплатно распространяемых систем пакетной обработки заданий. Окончательный выбор был сделан в пользу системы OpenPBS и с конца 2001 г. эта система введена в промышленную эксплуатацию. Почти годовой опыт ее эксплуатации подтвердил правильность сделанного выбора. Только после ввода в эксплуатацию единой диспетчерской системы набор разнотипных компьютеров превратился в мощный вы-

числительный ресурс, представляющий собой по сути дела гетерогенный кластер.

Важнейшим достоинством системы OpenPBS является поддержка вычислительных узлов разной конфигурации и архитектуры, что явилось определяющим фактором для ЦВВ РГУ. Кроме этого, следует отметить простоту и удобство работы с ней как для администратора системы, так и для конечных пользователей.

Для конечных пользователей работа с OpenPBS сводится к запуску программ через специального вида командный файл. В этом файле указывается запускаемая на исполнение программа и требуемые задаче ресурсы: количество и архитектура процессоров, время решения задачи, некоторые переменные окружения. В случае отсутствия в данный момент свободных процессоров требуемой архитектуры, задача ставится в очередь до их освобождения. В отличие от других систем пакетной обработки заданий PBS допускает возможность интерактивного просмотра результатов выполнения программы, в том числе и в графическом виде.

Для администратора диспетчерской системы предоставляются широкие возможности для динамического изменения параметров системы, таких как создание и уничтожение очередей, подключение и отключение вычислительных узлов, установка предельных лимитов для пользователей и др. В ЦВВ РГУ система сконфигурирована таким образом, что на каждом из 14 процессоров может выполняться не более одного счетного процесса. Таким образом, одновременно может обрабатываться не более 14 обычных однопроцессорных программ. Для параллельных программ не поддерживается механизм выполнения одной программы на процессорах разной архитектуры. Пользователь должен заранее определиться, на какой из 3-х перечисленных выше систем должна быть выполнена его программа, и поставить ее в соответствующую очередь. Таким образом, на системах Alpha и SUN одна программа может использовать не более двух процессоров, а на Linux-кластере не более 10.

К достоинствам системы PBS следует отнести также наличие средств для оперативного контроля состояния очередей, вычислительных узлов, а также системы регистрации выполненных заданий. В табл. 1 представлен результат опроса на терминале состояния выполняющихся задач (имеется и графический вариант команды).

Таблица 1

rsusu2.cc.rsu.ru

Job ID	Usernam	Queue	Jobnam	SessID	NDS	TSK	Req'd Time	S	Elap Time
8505..	ipoc	ALPHA	q	7556	1	1	167:0	R	09:24
8524..	victor	ALPHA	apbs1	9953	1	1	30:00	R	00:10
8457..	zabotin	LINUX	ggg1	2288	1	1	160:0	R	27:36
8458..	zabotin	LINUX	ggg2	1342	1	1	160:0	R	27:32
8459..	zabotin	LINUX	ggg3	1135	1	1	160:0	R	27:28
8460..	zabotin	LINUX	ggg4	1100	1	1	160:0	R	28:24
8462..	zabotin	LINUX	ggg7	1435	1	1	160:0	R	27:15
8464..	zabotin	LINUX	ggg	1607	1	1	160:0	R	25:20
8525..	onikit	LINUX	lpbs10	3929	1	1	03:00	R	00:03
8526..	chikin	LINUX	jun	2666	2	1	07:00	R	00:01
8376..	oleg	SUN	scdis	2346	1	1	72:00	R	30:37
8517..	orlova	SUN	apbs	27620	1	1	45:00	R	01:27

Система регистрации выполненных заданий позволяет получать полную статистическую информацию за любой промежуток времени, что также не маловажно для центров коллективного пользования. В табл. 2 представлена выдача недельной статистики по всем вычислительным системам (возможна выдача статистики по каждой вычислительной системе).

Таблица 2

A total of 8 accounting files will be processed.
The first record is dated 08/30/2002,
last record is dated 09/06/2002

Using TOTAL
****

Usernam	#jobs	CPUclk h ours	Wallclk hours	Pct. Eff.	Average #nodes	Average q-hours	Proc. Usage
ipoc	3	137.66	167.02	82	1.00	49.01	6.21
oleg	60	100.81	101.36	99	1.00	0.00	3.77
chikin	12	14.46	15.66	92	1.85	0.00	0.58
onikit	85	15.32	16.68	92	1.00	0.00	0.62
orlova	14	219.67	221.16	99	1.00	9.16	8.23
victor	23	50.94	40.69	100	1.00	0.25	1.51
ishevt	4	0.16	0.26	61	1.00	5.28	0.01
zabotin	21	1087.30	1088.14	100	1.00	2.00	40.48
TOTAL	222	1626.31	1650.98	99	1.00	1.55	61.42

Несмотря на все свои достоинства, система OpenPBS не лишена и некоторых недостатков. Возможно, они устранены в коммерческой версии, которая, однако, практически недоступна в виду ее дороговизны. К основным недостаткам системы можно отнести отсутствие разбиения заданий на классы, например, по времени решения. Это позволило бы присваивать более высокий приоритет отладочным заданиям с небольшим временем решения. Можно, конечно, организовать специальные очереди для таких заданий, но это приведет к значительному увеличению числа очередей, что вряд ли будет способствовать увеличению удобства работы с системой. В отличие от основного сервера, параметры функционирования которого можно менять динамически специальной командой, в OpenPBS отсутствует динамическое управление параметрами функционирования планировщика. Для их изменения требуется перезапуск системы, что зачастую бывает очень трудно сделать при непрерывном поступлении заданий, учитывая, что некоторые из них требуют очень большого времени решения (5-6 суток). В процессе эксплуатации OpenPBS выявилась также слабая ее защищенность в случае возникновения не штатных ситуаций, таких как аварийное выключение вычислительного узла, занятого выполнением некоторого задания. При этом зачастую происходит зависание системы, и пропадают все задания, которые выполнялись в данный момент на всех узлах. Некоторые возможности PBS представляются избыточными, например, поддержка множества серверов, в то время, как другие желательные возможности отсутствуют. Поэтому, в настоящее время ведется работа по разработке системы пакетной обработки, которая бы более адекватно отвечает потребностям организации потока заданий пользователей на вычислительных системах ЦВВ РГУ.

Вычислительные ресурсы ЦВВ РГУ достаточно интенсивно используются для решения различных научных и прикладных задач. За 8 месяцев текущего года обработано свыше 6000 заданий, которые использовали более 36000 часов процессорного времени. Средняя загрузка составила около 50%. Основными потребителями высокопроизводительных ресурсов являются научно-исследовательские подразделения РГУ – НИИ Физики, НИИФОХ, ЮГИНФО и естественно-научные факультеты – мехмат, физфак и химфак. В частности, сотрудники НИИФОХ используют установленный на Alpha DS20E пакет GAMESS для решения прикладных задач квантовой химии. Отдел астрофизики НИИФ и кафедра астрофизики используют вычислительные ресурсы для моделирования процессов в галактике. Кафедра радиофизики фи-

зического факультета для моделирования процессов в низкотемпературной плазме и расчета волноводов сложных профилей. Кафедра информатики и вычислительного эксперимента механико-математического факультета и ЮГИНФО для решения задач гидродинамики и моделирования гидросистем.

К сожалению, следует констатировать, что значительную часть потока заданий, выполняемых на вычислительных системах ЦВВ РГУ, представляет собой обычные однопроцессорные программы. На сегодняшний день весьма ограниченный круг пользователей владеет технологией разработки параллельных программ. В основном это сотрудники ЮГИНФО (вычислительного центра) РГУ и кафедры ИВЭ механико-математического факультета. По-видимому, такое положение дел сохранится еще достаточно длительное время, поскольку обучение параллельному программированию охватывает в настоящее время довольно ограниченный круг студентов.

В настоящее время курс параллельного программирования в рамках спецкурса «СуперЭВМ» проходят студенты 3-х групп механико-математического факультета обучающихся по специальности прикладная математика. Общая численность обучающихся около 75 человек. Для обучения используется многопроцессорная система nCUBE2 и компьютер SUN Ultra 10. В качестве рабочих мест используется компьютеры учебных классов локальной сети РГУ. Обучение производится по оригинальным пособиям, разработанным сотрудниками ЮГИНФО РГУ. Электронный вариант пособия размещен на сервере суперкомпьютерного центра РГУ. В сентябре 2002 г. с согласия авторов это пособие было также размещено на сервере Томского госуниверситета.

Телекоммуникационная сеть РГУ обеспечивают необходимую пропускную способность каналов удаленного доступа к ресурсам ЦВВ РГУ не только из внутренних подсетей распределенной телекоммуникационной сети РГУ (100 Мбит/сек), но и из созданных на базе сети РГУ научно-образовательных телекоммуникационных сетей Ростовской области и Южного федерального округа. Скоростной внешний канал телекоммуникационной сети РГУ (45 Мбит/сек через сеть RBNet) обеспечивает эффективный доступ к вычислительным ресурсам ЦВВ РГУ из научно-образовательных сетей других регионов России и зарубежья. В частности, многие сотрудники РГУ, выезжающие в длительные научные командировки за рубеж, как правило, продолжают активно работать с высокопроизводительными ресурсами ЦВВ

РГУ. В качестве другого примера удаленного использования вычислительных ресурсов ЦВВ можно привести пример успешного проведения практики по параллельному программированию для студентов Пермского педагогического госуниверситета в 2001 году.

Литература

1. *Белоконь А.В.*, *Букатов А.А.*, *Дацюк В.Н.*, *Крукиер Л.А.* Создание учебного центра по суперкомпьютерным технологиям на базе суперкомпьютерного центра РГУ // Тезисы докладов Всероссийской научно-методической конференции «Телематика'2000», Санкт-Петербург, 2000. С. 126–128.

СМЕШАННАЯ МОДЕЛЬ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ OPENMP & MPI В ПРОГРАММЕ ГАЗОВОЙ ДИНАМИКИ

А.Н. Быков, А.С. Жданов

РФЯЦ–ВНИИЭФ, г. Саров

Исследование смешанной модели параллельных вычислений проведено на примере трёхмерной задачи газовой динамики о разлете эллипсоида (размерность задачи – 96 точек по каждому из направлений). За основу взята программа на основе интерфейса MPI.

Тестирование параллельной программы в кластерной вычислительной системе показало, что ее эффективность падает с ростом числа задействованных процессоров. Падение эффективности происходит значительно медленней, если используется один процессор вычислительного узла.

Распределение вычислительной работы в программе на основе OpenMP производится на уровне процедур. Алгоритм большинства процедур представляет собой трёхмерные вычислительные циклы.

Проведено тестирование программ на основе MPI и OpenMP& MPI на разном числе процессоров. Определено влияние аппаратных издержек при работе с общей памятью на эффективность параллельной программы.

В ходе работ получены следующие результаты.

1 Определена методика параллельных вычислений на основе интерфейса OpenMP;

2. Изучены причины, приводящие к росту накладных расходов при

работе процессов, использующих общую память.

3. Получена сравнительная оценка быстродействия программ на основе MPI и OpenMP& MPI.

4. Показана целесообразность применения средств автоматического преобразования последовательных алгоритмов в параллельные на основе OpenMP. Проведено исследование возможностей средств автоматического распараллеливания Visual KAP.

5. Применение модели OpenMP не приводит к значительным изменениям алгоритма программы.

6. Смешанная модель OpenMP& MPI не обеспечивает автоматически существенного выигрыша во времени по сравнению с MPI.

7. Основной причиной снижения быстродействия реальной программы в кластерной многопроцессорной системе являются аппаратные издержки при работе с общей памятью.

ДИНАМИЧЕСКАЯ БАЛАНСИРОВКА РАСПРЕДЕЛЕННЫХ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ НА НЕСКОЛЬКИХ КЛАСТЕРАХ ПРИ ЧИСЛЕННОМ РЕШЕНИИ ЗАДАЧ С ПОМОЩЬЮ СТАТИСТИЧЕСКИХ МЕТОДОВ МОНТЕ–КАРЛО

В.В. Бычков, Ю.П. Галюк, С.Е. Журавлева, В.И. Золотарев, В.П. Мемнонов

СПбГУ

Использование многокластерных систем позволяет существенно раздвинуть пределы, устанавливаемые производительностью имеющегося у исследователей кластера, и переходить к численному моделированию все более сложных явлений. При этом, будучи связаны через Интернет, все или часть кластеров могут находиться в разных географических точках. Такой способ вычислений будем называть метакомпьютингом. Для реализации параллельных вычислений в таких распределенных системах весьма важно иметь эффективную динамическую балансировку загрузки процессоров. В используемом нами методе прямого статистического моделирования Монте-Карло (ПСМ) для численного моделирования течений в случае сравнительно простых конфигураций задачи могут распараллеливаться просто по независи-

мым реализациям, выполняемых на отдельных процессорах. В такой ситуации необходимо только передать файлы с результатами сначала со всех процессоров каждого кластера на соответствующий ведущий процессор, а затем с этих последних на какой-нибудь выбранный из них для финальных осреднений и выдачи окончательных результатов. Однако, учитывая возможность динамического изменения условий работы разных кластеров, в том числе и динамического изменения самой производительности процессоров на некоторых кластерах, например, из-за подключения новых приложений пользователей, статическая балансировка путем раздачи в начальный момент фиксированных заданий для каждого процессора с учетом их производительности не эффективна; необходима динамическая балансировка. Циркуляция текущей информации для автоматического управления распределенными вычислениями у нас осуществлялась с помощью обмена между процессорами небольшими файлами с информацией о количестве реализаций, оставшихся до накопления необходимой статистической выборки. По нашему Интернет-каналу производительностью 1Mb/s они проходили быстрее 0,01 с, тогда как прохождение файлов с результатами размером порядка 100Kb требовало времени вплоть до 20 с, видимо, составляло основной временной штраф Интернет-коммуникаций. В докладе будет представлена подробно оценка эффективности применяемой системы динамической балансировки загрузки процессоров, которая показала, что эта эффективность в условиях метакомпьютинга всего на 2-3 процента ниже, чем при работе на том же кластере без дополнительных межкластерных соединений. Рассматривается также альтернативный подход к динамической балансировке для оценки возможного эффекта communication bottleneck при одновременном обращении нескольких процессов к одним и тем же данным, а также вопросы повышении надежности распределенных вычислительных систем. В частности, показано, что в некоторых аварийных случаях с отдельными компьютерами или даже целыми кластерами можно продолжать решение задачи и получать нужную выборку на оставшихся кластерах с автоматическим увеличением времени их работы.

ВЫБОР АЛГОРИТМА СИНХРОНИЗАЦИИ МОДЕЛЬНОГО ВРЕМЕНИ ПРИ РЕШЕНИИ ЗАДАЧ ДИСКРЕТНОГО ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ НА МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ

Т.В. Вознесенская

МГУ им. М.В. Ломоносова

Введение

Важную роль в создании и разработке распределенных вычислительных систем играет имитационное моделирование. Имитационные модели таких систем – сложные ресурсоемкие программные комплексы. Требования ко времени их моделирования зачастую становятся критическими. Одним из путей сокращения времени моделирования является использование внутреннего параллелизма моделей. Перспективным направлением распараллеливания имитационных моделей является распределенное имитационное моделирование (РИМ) [1]. Создание распределенной системы имитационного моделирования (РСИМ) требует разработки специального алгоритма синхронизации (АС) распределенно исполняемых частей модели [1].

Задача синхронизации модельного времени

Отличительной особенностью программ имитационного моделирования является то, что поведение всех процессов должно быть согласовано в едином модельном времени, то есть в том времени, в котором «живет» объект моделирования. Говоря об имитационном моделировании, далее везде будем иметь в виду дискретно-событийное имитационное моделирование [2].

Каждое событие в модели имеет вычисляемую в процессе выполнения метку – модельное время его наступления.

Будем рассматривать имитационные модели, которые

- состоят из множества процессов, взаимодействующих между собой через прием и посылку сообщений;
- имеют частично упорядоченное в едином модельном времени множество событий: если модельное время события a меньше модельного времени события b, значит, событие a наступило раньше события b, и наоборот; одновременные события, вообще говоря, не упорядочены.

Последовательный прогон таких моделей происходит следующим образом [2]. Вначале вычисляются временные метки готовых к обработке событий. Такие события заносятся в список (календарь) событий и упорядочиваются по неубыванию модельного времени. Затем модельное время устанавливается равным метке первого события, а само событие обрабатывается. В результате появляются новые готовые к обработке события. Они заносятся в календарь. Причем события со временем, меньшим текущего появиться не могут. И так до тех пор, пока в календаре есть события.

При распределении компонентов имитационной модели по нескольким процессорам появляется распределенное модельное время и распределенный календарь событий. Такая распределенность заключается в том, что каждый процесс имеет свой счетчик модельного времени – локальные часы и свой календарь событий, и надо уметь согласовывать между собой часы и календари всех процессов. Для сохранения отношения порядка на множестве событий модели в этом случае необходим АС.

На сегодняшний день разработан ряд алгоритмов синхронизации. При проектировании распределенной системы имитационного моделирования необходимо выбрать АС, который будет в ней использован. Если в системе реализовано несколько АС, необходимо уметь выбирать наиболее эффективный из них для конкретной модели. Для некоторых АС надо уметь настраивать параметры для заданной модели. Однако, выбор АС для конкретной задачи затруднен, поскольку не существует ни единого метода сравнения АС из всех четырех классов [3], ни способа их формального описания.

Методика выбора наиболее эффективного АС

В работе [4] автором разработана математическая модель взаимодействия имитационной модели (в дальнейшем приложения) и АС. Это взаимодействие описано с помощью случайных процессов. Входными параметрами модели являются интенсивности обмена сообщениями между процессами приложения и скорости продвижения их локальных часов. Эти параметры зависят от свойств приложения. От АС зависит вид случайного процесса.

На основе разработанной модели автором предложена методика выбора наиболее эффективного АС для заданного приложения [4]. Критерий эффективности следующий: лучшим для данного приложения является тот АС, который быстрее других продвигает модельное

время, то есть такой $\overline{AC} \in M_{AC}$, для которого $T_t(\overline{AC}) = T = \max_{AC_j \in M_{AC}} T_t(AC_j)$, где $T_t(AC_j)$ – модельное время имитационной модели в заданный момент физического времени *t* при алгоритме синхронизации $AC_j \in M_{AC}$, M_{AC} – множество анализируемых алгоритмов синхронизации.

Литература

1. *Райтер Р.*, *Вальран Ж.С.* Распределенное имитационное моделирование дискретно-событийных систем // М.: Мир, ТИИЭР, т.77, №1, янв. 1989. С. 245–262.
2. *Емельянов В.В.*, *Ясиновский С.И.* Введение в интеллектуальное имитационное моделирование сложных дискретных систем и процессов. Язык РДО. М.: «АНВИК», 1998.
3. *Казаков Ю.П.*, *Смелянский Р.Л.* Об организации распределенного имитационного моделирования. // Программирование, №2, 1994. С. 45–63.
4. *Вознесенская Т.В.* Исследование эффективности алгоритмов синхронизации времени для систем распределенного имитационного моделирования // Дис… канд. ф.-м. наук, МГУ, Москва, 2001.

ОБ ОРГАНИЗАЦИИ ПОДГОТОВКИ ПО ПАРАЛЛЕЛЬНОМУ ПРОГРАММИРОВАНИЮ В УГАТУ

Р.К. Газизов, С.Ю. Лукащук

Уфимский государственный авиационный технический университет

К.И. Михайленко

Институт Механики Уфимского научного центра РАН

Введение

В 2000 году в рамках федеральной целевой программы «Государственная поддержка интеграции высшего образования и фундаментальной науки» в Уфе на базе Уфимского государственного авиационного технического университета (УГАТУ) был создан Башкирский региональный центр высокопроизводительных вычислений (БРЦ ВВ). Вместе с УГАТУ в создании центра приняли активное участие ведущие научно-исследовательские институты региона: Институт Механи-

ки УНЦ РАН, Институт математики с вычислительным центром РАН, Институт проблем сверхпластичности металлов РАН, Институт органической химии УНЦ РАН.

Одновременно с созданием центра остро встал вопрос о подготовке кадров в области высокопроизводительных вычислений. На момент создания БРЦ ВВ был оснащен двумя вычислительными кластерами: 10-процессорным экспериментальным кластером на основе процессоров Intel Pentium III–500 и 12-процессорным кластером на основе процессоров Alpha 21164EV5. Однако квалифицированных специалистов, способных эффективно использовать эти вычислительные мощности, в БРЦ ВВ явно было недостаточно.

Следует отметить, что проблема нехватки квалифицированных специалистов в области использования многопроцессорных вычислительных систем характерна отнюдь не только для Башкирского центра. За последние несколько лет кластерные системы получили достаточно широкую популярность благодаря своей относительно невысокой (по сравнению с суперкомпьютерами) стоимости при высокой производительности, хорошей масштабируемости и простоте. Поэтому количество кластерных вычислительных систем в российских научно-исследовательских и учебных организациях за последние годы существенно возросло. Именно поэтому проблема подготовки специалистов для работы на многопроцессорных системах была признана общероссийской по результатам работы всероссийского совещания по высокопроизводительным вычислениям, прошедшего в УГАТУ в 2000 г. при участии руководства Министерства образования РФ и ректоров ведущих вузов России.

Для решения указанной проблемы в УГАТУ в 2001 году была создана новая кафедра высокопроизводительных вычислительных технологий и систем (ВВТиС), сотрудниками которой начата подготовка специалистов в области использования многопроцессорных вычислительных систем (МВС).

Об организации в УГАТУ процесса подготовки специалистов для работы на многопроцессорных системах

Принципиальная схема организации подготовки специалистов по многопроцессорным вычислительным системам показана на рис. 1.

В настоящее время подготовка специалистов для работы на МВС ведется в УГАТУ по двум основным направлениям:

1) подготовка в области организации распределенных вычисли-

тельных сред и уделенного доступа к ним;

2) подготовка в области постановки и решения естественно-научных задач с использованием МВС.

Рис. 1

В рамках первого направления проводится обучение студентов старших курсов специальностей 220600 – Организация технической защиты информации и 220100 – Вычислительные машины, комплексы, системы и сети, подготовка по которым организована на факультете информатики и робототехники УГАТУ. Целью подготовки является обучение высококвалифицированных системных и проблемных программистов для технического и программного обеспечения МВС, подготовки кадров по созданию телекоммуникационных сред высокопроизводительных технологий и подготовка персонала по обслуживанию локальных и глобальных вычислительных сетей.

Подготовка предполагает, в частности, изучение студентами таких курсов, как «Сетевые операционные системы», «Корпоративные вычислительные сети», «Параллельные алгоритмы и системы» и ряд других.

Второе направление предполагает подготовку специалистов в области математического моделирования с использованием МВС. В УГАТУ подготовка в данной области начата для студентов направления 657100 и специальности 010200 – Прикладная математика, поскольку соответствующие государственные стандарты наиболее полно отвечают требованиям, предъявляемым к уровню подготовки специалистов в названной области. Программа подготовки включает, в частности, такие спецкурсы, как «Параллельное программирование», «Математические модели и методы высокопроизводительных вычислений» и другие.

Вместе с начавшейся подготовкой студентов старших курсов, начиная с 2001 года в УГАТУ в рамках направления 657100 – Прикладная математика открыта новая специализация – Математическое и программное обеспечение высокопроизводительных вычислительных систем, и соответствующим образом произведено увеличение общего плана приема. По новой специализации кафедра ВВТиС УГАТУ является выпускающей. Программа подготовки по специализации предполагает изучение студентами следующих курсов:

- архитектура современных многопроцессорных вычислительных машин;
- системное программное обеспечение МВС;
- параллельное программирование: языки и методы;
- параллельные алгоритмы вычислительной математики;
- математические модели и методы высокопроизводительных вычислений;
- математическое моделирование и вычислительный эксперимент;
- параллельные технологии в прикладных задачах механики, теории управления, экономики, экологии, обработки изображений и других областей.

Очевидно однако, что подготовить полноценного специалиста только преподаванием некоторого набора специальных курсов невозможно. Поэтому преподавателями кафедры ВВТиС совместно с сотрудниками БРЦ ВВ организован городской семинар по параллельному программированию, в работе которого принимают участие студенты, магистранты, аспиранты, сотрудники и преподаватели не только УГАТУ, но и других вузов города. Кроме того, под руководством преподавателей кафедры ВВТиС в тесном сотрудничестве с БРЦ ВВ ведется активная научно-исследовательская работа студентов, в рамках

которой разрабатываются параллельные численные алгоритмы и программы решения задач динамики жидкости, обратных задач математической физики, задач физики твердого тела, авиадвигателестроения, теплоэнергетики и др. В результате в 2002 году были успешно защищены несколько дипломных инженерных и квалификационных бакалаврских работ, выполненных с применением вычислительных кластеров БРЦ ВВ.

Об организации лабораторного практикума

Все преподаваемые в УГАТУ дисциплины, связанные с обучением параллельному программированию на МВС, помимо лекционных и практических занятий обязательно имеют лабораторный практикум. В настоящее время для проведения лабораторных работ используются дисплейный класс параллельного программирования, дисплейный класс кафедры ВВТиС, дисплейный класс кафедры математики и вычислительные ресурсы БРЦ ВВ (рис. 2). Все дисплейные классы УГАТУ объединены между собой корпоративной сетью университета и имеют выход на вычислительные кластеры БРЦ ВВ в режиме удаленного доступа. В учебном процессе имеется возможность использования двух кластеров БРЦ ВВ: Alpha-кластера (12×Alpha 21164 EV5 / 256 MB / 6×25 GB Raid / Fast Ethernet 100 Mbit/s) и Intel – кластера (32×PIII-1000 Dual / 1024 MB / PCI-адаптеры 132 Mbit/s). Оба кластера работают под управлением операционной системы Red Hat Linux 6.2.

Создание учебных параллельных программ в рамках лабораторного практикума организовано в три этапа. На первом этапе производится написание и первоначальная отладка программного кода. Этот этап выполняется на обыкновенных однопроцессорных персональных компьютерах, работающих под управлением ОС семейства Linux. Для написания программы используются компиляторы Fortran и C, а распараллеливание осуществляется средствами MPI (mpich-1.2.0). Для отладки механизмов обмена данными между отдельными процессами программы на данном этапе используется режим эмуляции, поддерживаемый MPI.

На втором этапе производится проверка правильности работы параллельной программы на двухпроцессорной рабочей станции. При этом прежде всего проверяется наличие ускорения в работе программы при использовании двух процессоров по сравнению с однопроцессорным режимом. Если ускорение отсутствует или оказывается слишком малым, производится переработка программного кода или модификация параллельного алгоритма.

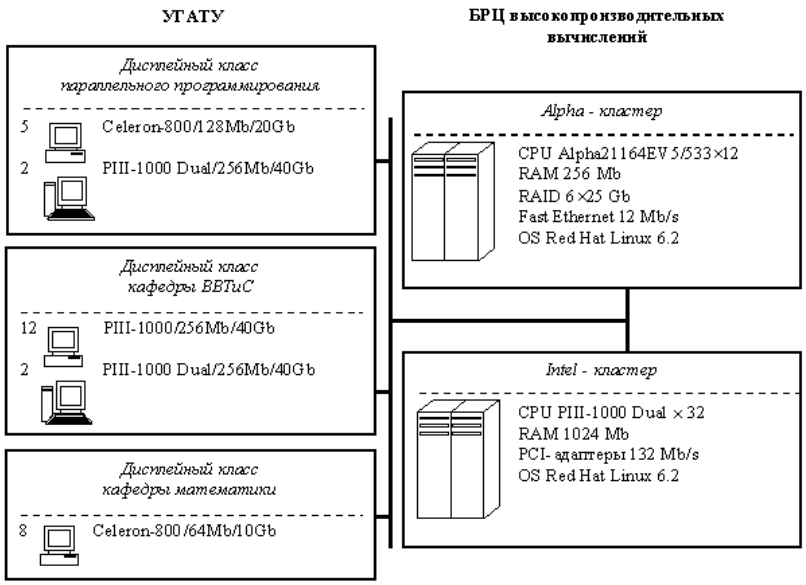


Рис. 2

После успешного прохождения второго этапа параллельная программа запускается на кластере в режиме удаленного доступа. На этом последнем этапе производится детальное исследование эффективности разработанной программы и анализ использованного параллельного алгоритма.

Описанная трехступенчатая технология создания параллельных программ хорошо зарекомендовала себя на практике. Она позволила практически полностью исключить кластерные системы из процесса отладки программ, тем самым освободив их для выполнения более важных вычислительных задач.

Заключение

В настоящее время в УГАТУ продолжаются работы по повышению качества подготовки студентов различных инженерных специальностей для работы на многопроцессорных вычислительных системах, совершенствуется и расширяется лабораторный практикум, разрабатываются программы новых курсов для переподготовки и повышения квалификации сотрудников вузов и НИИ региона. Активно ведется

подготовка к открытию новой специальности по математическому моделированию на высокопроизводительных вычислительных системах.

Работа выполнена при поддержке ФЦП «Интеграция», проект № У0011/735.

ПАРАЛЛЕЛЬНЫЙ ПОЛУЯВНЫЙ АЛГОРИТМ ЧИСЛЕННОГО РЕШЕНИЯ ЗАДАЧ ДИНАМИКИ ЖИДКОСТИ

Р.К. Газизов, С.Ю. Лукащук

Уфимский государственный авиационный технический университет

К.И. Михайленко

Институт Механики Уфимского научного центра РАН

Введение

Задачи вычислительной гидродинамики представляют собой обширный класс задач для решения которых применение высокопроизводительной вычислительной техники не только оправдано, но и необходимо. Об этом свидетельствует, в частности, знаменитый список проблем «большой вызов» [1], в котором задачи динамики жидкости и газа занимают одно из ведущих мест.

В последнее время в России и за рубежом все большее распространение получают кластерные вычислительные системы, производительность которых хотя и уступает производительности полноценных суперкомпьютеров, тем не менее значимо превышает производительность самых современных персональных компьютеров и рабочих станций. Такой рост популярности кластерных систем обусловлен как их относительно невысокой стоимостью, так и простотой и хорошей масштабируемостью.

Хорошо известно, что характерным недостатком недорогих кластерных вычислительных систем является весьма невысокая, по сравнению с пропускной способностью системной шины отдельного компьютера, пропускная способность среды передачи данных, объединяющей отдельные узлы кластера. Поэтому эффективный алгоритм решения задачи на кластерной вычислительной системе должен минимизировать обмен данными между узлами кластера. Кроме того, параллельный алгоритм должен обеспечивать максимально равномерную

загрузку всех процессоров системы, используемых для расчета [2].

Следует отметить, что далеко не каждый последовательный численный алгоритм может быть распараллелен, а класс алгоритмов, допускающих эффективное распараллеливание для кластерных систем, оказывается еще уже. Если ограничиться рассмотрением численных алгоритмов решения задач динамики жидкости, основанных на конечно-разностных схемах, то параллельные свойства таких алгоритмов во многом определяются видом лежащей в их основе численной схемы. Приведенным выше требованиям в наиболее полной степени удовлетворяют алгоритмы, основанные на явных конечно-разностных схемах. Однако присущие этим схемам недостатки, одним из которых является их условная устойчивость, ограничивают область применимости таких параллельных алгоритмов. С другой стороны, эффективные последовательные алгоритмы, использующие неявные конечно-разностные схемы, не могут быть эффективно модифицированы для использования на кластерных вычислительных системах. В этой связи значительный интерес представляют алгоритмы, основанные на полуявных численных схемах, так как в них существенно ослаблены недостатки явных численных схем при сохранении возможности эффективного распараллеливания.

Распараллеливание полуявного численного алгоритма

Полуявность алгоритма означает, что он основан на численной схеме, в которой для расчета некоторой искомой величины u в любой внутренней точке расчетной области используются значения этой величины в соседних узлах сетки не только на предыдущем временном слое, но и уже вычисленные значения этой величины на текущем слое (рис. 1):

$$u_{ij}^{n+1} = f(u_{ij}^{n}, u_{i+1\,j}^{n}, u_{ij+1}^{n}, u_{i-1\,j}^{n+1}, u_{ij-1}^{n+1}, \Delta x, \Delta y, \Delta t)$$

(здесь $\Delta x, \Delta y, \Delta t$ – шаги сетки по пространственным и временной переменным соответственно).

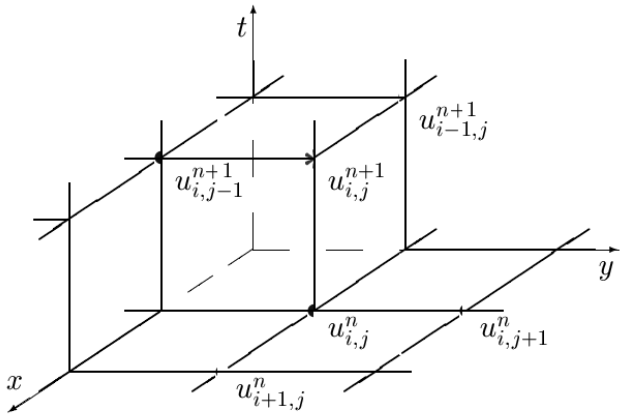


Рис. 1

В вычислительной гидродинамике алгоритмы, основанные на полуявных схемах, используются для расчета нестационарных неизотермических ламинарных и турбулентных течений сжимаемой и несжимаемой жидкости в двумерных и трехмерных областях сложной геометрии при различных краевых условиях (см., например, [3]).

Одним из наиболее продуктивных методов построения параллельных алгоритмов является метод пространственной декомпозиции расчетной области (domain decomposition method) [4]. Суть его достаточно проста: расчетная область разбивается на отдельные подобласти, число которых согласовано с количеством процессоров используемой вычислительной системы и расчет всех подобластей проводится по идентичному алгоритму; при этом для расчета каждой отдельной подобласти отводится свой процессор. Данный метод был использован нами для распараллеливания полуявного алгоритма.

В дальнейшем для простоты изложения будем считать, что расчетная область является прямоугольником. Кроме того, полагаем, что каждый процесс параллельного алгоритма выполняется на отдельном процессоре.

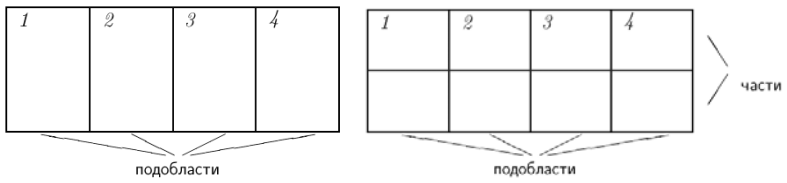
Наиболее простая декомпозиция расчетной области, отвечающая требованию минимального объема пересылок, предусматривает ее равномерное разбиение только в направлении одной из координатных осей (рис. 2,*а*). Однако эффективность параллельного алгоритма при такой декомпозиции никогда не будет превышать 50% (под эффектив-

ностью здесь понимается среднее время работы процессоров кластерной системы). В этом случае если каждый процессор на любом шаге по времени обсчитывает свою подобласть целиком, то полуявность алгоритма приводит к чередующейся работе процессоров, т.е. на каждом временном шаге половина процессоров будет простаивать.

В самом деле, после того, как первый процесс, выполняемый на первом узле, заканчивает расчет своей подобласти на первом временном слое, он предает необходимые для продолжения расчета результаты второму процессу. После получения этих результатов второй процесс начинает расчет своей подобласти. Однако в это время первый процесс простаивает: он ожидает получения результатов вычислений второй подобласти от второго процесса, так как они необходимы ему для продолжения расчетов своей подобласти на втором временном слое. Для остальных процессов реализуется аналогичная схема загрузки.

Улучшить загрузку позволяет такая организация вычислительного процесса, когда подобласть рассчитывается не целиком, а по частям, с промежуточной пересылкой данных.

Рассмотрим случай разбиения подобласти на две части, как это показано на рис. 2,*b*. В этом случае время простоя значительно сокращается. Для объяснения этого факта рассмотрим ход вычислительного процесса. После окончания расчета первой части первой подобласти, первый процесс отсылает необходимые результаты второму. В следующий момент времени работают два процесса одновременно. Первый процесс продолжает расчет второй части своей подобласти, а второй начинает расчет первой части своей.



а) *б*)

Рис. 2

Для продолжения расчета на втором временном слое первый процесс нуждается в результатах расчета только первой части второй подобласти. Благодаря этому и удается сократить время простоя. Более того, время простоя оказывается равным времени передачи сообщения

и в идеальном случае мгновенной передачи данных оно оказывается равным нулю.

Конечно, любая реальная кластерная система имеет отличное от нуля вполне определенное время передачи сообщения, в связи с чем представленная схема декомпозиции оказывается недостаточно эффективной.

Для непрерывной работы всех процессоров системы количество частей, на которые разбивается подобласть, должно быть не менее трех. На рис. 3 приведена диаграмма работы параллельной программы при таком разбиении.

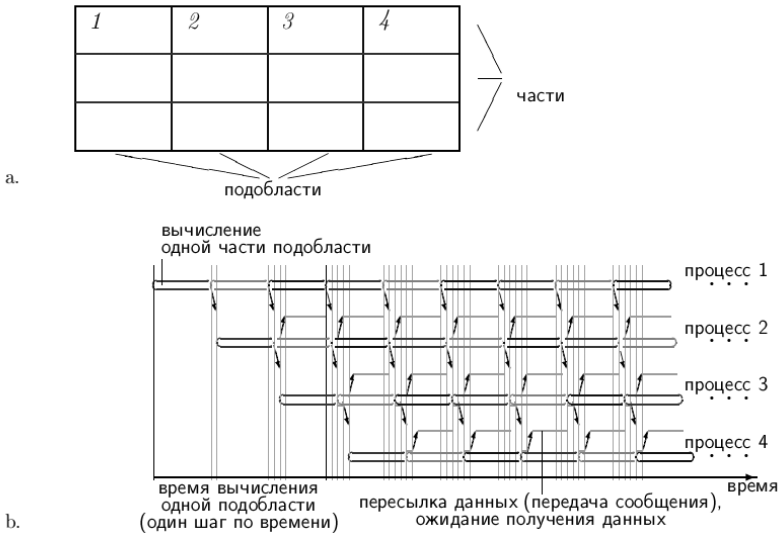


Рис. 3

Как хорошо видно из диаграммы на рис.3,*b*, дополнительное разбиение каждой расчетной подобласти не менее чем на три части позволяет подобрать условия, приводящие к полной загрузке участвующих в расчете узлов кластерной вычислительной системы.

Прежде чем перейти к описанию самого алгоритма, необходимо сделать одно важное замечание. При численном решении задач динамики жидкости существует необходимость сохранения больших объемов промежуточных результатов расчета (на различных, но далеко не всех временных шагах). В этой связи представляется целесообразным

освобождение одного процессора системы от непосредственных расчетов и выделение его для сбора данных со всех остальных процессоров и сохранения их на диске. Для определенности будем считать, что этот выделенный процессор и соответствующий ему процесс имеют номер нуль. Такая процедура позволяет освободить остальные процессоры от выполнения операции обмена данными с диском и улучшить тем самым равномерность их загрузки вычислительной работой. Все процедуры обмена данными с периферийными устройствами осуществляются через нулевой процесс, именно он осуществляет первоначальную рассылку данных, содержащих требуемую для них информацию о свойствах среды, начальных и граничных условиях и прочих исходных данных по остальным процессам.

В результате полуявный алгоритм будет представлять собой следующую последовательность шагов:

- нулевой процесс считывает с диска исходные данные и распределяет их по рабочим процессам. При этом можно производить действия над большими массивами данных, не помещающимися целиком в оперативной памяти, отведенной данному процессу;
- первый процесс производит расчет первой части своей подобласти на первом временном слое. В это время остальные процессы системы находятся в состоянии ожидания. По окончании расчета первый процесс отсылает требуемые результаты второму процессу;
- на следующем этапе работают уже два процесса: первый процесс рассчитывает вторую часть своей подобласти, а второй процесс - первую часть своей подобласти. По окончании расчета первый процесс пересылает данные второму процессу, необходимые ему для расчета второй части подобласти. Второй процесс передает третьему процессу данные, необходимые для расчета первой части третьей подобласти. Кроме того, второй процесс передает первому процессу данные, необходимые для расчета первой части первой подобласти на втором временном шаге;
- подобная последовательность действий продолжается и на следующих этапах, в том числе и после вступления в работу всех процессоров системы;
- если результаты расчета некоторого временного шага должны быть сохранены, то каждый процесс по окончании расчета всей

подобласти на указанном шаге передает результаты нулевому процессу для сохранения их на диске.

Об оценке границы эффективности параллельного алгоритма

Размеры пространственной подобласти и количество частей могут быть выбраны исходя из того, что время расчета подобласти должно быть не меньше суммарного времени всех пересылок данных. Последнее условие предполагает, что в любой момент времени может производиться только одна пересылка.

Формально указанное условие может быть записано в виде неравенства, связывающего время расчета одной части подобласти t_p и время передачи одного сообщения t_s:

$$t_p \geq 2(p-1)t_s + \frac{pt_\Delta}{n},$$

p – число процессоров, участвующих в вычислениях; n – количество частей; t_Δ – добавка, учитывающая периодическую передачу результатов расчета нулевому процессу для сохранения в файле.

При выполнении данного неравенства все обмены данными между процессами оказываются разнесенными во времени и не будут пересекаться. Эффективность вычислительного процесса можно повысить, если пересылки данных осуществлять посредством неблокирующих функций приема-передачи.

Величины, входящие в приведенное неравенство, в свою очередь могут быть выражены через характеристики кластерной системы и число узлов расчетной сетки. В результате указанное неравенство позволяет определить минимальные размеры каждой подобласти, при которых обеспечивается непрерывная работа всех процессоров системы.

Такая оценка была выполнена нами для Alpha-кластера Башкирского регионального центра высокопроизводительных вычислений. Кластер имеет следующую конфигурацию:

- количество узлов – 12;
- процессор на узле – Alpha21164EV5 с тактовой частотой 533 МГц;
- память на узле – 128 Мбайт;
- коммуникационная среда – FastEthernet;
- коммутатор – HP ProCurve 1600M.

В результате минимальный размер расчетной области, обеспечи-

вающий эффективную загрузку всех процессоров кластера, оценивается в 300×1100 узловых точек.

Заключение

В настоящее время проводится работа по программной реализации представленного алгоритма для кластерной вычислительной системы Башкирского регионального центра высокопроизводительных вычислений.

Литература

1. Grand Challenges: High performance computing and communications. A report by the Committee on Physical, Mathematical and Engineering Sciences, NSF/CISE, 1800 G. Street NW, Washington, DC 20550, 1991.
2. Foster I. Designing and Building Parallel Programs. – Addison-Wesley, 1995.
3. Griebel M., Dornseifer T., Neunhoeffer T. Numerical Simulation in Fluid Dynamics. –SIAM, 1998.

ОЦЕНКА СЛОЖНОСТИ КОММУНИКАЦИОННЫХ ОПЕРАЦИЙ В КЛАСТЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ*

А.В. Гергель, Р.В. Виноградов

Нижегородский государственный университет им. Н.И. Лобачевского

Эффективность параллельных вычислений во многом определяется трудоемкостью коммуникационных операций, выполняемых в параллельных программах. В этой связи важным представляется разработка моделей и методов оценки сложности операций передачи данных в многопроцессорных вычислительных системах.

В данной работе приводится разработанная модель оценки трудо-

* Работа выполнена в рамках Федеральной Целевой Программы «Интеграция»

емкости коммуникационных операций в кластерных системах и описываются результаты вычислительных экспериментов, подтверждающих адекватность предлагаемой модели (при проведении экспериментов для организации параллельных вычислений использовалась библиотека MPI [1]).

Проводя анализ существующих методов оценки сложности можно выделить тесты коммуникационных операций, разработанные в НИВЦ МГУ [2]. Эти тесты позволяют оценить экспериментально время подготовки данных для передачи по сети вычислительного кластера (*латентность*), скорость передачи данных (*пропускную способность*), а также определить время выполнения основных операций MPI.

Для оценки времени выполнения операции передачи данных в [2] предлагается соотношение (*модель А*):

$$t_{пд}(m) = t_н + m / R\,, \qquad (1)$$

здесь $t_{пд}(m)$ – время передачи данных объема m байт, $t_н$ – латентность, R – пропускная способность сети). Данная модель позволяет достаточно точно оценить время выполнения операции передачи данных. С другой стороны, в этой модели не учитывается ряд эффектов, имеющих место при выполнении коммуникационных операций. Так, в большинстве сетей для передачи данных используется *метод передачи пакетов*, в соответствии с которым передаваемые данные пересылаются блоками (*пакетами*) одинакового размера. Как правило, каждый пакет содержит помимо исходных передаваемых данных некоторый набор служебных полей (*заголовок пакета*). Учитывая приведенные обстоятельства, в [3] приводится уточненное выражение (*модель В*), позволяющее определить время передачи данных между двумя процессорными узлами, длина маршрута между которыми равна ℓ:

$$t_{пд}(m) = t_н + m * t_k + t_c * \ell\,, \qquad (2)$$

где t_k – время передачи одного байта данных через сетевой канал (определяется пиковой пропускной способностью канала, то есть $t_k = 1/R$), t_c – время передачи служебных данных (заголовков, служебной информации), ℓ – длина маршрута, связывающего узлы.

Данная модель позволяет более точно оценить время выполнения операции передачи данных, однако, как и ранее в первой модели, время подготовки данных $t_н$ предполагается постоянным (не завися-

щим от объема передаваемых данных), время передачи служебных данных t_c не зависит от количества передаваемых пакетов и т.п.

Учитывая все приведенные замечания, была предложена новая модель (*модель С*), в которой время передачи данных между двумя процессорами определяется в соответствии со следующими выражениями:

$$t_{пд} = \begin{cases} m \cdot t_к + (TCP + IP + FE) \cdot n \cdot t_к + t_{нач_0} + m \cdot t_{нач_1}, n = 1 \\ m \cdot t_к + (TCP + IP + FE) \cdot n \cdot t_к + t_{нач_0} + (MTU - IP - TCP - FE) \cdot t_{нач_1}, n > 1 \end{cases}$$

$$n = \left\lceil \frac{m}{MTU - TCP - IP - FE} \right\rceil. \quad (3)$$

В приведенных соотношениях TCP есть размер заголовка пакета протокола передачи данных TCP, IP – размер заголовка пакета протокола *IP*, *FE* – размер заголовка пакета протокола сети Fast Ethernet, *MTU* – максимальный размер пакета, который может быть доставлен в сети Fast Ethernet (по умолчанию для операционной сети MS Windows *MTU* =1500 байт), n – число передаваемых пакетов (размеры всех заголовков должны указываться в байтах).

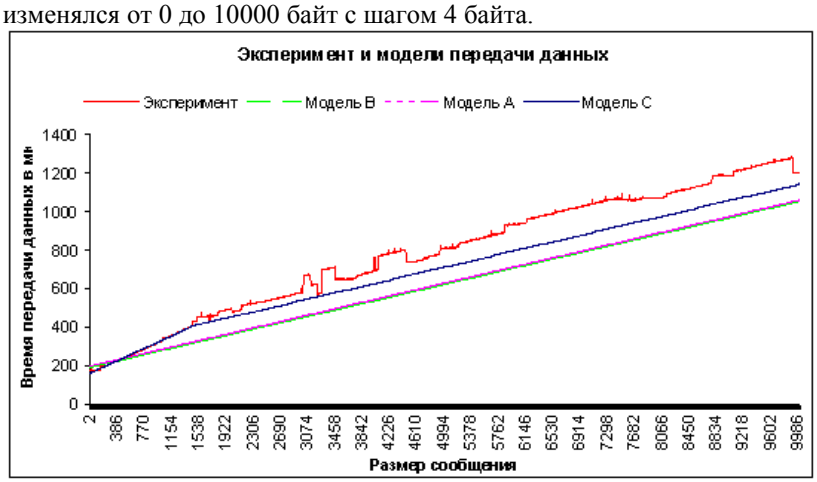
В предлагаемой модели время подготовки данных к передаче по сети предлагается оценивать при помощи линейной зависимости вида:

$$t_н = t_{нач_0} + S \cdot t_{нач_1},$$

где параметр S определяется в зависимости от программных и аппаратных свойств сети вычислительного кластера. При передаче малых сообщений, представляемых в виде одного пакета, в модели значение полагается S равным m, а значения величин $t_{нач_0}$ и $t_{нач_1}$ оцениваются по результатам вычислительных экспериментов при помощи линейной аппроксимации времен передачи сообщений размера от 0 до *MTU*. При больших объемах передаваемой информации значение S принимается равным $MTU - TCP - IP - FE$, поскольку процессы подготовки второго и следующего пакетов могут совмещаться с процедурой передачи предшествующих пакетов.

Для проверки адекватности предложенной модели реальным процессам передачи данных в сети многопроцессорного кластера были проведены вычислительные эксперименты на кластере Нижегородского университета. В состав кластера входят рабочие станции, оснащенные процессорами Intel Pentium 4 (1300 Мгц) и соединенные сетью

Fast Ethernet (100M бит). В ходе экспериментов осуществлялась передача данных между двумя процессорами кластера, размер передаваемых сообщений варьировался от 0 до 8 Мб. Для получения более точных оценок выполнение каждой операции осуществлялось многократно (более 100000 раз), после чего результаты временных замеров усреднялись. Для иллюстрации ниже приведен результат одного эксперимента, при проведении которого размер передаваемых сообщений изменялся от 0 до 10000 байт с шагом 4 байта.



Зависимость экспериментального времени и времени, полученного по моделям A, B, C от объема данных

В таблице приводятся ряд числовых данных по погрешности рассмотренных моделей трудоемкости коммуникационных операций по результатам вычислительных экспериментов (величина погрешности дается в виде относительного отклонения от реального времени выполнения операции передачи данных).

В результате приведенных данных можно заключить, что использование новой предложенной модели позволяет оценивать время выполняемых операций передачи данных с более высокой точностью.

В заключение сформулируем основные результаты работы:

– проведен анализ существующих методов оценки сложности коммуникационных операций;
– разработана новая модель оценки сложности операций передачи данных;
– проведены вычислительные эксперименты для проверки соот-

ветствия рассмотренных моделей реальным процессам передачи данных в сети многопроцессорного кластера.

Объем сообщения в байтах	Время передачи данных, мкс	Погрешность теоретической оценки времени выполнения операции передачи данных, %		
		Модель A	Модель B	Модель C
32	172,0269	–16,36	–12,45	3,55
64	172,2211	–17,83	–13,93	0,53%
128	173,1494	–20,39	–16,50	–5,15
256	203,7902	–7,70	–4,40	0,09
512	242,6845	0,46	3,23	–1,63
1024	334,4392	14,57	16,58	0,50
2048	481,5397	22,33	23,73	5,05
4096	770,6155	28,55	29,42	18,13

Авторы работы выражают свою благодарность Сенину А.В., на внимание и дружескую поддержку выполненной работы. Следует отметить также, что начальные работы по оценки предлагаемой модели C и проведение вычислительных экспериментов для вычислительного кластера Института прикладной физики РАН были выполнены в дипломной работе Дрейбанда М.

Литература

1. *Group W.*, *Lusk E.*, *Skjellum A.* Using MPI. Portable Parallel Programming with the Message-Passing Interface. MIT Press, 1994.
2. *Андреев А.Н.*, *Воеводин В.В.* Методика измерения основных характеристик программно-аппаратной среды (www.dvo.ru/bbc/benchmarks.html).
3. *Гергель В.П.*, *Стронгин Р.Г.* Основы параллельных вычислений для многопроцессорных вычислительных систем. Н.Новгород: Изд-во ННГУ, 2001

РАЗРАБОТКА ИНТЕГРИРОВАННОЙ СРЕДЫ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ ДЛЯ КЛАСТЕРА НИЖЕГОРОДСКОГО УНИВЕРСИТЕТА*

В.П. Гергель, А.Н. Свистунов

Нижегородский государственный университет им. Н.И. Лобачевского

В работе рассматриваются проблемы создания интегрированной среды высокопроизводительных вычислений для кластера Нижегородского университета, который был предоставлен ННГУ в рамках академической программы Интел в 2001 г. [3].

В состав кластера входит:

- 2 вычислительных сервера, каждый из которых имеет 4 процессора Intel Pentium III 700 Мгц, 512 MB RAM;
- 12 вычислительных серверов, каждый из которых имеет 2 процессора Intel Pentium III 1000 Мгц, 256 MB RAM;
- 12 рабочих станций на базе процессора Intel Pentium 4 1300 Мгц, 256 MB RAM.

Важной отличительной особенностью кластера является его неоднородность (гетерогенность). В состав кластера входят рабочие места, оснащенные процессорами Intel Pentium 4 и соединенные относительно медленной сетью (100 Мбит), а также вычислительные 2 и 4-процессорные сервера, обмен данными между которыми выполняется при помощи быстрых каналов передачи данных (1000 Мбит)

Основной операционной системой, установленной на узлах кластера является ОС Windows (на рабочих станциях установлена Windows 2000 Professional, на серверах установлена Windows 2000 Advanced Server).

Дополнительная информация о структуре кластера ННГУ доступна в сети Интернет по адресу http://www.software.unn.ac.ru/cluster. htm.

Эффективное использование быстродействующего компьютерного оборудования предполагает решение двух основных проблем, возникающих при эксплуатации кластера - проблему предоставления удаленного доступа к вычислительной среде кластера и проблему эффективного управления и мониторинга вычислительных задач, выпол-

* Работа выполнена в рамках Федеральной Целевой программы «Интеграция»

няющихся на кластере [1].

На сегодняшний день известно довольно много различных программных систем, позволяющих решать отмеченные проблемы. Большинство подобных систем традиционно разрабатывались для ОС UNIX, однако, в последнее время появились подобные системы и для ОС семейства Windows. К числу таких систем относится, например LSF (Load Sharing Facility, http://www.platform.com), или Cluster CoNTroller (http://www.mpi-softtech.com/).

Однако использование готовых систем управления кластером затруднено рядом обстоятельств. Прежде всего, это высокая стоимость подобных систем, достигающая, для кластера подобного кластеру Нижегородского университета, десятков тысяч долларов. Вторым, не менее важным обстоятельством, является закрытость подобных систем, осложняющая проведение некоторых исследовательских работ. Дело в том, что кроме задачи обеспечения функционирования кластерной системы ставилась еще и задача создания испытательного стенда для проведения экспериментов по апробации различных алгоритмов планирования задач на кластерных системах. Для кластера Нижегородского университета планирование распределения вычислительной нагрузки по узлам кластера является практически важной задачей, в силу его неоднородности.

Отмеченные факторы приводят к необходимости создания собственных средств поддержки организации высокопроизводительных вычислений на кластере.

Разрабатываемая программная система должна была решить еще одну важную задачу - задачу интеграции вычислительного кластера ННГУ с вычислительным кластером Института прикладной физики РАН (ИПФ РАН). К числу обстоятельств, затрудняющих такую интеграцию, следует отнести тот факт, что в отличие от вычислительного кластера университета, вычислительный кластер ИПФ РАН в качестве базовой операционной системы использует один из клонов UNIX[4].

При построении системы управления кластером со стороны потенциальных пользователей были определены следующие требования:

- реализация по крайней мере минимального набора операций (загрузка задачи, добавление задачи в очередь задач, получение текущего статуса очереди задач и текущей выполняемой задачи, получение результатов вычислений);
- простой и удобный способ доступа, не требующий установки на рабочих станциях пользователей какого-либо специального

программного обеспечения, позволяющий получить доступ к системе из любой точки;

– собственная система авторизации пользователей не связанная напрямую с системой авторизации операционной системы;
– наличие системы очередей задач;
– сохранение задач пользователя после их выполнения и возможность их повторного запуска;
– автоматическое сохранение результатов работы.

При построении программной системы за основу была взята сложившаяся архитектура системы мониторинга и управления[2] (рис.)

Она включает, как правило, следующие составные части:

– компонент, взаимодействующий с пользователем (Менеджер доступа), позволяющий ставить задачи в очередь, удалять задачи из очереди, просматривать статус задач и т.д., а также ведущий базу данных пользователей и базу данных результатов;
– компонент, оперирующий с очередью заданий (Диспетчер заданий), распределяющий задания по узлам кластера и ставящий их в очередь для выполнения;
– компонент, обеспечивающий мониторинг кластера (Супервизор кластера) и непосредственное выделение ресурсов.

В настоящее время разработан и внедрен опытный вариант системы, обладающий следующими возможностями:

– поддержка минимально – необходимого набора операций по управлению задачами пользователей;
– возможность доступа к системе с любого компьютера, подключенного к сети Интернет;
– отсутствие необходимости установки на компьютере пользователя специального программного обеспечения. Использование в качестве клиента web-браузера, telnet-клиента, различных специализированных программ;
– хранение очереди заданий (как ждущих своей очереди на выполнение, так и уже завершившихся, сформировавших результат) во внешней базе данных, что позволяет обеспечить устойчивость системы в случае сбоя;
– возможность замены планировщика и изменения стратегии планирования;

Ведение статистики использования задачами пользователей вычислительных ресурсов кластера;

В настоящее время система активно используется широким кругом пользователей и сотрудников Центра компьютерного моделирования. Возможности, предоставляемые системой, используются при разработке и эксплуатации учебных и исследовательских программных комплексов, таких как программная система для изучения и исследования параллельных методов решения сложных вычислительных задач (ПараЛаб) и система параллельной многоэкстремальной оптимизации Абсолют Эксперт.

В настоящее время работа над системой продолжается в нескольких направлениях:

– расширение функциональности системы, производящееся в тесном контакте с конечными пользователями;

– исследования в области различных алгоритмов планирования;
– оптимизация процесса мониторинга ресурсов кластера;
– исследование подходов к созданию шлюза между кластером ННГУ и кластером ИПФ РАН.

Литература

1. *Rajkumar Buyya*. High Performance Cluster Computing. Volume1: Architectures and Systems. Volume 2: Programming and Applications. Prentice Hall PTR, Prentice-Hall Inc., 1999.
2. *W. Saphir*, *L.A. Tanner*, *B. Traversat.* Job Management Requirements for NAS Parallel Systems and Clusters. NAS Technical Report NAS-95-006 February 95.
3. *Гергель В.П.*, *Стронгин Р.Г.* Высокопроизводительный вычислительный кластер Нижегородского университета. Материалы конференции Relarn. 2002. Н. Новгород.
4. *Дрейбанд М.С.* Вычислительный кластер ИПФ РАН. Материалы конференции Relarn. 2002. Н. Новгород.

ПРОГРАММНАЯ СИСТЕМА ДЛЯ ИЗУЧЕНИЯ И ИССЛЕДОВАНИЯ ПАРАЛЛЕЛЬНЫХ МЕТОДОВ РЕШЕНИЯ СЛОЖНЫХ ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ*

В.П. Гергель, А. Сибирякова

Нижегородский государственный университет им. Н.И. Лобачевского

В работе рассматривается программная система Параллельная Лаборатория (сокращенное наименование ПараЛаб), обеспечивающая возможность проведения вычислительных экспериментов с целью изучения и исследования параллельных алгоритмов решения сложных вычислительных задач. Данная программная система может быть использована для организации лабораторного практикума по различным учебным курсам в области параллельного программирования. В рамках такого лабораторного практикума система ПараЛаб обеспечивает

* Работа выполнена при поддержке Фонда содействия развитию малых форм предприятий в научно-технической сфере

возможность моделирования многопроцессорных вычислительных систем с различной топологией сети передачи данных, получения визуального представления о вычислительных процессах и операциях передачи данных, происходящих при параллельном решении разных вычислительных задач, построения оценок эффективности изучаемых методов параллельных вычислений. Проведение такого практикума может быть организовано на «обычных» однопроцессорных компьютерах, работающих под управлением операционных систем MS Windows 2000 или MS Windows XP (режим многозадачной имитации параллельных вычислений). Кроме режима имитации в системе ПараЛаб может быть обеспечена процедура удаленного доступа к имеющейся многопроцессорной вычислительной системе для выполнения экспериментов в режиме «настоящих» параллельных вычислений для сопоставления результатов имитации и реальных расчетов.

Итак, система ПараЛаб есть интегрированная среда для изучения и исследования параллельных алгоритмов решения сложных вычислительных задач. Широкий набор имеющихся средств визуализации процесса выполнения эксперимента и анализа полученных результатов позволяет изучить эффективность использования тех или иных алгоритмов на разных вычислительных системах, сделать выводы о масштабируемости алгоритмов и вычислить возможное ускорение процесса параллельных вычислений.

В рамках системы пользователю предоставляется возможность смоделировать («собрать») параллельную вычислительную систему. Для этого можно *выбрать топологию* сети передачи. В системе ПараЛаб реализован ряд наиболее известных топологий, среди которых Линейка, Кольцо, Решетка, Гиперкуб и Полный граф. Можно *задавать число процессоров* в выбранной топологии, *определять производительность* процессора (предполагается, что вычислительная система является однородной, то есть, все процессоры обладают одинаковой производительностью). Кроме того, можно определять такие важные параметры многопроцессорной вычислительной системы, как *характеристики коммуникационной среды* (латентность сети и ее пропускная способность) и *способ коммуникации* (в числе реализованных в системе метод передачи сообщений и метод передачи пакетов).

На следующем шаге пользователю предоставляется возможность *осуществить постановку задачи*, то есть выбрать одну из имеющихся в системе задач, для которой существуют реализованные параллельные алгоритмы решения. В системе ПараЛаб обеспечивается возмож-

ность решения задач сортировки линейного массива данных, матричного умножения и обработки графов. Эти задачи относятся к числу типовых вычислительных проблем и широко используются в качестве учебных примеров при изучении проблематики параллельных вычислений. Есть возможность указать конкретный *метод решения задачи* – для сортировки данных реализованы параллельные обобщения пузырьковой сортировки, сортировки Шелла и быстрой сортировки, для задачи матричного умножения реализован ленточный алгоритм, алгоритмы Фокса и Кэннона, среди задач обработки графов выделены алгоритм Прима поиска минимального охватывающего дерева и алгоритм Дейкстры поиска кратчайшего пути [1]. Можно ввести *параметры задачи* (размер массива для задач сортировки, размерность матриц для матричных операций и т.п.).

Далее пользователь системы может *установить параметры визуализации*: выбрать приемлемый темп демонстрации, тип отображения пересылки данных и процессор для более детального наблюдения за его действиями. Для наблюдения за процессом выполнения эксперимента в рамках системы ПараЛаб предусмотрены различные формы графического представления результатов выполнения итераций параллельного алгоритма (рис. 1).

В левой части *окна вычислительного эксперимента* выделена область «Выполнение эксперимента» для демонстрации процессоров вычислительной системы, объединенных в ту или иную топологию, и данных, расположенных на каждом из имеющихся процессоров. В ходе вычислений в этой же области показывается выполняемый взаимообмен данными между процессорами системы.

В правой верхней части окна отображается текущее состояние объекта, который является результатом решаемой вычислительной задачи. В зависимости от того, какой эксперимент выполняется, эта область носит название «Текущее состояние массива» (при выполнении алгоритма сортировки) или «Результат умножения матриц» (при выполнении матричного умножения).

В правом нижнем углу располагается ленточный индикатор выполнения эксперимента и его текущие временные характеристики. Дополнительно в отдельном окне могут быть подробно представлены вычисления, которые производит конкретный выбранный процессор.

В зависимости от изучаемой задачи, можно *выполнить эксперимент* только для *активного окна* или *для всех имеющихся окон экспериментов* (в режиме разделения времени). Последний способ выпол-

нения экспериментов позволяет наглядно сравнивать динамику решения задачи различными методами, на разных топологиях, с разными параметрами исходной задачи. Кроме того, в системе имеется возможность выполнения последовательности экспериментов (*серии*), требующих длительных вычислений, в автоматическом режиме с запоминанием результатов для организации последующего анализа полученных данных. Это обеспечивает возможность накопления экспериментальных данных большого объема для принятия более обоснованных выводов при изучении и исследовании параллельных алгоритмов решения вычислительных задач.

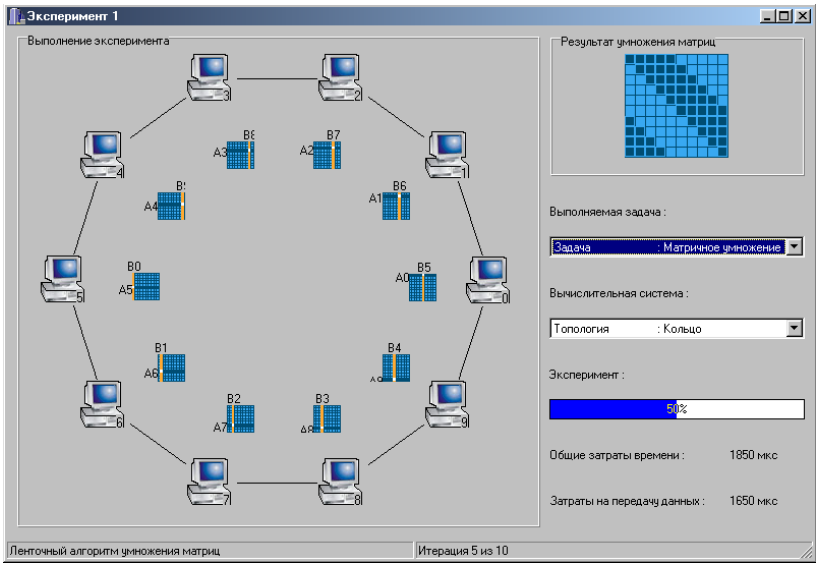


Рис 1. Окно вычислительного эксперимента

Одной из важных характеристик системы ПараЛаб является возможность *накапливать и анализировать результаты* выполненных экспериментов. По запомненным результатам в системе предусматривается возможность графического построения зависимостей временных характеристик от параметров задачи и вычислительной системы (рис. 2).

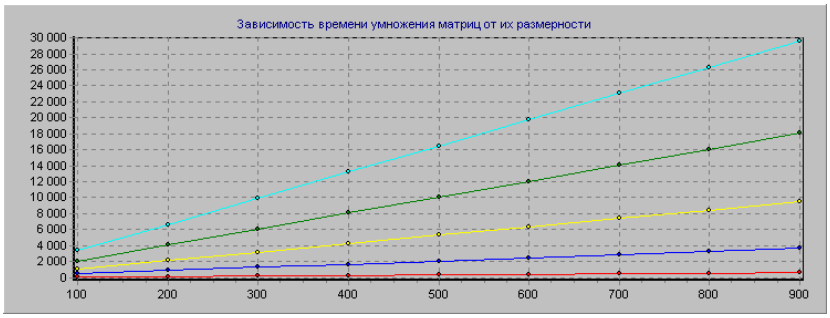


Рис. 2. Графическое построение зависимостей

При построении зависимостей временных характеристик от параметров задачи и вычислительной системы для экспериментов, выполненных в режиме имитации, используются теоретические оценки в соответствии с имеющимися моделями параллельных вычислений [1–3]. Для реальных экспериментов на многопроцессорных вычислительных системах зависимости строятся по совокупности результатов проведенных вычислительных экспериментов. Любой из проведенных ранее экспериментов может быть восстановлен для повторного проведения. Кроме того, обеспечена возможность ведения журнала экспериментов с записью туда постановки задачи, параметров вычислительной системы и полученных результатов.

Одной из важнейших характеристик системы является возможность *выбора типа эксперимента*. Эксперимент может быть выполнен *в режиме эмуляции*, т.е. проведен на одном процессоре без использования каких – либо специальных программных средств типа интерфейса передачи сообщений (message passing interface – MPI). Кроме того, обеспечена возможность проведения *реального вычислительного эксперимента* в каждом из трех вариантов:

– на одном компьютере, где имеется библиотека передачи сообщений MPI (многопоточное выполнение эксперимента); для данной библиотеки имеются общедоступные реализации, которые могут быть получены в сети Интернет и установлены на компьютере под управлением операционных систем MS Windows 2000 или MS Windows XP (требование данного типа операционных систем определяется условиями разработки системы ПараЛаб);

– на реальной многопроцессорной кластерной вычислительной системе;
– в режиме удаленного доступа к вычислительному кластеру.

Если проводится реальный эксперимент на многопроцессорной вычислительной системе или в режиме удаленного доступа, есть *возможность выбора типов вычислительных узлов* (например, кластер Нижегородского университета состоит из однопроцессорных рабочих станций и многопроцессорных серверов с общей памятью).

Реализуемые системой ПараЛаб процессы изучения и исследований ориентированы на активное освоение основных теоретических положений и способствуют формированию у пользователей своих собственных представлений о моделях и методах параллельных вычислений путем наблюдения, сравнения и сопоставления широкого набора различных визуальных графических форм, демонстрируемых в ходе выполнения вычислительного эксперимента.

Разработка программной системы осуществлялась в среде Borland C++ Builder и может функционировать под управлением операционных систем MS Windows 2000 или MS Windows XP на компьютере типовой конфигурации, совместимом с IBM PC Pentium. На текущий момент времени в опытной эксплуатации имеется версия системы ПараЛаб 1.1, реализующей весь рассмотренный выше набор возможностей. Для успешного освоения система ПараЛаб содержит краткое руководство пользователя.

В Нижегородском университете система ПараЛаб использовалась для организации лабораторного практикума по учебному курсу «Многопроцессорные системы и параллельное программирование», который читается для студентов факультета вычислительной математики и кибернетики с 1995 г. Данный курс входит в программу межфакультетской магистратуры “Математические модели, методы и программное обеспечение современных компьютерных технологий», ориентированной на подготовку квалифицированных специалистов для высокотехнологичных производств информационной индустрии. Для курса разработана расширенная учебная программы, подготовлены электронные презентации для проведения лекционных занятий и опубликовано учебное пособие [3].

Литература

1. *Kumar V.*, *Grama A.*, *Gupta A.*, *Karypis G.* Introduction to Parallel Computing / The Benjamin/Cummings Publishing Company, Inc., 1994.

2. *Rajkumar Buyya*. High Performance Cluster Computing. Volume 1: Architectures and Systems. Volume 2: Programming and Applications. Prentice Hall PTR, Prentice-Hall Inc., 1999.
3. *Гергель В.П.*, *Стронгин Р.Г.* Основы параллельных вычислений для многопроцессорных вычислительных систем. Учебное пособие – Нижний Новгород; Изд-во ННГУ им. Н.И.Лобачевского, 2001.

ПАРАЛЛЕЛЬНЫЕ РЕКУРСИВНЫЕ АЛГОРИТМЫ МНОГОЭКСТРЕМАЛЬНОЙ ОПТИМИЗАЦИИ*

В.А. Гришагин, А.А.Филатов

Нижегородский государственный университет им.Н.И. Лобачевского

Рассматривается конечномерная задача оптимизации

$$f(y) \to \min, \quad y \in Q \subseteq R^N, \tag{1}$$

$$Q = \{y \in D : g_j(y) \le 0, \quad 1 \le j \le m\}, \tag{2}$$

$$D = \{y \in R^N : y_i \in [a_i, b_i],\ 1 \le i \le N\}, \tag{3}$$

в которой целевая функция $f(y)$ и зависимости $g_j(y)$, $1 \le j \le m$, задающие функциональные ограничения (3.2), являются многоэкстремальными. В указанной задаче требуется найти ее глобально-оптимальное решение.

Многоэкстремальные оптимизационные задачи обладают высокой степенью вычислительной сложности. Основными факторами, определяющими данную сложность, являются размерность (количество варьируемых параметров) и наличие функциональных ограничений (2), многоэкстремальность которых может порождать невыпуклые, неодносвязные и даже несвязные допустимые области Q.

Для преодоления высокой трудоемкости задачи (1)–(3) возможно использование различных идей редукции сложности, например, применение схем редукции размерности [1,2] или различных вариантов метода штрафных функций для редуцирования задачи с ограничениями к задаче без ограничений. Важнейшим резервом для ускорения процесса анализа и расширения круга исследуемых задач является

* Данная работа выполнена при поддержке РФФИ (грант № 01-01-00587)

разработка параллельных алгоритмов, ориентированных на использование ресурсов многопроцессорных, в том числе кластерных вычислительных систем.

В настоящей работе предложена алгоритмическая схема, обеспечивающая параллельную реализацию многошаговой схемы редукции размерности [1, 2] в сочетании с индексным подходом к учету ограничений [2] и применением высокоэффективных характеристических алгоритмов глобального поиска [3].

Многошаговая схема редукции размерности базируется на соотношении

$$\min_{y \in Q} f(y) = \min_{y_1 \in \Pi_1} \min_{y_2 \in \Pi_2(u_1)} \cdots \min_{y_N \in \Pi_N(u_{N-1})} f(y) \qquad (4)$$

где $u_i = (y_1, ..., y_i)$, $v_i = (y_{i+1}, ..., y_N)$, а области минимизации $\Pi_{i+1}(u_i)$ формируются на основе сечений

$$S_1 = Q, \quad S_{i+1}(u_i) = \{(u_i, v_i) \in Q\}, \quad S_{i+1}(u_i) = \{(u_i, v_i) \in Q\}, \quad 1 \le i \le N-1$$

множества Q и определяются как

$$\Pi_{i+1}(u_i) = \{y_{i+1} \in R^1 : \exists (y_{i+1}, v_{i+1}) \in S_{i+1}(u_i)\}.$$

Соотношение (4) позволяет свести решение исходной многомерной задачи к решению семейства рекурсивно связанных одномерных подзадач, для которых возможно применение эффективных алгоритмов глобального поиска. Использование в качестве одномерных алгоритмов оптимизации параллельных характеристических методов глобального поиска [3] позволяет при решении многомерной задачи (1)–(3) использовать до 2^N параллельно работающих процессоров и добиться высокой эффективности распараллеливания [5].

Для решения задач со сложными ограничениями в структуру многошаговой схемы был встроен индексный метод [2] учета ограничений, который в отличие от классического метода штрафных функций не требует подбора каких-либо констант типа константы штрафа или «выравнивающих» коэффициентов при ограничениях и допускает естественное обобщение характеристических алгоритмов на указанный класс задач.

При проведении вычислительных экспериментов были реализованы две схемы распараллеливания решающих правил характеристиче-

ских алгоритмов, обеспечивающих выбор координат новых испытаний (вычислений значений целевых функций одномерных подзадач). В синхронной схеме координаты испытаний выбирались после завершения всех предшествующих параллельно исполняемых испытаний, а в асинхронной реализации [4] освободившийся процессор получал координату нового испытания, не ожидая завершения работы других процессоров.

Итоги эксперимента подтвердили применительно к многоэкстремальным задачам с ограничениями эффективность предложенной схемы распараллеливания, отмеченную ранее [5] для безусловных многоэкстремальных задач. Экспериментально также подтверждены теоретические результаты [4] ускорения вычислений при асинхронной схеме по сравнению с синхронной.

Литература

1. *Стронгин Р.Г.* Численные методы в многоэкстремальных задачах. Информационно- статистический подход. М.: Наука, 1978.
2. *Strongin R.G.*, *Sergeyev Ya.D*. Global optimization with non-convex constraints: Sequential and parallel algorithms, Kluwer Academic Publishers, Dordrecht, Netherlands. 2000.
3. *Strongin R.G.*, *Sergeyev Ya.D.*, *Grishagin V.A.* Parallel Characteristical Algorithms for Solving Problems of Global Optimization // Journal of Global Optimization,10, 1997. P. 185–206.
4. *Sergeyev Ya.D.*, *Grishagin V.A*. Parallel asynchronous global search and the nested optimization scheme, Journal of Computational Analysis & Applications, 3(2), 2001. P. 123–145.
5. *Гришагин В.А.*, *Песков В.В*. Повышение эффективности параллельных рекурсивных схем редукции размерности // Высокопроизводительные параллельные вычисления на кластерных системах: Матер. Международ. научно-практич. семинара / Н.Новгород: Изд-во ННГУ. 2002. С. 56–60.

ПОСТРОЕНИЕ РАСПИСАНИЙ ПАРАЛЛЕЛЬНОГО

ВЫПОЛНЕНИЯ ЗАДАЧ В КЛАСТЕРНЫХ СИСТЕМАХ

А.М. Данильченко, С.Н. Защипас

Житомирский инженерно-технологический институт,
Житомир, Украина

Введение

В последнее время во всем мире бурно развивается относительно новая область вычислительной индустрии: метакомпьютинг. Понятие метакомпьютинг – это абстракция с помощью которой компьютерные кластеры и отдельные компьютеры в распределенной системе могут быть представлены в виде единого виртуального компьютерного ресурса [1–3]. В свою очередь компьютерный кластер есть набор полноценных компьютеров-узлов, которые связаны между собою сетью и предоставлены конечному пользователю в виде единого вычислительного ресурса [4,5].

Обычно кластеры строятся с целью создания высокопроизводительной и высоконадежной вычислительной системы. Очевидно, что метакомпьютер и кластер есть подобными, друг другу, абстракциями, хотя кластер может быть составной частью метакомпьютера.

Развитие метакомпьютинга можно связать прежде всего с возростающими требованиями человечества к вычислительным системам. Причем с экономической точки зрения метакомпьютеры или кластеры часто являются более выгодными чем специализированные суперкомпьютеры, которые создаются в единичном экземпляре. В дальнейшем мы будем рассматривать кластерную систему (КС), как базовую абстракцию метакомпьютерных систем.

Математическая модель построения расписания

Использование кластерных систем накладывает определенные требования на программное обеспечение, которое на них выполняется. Эффективность использования таких систем зависит от технических характеристик системы и задач, которые на ней решаются.

Задачи, которые решаются в кластерной системе можно условно разделить на два типа:

– последовательные задачи, алгоритмы которых слабо поддаются распараллеливанию. Задача такого типа обычно полностью выполняется на какому-то одном узле кластера. В кластерной

системе может одновременно (параллельно) выполняться много подобных задач;

– задачи, которые состоят из частей которые можно выполнять параллельно. Для программирования таких задач обычно используют специальные библиотеки, которые поддерживают распараллеливание данных (MPI,PVM [7,8]). Для диспетчеризации таких задач можно использовать модель динамического составления расписаний.

Рассмотрим процесс решения задач второго типа в кластерной системе.

Пусть существует множество данных A которое должно быть преобразовано в множество данных A'. Для подобного преобразования необходимо использовать алгоритм P, причем A должно быть разбито определенным образом на n подмножеств: $A = A_1 \bigcup A_2 \bigcup ... \bigcup A_n$ так, что преобразование $P(A_i) = A_i'$, где $i \in [1; n]$ и $A_1' \bigcup A_2' \bigcup ... \bigcup A_n' = A'$

Далее над каждым из множеств A_i' проводятся некоторые вычисления при помощи функции $F(A_i')$ и полученные частичные результаты используются для формирования окончательного результата на ведущем узле.

В кластерной системе сформированную задачу можно представить следующей последовательностью этапов:

1 этап: Формирование подмножеств A_i.

2 этап: Передача A_i на свободный узел кластера N_j, где $j \in [1, m]$, m – количество узлов кластера.

3 этап: Выполнение алгоритма P на узле N_j.

4 этап: Возвращение результата на ведущий узел КС.

5 этап: Формирование задач для вычисления функции $F(A_i')$.

6 этап: Передача данных на свободный узел кластера для вычислений

7 этап: Вычисление функции $F(A_i')$.

8 этап: Возвращение частичных результатов на ведущий узел КС.

9 этап: Формирование общего результата.

Так как операции передачи информации на свободный узел кла-

стера, выполнение алгоритма P на узле N_i и возвращение результата на ведущий узел КС (2-4 этапы) выполняются строго последовательно и полностью блокируют канал передачи и узел кластера, то эти операции можно объединить в одну с общим временем выполнения. Аналогичное преобразование можно осуществить с 6-8 этапами. Тогда весь процесс превращения данных в КС можно представить пятью этапами.

1. Последовательное формирования подмножеств A_i.
2. Параллельная передача данных и выполнение алгоритма P на узле N_i.
3. Последовательное формирование данных для вычисления $F(A_i')$.
4. Параллельная передача данных и вычисление $F(A_i')$.
5. Последовательное формирование общего результата.

Время обработки данных на каждом этапе обозначим через t_j^k, $k = 1,2,3,4,5 \quad j \in [1,n]$. Тогда процесс решения задачи оптимизации роботи КС сводится к минимизации общего времени выполнения пятиэтапных задач. Если количество узлов кластера $m \geq n$ – количества подмножеств данных (для каждого подмножества всегда существует свободный узел), то функцию цели можно записать следующим образом (1):

$$\max_{1 \leq p \leq q \leq n} \left(\sum_{j=1}^{p} t_{i_j}^1 + t_{i_p}^2 + \sum_{j=p}^{q} t_{i_j}^3 + t_{i_q}^4 + \sum_{j=q}^{n} t_{i_j}^5, \sum_{j=1}^{n} t_{i_j}^1 + \sum_{j=p}^{q} t_{i_j}^3 + t_{i_q}^4 + \sum_{j=q}^{n} t_{i_j}^5, \right.$$

$$\left. \sum_{j=1}^{p} t_{i_j}^1 + t_{i_p}^2 + \sum_{j=p}^{n} t_{i_j}^3 + \sum_{j=1}^{n} t_{i_j}^5, \sum_{j=1}^{n} t_{i_j}^1 + \sum_{j=1}^{n} t_{i_j}^3 + \sum_{j=1}^{n} t_{i_j}^5 \right) \to \min_{\pi_i}, \qquad (1)$$

где $\pi_i = (i_1, ..., i_n)$ - перестановка чисел от 1 к n.

В общем случае сформулированная задача является NP полной в сильном смысле даже для четырех этапов ($t_{i_j}^5 = 0$). Однако, при выполнении хотя бы одного из условий (2;3) (что часто имеет место в практических задачах)

$$t_{i_p}^2 \leq \sum_{j=p+1}^{n} t_{i_j}^1 + \sum_{j=1}^{p-1} t_{i_j}^3, \qquad (2)$$

$$t_{i_q}^4 \le \sum_{j=q+1}^{n} t_{i_j}^3 + \sum_{j=1}^{q-1} t_{i_j}^5 \qquad (3)$$

первый член под знаком максимума в выражении (1) можно отбросить, и задача минимизация функции (1) сводится к задаче минимизации функции (4) если выполняется условие (2)

$$\max_{1\le q\le n}\left(\sum_{j=1}^{q} t_{i_j}^3 + t_{i_q}^4 + \sum_{j=q}^{n} t_{i_j}^5 \right) \to \min_{\pi_i} \qquad (4)$$

и к задаче минимизации функции (5) если выполняется условие (3)

$$\max_{1\le p\le n}\left(\sum_{j=1}^{p} t_{i_j}^1 + t_{i_p}^2 + \sum_{j=p}^{n} t_{i_j}^3 \right) \to \min_{\pi_i}. \qquad (5)$$

В литературе подобные задачи известны, как задача «О редакторе» [6] и довольно хорошо исследованы. Например, оптимальное решение функции (5) находится при помощью следующего алгоритма: сначала, на обработку в КС, запускаются подзадачи (подмножества) для которых $t_j^1 \le t_j^3, j \in [1;n]$ в последовательности возрастания величин $t_j^1 + t_j^2$, а далее подзадачи для которых $t_j^1 > t_j^3$ в последовательности уменьшения величин $t_j^2 + t_j^3$.

Аналогично находится и решение функции(4).

В случае если $n > m$ рассматриваемая задача есть NP-полной в сильном смысле даже для двух этапов. Для решения этой задачи применяется эвристический алгоритм, в котором на первом этапе все подзадачи упорядочиваются по алгоритму 1, а в случае, если для выполнения подзадачи на параллельных этапах свободный узел кластера отсутствует, то подзадача становится в очередь и запускается на первом свободном узле.

Следует отметить, что для решения поставленной задачи не нужно знание точного времени обработки подзадач на этапах, а достаточно информации об относительном времени выполнения (больше-меньше).

Для решения подобного класса задач был написан модуль на языке С, который применяется для упорядочения запуска подзадач на выполнение в рамках кластерной системы. Данный модуль может применятся как для программ написанных на С так и для программ написа-

ных на других языках (Pascal,Java,Perl).

Литература

1. *Larry Smarr*, *C.E. Catlett*. Metacomputing. Communications of the ACM, 35(6):44-52, June 1992.
2. *G.C. Fox*, *R.D. Williams*, *P.C. Messina*. Parallel Computing Works! Morgan Kaufmann Publishers, Inc., 1994.
3. Что такое Мета-компьютеринг? http://www.parallel.ru/computers/ rewievs/meta-computing.html.
4. Что такое Beowulf? http://www.parallel.ru/computers/rewievs/ beowulf.html.
5. *В. Савяк*. Эффективные кластерные решения. http://www.ixbt.com /cpu/clustering.shtml.
6. *Танаев В.С.*, *Шкурба В.В.* Введение в теорию расписаний. М.: Наука, 1975.
7. MPI:A Message-Passing Interface Standard –http://parallel.ru/docs/ Parallel/mpi1.1/mpi-report.html.
8. PVM: Parallel Virtual Machine – http://www.epm.ornl.gov/pvm/

ЧИСЛЕННОЕ МОДЕЛИРОВАНИЕ ТЕЧЕНИЙ СТАТИСТИЧЕСКИМИ МЕТОДАМИ МОНТЕ–КАРЛО СОВМЕСТНО С РЕШЕНИЕМ УРАВНЕНИЙ НАВЬЕ–СТОКСА

С.В. Денисихин

Балтийский государственный технический университет «Военмех», Санкт-Петербург

С.Е. Журавлева, В.П. Мемнонов

СПбГУ

В современных высокотехнологичных устройствах все чаще встречаются потоки в переходном режиме. Одним из методов численного решения подобных задач является метод прямого статистического моделирования (ПСМ). Однако в большинстве задач потоки в переходном режиме составляют только часть течения. Например, в системах магнитной записи винчестерного типа наряду с очень узким кана-

лом между магнитной головкой и жестким диском, высотой порядка одного-двух длин свободного пробега молекул в воздухе, имеется еще обтекание самой головки с линейным размером L порядка 10^{-3}м. При таких характерных размерах течение, за исключением весьма малых областей около входа и выхода из канала, следует описывать с помощью уравнений Навье–Стокса. Используя нестационарные решения этих уравнений, на границах этих малых областей мы получаем текущие значения плотности и скорости воздуха, которые дают возможность промоделировать микроскопические потоки внутрь той части, где решение строится методом ПСМ. Тем самым для него мы получаем реалистические граничные условия. Такой подход до сих пор не был осуществлен в пока еще немногочисленных публикациях, связанных с расчетами новейших систем магнитной записи.

Следует, правда, отметить, что пока мы действуем методом последовательных приближений. За нулевое приближение берется нестационарное решение уравнений Навье–Стокса во всей области, включая и канал. Предполагаем, что влияние той части течения в канале и около его входов, где это решение в локальном замкнутом объеме может быть плохим, на остальное течение достаточно быстро убывает с расстоянием. Далее на границах областей около входов в канал с размером l порядка нескольких десятков длин свободного пробега, то есть l~10^{-6} м, применяем макропараметры этого нулевого приближения для моделирования микроскопических потоков молекул внутрь этих областей и используем их в качестве граничных условий для получения решения там методом ПСМ. В свою очередь получаемые из этого решения макропараметры на границе вместе с этими величинами нулевого приближения дают возможность определить на границе разности макропараметров и, в принципе, в следующем навье-стоксовском, возможно, уже линеаризованном приближении их уточнить. Но практически в этой задаче важнейшими параметрами являются давление на магнитную головку, распределение давления вдоль нее, а также распределение плотности и средней скорости вдоль канала. Эти величины были получены и будут представлены в докладе. Представлены также относительные разности макропараметров в граничных ячейках для нулевого и первого приближений, которые оказались настолько малыми, что позволили ограничиться этими первыми приближениями.

ПАРАЛЛЕЛЬНЫЕ БЛОЧНЫЕ МЕТОДЫ РЕШЕНИЯ ДИНАМИЧЕСКИХ ЗАДАЧ НА SIMD СТРУКТУРАХ

О.А. Дмитриева

Донецкий национальный технический университет

Введение

В последнее время появилось большое количество численных методов интегрирования дифференциальных уравнений, которые ориентированы на вычислительные системы с параллельной архитектурой. Однако практическое использование большинства методов не всегда оправдано, поскольку многие из них либо обладают численной неустойчивостью, либо имеют очень сложную структуру, приводящую к потере эффективности. К методам, лишенным указанных недостатков, можно отнести блочные [1]. Блочным будем называть метод, при котором для блока из k точек новые k значений функции вычисляются одновременно. Эта особенность методов, во-первых, согласуется с архитектурой параллельных вычислительных систем, а, во-вторых, позволяет вычислять коэффициенты разностных формул не в процессе интегрирования, а на этапе разработки метода, что значительно увеличивает эффективность счета. В то же время для этих методов отсутствуют оценки погрешностей и условия сходимости. Поэтому данная работа посвящена исследованию сходимости и получению оценок погрешностей параллельных блочных методов решения задачи Коши для обыкновенных дифференциальных уравнений. Рассмотрим решение задачи Коши

$$\frac{dx}{dt} = f(t,x),\ x(t_0) = x_0 \qquad (1)$$

k точечным блочным разностным методом. Множество M точек равномерной сетки t_m, $m = 1,2,\ldots,M$ и $t_M = T$ с шагом τ разобьем на N блоков, содержащих по k точек, при этом $kN \geq M$. В каждом блоке введем номер точки $i = 0,1,\ldots,k$ и обозначим через $t_{n,i}$ точку n-го блока с номером i. Точку $t_{n,0}$ назовем началом блока n, а $t_{n,k}$ – концом блока. Очевидно, что имеет место $t_{n,k} = t_{n+1,0}$. Условимся начальную точку в блок не включать. Если для расчета значений в новом блоке используется только последняя точка предшествующего – будем говорить об

одношаговых, а если все точки предшествующего блока – многошаговых блочных методах.

В общем случае уравнения многошаговых разностных методов для блока из k точек с учетом введенных обозначений можно записать в виде

$$\frac{u_{n,i}-u_{n,0}}{i\tau}=\sum_{j=1}^{k} b_{i,j}F_{n-1,j}+\sum_{j=1}^{k}a_{i,j}F_{n,j},\ i=\overline{1,k},\ \ n=1,2,\dots N, \qquad (2)$$

где $F_{n,j}=f(t_n+j\tau,u(t_n+j\tau))$.

Для одношаговых разностных методов уравнения могут быть представлены как

$$\frac{u_{n,i}-u_{n,0}}{i\tau}=b_iF_{n,0}+\sum_{j=1}^{k}a_{i,j}F_{n,j},\ i=\overline{1,k},\ n=1,2,\dots N, \qquad (3)$$

Погрешность аппроксимации многошаговых блочных методов

Выражение для погрешности аппроксимации рассматриваемого разностного метода (2) на решении уравнения (1) запишем следующим образом

$$r_{n,i}=-\frac{x_{n,i}-x_{n,0}}{i\tau}+\sum_{j=1}^{k} b_{i,j}x'_{n-1,j}+\sum_{j=1}^{k}a_{i,j}x'_{n,j},\ i=\overline{1,k}, \qquad (4)$$

где

$$x_{n,i}=x(t_n+i\tau),\ x'_{n,j}=f(t_{n,j},x_{n,j})=f(t_n+jt,x(t_n+j\tau)),$$
$$x'_{n-1,j}=f(t_{n-1,j},x_{n-1,j})=f(t_n-(k-j)\tau,\,x(t_n-(k-j)\tau)).$$

Воспользуемся подходом, предложенным в работе [2]. Раскладывая $x_{n,i}$, $x'_{n,j}$ и $x'_{n-1,j}$ в ряды Тейлора в окрестности точки t_n, и учитывая, что конечная точка предыдущего блока совпадает с начальной точкой следующего блока, будем иметь

$$r_{n,i}=-\sum_{l=1}^{p}\frac{(i\tau)^{l-1}}{l!}x^{(l)}{}_{n,0}+\sum_{j=1}^{k-1}b_{i,j}\sum_{l=1}^{p}\frac{((j-k)\tau)^{l-1}}{(l-1)!}x^{(l)}_{n,0}+b_{i,k}x'_{n,0}+$$
$$+\sum_{j=1}^{k}a_{i,j}\sum_{l=1}^{p}\frac{(j\tau)^{l-1}}{(l-1)!}x^{(l)}_{n,0}+O(\tau^p). \qquad (5)$$

Сгруппируем члены с одинаковыми производными и изменим порядок суммирования в последнем выражении, тогда получим

$$r_{n,i} = x'_{n,0}\Big(\sum_{j=1}^{k} b_{i,j} + \sum a_{i,j} - 1 \Big) + \sum_{l=2}^{p} \frac{\tau^{l-1}}{(l-1)!} x_{n,0}^{(l)} \times$$

$$\times \Big[\sum_{j=1}^{k} a_{i,j} * j^{l-1} + \sum_{j=1}^{k-1} b_{i,j} (j-k)^{l-1} - \frac{i^{l-1}}{l} \Big] + O(\tau^{p}). \qquad (6)$$

Отсюда следует, что погрешность аппроксимации имеет порядок p, если выполнены условия

$$\sum_{j=1}^{k} b_{i,j} + \sum_{j=1}^{k} a_{i,j} = 1,$$

$$\sum_{j=1}^{k} j^{l-1} a_{i,j} + \sum_{j=1}^{k-1} (j-k)^{l-1} b_{i,j} = \frac{i^{l-1}}{l}, \quad i = \overline{1,k},\ l = \overline{2,p}. \qquad (7)$$

Система уравнений (7) для каждого фиксированного i содержит p уравнений и $2k$ неизвестных $b_{i,1}, b_{i,2}, ..., b_{i,k}, a_{i,1}, a_{i,2}, ..., a_{i,k}$. Потребуем, чтобы $p = 2k$, тогда из системы (7) можно будет определить все неизвестные коэффициенты. Отсюда следует, что наивысший порядок аппроксимации многошагового k-точечного блочного метода равен 2k. Его погрешность в соответствии с (6) определяется формулой

$$r_{n,i} = \frac{\tau^{2k}}{(2k)!} x_{n,0}^{(2k+1)} \left[\sum_{j=1}^{k} j^{2k} a_{i,j} + \sum_{j=1}^{k-1} (j-k)^{2k} b_{i,j} - \frac{i^{2k}}{2k+1} \right] + \qquad (8)$$

$$+ O\left(\tau^{2k+1}\right),\ i = \overline{1,k}\ .$$

Для одношаговых блочных методов аппроксимацию порядка p обеспечивают следующие условия [3]

$$b_i + \sum_{j=1}^{k} a_{i,j} = 1, \quad \sum_{j=1}^{k} j^{l-1} a_{i,j} = \frac{i^{l-1}}{l}, \quad i = \overline{1,k},\ l = \overline{2,p}. \qquad (9)$$

Система уравнений (9) для каждого фиксированного i содержит p уравнений и k+1 неизвестных $b_i, a_{i,1}, a_{i,2}, ..., a_{i,k}$. Если потребовать, чтобы p=k+1, тогда из системы (9) можно будет определить все неизвестные коэффициенты. Отсюда следует, что *наивысший порядок аппроксимации одношагового k-точечного блочного метода равен* k+1. Его погрешность определяется формулой

$$r_{n,i} = \frac{\tau^{k+1}}{(k+1)!} x_{n,0}^{(k+2)} \left[\sum_{j=1}^{k} j^{k+1} a_{i,j} - \frac{i^{k+1}}{k+2} \right] + O\left(\tau^{k+2}\right),\ i = \overline{1,k}\ . \qquad (10)$$

Примеры многоточечных разностных методов

Элементы b_{ij} и $a_{i,j}$,i,j = 1,2,…,k матриц B и A соответственно можно найти в зависимости от выбранного метода, решая системы (7) или (9) для конкретной размерности блока k. Решение систем выполним с помощью пакета *Mathematica®* (Wolfram Research Inc.). Получившиеся при этом коэффициенты блочных методов будут иметь следующий вид:

– для двухточечного блока многошагового метода

$$\mathbf{B} = \left\{ \{-\frac{1}{24}, \frac{13}{24}\}, \{0, \frac{1}{6}\} \right\}, \qquad \mathbf{A} = \left\{ \{\frac{13}{24}, -\frac{1}{24}\}, \{\frac{2}{3}, \frac{1}{6}\} \right\};$$

– для четырехточечного блока многошагового метода

$$\mathbf{B} = \{ \{ -\frac{191}{120960}, \frac{1879}{120960}, -\frac{353}{4480}, \frac{68323}{120960}\}, \{0, \frac{1}{1512}, -\frac{1}{105}, \frac{167}{840}\},$$
$$\{-\frac{13}{13440}, \frac{39}{4480}, -\frac{171}{4480}, \frac{2777}{13440}\}, \{\frac{2}{945}, -\frac{16}{945}, \frac{2}{35}, -\frac{53}{1890}\}\}$$

$$\mathbf{A} = \{ \{\frac{68323}{120960}, -\frac{353}{4480}, \frac{1879}{120960}, -\frac{191}{120960}\}, \{\frac{586}{945}, \frac{167}{840}, -\frac{1}{105}, \frac{1}{1512}\},$$
$$\{\frac{1299}{4480}, \frac{369}{896}, \frac{337}{2688}, -\frac{3}{896}\}, \{\frac{446}{945}, \frac{2}{35}, \frac{362}{945}, \frac{139}{1890}\}\}.$$

В случае выбора одношагового метода с размерностью блока $k = 4$ элементы матрицы A и вектора B, примут следующий вид

$$\mathbf{B} = \{ \frac{251}{720}, \frac{29}{180}, \frac{9}{80}, \frac{7}{90} \}$$

$$\mathbf{A} = \{ \{\frac{323}{360}, -\frac{11}{30}, \frac{53}{360}, -\frac{19}{720}\}, \{\frac{31}{45}, \frac{2}{15}, \frac{1}{45}, -\frac{1}{180}\},$$
$$\{\frac{17}{40}, \frac{3}{10}, \frac{7}{40}, -\frac{1}{80}\}, \{\frac{16}{45}, \frac{2}{15}, \frac{16}{45}, \frac{7}{90}\}\}$$

Аналогично определяются коэффициенты разностных уравнений для блоков с любым количеством точек. Погрешности аппроксимации многошагового или одношагового блочных методов могут быть получены по формулам (8) или (10). В качестве примера можно привести значения погрешностей для четырехточечных многошагового

$$r_{n,1} = -\frac{2497\tau^8}{3628800} x_{n,0}^{(9)} + O\left(\tau^9\right), r_{n,2} = \frac{23\tau^8}{226800} x_{n,0}^{(9)} + O\left(\tau^9\right),$$

$$r_{n,3} = -\frac{27\tau^8}{44800} x_{n,0}^{(9)} + O\left(\tau^9\right), r_{n,4} = O\left(\tau^9\right)$$

и одношагового

$$r_{n,1} = -\frac{3\tau^5}{160} x_{n,0}^{(6)} + O\left(\tau^6\right), r_{n,2} = -\frac{\tau^5}{180} x_{n,0}^{(6)} + O\left(\tau^6\right), r_{n,3} =$$

$$= -\frac{3\tau^5}{160} x_{n,0}^{(6)} + O\left(\tau^6\right), r_{n,4} = O\left(\tau^6\right).$$

блочных методов в соответствующих точках блока. Для оценки полученных методов исследовались вопросы сходимости и оценки погрешности. Доказана следующая

Теорема. Пусть правая часть уравнения (1) $f(t,x)$ удовлетворяет условию Липшица по второму аргументу с константой L, и r_n – невязка многошагового к- точечного блочного метода (2) определенная согласно (6) с оценкой

$$\| r_n \|_c = O(\tau^{2k}) \tag{11}$$

Тогда при $\tau < \tau_0 = \dfrac{1}{kL\|A\|_c}$ и $kn\tau \le T$ для погрешности метода имеет место оценка

$$\|z_n\|_c \le C\tau^{2k} \frac{E^{T\left(L\|\mathbf{B}\|_c + L\|\mathbf{A}\|_c\right)} - 1}{\left(L\|\mathbf{B}\|_c + L\|\mathbf{A}\|_c\right)k} + E^{TL\left(\|\mathbf{B}\|_c + \|\mathbf{A}\|_c\right)} \|z_0\|_c . \tag{12}$$

Следствие. Если разностное уравнение (2) аппроксимирует исходное уравнение (1), то решение разностной задачи (2) сходится при $\tau \to 0$ к решению исходной задачи (1), причем порядок точности совпадает с порядком аппроксимации.

Алгоритмы параллельного решения нелинейной разностной задачи

Для вычисления приближенных значений решения задачи Коши (1) необходимо решить нелинейную систему уравнений (2). Для этого используется либо метод простой итерации

$$u_{n,i,0} = u_{n,0} + i\tau\sum_{j=1}^{k} c_{i,j} F_{n-1,j},\ i=\overline{1,k},\ n=1,2,\ldots,$$

$$u_{n,i,s+1} = u_{n,0} + i\tau(\sum_{j=1}^{k} b_{i,j} F_{n-1,j} + \sum_{j=1}^{k} a_{i,j} F_{n,j,s}),\ i=\overline{1,k}, \qquad (13)$$

$$s=\overline{0,2k-1},\ n=1,2,\ldots,$$

который позволяет проводить вычисления параллельно для каждого узла блока. После выполнения $2k$ шагов вычислений по формулам (13) локальная ошибка в узлах блока будет иметь порядок $O(\tau^{2k+1})$.

Для одношаговых блочных методов формулы итерационного процесса могут быть получены в следующем виде

$$u_{n,i,0} = u_{n,0} + i\tau F_{n,0},\ i=\overline{1,k},\ n=1,2,\ldots,$$

$$u_{n,i,s+1} = u_{n,0} + i\tau(b_i F_{n,0} + \sum_{j=1}^{k} a_{i,j} F_{n,i,s}),\ i=\overline{1,k}, \qquad (14)$$

$$s=\overline{0,k-1},\ n=1,2,\ldots$$

Локальная ошибка в узлах блока будет иметь порядок $O(\tau^{k+2})$ после выполнения $k+1$ шагов вычислений по формулам (14) Последующие итерации повышения порядка точности результатов не дадут. Система уравнений (2) может быть решена также с помощью метода Ньютона.

Заключение

Предложенные и обоснованные в статье блочные методы решения обыкновенных дифференциальных уравнений могут быть без особых затруднений распространены и на решение систем уравнений. Также нужно отметить, что к особенностям методов, приведенных в статье, можно отнести, во-первых, хорошую согласованность с архитектурой параллельных вычислительных систем, поскольку имеется возможность одновременного вычисления новых значений функции для всех точек блока. А, во-вторых, коэффициенты разностных формул при этом вычисляются не в процессе интегрирования, а на этапе разработ-

ки метода, что значительно увеличивает эффективность счета.

Литература

1. Системы параллельной обработки: Пер. с англ. / Под. ред. Д. Ивенса. М.: Мир, 1985.
2. *Дмитриева О.А.* Анализ параллельных алгоритмов численного решения систем обыкновенных дифференциальных уравнений методами Адамса–Башфорта и Адамса–Моултона // Математическое моделирование. 2000. Т. 12, №5. С. 81–86.
3. *Feldman L.*, *Dmitrieva O.*, *Gerber S.* Abbildung der blockartigen Algoritmen auf die Parallelrechnerarchitekture. 16 Symposium Simulationstechnik ASIM 2002, Rostock, 10.09 bis 13.09 2002. Erlangen: Gruner Druck, 2002. P. 359–364.

ПАРАЛЛЕЛЬНЫЕ ФЕЙЕРОВСКИЕ МЕТОДЫ ДЛЯ СИЛЬНО СТРУКТУРИРОВАННЫХ СИСТЕМ ЛИНЕЙНЫХ НЕРАВЕНСТВ И УРАВНЕНИЙ

И.И. Еремин, Л.Д. Попов

ИММ УрО РАН, г.Екатеринбург

Моделирование реальных экономических, технических, социальных, биологических и других процессов зачастую приводит к необходимости решить систему линейных неравенств и уравнений большой размерности с сильно структурированной матрицей коэффициентов. Под свойством сильной структурированности матрицы ниже понимается наличие яркой специфики в расположении ее ненулевых элементов в предположении, что общее их количество чрезвычайно мало (последнее условие обычно называют разреженностью матрицы). Примером могут служить блочно-диагональные матрицы с различным типом окаймления, матрицы циклической связности и многие другие.

Один из общих подходов к построению численных методов анализа больших систем неравенств и уравнений, в частности линейных, опирается на теорию фейеровских отображений и свойства порождаемых ими итерационных процессов [1, 2]. Среди таких отображений для эффективного учета особенностей расположения ненулевых коэффициентов разреженных матриц более всего подходят отображения с

несовпадающими пространствами образов, изучение которых было недавно начато в [3]. В докладе этот подход получает дальнейшее развитие, при этом основными являются вопросы компьютерной реализации и распараллеливания вычислений. Последние обсуждается применительно к вычислительным сетям, составленным из микропроцессоров с локальной памятью, соединенных друг с другом линками непосредственно или опосредованно через коммутирующее устройство, позволяющее элементам сети обмениваться между собой сообщениями. В частности, к такому типу многопроцессорных систем относятся системы МВС-100 и МВС-1000.

Очень важно, что при построении фейеровских операторов применительно к системам линейных неравенств и задачам линейного программирования (последние могут быть сведены к системе линейных неравенств с сильно структурированной матрицей коэффициентов) все особенности структуры исходных данных могут быть учтены, выстроены в сегменты параллельных для обработки блоков при минимальных обменах между ними. Это дает широкий простор для использования принципа распараллеливания и порождает определенный оптимизм в использовании фейеровских процессов для решения больших задач на многопроцессорных компьютерах самой разной архитектуры. Некоторый опыт практического применения методов фейеровского типа изложен в [4, 5].

Работа выполнена при поддержке РФФИ, гранты № 00-15-96041, № 01-01-00563, а также целевой программы поддержки междисциплинарных проектов, выполняемых в содружестве ученых УрО РАН и СО РАН.

Литература

1. *Еремин И.И.*, *Мазуров В.Д.* Нестационарные процессы математического программирования. М.: Наука, 1979.
2. *Еремин И.И.* Теория линейной оптимизации. Екатеринбург: УрО РАН, 1999.
3. *Еремин И.И.* Синтез фейеровских отображений с несовпадающими пространствами их образов // ДАН. Т. 378, №1, 2001. С. 11–13.
4. *Бердникова Л.Д.*, *Попов Л.Д.* О применении декомпозиции при реализации фейеровских методов решения больших систем линейных неравенств на МВС-100. Сб. Алгоритмы и программные средства параллельных вычислений. Екатеринбург: УрО РАН, 2000, №4. С. 51–62.

5. *Попов Л.Д.* Вопросы реализации методов ЛП в транспьютерных сетях. Сб. Алгоритмы и программные средства параллельных вычислений. Екатеринбург: УрО РАН, 1995, №1. С. 148–156.
6. *Голуб Дж.*, *Ван Лоун Ч.* Матричные вычисления. М.: Мир, 1999.
7. *Ортега Дж.* Введение в параллельные и векторные методы решения линейных систем / Пер. с англ. М.: Мир, 1991.

ТЕСТИРОВАНИЕ КЛАСТЕРНЫХ СИСТЕМ ТГУ И ИОА СО РАН С ПОМОЩЬЮ ПАКЕТА LINPACK

А.О. Есаулов, Н.В. Дмитриева

Томский государственный университет

Введение

Если бы за последние 25 лет авиационная промышленность развивалась столь же стремительно, как и вычислительная техника, то Боинг-767 можно было бы приобрести сегодня за 500 долларов и облететь на нем земной шар за 20 минут, израсходовав при этом 19 литров горючего. По этой аналогии, хотя и не совсем точной, можно судить о темпах снижения стоимости, энергопотребления и роста быстродействия вычислительных машин. За 25 лет скорость вычислений возросла в 200 раз, при этом размеры и потребление электроэнергии у современных ЭВМ стали в 10000 раз меньше, чем у машин сравнимой производительности 25-летней давности.

На всех этапах развития вычислительной техники существовали задачи, для решения которых было недостаточно использовать вычислительные возможности рядовых компьютеров. Это обстоятельство приводило к появлению «очень мощных», сверх- или суперкомпьютеров. Так, первая промышленная электронная вычислительная машина UNIVAC I, установленная в 1951 г. в Бюро переписи населения США, работала примерно в 3 раза быстрее современных ей рядовых машин и в несколько тысяч раз превосходила их по размерам. На сегодняшний день серийные компьютеры, несмотря на превосходство над суперкомпьютерами 60-70-х годов, в значительной степени уступают современным суперкомпьютерам по производительности, размерам и, разумеется, стоимости [3].

Сегодня примеры использования суперкомпьютеров можно найти

не только в стенах военных лабораторий и государственных учреждений. Вот лишь небольшой список областей человеческой деятельности, где использование суперкомпьютеров действительно необходимо:

- автомобилестроение;
- нефте- и газодобыча;
- фармакология;
- прогноз погоды и моделирование изменения климата;
- сейсморазведка;
- проектирование электронных устройств;
- синтез новых материалов;
- и многие, многие другие.

1. Об измерении производительности

1.1. Пиковая и реальная производительность

Естественной характеристикой вычислительной системы является ее производительность, то есть объем вычислений, производимых в единицу времени. Эта характеристика чаще всего используется для сравнения различных типов компьютеров между собой. Без точного знания производительности компьютера исследователь не имеет возможности судить о том, сколько времени потребуется для решения той или иной задачи, а, следовательно, и о целесообразности использования данной машины.

Исторически первым способом оценки производительности было определение пиковой или технической производительности, представляющей собой теоретический максимум быстродействия компьютера при идеальных условиях. Данный максимум определяется как число операций, выполняемое в единицу времени всеми имеющимися в компьютере обрабатывающими логико-арифметическими устройствами. В современных условиях значение производительности измеряется в миллионах операций в секунду – MIPS (millions instructions per second) или миллионах операций с плавающей точкой в секунду – MFLOPS (millions floating point operations per second).

Пиковое быстродействие достигается при обработке бесконечной последовательности не связанных между собой и не конфликтующих при доступе в память команд (то есть, когда результат любой операции не зависит от действий, выполненных другими командами). При этом в современных компьютерах предполагается, что все операнды выбираются из внутрикристальной кэш-памяти данных, а команды - из кэш-памяти команд. Разумеется, подобная ситуация чисто гипотетическая и

на практике ни одна система не в состоянии работать сколько-нибудь длительное время с пиковой производительностью, хотя и может приближаться к этой величине.

Пиковая производительность является единственной по-настоящему объективной оценкой (для ее определения необходимо знать всего несколько параметров ВС) и совершенно не зависит от выполняемой программы. Речь идет о тактовой частоте процессора, которая для подавляющего большинства современных компьютеров определяет темп формирования результатов на выходе арифметического конвейера, о числе арифметических конвейеров и о числе процессоров в системе. Чтобы определить пиковую производительно машин, надо умножить тактовую частоту на количество параллельно выполняемых операций. Например, арифметическое устройство Pentium каждый такт может формировать один результат 64-битной операции с плавающей точкой или два 32-битовых результата целочисленных операций. Следовательно, для Pentium-90 с тактовой частотой 90 МГц пиковая производительность равна 90 MFLOPS при выполнении вычислений с плавающей точкой и 180 MIPS при целочисленной 32-разрядной обработке. Умножив это значение на число процессоров Pentium в ВС, получим ее пиковую производительность при обработке соответствующих операндов.

При выполнении реальных прикладных программ эффективная (реальная) производительность компьютера может весьма существенно (до нескольких раз) быть меньше пиковой. Это объясняется тем, что современные высокопроизводительные микропроцессоры имеют сложную архитектуру (суперконвейерная и суперскалярная обработка, многоуровневая память, и т.д.). Характеристики их функционирования на уровне внутренних устройств существенно зависят от программы и обрабатываемых данных. Поэтому невозможно с необходимой точностью оценить производительность только на основании тактовой частоты их работы, числа затрачиваемых на выполнение одной команды тактов процессора и числа устройств обработки [1].

1.2. Тесты для измерения реальной производительности

Существующие тестовые наборы можно разбить на три группы. Первую группу тестов измерения производительности составляют тесты производителей, разрабатываемые компаниями-изготовителями компьютеров для внутреннего применения - оценивания качества собственных продуктов. Главная особенность данных тестов заключается в том, что они ориентированы на сравнение ограниченного множества

однотипных компьютеров, часто относящихся к одному семейству. Эти тесты позволяют разработчикам компьютеров оптимизировать структурно-технические решения. Например, для оценки производительно микропроцессоров с архитектурой x86 компания Intel в 1992 г. предложила индекс производительности iCOMP (Intel Comparative Microprocessor Performance). В качестве эталонного процессора принят 486 SX-25, значение индекса для которого равно 100. Индекс iCOMP определяется при выполнении смеси операций, состоящей из 67% операций над 16 разрядными целыми, 3% операций над 16 разрядными числами с плавающей точкой, 25% – над 32-разрядными целыми и 5% над 32 разрядными числами с плавающей точкой. К примеру, индексы iCOMP для микропроцессоров 486 SX2-50, Pentium-100 и Pentium-166 равны 180, 815 и 1308 соответственно. Следует отметить, что индекс iCOMP оценивает производительность микропроцессора как такового, а не вычислительной установки, включающей еще оперативную память и внешние устройства.

В IBM имеются специализированные тестовые пакеты для измерения производительности компьютеров с архитектурой мэйнфреймов семейства System/370, System/390, а также тесты для компьютеров с архитектурой AS/400.

Тесты производителей являются почти идеальным средством оценивания быстродействия и технико-экономических показателей систем с одной и той же архитектурой, но разными средствами ее реализации, однако они не могут быть в чистом виде использованы для других компьютеров – сказывается слишком явная ориентация тестов на конкретную «фирменную» архитектуру. Эти тесты используются создателями систем оценки различных вариантов реализации.

Вторую группу составляют стандартные тесты. Стандартные тесты, разработанные для сравнения широкого спектра компьютеров, часто претендуют на роль полностью универсальных средств измерения производительности. В основе подобных амбиций лежит то, что тесты этой категории – продукт деятельности независимых аналитиков (например, Джека Донгарры, предложившего тестовый пакет Linpack), или групп, объединяющих крупнейших производителей компьютеров (SPEC, TPC), что практически исключает возможность ориентации стандартного теста на конкретного поставщика компьютеров. Например, для оценки серверов, обрабатывающих транзакции в реальном времени, в так называемых OLTP-системах, используется тестовый набор компании Transaction Processing Performance Council (TPP-C).

Производительность серверов оценивается числом транзакций, исполняемых за минуту (tpmC).

Третья группа тестов состоит из пользовательских тестов, учитывающих специфику конкретного применения ВС. Пользовательские тесты создаются крупными компаниями, специализирующимися на внедрении компьютерных технологий, или совместными усилиями группы пользователей, объединенных сходством решаемых задач. Эти средства предназначены специально для выбора компьютеров и программного обеспечения, наиболее подходящих под определенные прикладные задачи. В принципе такой подход позволяет получить наиболее точные оценки производительности для конкретного класса приложений, хотя и сопряжен со значительными усилиями пользователей по созданию тестовых программ и проведению испытаний компьютеров.

Сегодня наиболее распространенными являются наборы тестов Linpack, а также наборы тестов компании SPEC (Standard Performance Evaluation Corporation) – SPEC (89, 92 и 95), и тесты Transaction Processing Performance Council.

1.3. Тесты Linpack (LINear equations software PACKage)

Linpack, впервые опубликованный в 1976 году, явился детищем Джека Донгарры из Университета штата Теннеси.

Этот набор тестов представляет собой совокупность программ решения задач линейной алгебры. В качестве параметров используются: порядок матрицы (например, 100x100, 1000x1000), формат значений элементов матриц (одинарная или двойная точность представления элементов матриц), способ компиляции (с оптимизацией или без оптимизации), а также возможность применения оптимизированной библиотеки стандартных функций. Вообще говоря, на тестах Linpack при больших размерностях обрабатываемых матриц почти все компьютеры демонстрируют производительность в диапазоне от 0.8 до 0.95 от пикового значения. Это наблюдение объясняется тем, что проблемная ориентация большинства современных компьютеров расчет по формулам при решении хорошо формализованных задач.

По результатам тестирования с использованием тестов Linpack с июня 1993 г. в университете Маннгейма (Mannheim) в Германии и в Netlib в США ведется список Top-500, включающий 500 самых производи тельных компьютерных установок в мире. Этот список легко найти в Internet по его названию.

HPL представляет собой пакет, ориентированный на решение сис-

тем линейных алгебраических уравнений большой размерности на компьютерах с распределенной памятью (MPP-компьютеры). При этом матрица системы заполняется случайными вещественными числами с двойной точностью (8 байт). Для пакета необходим параллельный компьютер, на котором установлена система MPI (Message Passing Interface).

В основу тестов HPL положены методы LU-факторизации [2].

Пакет проводит расчеты с применением различных методик и на основе времени, на них затраченного определяет производительность. По сути дела каждый раз, когда говорят о производительности, полученной с использованием тестов типа Linpack, имеют в виду максимальную производительность, достигнутую для некоторого сочетания параметров. Лучшая производительность, которую можно достигнуть, зависит от большого числа параметров. Оптимальный набор параметров, вообще говоря, свой для конкретной вычислительной системы. Исчерпывающую информацию, касающуюся HPL, а также последнюю версию пакета можно найти на **www.netlib.org/benchmark/hpl**.

Пакет HPL базируется на стандартных библиотеках процедур линейной алгебры BLAS (Basic Linear Algebra Subroutines), ATLAS (Automatic Linear Algebra Subroutines) и VSIPL (Vector Signal Image Processing Library).

BLAS (**www.netlib.org/blas**) представляет собой библиотеку «строительных блоков», то есть основных процедур векторной и матричной алгебры. Эта библиотека, как правило, распространяется в виде исходных подпрограмм на языке C или на Fortran, а ответственность за компиляцию для конкретной Unix системы лежит на пользователе. Существует набор откомпилированных вариантов пакета BLAS, оптимизированных для тех или иных машинных систем, однако использование этих версий пакета не рекомендуется из-за возможности неоптимизированности версии. Альтернативным способом является использование пакетов автоматического генерирования кода, таких как ATLAS (**www.netlib.org/atlas**). Этот пакет автоматически создает вариант BLAS, оптимальный для рассматриваемой системы.

VSIPL представляет собой интерфейс программиста (API) и является открытым стандартом, поддерживаемым продавцами аппаратного и программного обеспечения, а также академическими и правительственными лабораториями. Более подробную информацию по алгоритмам, входящим в VSIPL, можно найти по адресу **www.vsipl.org.**

2. Тестирование кластеров ТГУ и ИОА СО РАН

С помощью пакета HPL производилось тестирование кластеров ТГУ и ИОА СО РАН. Кластер ТГУ(http://**cluster.tsu.ru**) расположен в Интернет-центре госуниверситета. Девять его двухпроцессорных элементов Pentium-III-650 MGz объединены сетью 100 Мбит-Ethernet. После нескольких запусков тестов HPL для кластера ТГУ определены параметры, при которых достигается максимальная производительность:

1	# of problem size
14400	Ns
1	# of NBs
180	NBs
1	# of process grids (P x Q)
3	Ps
6	Qs

Максимально достигнутая производительность для кластера ТГУ составила 5,015 GFlops при пиковой производительности 11,7 Gflops.

Второй тестируемый кластер установлен в Институте оптики атмосферы (http://**cluster.iao.ru**); 10 его 2-х процессорных элементов Pentium-III-1GGz объединены в сеть 1 Гбит-Ethernet. Оптимальными параметрами являются следующие:

1	# of problem size
30000	Ns
1	# of NBs
75	NBs
1	# of process grids
5	Ps
4	Qs

Максимально достигнутая производительность – 11,3 GFlops при пиковой производительности 20Gflops.

Заключение

Выполнено тестирование высокопроизводительных вычислительных ресурсов Томского государственного университета и Института оптики атмосферы СО РАН на предмет установления их реальной производительности. Тестирование проведено с помощью пакета HPL.

Получено, что максимально достигнутая производительность кластера ИОА более чем в два раза выше производительности кластера

ТГУ объясняется более высокими характеристиками вычислительных узлов (производительность процессора и объем оперативной памяти) и более высокой пропускной способностью сети кластера ИОА.

Литература

1. *Корнеев В.В*. Параллельные вычислительные системы. М.: Нолидж, 1999.
2. *Ортега Дж.* Введение в параллельные и векторные методы решения линейных систем. М.: Мир, 1991.
3. Современный компьютер / Сб. научно-популярных статей. Пер. с англ. под ред. В.М. Курочкина. М.: Мир, 1986.
4. www.netlib.org.
5. www.parallel.ru

КОМПЛЕКС МОНИТОРИНГА РАСПРЕДЕЛЕННЫХ ИНФОРМАЦИОННО-ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

С.А. Жуматий, А.А. Кальянов

НИВЦ МГУ им. М.В. Ломоносова

В эпоху Internet компьютеры часто работают без постоянного контроля человеком. Например, web- и файл-серверы работают автономно на протяжении месяцев и о возникших на них неполадках можно узнать только по репликам разгневанных пользователей. Подобные проблемы могут возникать и с другой вычислительной техникой, а так как не всегда есть возможность постоянно проверять ее работоспособность, то возникает проблема оперативного обнаружения проблем в сетях ЭВМ. Очевидно, что подобные проблемы должны ликвидироваться в самые короткие сроки, поэтому задача весьма актуальна. В настоящее время в НИВЦ МГУ реализуется проект по обеспечению мониторинга в вычислительных и информационных сетях, которому и посвящена данная работа.

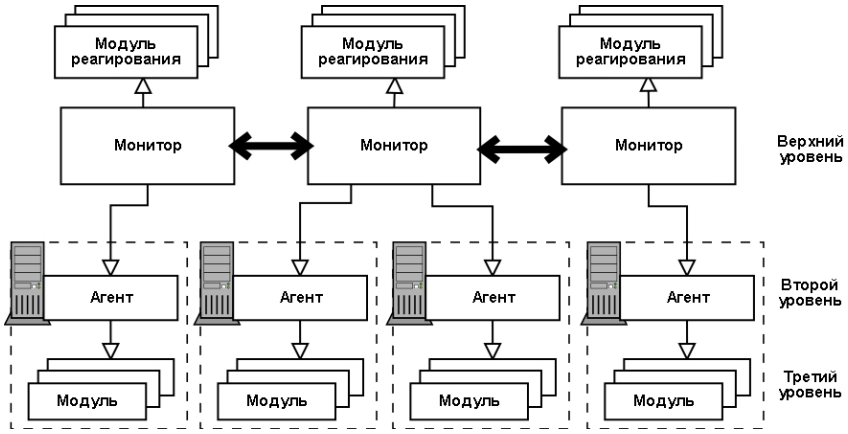
Представляемый комплекс предназначен для мониторинга информационных и вычислительных ресурсов. Мониторинг означает отслеживание каких-либо параметров компьютеров и сервисов в сети, например доступности web-сервера. Главными задачами при создании комплекса ставились:

- оперативность обнаружения неполадок, отказов;

– гибкие возможности оповещения;
– возможность предприятия попытки автоисправления неполадок;
– надежность работы комплекса;
– возможность настройки на новые (нестандартные) параметры.

Для решения этих задач была разработана архитектура, описанная ниже.

Комплекс мониторинга представляет собой распределенную сетевую систему. Общая схема представлена на рисунке. Основу комплекса составляют мониторы (верхний уровень), которые находятся в постоянном взаимодействии друг с другом. Эти мониторы общаются с агентами (второй уровень), работающими на контролируемых ЭВМ. Наличие в комплексе нескольких мониторов обеспечивает повышенную надежность работы комплекса. В случае выхода из строя одного из мониторов или его недоступности, остальные мониторы попытаются взять на себя его функции, проведя автоматическую переконфигурацию комплекса и перераспределив обязанности по мониторингу. Все взаимодействие происходит по протоколу TCP/IP.



Агенты на контролируемых ЭВМ отслеживают собственно значения заданных параметров. Для этого они вызывают модули (третий уровень). В данный момент в комплекс включены модули для отслеживания доступности компьютера по сети, работоспособности web-сервера, работоспособности файл-сервера (nfs), уровня заряда батареи

UPS, напряжения питания UPS, температуры UPS (последние 3 параметра отслеживаются только при наличии работающей системы nut, содержащей драйвера для работы с UPS). Кроме того, разработаны модули для контроля сервисов вычислительных кластеров (работоспособность системы очередей, загрузка процессоров). Разрабатываются модули для отслеживания температуры процессора и скорости вращения кулера.

Все значения параметров мониторинга, как отмечалось выше, получаются в результате работы модулей, которые запускаются агентами. За получение одного или нескольких параметров отвечает один модуль. Интерфейс между агентом и модулем фиксирован, что означает возможность написания своих специфичных модулей для отслеживания нестандартных параметров. Модуль вызывается с любыми настраиваемыми администратором аргументами, а на стандартный вывод каждый модуль должен вывести текстовый блок, в котором содержится информация о значении параметров в предопределенном формате. Таким образом, можно подключать любое количество модулей с заданным интерфейсом и отслеживать значения любых параметров.

В случае, если комплекс обнаружил неполадку (иначе говоря, значение какого-либо из параметров вышло за допустимые пределы), то включается механизм реагирования. Это может быть как оповещение по e-mail или через sms, так и запуск какой-либо программы для оперативного реагирования (например перезапуск web-сервера).

При возникновении сбоев, связанных с неполадками сети, комплекс предпринимает попытку определить, где именно произошел сбой и информировать об этом администраторов.

Так как возможны ситуации, когда допустим кратковременный выход параметра из допустимого диапазона (например, отключение питания на 3-4 секунды при работающем UPS), то в комплексе предусмотрена возможность ожидания определенного количества сбоев подряд и только после этого включения механизма реагирования. После того, как реагирование активировано, параметр отмечается как сбойный и в дальнейшем реагирование этот сбой не происходит. Если значение параметра возвращается к нормальному (возможно также после нескольких успешных проверок подряд), то включается соответствующее реагирование и параметр вновь помечается как рабочий.

Реагирование происходит с помощью вызова монитором одного или нескольких модулей реагирования. В настоящее время в комплекс включены следующие модули реагирования:

– оповещение по e-mail;
– оповещение по sms;
– запись в log-файл;
– формирование файла текущего состояния в формате html;
– запуск программы.

Описанная архитектура комплекса продиктована требованиями надежности (дублирование головных мониторов), снижения нагрузки на сеть (разделение головных мониторов и агентов), облегчения настройки и гибкости (разделение агентов и модулей). Каждый компонент выполняет свои функции:

– мониторы производят самую сложную работу по конфигурированию и настройке системы в целом, динамическому перераспределению обязанностей, оперативному реагированию и информированию администраторов о неполадках, а также по представлению результатов в удобном для пользователя виде;
– агенты минимизируют нагрузку на сеть путем обработки вывода каждого из вызванных модулей и объединения результатов, требующихся определенному монитору, в один логический пакет;
– модули выполняют сугубо узкую задачу по получению одного или нескольких параметров и могут быть любого уровня сложности;
– модули реагирования отвечают за оповещение о сбоях и возвратах к нормальной работе, а также запуск программ для оперативного исправления или коррекции сбоев (если это возможно).

Комплекс мониторинга внедрен на трех кластерах МГУ (см. http://parallel.ru/cluster). Реализованная версия работает под управлением операционной системы Linux. Как показала практика, за счет использования представленной архитектуры комплекса удается реально повысить надежность работы и минимизировать сетевой трафик, что немаловажно как для глобальных, так и для локальных сетей.

Работа выполняется при поддержке РФФИ, грант №02-07-90442.

ПАРАЛЛЕЛЬНАЯ И РАСПРЕДЕЛЕННАЯ СИСТЕМА ИМИТАЦИИ DOOMS.NET

Е.Б. Замятина, А.Х. Фатыхов, М.Х. Фатыхов

Пермский государственный университет

Введение

Метод имитационного моделирования широко применяется в различных областях науки для анализа сложных систем и требует значительных затрат времени и вычислительных ресурсов. Для сокращения времени, которое необходимо на проведение имитационного эксперимента, ведутся работы над созданием инструментальных средств имитационного моделирования, предназначенных для выполнения имитации на нескольких процессорах многопроцессорных ВС или на нескольких компьютерах в сети (примером таких разработок может, к примеру, служить система CHIMERA-P, г. Новосибирск). На кафедре математического обеспечения ВС Пермского государственного университета разрабатывается система имитации DOOMS.Net, с помощью которой можно строить и исследовать параллельные и распределенные имитационные модели.

Формальная имитационная модель

Имитационная модель DOOMS.Net является объектно-ориентированной, то есть представляет собой совокупность объектов, каждый из которых имеет свой сценарий поведения и обменивается сообщениями с другими объектами посредством каналов. Инструментальное средство DOOMS.Net поддерживает дискретное событийно-ориентированное моделирование.

Формально имитационную модель можно представить так: $M = \{O, MO, Ch\}$, где O – множество объектов obj_i ($i = 1,2,\ldots,N$), обменивающихся сообщениями друг с другом. MO – множество менеджеров объектов mo_k ($k = 1,..,M$), а Ch – множество каналов ch_j ($j = 1,2,..L$), используя которые объект передает и получает сообщения ($Ch = In \cup Out$, In и Out – множества входных и выходных каналов). Каждый объект отображает отдельный компонент моделируемой системы и его можно представить следующим образом: $\{Bhv, Tloc, St, P\}$, где Bhv – набор событий, определяющий поведение объекта; $Tloc$ – локальные часы; St – множество состояний (объекты могут нахо-

диться в активном состоянии или в состоянии ожидания); P – набор свойств (переменные, за изменением значений которых можно наблюдать во время имитационного эксперимента). Передачей сообщений между объектами (и менеджеров объектов более низкого уровня) управляет менеджер объектов. Менеджер объектов является основным хранителем информации о связях между объектами (то есть о структуре системы). Менеджеры позволяют строить иерархические имитационные модели (см. рис.)

Над моделью определены операции: $M = M + Obj$ – добавление объекта, $M = M - Obj$ – удаление объекта, $O = O + ch$ – добавление канала связи, $O = O - ch$ – удаление канала связи. Операции над моделью дают возможность изменять ее динамически в ходе процесса имитации.

По сути дела мы имеем графовую модель $G = (V,E)$, вершинами которой является объекты и менеджеры ($V = O \cup MO$), а дугами – каналы связи Ch.

Для сбора статистической информации о поведении модели используются специальные объекты-шпионы (Spy). Объекты-шпионы

описываются отдельно от имитационной модели, они могут быть подсоединены к отслеживаемым элементам модели (свойствам объектов и их событиям) в любой момент времени в ходе имитационного эксперимента или отключены, если в наблюдении за изменением конкретного события или свойства объекта нет необходимости. Таким образом, алгоритм исследования, который выполняют объекты-шпионы, отделен от алгоритма имитации. Алгоритмы исследования и имитации также могут выполняться на различных процессорах параллельно.

Алгоритм имитации

Управляет работой объектов во времени и синхронизирует их работу алгоритм имитации, который является распределенным.

Распределенный алгоритм имитации предполагает, что поведение объектов должно быть согласовано в едином модельном времени. В системе DOOMS.Net реализован консервативный алгоритм синхронизации модельных объектов.

Каждый объект хранит свое собственное локальное модельное время $Tloc_{obj}$. Объект работает по собственному сценарию поведения до тех пор, пока он не изменяет свое локальное время при выполнении некоторого события или не пожелает принять сообщение (в этих случаях он может быть переведен в состояние ожидания). Таким образом, операция чтения сообщения из буфера и изменение локального времени являются двумя контрольными точками, в промежутках между которыми объект «живет» независимо от других объектов.

Как уже говорилось ранее, объекты связаны друг с другом с помощью каналов. У каждого объекта существуют входные ($Ch_i \in In$, где In – множество входных каналов), и выходные каналы ($Ch_j \in$ Out, где Out – множество выходных каналов), через которые он принимает и посылает сообщения.

Алгоритм синхронизации состоит в том, что локальное время объекта In не может быть изменено или сообщение не будет принято им к рассмотрению до тех пор, пока существуют объекты, связанные выходными каналами с Obj, локальное время которых меньше, чем $Tloc_{obj.}$

Программная реализация системы

Рассмотрим подробнее некоторые вопросы реализации системы. Поскольку имитационная модель является иерархической, ее удобно

хранить как xml-документ. В *xml*-документе описываются типы, объекты, менеджеры, связи, а также сам алгоритм моделирования.

Для реализации DOOMS была выбрана технология .Net. Это позволяет реализовать DOOMS не только под управлением Windows, но и под управлением других ОС (например, LINUX), если будет обеспечена поддержка .Net CLR (Common Language Runtime). .Net - является дальнейшим развитием COM, DCOM, COM+ технологии и поэтому дает возможность использовать транзакции, пул объектов, WEB сервисы, обеспечивает межъязыковую интероперабельность и достаточно высокую безопасность. При реализации интерфейса передачи сообщений использован стандарт MPI.

Каждый объект располагает обработчиками событий, собственным набором методов, набором свойств, списком каналов, локальным временем. Объекты могут наследоваться. Это позволяет создавать новые типы, используя существующий код. В системе можно создавать библиотеки типов. Во время имитационного прогона объекты могут порождать объекты нового типа, создавать новые экземпляры объектов, изменять конфигурацию объектов, выполняя операции над моделью.

В состав программных средств DOOMS.Net включен графический редактор моделей (DOOMS EDITOR). С помощью графического редактора можно:

- создавать новые и изменять существующие типы объектов;
- добавлять и удалять объекты;
- изменять поведение объектов;
- подсоединять объекты к тому или иному менеджеру;
- создавать каналы связи;
- изменять направление передачи сообщения по каналу.

Решение вопросов балансировки

Поскольку система имитации DOOMS проектировалась как параллельная (или распределенная), то среди программных средств, обеспечивающих ее функционирование, разработаны средства, позволяющие выполнить равномерную загрузку процессоров (или компьютеров).

Первоначальное распределение компонентов модели выполняется с помощью визуальной среды DOOMS EDITOR. Первоначальное распределение предполагает выделение в графовой модели сильносвязанных компонентов (по количеству связей между объектами) и выполне-

ние процедуры автоматического распределения объектов по менеджерам. Предусмотрено также и автоматическое распределение менеджеров (со своими объектами) по отдельным процессорам (или узлам сети).

После выполнения имитационного прогона выполняется сбор статистики об интенсивности обмена сообщениями между объектами. Используя эту характеристику, система DOOMS может перераспределить компоненты модели по процессорам. Для перераспределения используется генетический алгоритм.

Заключение

Система имитации DOOMS продолжает серию работ по созданию средств автоматического проектирования и использования методов имитационного моделирования, которые ранее велись на кафедре.

В настоящее время идет разработка стандартных библиотек типов объектов и стандартных объектов-шпионов для моделирования и перераспределения информационных потоков в локальных и глобальных сетях. Поскольку имитационная модель является объектно-ориентированной и динамически изменяемой (за счет реализации операций над моделью), она достаточно адекватно отображает предметную область. Реализация распределенного алгоритма делает систему эффективной, поскольку позволяет выполнять параллельное или распределенное моделирование. В настоящее время разрабатывается версия системы имитации для кластерной МВС 1000.

Литература

1. *Миков А.И.* Автоматизация синтеза микропроцессорных управляющих систем. Иркутск: Изд-во ИГУ, 1987.
2. *Замятина Е.Б.* Реализация информационных процедур и условий моделирования. В кн.: «Моделирование вычислительных систем и процессов. Пермь: ПГУ, 1989.
3. *Костенко В.А.*, *Смелянский Р.Л.*, *Трекин А.Г.* Синтез структур вычислительных систем реального времени с использованием генетических алгоритмов / Программирование, №5, 2000.

МОДЕЛИРОВАНИЕ СТАТИЧЕСКОГО РАСПРЕДЕЛЕНИЯ АБРИКОСОВСКИХ ВИХРЕЙ В СВЕРХПРОВОДЯЩЕЙ ПЛЕНКЕ*

В.В. Заскалько, И.Л. Максимов

Нижегородский государственный университет им. Н.И. Лобачевского

Одной из задач физики конденсированных систем является решение проблемы достижения высоких критических плотностей тока в сверхпроводящих материалах. Решение этой задачи позволит обеспечить оптимальные параметры технических устройств на основе высокотемпературных сверхпроводников, используемых в современной энергетике. Одним из подходов решения данной задачи является численное моделирование вихревых структур возникающих в сверхпроводнике, помешенном во внешнее магнитное поле или при пропускании транспортного тока.

Как известно магнитное поле проникает в сверхпроводник II -го рода в виде элементарных квантов потока – абрикосовских вихрей [1]. Одиночный вихрь несет квант магнитного потока Φ_0 ($\Phi_0 = \pi\hbar c/e$), который может быть направлен как в вдоль положительного направления оси z (ось параллельная внешнему магнитному полю H), так и против него. Вихри, имеющие магнитный поток, направленный против положительного направления оси z, будем далее называть антивихрями.

Рассмотрим тонкопленочную полоску сверхпроводника II-го рода толщины, ширины W и бесконечной длины. В предположении о вязком характере движения вихря в пленке (характеризующемся коэффициентом вязкости ν) уравнение движения k-го вихря имеет вид [1]:

$$\nu \frac{d\mathbf{r}_k}{dt} = \mathbf{F}_k, \qquad (1)$$

где $\mathbf{r}_k$ – радиус-вектор в плоскости пленки, определяющий положение центра k-го вихря, $\mathbf{F}_k$ – суммарная сила, действующая на этот вихрь со стороны других вихрей и внешнего магнитного поля. $k = 1, 2..., N$

* Работа выполнена при поддержке Фонда содействия развитию малых форм предприятий в научно-технической сфере

$= = N_+ + N_-$, N_+, N_- – соответственно количество вихрей и антивихрей в пленке.

Для решения (1) применена схема последовательных приближений, суть которой состоит в следующем.

В начальный момент времени вихри (антивихри) равномерно распределяются между процессорами (рис.1). Стрелками указано взаимодействие k-го вихря с его ближайшим окружением. Затем каждому процессору, на текущем итерационном шаге, поручается рассчитать смещение вихря, при этом для расчёта необходимо знать положение всех вихрей на предыдущем итерационном шаге. Для этого после обработки своей части каждый процесс отсылает свой результат всем остальным и принимает данные о положении остальных вихрей с других процессоров. Данная итерационная процедура продолжается до тех пор, пока не установится статическое состояние системы вихрей в пленке.

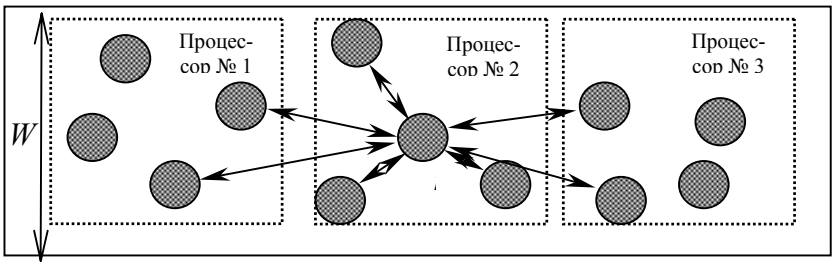


Рис. 1

Оценим вычислительную сложность задачи C. Очевидно, она квадратично пропорциональна числу вихрей: $C \sim N^2$, В случае одного процессора время, затраченное на один итерационный шаг $t_1 = aN^2$, где a константа прямо пропорциональная вычислительной трудоёмкости расчёта межвихревого взаимодействия и обратно пропорциональная скорости процессора. При параллельном расчёте время итерационного шага уменьшается обратно пропорционально числу процессоров P: $t_p = aN^2/P$, в то время как сложность коммуникационного обмена, характеризуемая временем t_c линейно растет с N: $t_c = bN$; здесь b – константа связанная с временем затрачиваемым на пересылку одного кванта информации. Простой расчет [2] показывает, что эффективность работы программы характеризуется параметром $\varepsilon = 1/(1 + bP/(aN))$. В случае $a = b$ (машины с общей памятью), эф-

фективность будет не ниже 50%. На практике для кластерных систем $b>>a$, но при этом $N \square P$ поэтому эффективность определяется пределом отношения b/N. В процессе моделирования происходит перераспределение и изменение числа вихрей на процессорах, которое компенсируется их динамической балансировкой.

В результате проведенных расчётов изучены двухмерные вихревые структуры, возникающие в сверхпроводящем образце. Наиболее устойчивой конфигурации соответствует максимальный критический ток. На рис. 2 приведен график зависимости параметра ε эффективности работы от количества вихрей в плёнке (P=2) в случае двух процессоров.

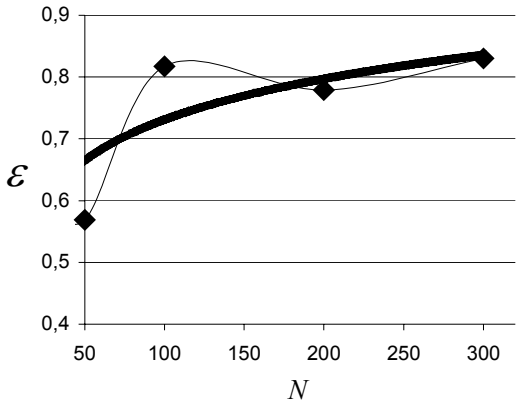


Рис. 2

Точками отмечены экспериментально полученные значения параметра ε, соединенные сглаженной кривой. Жирная линия соответствует аппроксимации точек теоретической кривой. Как видно из графика, в физически интересном случае, когда число вихрей порядка $\sim 10^2$ на образец, программа реализованная с использованием стандарта MPI [3], обеспечивает высокую эффективность работы на многопроцессорных системах. Нерегулярное поведение связано с высокой сложностью и нелинейностью алгоритма, моделирующего статические вихревые структуры.

Литература

1. *Шмидт В.В.* Введение в физику сверхпроводников М.: Наука, 1982.
2. *Гергель В.П., Стронгин Р.Г.* Основы параллельных вычислений для многопроцессорных вычислительных систем. Н.Новгород: Изд-во ННГУ, 2001.
3. *Group W., Lusk E., Skjellum A.* Using MPI. The MIT Press, Cambridge, Massachusetts, London, England.

ИСПОЛЬЗОВАНИЕ СРЕДСТВ MPI В ПРОГРАММНОЙ СИСТЕМЕ ДЛЯ МОДЕЛИРОВАНИЯ ДИНАМИКИ ДИСКРЕТНЫХ АКТИВНЫХ СРЕД*

М.И. Иванченко, О.И. Канаков, К.Г. Мишагин
Г.В. Осипов, В.Д. Шалфеев

Нижегородский государственный университет им. Н.И. Лобачевского

Введение

Сегодня задачи изучения пространственно-временной динамики нелинейных активных сред находятся в центре внимания исследователей в различных областях науки. Достаточно указать в этой связи проблемы турбулентности, эволюции биологических и химических возбудимых сред, задачи нейродинамики, социально-экономических наук, а также целый ряд приложений в радиофизике, электро- и радиотехнике и связи (распределенные энергосети, распределенные джозефсоновские контакты, сети синхронизации, фазированные антенные решетки и др.).

Решение этих задач требует моделирования коллективной динамики ансамблей (решеток), состоящих из большого числа связанных между собой активных элементов, обладающих как простой, так и сложной индивидуальной динамикой. Отметим, что пространственно-дискретные модели зачастую не являются решеточной аппроксимацией каких либо распределенных уравнений, а отражают дискретную

* Работа выполнена при поддержке Фонда содействия развитию малых форм предприятий в научно-технической сфере

природу реальных систем (таких, например, как сердечная ткань или сети синхронизации). При этом в решеточных системах могут наблюдаться эффекты, принципиально не реализуемые или структурно-неустойчивые в распределенных системах, например, дискретные бризеры (пространственно-локализованные осциллирующие решения).

Назначение и возможности пакета

Представленный программный комплекс предназначен для моделирования на параллельных вычислительных системах динамики двумерных решеточных ансамблей, в которых каждый элемент связан с четырьмя своими ближайшими соседями. Динамика базового элемента может задаваться отображением или системой обыкновенных дифференциальных уравнений (ОДУ) любого порядка в форме Коши. В уравнениях допускается явная зависимость от времени. Для интегрирования ОДУ используется алгоритм, основанный на разностной схеме Рунге-Кутты четвертого порядка. Однако, пакет допускает «подключение» других алгоритмов счета. Для этого каждый временной шаг метода должен быть разделен на некоторое количество этапов, выполнение каждого из которых на каком-либо элементе информационно зависит от результата только непосредственно предшествующего этапа на соседних элементах. Заданные пользователем уравнения динамики базового элемента применяются единообразно ко всем узлам решетки. Граничные условия могут быть типа Неймана, Дирихле или периодическими. Допускается задание произвольного распределения на решетке любого числа параметров, что позволяет моделировать пространственно-неоднородные системы. Начальные условия также могут задаваться произвольным образом.

Пакет содержит средства регистрации минимальных и максимальных значений переменных состояния элементов решетки за время интегрирования, осциллограмм динамики отдельных элементов, позволяет делать «снимки» состояния всей решетки в фиксированные моменты времени. Имеется прототип обработчика, позволяющего вычислять значение старшего ляпуновского показателя траектории системы в процессе моделирования. Возможно создание и подключение других обработчиков.

Организация параллельных вычислений

Локальность связей в решетке позволяет эффективно использовать в программной системе методы параллельных вычислений. Переносимая реализация параллельного алгоритма в представленном пакете

основана на спецификации MPI. Решетка активных элементов разбивается на прямоугольные неперекрывающиеся блоки достаточно большого размера, каждый из которых назначается для расчета соответствующему процессу. Для организации обмена информацией между процессами создается виртуальный коммуникатор с двумерной топологией с помощью функции MPI_Cart_create.

Каждый процесс хранит информацию о состоянии элементов выделенного ему блока решетки в два последовательных момента времени (текущий и предшествующий), а также имеет четыре краевых буфера для приема информации от соседних процессов.

Все элементы блока, являющегося «зоной ответственности» данного процесса, могут быть разделены на пограничные и внутренние. Для вычисления состояния пограничных элементов в очередной момент времени необходимы значения переменных состояния пограничных элементов одного или двух соседних процессов в предыдущий момент. Соответствующая информация должна быть своевременно получена от соседних процессов и помещена в краевые буферы. Напротив, внутренние элементы не требуют для вычисления итерационного шага информации от других процессов. Это позволяет осуществлять обработку внутренних элементов параллельно с операциями обмена информацией. Поэтому для обмена используются функции неблокирующего приема и передачи Irecv и Isend.

Шаг итерационного алгоритма на каждом из процессов осуществляется по следующей схеме. Допустим, что после завершения предыдущего шага известно состояние всех элементов, за которые отвечает данный процесс, в момент времени i. Кроме того, приняты от соседних процессов и помещены в краевые буферы переменные состояния их пограничных элементов, относящиеся к тому же моменту времени. Сначала выполняется итерация пограничных элементов, в результате чего становится известным их состояние в i+1-й момент времени. После этого инициируются неблокирующие операции передачи только что вычисленных значений соседним процессам, а также приема в краевые буферы аналогичной информации от соседей. Параллельно осуществляется расчет новых состояний внутренних элементов. По завершении обработки внутренних элементов и процессов обмена с соседями (ожидание завершения приема/передачи обеспечивается функцией MPI_Wait_all) процессу известны переменные состояния всех «своих» элементов в i+1-й момент времени, а в буферах находятся переменные состояния пограничных элементов соседей в тот же

момент времени. Таким образом, процесс готов к выполнению следующего итерационного шага по описанной схеме.

Итерационный шаг метода Рунге-Кутты четвертого порядка реализован в четыре этапа. Каждый из них использует данные о результатах предыдущего этапа, получаемые от соседних элементов.

Примеры применения пакета

В качестве тестовых задач, иллюстрирующих возможности пакета, были рассмотрены следующие две.

1. Исследование динамики решетки элементов первого порядка маятникового типа с взаимной нелинейной связью при наличии точечного воздействия. Модель задается системой ОДУ

$$\dot{\varphi}_{i,j} + \sin\varphi_{i,j} = \gamma_{i,j} - \delta(\sin(\varphi_{i,j} - \varphi_{i+1,j}) + \sin(\varphi_{i,j} - \varphi_{i-1,j}) + \\ + \sin(\varphi_{i,j} - \varphi_{i,j+1}) + \sin(\varphi_{i,j} - \varphi_{i,j-1})), \quad i = 1...M,\ j = 1...N, \tag{1}$$

где φ_{ij} – переменные состояния (фазы), $\gamma_{ij} > 0$ и $\delta > 0$ – параметры; граничные условия – неймановские, $M = N = 20$. На всех элементах, кроме центрального, $\gamma_{ij} = \gamma_1 < 1$, на центральном $\gamma_{ij} = \gamma_2 > 1$. Такая система может моделировать, например, фазированную антенную решетку с распределенными генераторами, в которой один из элементов вышел из строя, либо энергосеть, в которой неисправен один из электрогенераторов (явление «перекоса фаз»).

Были обнаружены следующие варианты динамических режимов в системе:

- синхронный режим: система приходит в состояние равновесия; элементы расстроены по фазе, но значения фазовых расстроек неизменны во времени. Неустойчивость в центральном элементе оказывается полностью подавленной. Ляпуновский показатель отрицателен;
- колебательный режим: в системе имеется устойчивый вращательный предельный цикл. Связь между элементами не может обеспечить режима глобальной синхронизации и возмущенный элемент совершает вращательное движение (отсутствует синхронизация по частоте); фазы остальных элементов колеблются около некоторых фиксированных значений значений (дрейф фазы отсутствует), причем амплитуда колебаний резко спадает при удалении от центрального элемента. Ляпуновский показатель равен нулю;
- вращательный режим: в фазовом пространстве системы су-

ществует устойчивый инвариантный тор. Все элементы совершают вращательное движение. Ляпуновский показатель также равен нулю.

2. Демонстрация существования бризерного решения в консервативной решеточной системе.

В форме Коши уравнения движения системы имеют вид

$$\begin{aligned} \dot{x}_i &= y_i, \\ \dot{y}_i &= -f(x_i) + a(x_{i-1} - 2x_i + x_{i+1}), \quad i = 1, \ldots, N, \end{aligned} \tag{2}$$

где $f(x) = x - x^2 + x^3/4$, $a = 0{,}1$, $N = 3000$.

Известно [1], что такая система имеет пространственно-локализованное решение, реализующееся при начальных условиях $x_{1501}(t = 0) = 2{,}3456$, $x_{i\neq 1501}(t = 0) = y_i(t = 0) = 0$. Этот результат был повторен с помощью представленного пакета. На рис. 1 (t = 0–30), 2 (t = 230–260), 3 (t = 260–290) показаны минимальные и максимальные значения переменных x_i элементов решетки за три различных интервала времени.

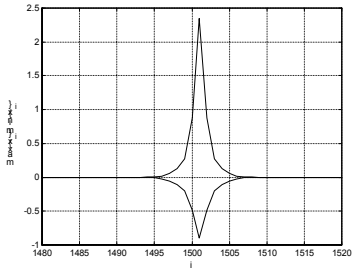


Рис. 1

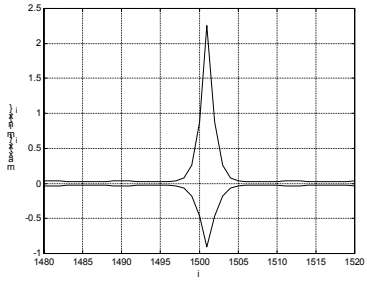


Рис. 2

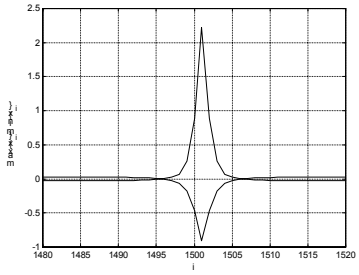


Рис. 3

Как видно, лишь небольшая часть энергии начального возмущения излучается в решетку в течение некоторого начального промежутка времени; в дальнейшем локализованное решение с хорошей точностью сохраняет свою амплитуду.

Заключение

Подводя итог, отметим, что представленный программный продукт, используя преимущества параллельного программирования и будучи переносимым благодаря использованию MPI, является весьма эффективным средством для изучения процессов, происходящих в дискретных активных средах, что на настоящий момент является одной из самых актуальных задач нелинейной динамики, нелинейной физики.

Литература

1. *Flach S.*, *Willis C.R.* Discrete Breathers // Physics Reports. 1998. V. 295, №5.

МЕТОДЫ АНАЛИЗА И ПРЕДСКАЗАНИЯ ЭФФЕКТИВНОСТИ DVM-ПРОГРАММ

В.Н. Ильяков, Н.В. Ковалева, В.А. Крюков

Институт прикладной математики им. М.В. Келдыша, РАН

Эффективность выполнения параллельных программ на многопроцессорных ЭВМ с распределенной памятью определяется следующими основными факторами:

- степенью распараллеливания программы – долей параллельных вычислений в общем объеме вычислений;
- равномерностью загрузки процессоров во время выполнения параллельных вычислений;
- временем, необходимым для выполнения межпроцессорных обменов;
- степенью совмещения межпроцессорных обменов с вычислениями.

Методы и средства отладки производительности параллельной программы существенно зависят от той модели, в рамках которой разрабатывается параллельная программа.

Существенным достоинством DVM-модели является то, что в любой момент выполнения программы на любом процессоре всегда известно, какой участок программы выполняется - последовательный или параллельный, а также известны все точки программы, в которых выполняются операции, требующие синхронизации процессоров. Поэтому имеется возможность количественно оценить влияние на эффективность выполнения программы каждого из четырех перечисленных выше факторов. Кроме того, ускорение параллельного выполнения программы (speedup) можно вычислить, даже не выполняя запуска на одном процессоре, что часто затруднительно сделать из-за длительности такого выполнения, а иногда и невозможно сделать из-за нехватки оперативной памяти.

Для анализа и отладки эффективности выполнения DVM-программ созданы инструментальные средства, функционирующие следующим образом. *Система поддержки выполнения DVM-программ* во время выполнения программы на многопроцессорной ЭВМ (или сети ЭВМ) накапливает информацию с временными характеристиками в оперативной памяти процессоров, а при завершении выполнения программы записывает эту информацию в файл, который затем обрабатывается на рабочих станциях в среде Windows или UNIX специальным инструментом - *визуализатором производительности*.

С помощью визуализатора производительности пользователь имеет возможность получить временные характеристики выполнения его программы с различной степенью подробности.

Основной проблемой, с которой сталкиваются пользователи при отладке эффективности выполнения своих программ на многопроцессорных ЭВМ, является нестабильность их выполнения, и, как следствие, нестабильность временных характеристик. В докладе анализируются причины такой нестабильности и методы борьбы с нею.

Особенностью DVM-подхода является то, что основная работа по реализации модели выполнения параллельной программы (например, распределение данных и вычислений) осуществляется динамически системой поддержки выполнения DVM-программ. Это позволяет на основании информации о выполнении DVM-программы на однопроцессорной ЭВМ посредством моделирования работы системы поддержки предсказать характеристики выполнения этой программы на многопроцессорной ЭВМ с заданными параметрами (производительностью процессоров и коммуникационных каналов).

В состав DVM-системы входит специальный инструмент – *пре-*

диктор, который предназначен для анализа и отладки производительности DVM-программ без использования реальной многопроцессорной машины (доступ к которой обычно ограничен или сложен).

Предиктор представляет собой систему обработки трассировочной информации, собранной системой поддержки выполнения DVM-программ во время прогона программы на инструментальной ЭВМ. По трассировке и заданным пользователем параметрам целевой ЭВМ, он вычисляет временные характеристики выполнения на ней данной программы. При этом предиктор моделирует параллельное выполнение DVM-программ и является, фактически, моделью многопроцессорной ЭВМ, системы поддержки выполнения DVM-программ и используемой ею библиотеки передачи сообщений (MPI). С помощью предиктора пользователь имеет возможность получить оценки временных характеристик выполнения его программы на многопроцессорной ЭВМ с различной степенью подробности.

В докладе анализируются проблемы, возникающие при моделировании параллельного выполнения программ, и рассматриваются пути их преодоления.

МОНИТОРИНГ ВЫПОЛНЕНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ НА КЛАСТЕРЕ PARC

Г.Г. Исламов, С.А. Мельчуков, М.А. Клочков
О.В. Бабич, Д.А. Сивков

Удмуртский государственный университет, г. Ижевск

Введение

Мы проводим исследования по проекту «Создание научно-методического обеспечения подготовки специалистов в области высокопроизводительных кластерных технологий» в рамках ФЦП «Интеграция науки и высшего образования России на 2002-2006 годы».

В учебном пособии [1] описаны средства написания параллельных программ на языке Си в операционной среде Linux с использованием возможностей интерфейса обмена сообщениями MPI и системы PVM. Здесь представлены средства визуализации, мониторинга выполнения и отладки параллельных программ на кластере PARC высокопроизводительных компьютеров Удмуртского госуниверситета и демонстри-

руется применение этих средств при анализе учебных алгоритмов монографии [2], а также алгоритмов Дейкстры и Флойда решения задачи о кратчайшем расстоянии в ориентированном графе.

Из всего многообразия имеющихся программных средств мы выделяем AIMS (The Automated Instrumentation and Monitoring System) [3], XMPI [4], XPVM [5] и TotalView [6]. Удобный графический интерфейс этих средств позволяет проводить тщательный анализ и мониторинг параллельных процессов с целью улучшения их производительности, способствует пониманию их поведения и взаимодействия, а также позволяет оценивать степень загрузки процессоров при выполнении параллельной программы. Несмотря на то, что в первых трех средствах отсутствуют возможности отладки параллельных программ в ходе их исполнения, использование AIMS, XPVM и XMPI дает значительный выигрыш при обучении методам разработки параллельных программ.

AIMS

Средство мониторинга AIMS создано NASA Ames Research Center в рамках High Performance Computing and Communication Program. Оно позволяет измерять и анализировать производительность программ, написанных на Фортране и Си с использованием двух библиотек обмена сообщениями: MPI и PVM и исполняемых на IBM SP2, Sun Sparc, SGI, SGI/Power Challenge, HP, SGI ORIGIN/2000 и Linux. Соответствующие версии разрабатываются для CRAY T3E и Macintosh/MachTen.

AIMS состоит из трех основных компонент: обработчика исходного кода, который автоматически до процесса компиляции и компоновки вставляет в исходный текст программы вызовы подпрограмм мониторинга, библиотеки, позволяющей собирать данные о ходе выполнения программы (временные затраты на выполнение процедур, передачу сообщений и синхронизацию) и соответствующего инструментария для отображения этих данных.

В AIMS применяется новая технология предварительной обработки исходного кода, мониторинга и визуализации процесса выполнения параллельной программы. Имеется возможность графической анимации файла трассировки полученного после выполнения обработанной программы, что позволяет пользователю устанавливать взаимосвязь отдельных событий с фрагментами исходного кода и многопроцессорной системой. Статистические данные, полученные в системе, могут быть переданы в Microsoft Excel или AVS для дальнейшей обработки.

Процесс работы с программой AIMS состоит из трёх этапов: обработка с помощью модуля xinstrument исходного MPI или PVM кода, компиляция и выполнение обработанной программы, мониторинга и анализа результатов работы с применением различных дополнительных средств AIMS.

Во время первого этапа, после обработки программы, пользователь должен скомпоновать полученный код с библиотекой монитора AIMS для генерации исполняемого приложения.

На втором этапе обработчики событий фиксируют результаты в буфере памяти соответствующего узла многопроцессорной системы, содержимое которого записывается на диск в результирующий файл трассировки. На данном этапе происходит непосредственный сбор информации о выполнении программы.

На третьем этапе, обрабатывая файлы трассировки модулем AIMS View Kernel, можно получить следующую информацию об активных процедурах, об обращениях к диску, о сообщениях, рассылаемых между узлами многопроцессорной системы, о маршрутах сообщений в сети рабочих станций.

Другой модуль AIMS tally генерирует итоговую статистику использования ресурсов и информацию о поведении программы.

Для отображения конфигурации сети процессоров имеется средство sysconfig.

AIMS рассматривает параллельную программу как конструкцию логических элементов. Наиболее важным при работе с системой AIMS оказывается собрать информацию о следующих элементах программы: приема и посылка сообщений и синхронизация процессов. Кроме того, полезно обработать циклы и ветвящиеся конструкции.

XPVM

XPVM – графическая консоль и среда управления PVM. Она предоставляет графический интерфейс для команд PVM-консоли и информацию о ходе решения задачи с одновременным отображением анимированных диаграмм. Эти диаграммы показывают взаимодействие процессов параллельной PVM-программы.

Для анализа параллельной программы в XPVM, пользователю достаточно откомпилировать программу, подключив PVM-библиотеки, которые используются для получения отладочного вывода во время выполнения. Любая задача, запущенная из XPVM, вернет информацию для анализа о последовательности событий в реальном

времени, либо в пошаговом режиме. Информация будет извлечена из сохраненных при выполнении программы файлов отладочного вывода.

Рабочие узлы могут быть присоединены к PVM или отсоединены с помощью меню. XPVM отображает активность рабочих машин и загрузку сети, очередь сообщений процессов, количество простаивающих задач. Имеется возможность просматривать историю событий и вывод родительской задачи. Безусловно, визуализация взаимодействия задач PVM помогает при отладке программ. Особенно полезным является отображение выполнения процессов с указанием используемого узла и передач сообщений между процессами. Такая визуализация выполнения позволяет легко и быстро определить, когда и какие процессы получали и передавали сообщения, таким образом, прослеживая ход выполнения программы в целом. Оптимальность использования ресурсов вычислительного комплекса позволяет определить диаграмма соотношения количества простаивающих процессов и процессов, осуществляющих эффективный счет. На этой диаграмме данное соотношение указывается в каждый момент времени выполнения программы.

Однако, XPVM, не лишена и недостатков. В отличие от TotalView, XPVM не является средой отладки, что выражается, прежде всего, в том, что в ней не поддерживается отладочный вывод и средства просмотра состояния переменных анализируемой программы.

XMPI

XMPI – система, обеспечивающая графический интерфейс между пользователем и средой параллельных вычислений. Выполнение параллельных задач на кластерных системах чаще всего происходит в фоновом режиме без вывода какой-либо промежуточной информации. Иными словами, при выполнении большого проекта пользователь не в состоянии оперативно взаимодействовать со своей задачей.

Однако, во время отладки программы и в учебном процессе некоторый инструментарий взаимодействия и мониторинга необходим. Необходимы также средства анализа производительности полученной параллельной программы.

XMPI может использоваться для запуска MPI-программ на реализации LAM, мониторинга их процессов и сообщений, а также для создания и анализа отладочных файлов трассировки. Эта система может использоваться при отладке программы для просмотра передаваемых сообщений в конкретный момент времени.

XMPI предназначен, в первую очередь, для оценки эффективности параллельных программ. Такие средства как трассировка, диаграмма использования процессов и матрица сообщений позволяют приблизительно оценить соотношение времени, затраченного на эффективный счет и времени, затраченного на служебные операции. XMPI позволяет удобно запускать MPI-программы на гетерогенных кластерах, предоставляет графические средства построения схемы загрузки узлов параллельной машины.

В Удмуртском госуниверситете XMPI широко используется на кластере PARC как сотрудниками, так и студентами, отлаживающими учебные программы.

TotalView

TotalView – оригинальный продукт фирмы Etnus LLC (см. сайт www.etnus.com), предназначенный для отладки, анализа и улучшения производительности сложных последовательных, параллельных и многопоточных приложений. Может быть успешно применен в среде Unix/Linux на кластерных, однопроцессорных и массивно-параллельных компьютерных системах.

Он поддерживает наиболее популярные модели/библиотеки параллельного программирования: Threads, OpenMP, PVM, MPI, HPF и др., имеет богатый и удобный в обращении графический интерфейс для X-Windows. А также позволяет работать с командной строки. Для каждого процесса/потока имеется возможность просматривать исходный и ассемблерный код, точки вызова функций, стек вызванных процедур, стек регистров и переменных, значения переменных и массивов программы, очереди сообщений MPI и т.д. При этом в процессе отладки можно легко модифицировать значения адресов, массивов и переменных. Данная система отладки содержит мощную гипертекстовую справочную систему. TotalView версии 5.0 поддерживает операционные системы Linux RedHat, AIX, IRIX, Solaris и др. соответственно на платформах Compaq Alpha и Intel x86, IBM SP, SGI MIPS, Sun SPARC, понимает программы на языках C, C++, Fortran 77/90, Assembler.

Подобно многим Unix-отладчикам TotalView нуждается в отладочной информации. Поэтому компиляцию отлаживаемой программы следует проводить с опцией «–g» (и без опции оптимизации). В противном случае отладка будет происходить не на уровне исходного, а на уровне ассемблерного кода. Имеется возможность проводить отладку программы на удаленном узле кластерной системы и под систе-

мой PBS управления заданиями в распределенной среде. Для работы с графической системой отладки TotalView необходимо иметь трехкнопочную мышь или ее эмулятор для обычной мыши.

Для управления выполнением отлаживаемой программы можно использовать точки останова процессов/потоков, которые могут быть установлены как в исходный, так и в ассемблерный код. Кроме этого существуют возможность устанавливать точки останова, распределенные на все процессы параллельной программы, например, установление барьера для процессов и т.п. Сложные приложения могут содержать большое число ресурсных файлов и переменных, для доступа к которым имеется удобный интерфейс. Команды управления отладкой могут быть применены на уровне группы процессов, отдельного процесса или группы потоков. Имеется возможность получения детальной информации о процессах или потоках процессов, подпрограммах, указателях, переменных, массивов, адресов, элементов массивов структур, строки исходного кода, которая при обычной отладке для удобства не отображается.

Универсальный отладчик TotalView позволяет увидеть значения локальных и глобальных переменных, регистров, областей памяти, машинные команды, менять форматы представления данных для числовых значений переменных.

TotalView может быть использован для анализа аварийного файла core, возникающего после сбоя выполнения программы. По умолчанию отладчик не передает аргументы в отлаживаемую программу. Требуется опция «-a» после имени программы, вызываемой вместе с TotalView с командной строки.

TotalView представляет собой мощный и универсальный отладчик параллельных программ. В то же время, TotalView достаточно сложен, и изучение его в деталях может потребовать значительного времени. Необходимо отметить, также, что в отличие от других рассмотренных систем он является коммерческим продуктом.

Литература

1. PARC – кластер высокопроизводительных компьютеров / Исламов Г.Г., Мельчуков С.А., Клочков М.А., Бабич О.В., Сивков Д.А. Учебно-методическое пособие. Ижевск: Изд-во УдГУ, 2001.
2. *Корнеев В.Д.* Параллельное программирование в MPI. Новосибирск: Изд-тво СО РАН, 2000.
3.??AIMS. http://science.nas.nasa.gov/Software/AIMS.

4. XMPI. http://www.lam-mpi.org, http://www.osc.edu/lam.html, http://www.mpi.nd.edu/lam/
5. XPVM. http://www.netlib.org/pvm3/xpvm/index.html, http://www. netlib.org/ukt/icl/xpvm/xpvm.html.
6. Totalview. http://www.etnus.com/Products/TotalView/index.html, http://www.llnl.gov/computing/tutorials/workshops/workshop/totalview/main.html.

СИСТЕМА ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ НА ТИПОВЫХ АЛГОРИТМИЧЕСКИХ СТРУКТУРАХ

Т.Е. Истомин

МГУ им. М.В. Ломоносова

Введение

Работа посвящена созданию интегрированной среды разработки параллельных приложений. Используемый подход позволяет собирать крупные параллельные программы из небольших готовых блоков. Этот подход существует уже много лет и показал свою применимость и эффективность. Данная работа, кроме реализации инструментального средства, продолжает развитие подхода, внося в него ряд новшеств.

Цели

Система представляет собой инструментальное средство поддержки высокоуровнего параллельного программирования, основанного на повторном использовании проектных решений и шаблонов алгоритмов (типовых алгоритмических структур).

Основными целями этой системы являются:

- достижение простоты проектирования и наглядности представления программ;
- предоставление широких возможностей повторного использования наработок;
- избавление от некоторых рутинных этапов процесса создания параллельной программы.

Описание подхода

Типовая алгоритмическая структура (ТАС) – это параметризуемый параллельный алгоритм, шаблон, фиксирующий схему решения некоторого класса задач. Программист использует эту схему как готовое

проектное решение при анализе своей задачи, а также использует скелет алгоритма, который предлагается разработчиком данной ТАС, наполняет его функциональностью и настраивает под свои нужды. ТАС с точки зрения пользователя представляет собой «ящик» со входом, выходом и набором параметров. Различаются структурные P_s и функциональные P_f параметры. Структурные параметры уточняют топологию алгоритма (например, размерности величин), позволяют масштабировать структуру. Функциональные параметры наполняют ТАС вычислительными операциями, необходимыми для решения поставленной задачи. ТАС с подставленными параметрами будем называть *экземпляром* типовой алгоритмической структуры, или *обобщенной функцией*.

Всем ТАС (кроме *Seq*) в качестве функциональных параметров разрешается передавать обобщенные функции. Поэтому программа на ТАС представляет собой иерархическую структуру и изображается в виде дерева, в вершинах которого находятся экземпляры типовых алгоритмических структур, а ребра – это связи типа «является функциональным параметром». В листьях дерева программы всегда находятся последовательные процедуры, представляемые ТАС *Seq*.

Приведем примеры типовых алгоритмических структур.

Map – применить ко всем.

Данной структурой реализуется схема независимого по данным параллелизма. К каждому элементу множества применяется указанная в параметрах функция, не зависящая от остальных элементов.

Reduce – редукция.

Эта ТАС представляет собой шаблон редукции, то есть операции $X \to y$, где X – множество однотипных величин, а y – скалярная величина. Суммирование элементов массива, поиск максимального элемента – примеры операций, которые могут быть реализованы по схеме редукции.

Farm – «плантация».

Типовая алгоритмическая структура *Farm* используется для создания нескольких параллельно работающих копий некоторого вычислительного элемента. Менеджер («фермер»), принимая из входного потока порцию данных, отправляет ее на обработку свободному рабочему процессу. После обработки каждый из рабочих процессов отправляет результат менеджеру или сразу в выходной поток. Существует два варианта этой структуры – с сохранением и без сохранения порядка выдачи результатов.

Pipe – конвейер.

Реализует композицию обобщенных функций по принципу конвейера. f_1 o f_2 o ... o f_N.

Comp – последовательная композиция.

Реализует последовательную композицию обобщенных функций f_1 o f_2 o ... o f_N.

Типовые алгоритмические структуры могут параметризовываться типами данных, способами декомпозиции/сборки данных, способами рассылки данных, типом виртуальной топологии, количеством процессоров.

Устройство системы

Система состоит из двух частей: библиотеки компонентов и конструктора параллельного алгоритма.

Библиотечная часть содержит все сущности, которыми может манипулировать пользователь в системе. Каждая из них представлена в виде компонента, состоящего из трех частей:

- класс на целевом языке – готовая к использованию реализация одной из абстракций;
- метаинформация в формате XML – необходимые конструктору знания о компоненте;
- и, возможно, плагин к конструктору.

Сущности могут быть следующих видов:

- типовая алгоритмическая структура;
- протокол коммуникаций (broadcast, scatter,...);
- структура данных (массив, изображение,...);
- операция над данными (например декомпозиция).

Конструктор предназначен для параметризации сущностей (создания экземпляров), связывания их между собой с проверкой корректности. После того как программа собрана в конструкторе, генерируется код с использованием классов из библиотеки системы.

Особенности

- компонентно-ориентированный стиль программирования;
- графическое представление программы;
- возможность использования как стандартных компонентов, так и созданных самостоятельно;
- возможность смены целевых языка и библиотеки коммуникаций;

– гибкость – настраиваемость и масштабируемость встроенных компонентов;

Конструктор реализуется на языке Java, библиотека классов – на C++ и MPI. Планируется обеспечить возможность использования системы через Web.

Работа выполняется при финансовой поддержке РФФИ, грант №01-07-90056.

Литература

1. *Берзигияров П.К.* Программирование на типовых алгоритмических структурах с массивным параллелизмом. Вычислительные методы и программирование. 2001. Т. 2. 1. С. 96–112, http://num-meth.srcc.msu.su/zhurnal/tom_2001/pdf/art2_1.pdf.
2. *MacDonald S.* From Patterns to Frameworks to Parallel Programs, Ph.D. thesis, University of Alberta, Edmonton, Alberta, Canada, 2002.
3. *Goswami D.* Parallel Architectural Skeletons: Re-Usable Building Blocks for Parallel Applications, Ph.D. thesis, University of Waterloo, Waterloo, Ontario, Canada, 2001.
4. *Zavanella A.* Skel-BSP: Performance Portability for Parallel Programming.
5. *Skillicorn D.*, *Daneluto M.*, *Pelagatti S.*, *Zavanella A.* Optimizing Data-Parallel Programs Using the BSP Cost Model.
6. *MacDonald S.* Parallel Object-Oriented Pattern Catalogue.
7. *Coudarcher R., Serot J.*, *Derutin J.* Implementation of a Skeleton-Based Parallel Programming Environment Supporting Arbitrary Nesting.

АДАПТАЦИЯ СХЕМЫ ГОДУНОВСКОГО ТИПА ДЛЯ КОМПЬЮТЕРОВ С МНОГОПРОЦЕССОРНОЙ АРХИТЕКТУРОЙ

П.В. Кайгородов, О.А. Кузнецов

Институт астрономии РАН, Москва

Введение

В 2001-м году в Институте Астрономии РАН был построен вычислительный кластер, с пиковой производительностью 28,8 GFlops.

Данный кластер состоит из 18 узлов, каждый из которых содержит процессор Pentium-4 1600 MHz, 512 MB памяти. Для осуществления газодинамических расчетов на данном кластере нами были проведены работы по адаптации схемы годуновского типа на многопроцессорный компьютер.

В докладе представлена методика адаптации схемы Роу–Ошера на кластере ИНАСАН. Освещены проблемы возникающие при расчетах с большим числом процессоров, а так же различные подходы к оптимизации кода, позволяющие повысить эффективность расчетов.

В докладе так же представлены результаты тестирования производительности трехмерного вычислительного кода, использующего схему Роу–Ошера для моделирования газодинамики в двойных звездных системах. Приведено сравнение производительности при различных типах разбиения вычислительной сетки, при различных количествах задействованных процессоров.

Была протестирована производительность данного кода на вычислительном кластере МСЦ МВС1000М, при этом число процессоров достигало 300. Тесты показали практичеки линейный рост производительности с увеличением числа процессоров.

ТРАНСЛЯЦИЯ ПРОГРАММНОГО КОДА В ГЕТЕРОГЕННОЙ СИСТЕМЕ

Н.Г. Ковтун, Л.К. Бабенко, А.Г. Чефранов
ТРТУ, УНЦ СИБ, г. Таганрог
А.Ю. Коробко
НИИ ПМА КБНЦ РАН, г. Нальчик

Введение

Постоянно увеличивающаяся сложность математических расчетов предъявляет постоянно растущие требования к производительности вычислительных систем. В настоящее время считается, что основное повышение производительности может быть достигнуто не за счет повышения производительности отдельных процессоров, а за счет создания сложных параллельных систем обработки информации. Однако повышение сложности архитектур вычислительных систем неизменно влечет за собой усложнение систем программирования.

Наибольшее распространение приобретают сейчас кластерные системы. Такая архитектура предполагает наличие в системе большого

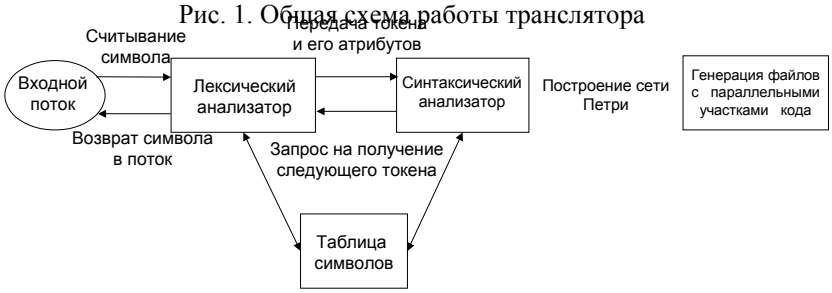
количества процессоров, работающих параллельно и асинхронно. Различные части решаемой задачи загружаются на различные процессоры в виде отдельных процессов с последующим объединением полученных результатов. При таком подходе часто возникает проблема выделения в задаче частей, которые могут функционировать параллельно. Частично решение этой проблемы может возлагаться на операционную систему, под управлением которой работает кластер, однако эффективность такого подхода невысока.

Гораздо более эффективным является выявление параллельных участков кода на этапе создания программы. Частично это можно сделать на этапе программирования. Однако прямое написание программ для параллельных систем очень трудоемко. Для облегчения задачи программиста часть задачи распараллеливания можно перенести на этап трансляции программы. В этом случае транслятор на основе анализа зависимостей по данным в программе формирует указания операционной системе о возможности параллельного запуска отдельных ветвей программы.

В данной работе рассматриваются вопросы построения подобного транслятора для гетерогенного кластера. Он осуществляет анализ программы с применением математического аппарата сетей Петри и выделяет в ней независимые по данным участки. Такие ветви программы могут выполняться параллельно, поскольку взаимодействие между ними осуществляется только на уровне входных и выходных данных. Разрабатываемый транслятор имеет модульную структуру и поэтому может быть достаточно легко перенастроен на использование другого языка программирования.

Степень оптимизации кода у различных трансляторов значительно отличается. Однако существуют простые приемы оптимизации, которые существенно уменьшают время работы целевой программы, не сильно замедляя работу транслятора.

Рис. 1. Общая схема работы транслятора



Зависимости

Чтобы выяснить, какие участи кода могут выполняться параллельно, следует уделить внимание типам зависимостей в исходной программе.

Зависимость – отношение между двумя вычислениями, которое накладывает ограничения на порядок их следования. Анализ зависимости идентифицирует эти ограничения, которые используются, когда нужно определить, может ли конкретное преобразование применяться без изменения семантики вычислений.

Существует два типа зависимостей: *зависимость по управлению* и *зависимость по данным.* Между выражением S_1 и выражением S_2, существует зависимость по управлению (обозначается $S_1 \xrightarrow{c} S_2$), если выражение S_1 определяет, будет ли выполнено S_2. Например:

```
if (a = 3) then
        b = 10
end if
```

Между двумя выражениями существует зависимость по *данным*, если они не могут быть выполнены одновременно из-за конфликта использования одной и той же переменной. Зависимости по данным делятся на три типа: *зависимость по потоку* (также называемый *истинной зависимостью), антизависимость* и *зависимость выходных данных* S_4 истинно зависит от S_3 (обозначается $S_3 \rightarrow S_4$), если сначала должно быть выполнено S_3, так как оно записывает значение, которое затем читается S_4. Например:

```
a = c*10
d = 2*a + c.
```

S_6 имеет антизависимость от S_5 (обозначается $S_5 \dashrightarrow S_6$), если S_6 записывает переменную, которая читается S_5:

```
e = f*4 +g
g = 2* h
```

Антизависимость не ограничивает выполнение так сильно, как зависимость по потоку. Код выполнится правильно, если выполнение S_6 задержано, до окончания S_5. Альтернативное решение состоит в том, чтобы использовать два участка памяти g_5 и g_6 для хранения значений, читаемых в S_5 и записываемых в S_6, соответственно. Если S_6 раньше заканчивает запись, старое значение будет все еще доступно в g_5.

Зависимость выходных данных выполняется, когда оба выражения

записывают одну переменную:

```
a = b*c
a = d+e
```

Это обозначается, как $S_7 \dashrightarrow S_8$. Так же, как и при антизависимости, дублирование хранения данных позволяет выражениям выполняться одновременно. В этом коротком примере нет никакого промежуточного использования **a** и нет передачи управления между этими двумя назначениями, так что вычисление в S_7 избыточно и может фактически быть устранено.

Четвертое возможное отношение, называется *зависимостью входных данных*. Оно имеет место, когда осуществляется чтение при двух обращениях к одному участку памяти. Хотя зависимость входных данных не налагает никаких ограничений на порядок, транслятор может сделать это сам с целью оптимизации размещения данных на процессорах.

В случае зависимостей по данным, которые записываются $X \Rightarrow Y$, мы несколько неточны: зависимости формируются не самими выражениями в целом, а ссылками на отдельные переменные в пределах выражения. В приведенном выше примере зависимости выходных данных b, c, d, и e могут читаться из памяти в любом порядке, и результаты $b*c$ и $d + e$ могут быть выполнены, как только их операнды считались из памяти. $S_7 \dashrightarrow S_8$ фактически означает, что сохранение значения $b*c$ в **a** должно предшествовать хранению значения $d + e$ в **a**. Когда возникнет потенциальная двусмысленность, необходимо различать ссылки на различные переменные в пределах выражения.

Представление зависимостей

Чтобы получить информацию о зависимостях для участка кода, транслятор формирует *граф зависимостей*. Обычно каждый узел в графе представляет одно выражение. Дуга между двумя узлами указывает, что между вычислениями, представленными вершинами, есть зависимость.

Пример кода:

$a = 1;$
$b = a + 1;$
$a = 1;$

$b = a + 1$;
$c = a + b$;

На рис. 2 представлен граф зависимостей, который можно получить из выше приведенного кода.

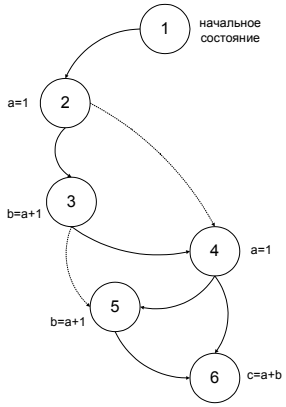


Рис. 2. Граф зависимостей

Используя зависимости, из этого графа можно удалить связи (2, 4) и (3, 5) (рис. 3). Проведя эквивалентные преобразования над графом, получим сеть Петри, изображенную на рис. 4.

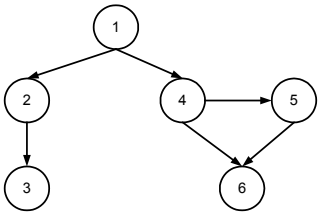


Рис. 3. Граф после отсечения связей

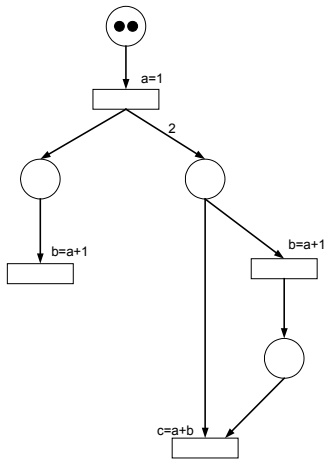


Рис. 4. Сеть Петри

Алгоритм распараллеливания на основе анализа зависимостей по данным

Алгоритм автоматического распараллеливания линейных программ заключается в итерационном анализе множества операторов присваивания P. Результатом работы алгоритма является матрица зависимостей программы. Для переменной Θ определим следующие атрибуты: $L(\Theta)$ – номер последнего оператора присваивания, в левой части которого встречалась переменная Θ; $R(\Theta)$ – номер последнего оператора присваивания, в правой части которого встречалась переменная Θ.

Если переменная ранее не использовалась в правой и/или левой части, то соответствующие атрибуты будут равны $\varnothing$.

Алгоритм 1. Построение матрицы зависимостей

Алгоритм позволяет по исходной программе построить матрицу зависимостей $S = \{s_{ij}\}$ и вектор начального состояния S^0.

Алгоритм состоит из цикла, в каждой итерации которого осуществляется проверка на зависимость текущего оператора присваивания. В строке 2 временной переменной t присваивается значение 1, затем, если будет найдена хоть одна зависимость, то t будет присвоено значение 0. Эта переменная используется как флаг, сигнализирующий о том, что данный узел будет входить в начальное состояние (строка 20). В строках с 4 по 7 осуществляется проверка на наличие зависимости по результату, в строках 8-11 – антизависимости. После этого обновляется значение атрибута L переменной. В строках 13-19 организован цикл по переменным из правой части оператора присваивания. Здесь каждая переменная проверяется на наличие истиной зависимости, и обновляется значение атрибута R.

```
1  : from i = 1 to n
2  :   t ← 1
3  :   Θ ← l_i
4  :   if L(Θ) ≠ ∅ then
5  :     o_{L(Θ),i} ← Θ
6  :     t ← 0
7  :   end if
8  :   if R(Θ) ≠ ∅ then
```

```
9  :      a_{R(Θ),i} ← Θ
10 :      t ← 0
11 :    end if
12 :    L(Θ) ← 0
13 :    for Θ ∈ R_i do
14 :      if L(Θ) ≠ ∅ then
15 :        t_{L(Θ),i} ← Θ
16 :        t ← 0
17 :      end if
18 :      R(Θ) ← i
19 :    end for
20 :    s_i^0 ← t
21 : end for
```

Заключение

Разработанный транслятор помогает эффективно использовать время программиста и машинные ресурсы. Теперь разработчики программного обеспечения могут не заботиться об архитектуре системы. Транслятор на основе анализа зависимостей по входным данным в программе может формировать указания операционной системе о возможности параллельного запуска отдельных ветвей программы.

Работа ведется при поддержке РФФИ (номера 00-07-90300, 01-07-90211, 02-07-06057, 02-07-06056).

Литература

1. *Ахо А.В.*, *Сети Р.*, *Ульман Дж.Д.*, Компиляторы: принципы, технологии, инструменты. М.: «Вильямс», 2001.
2. *Петерсон Дж.* Теория сетей Петри и моделирование систем. М.: Мир, 1984.
3. *Ачасова С.М.*, *Бандман О.Л.* Корректность параллельных вычислительных процессов. Новосибирск: Наука, 1990.

ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ ДИСТАНЦИОННОЙ РАЗРАБОТКИ И ТЕСТИРОВАНИЯ ПАРАЛЛЕЛЬНЫХ ПРИЛОЖЕНИЙ

А.В. Комолкин, С.А. Немнюгин, А.В. Захаров
СПбГУ
О.Л. Стесик
Междисциплинарный центр СПбГУ

Введение

Темпы разработки прикладного программного обеспечения для многопроцессорных вычислительных систем на настоящем этапе значительно уступают темпам развития самих высокопроизводительных вычислительных систем. На наш взгляд, это вызвано отсутствием у подавляющего большинства программистов навыков "параллельного" мышления, умения проектировать программы для многопроцессорного исполнения. Приобретение таких знаний требует значительного времени для непосредственной работы с многопроцессорной вычислительной системой, что зачастую невозможно, особенно для тех, кто только учится программировать. Проблема расширения круга пользователей высокопроизводительных вычислительных систем все чаще решается путем организации Web-порталов к кластерам и суперкомпьютерам, установки систем очередей и ориентации пользователей на работу с многопроцессорными системами через Интернет. Такое решение должно способствовать повышению эффективности *использования готовых приложений*, так как оно обеспечивает оптимальный режим работы высокопроизводительных ресурсов, но *разработка новых приложений* в рамках такого подхода становится весьма продолжительной и проблематичной.

Мы предлагаем проект системы, которая могла бы облегчить начальные стадии разработки параллельных приложений. Основной идеей проекта является использование специализированного комплекса программ на языке Java, предназначенного для разбора грамматических конструкций программ, анализа использования средств передачи сообщений и эмуляции процесса исполнения пользовательской программы на многопроцессорной вычислительной системе. Использование языка Java позволяет перенести начальные стадии разработки приложений на компьютер пользователя, не предъявляя к последнему практически никаких требований. Так как основой системы является эмуляция – эмуляция компилирования и эмуляция параллельного ис-

полнения, – то в ее систему могут быть включены дополнительные функции, как информативные, так и контролирующие.

Стадии разработки приложения и состав программного комплекса

Разработка приложения для многопроцессорной вычислительной системы, как правило, включает в себя 3 основных этапа:

- разработку алгоритма;
- написание текста параллельной программы;
- компиляцию и отладку.

Предлагаемый комплекс содержит к настоящему времени 4 относительно независимых программы, обеспечивающих или поддерживающих все три стадии разработки:

- портативный справочник параллельного программиста;
- интерактивный редактор;
- верификатор параллельных программ;
- эмулятор многопроцессорной вычислительной системы.

Программы имеют общую оболочку, и могут использоваться в комплексе, например по такой схеме: создание текста программы на исходном языке программирования (в текущий момент реализовано подмножество языка FORTRAN-77) с помощью интерактивного проверяющего редактора и портативного справочника, верификация программы на корректность языковых конструкций, трансляция и исполнение полученного кода на эмуляторе многопроцессорной вычислительной системы.

Интерактивный редактор используется в начале обучения и ориентирован на усвоение «беспроигрышного» стиля программирования. Более опытные пользователи имеют возможность обращаться к инструментарию, минуя интерактивный редактор, загружая предварительно подготовленный текст программы непосредственно в верификатор программ. Верификатор выполняет разбор конструкций программы и, при условии правильности загруженного текста, производит построение и компиляцию аналога исходной программы на языке Java. Эмулятор параллельной вычислительной системы обеспечивает запуск исполняемого аналога исходной программы и отслеживает процесс ее исполнения. Портативный справочник представляет собой комплект словарей с необходимым минимумом сведений по программированию в поддерживаемых системах параллельного программирования, а также набор справочных ссылок на источники более подробной информа-

ции, доступные при наличии выхода в Интернет.

Принципы работы компонентов комплекса

Портативный справочник является текстовой справочной системой, организованной по принципу «дерева», связанного с определенной файловой структурой. Предопределенным является только состав каталога верхнего уровня, названия подкаталогов которого определяют разделы справочника. Файлы-листья содержат в себе названия описываемых разделов. Такая организация позволяет легко вносить изменения и дополнения в справочник, не прибегая к изменениям в программе.

Интерактивный редактор представляет собой панель с текстовым полем и выпадающим меню, содержащим ключевые слова (наименования операторов) исходного языка программирования. Текст программы формируется из выбранных ключевых слов (ключевые слова, набранные вручную, не включаются в текст программы). Выбор оператора, предполагающего наличие парной закрывающей конструкции (например, «DO» и «END DO») приводит к включению в текст программы этой закрывающей конструкции и ограничению поля ввода пространством между ними. Готовый текст при нажатии кнопки «Verify» передается верификатору программного текста и, если верификатор не находит в нем ошибок, делает возможным переход к вызову компилятора. Компилятор вызывает преобразователь текста с исходного языка (FORTRAN-77) на язык Java и компиляцию последнего в байт-код. При успешной компилятора появляется возможность исполнить полученный байт-код на компьютере пользователя (например, в виртуальной Java-машине в Интернет-обозревателе Netscape), эмулируя параллельное исполнение на произвольном числе процессоров.

Реализация эмуляции и преобразования кода

Для эмуляции работы программы, написанной на исходном языке, в виртуальной машине Java создается комплекс Java-классов, описывающих действия и свойства компонентов программы на исходном языке. Так, для эмуляции работы MPI-системы, создается класс MPI, методы которого представляют собой описание действий, производимых вызовами библиотеки MPI. Все переменные исходной программы представляются объектами специального класса, поля и методы которого позволяют отслеживать все изменения состояния данной переменной в процессе исполнения программы. Эмуляция многопроцессорности обеспечивается легковесными процессами (нитями), под-

держка которых – одно из несомненных достоинств Java.

FORTRAN-77, принятый на настоящем этапе в качестве базового языка исходных программ, реализуется эмулирующей системой со строгим контролем и с ограничениями. В частности, осуществляется контроль использования COMMON-блоков, правильности передачи процедурам и функциям фактических параметров и их использования, присваивания новых значений параметрам цикла или константам и некоторых других запрещенных действий, которые описаны в стандартах FORTRAN-77 и 90, но не контролируются их исполняющими системами. Введены ограничения, связанные, главным образом, с громоздкостью полной реализации препроцессора FORTRAN-Java и исполняющей системы. В частности, упрощены конструкции языка FORTRAN-77, исключена работа со строками.

Задачей преобразователя кода является построение конструкций, заменяющих вызовы соответствующих подпрограмм обращениями к методам созданного класса. Эта задача решается в два этапа: разбор структур на исходном языке и построение описания исходной программы на xml-подобном языке, и, затем, перевод xml-описания программы на язык Java.

Заключение

Задача активного обучения принципам и методам создания параллельных алгоритмов и программ, их отладки и оптимизации, как показали несколько лет педагогической практики, является непростой даже в форме очного обучения в небольшой (8-10 учащихся) группе при непосредственной работе учащихся с учебной высокопроизводительной системой. Комплекс электронных учебных пособий, помогающий быстро отыскать необходимую справку, проверить, не прибегая к компиляции, программный код на наличие типичных ошибок, получить и оценить построенную в рамках учебной программы схему обмена сообщениями с точки зрения тупиковых ситуаций, поднимет эффективность очного обучения, в дистанционном же обучении он просто необходим.

Несмотря на неизбежные ограничения, предлагаемый комплекс обладает рядом очевидных преимуществ перед реальной многопроцессорной системой: он позволяет произвести разбор программы еще на стадии разработки исходного текста на языке FORTRAN; исполнение контролируется и аварийная ситуация сопровождается трассировкой вызовов процедур и состояния системы передачи сообщений; для эму-

лятора не нужны кластер, FORTRAN, MPICH и система очередей. Эмулятор помогает понять особенности отдельных операций обмена, тогда как использование реального мультипроцессора требует наличия глубокого понимания сути вызываемых процедур.

Авторы выражают благодарность Российскому фонду фундаментальных исследований за финансовую поддержку (грант №02-07-90332в).

ИЕРАРХИЧЕСКАЯ СИСТЕМА УПРАВЛЕНИЯ РАСПРЕДЕЛЕННЫМИ ВЫЧИСЛИТЕЛЬНЫМИ РЕСУРСАМИ ВЫСОКОЙ ПРОИЗВОДИТЕЛЬНОСТИ НА ОСНОВЕ GLOBUS TOOLKIT

В.В. Корнеев, А.В. Киселев, А.В. Баранов, Е.Л. Зверев

ФГУП НИИ «Квант», г. Москва

В настоящее время, в связи с распространением кластерных вычислительных систем, все более широкому кругу пользователей становятся доступными высокопроизводительные многопроцессорные установки (супер-ЭВМ). Управление вычислительными ресурсами супер-ЭВМ осуществляется с помощью специальных управляющих систем, таких как PBS, DQS, СУПЗ МВС-1000 и др. Каждая из этих систем имеет свои логику работы, алгоритмы планирования очередей и заданий и пользовательский интерфейс.

На сегодняшний день пользователь зачастую имеет возможность выполнять свои задания на нескольких вычислительных установках, причем количество доступных установок постоянно увеличивается. Различные системы управления вычислительными ресурсами имеют различные интерфейсы взаимодействия с пользователем, применяют различные механизмы планирования и политики безопасности. Для работы на каждой конкретной установке пользователь должен производить локальную регистрацию на ней, копирование необходимых для работы файлов исходных данных и программ, запуск заданий на счет в соответствии с правилами системы планирования, используемой данной вычислительной установкой, и, наконец, получать результаты выполнения своих заданий. При наличии у пользователя возможности использовать ресурсы нескольких различных вычислительных устано-

вок, возникают сложности с регистрацией (необходимо «помнить» или хранить пароли или ключи), с поддержкой правильной настройки рабочей среды на каждой вычислительной установке, с необходимостью взаимодействия с системами планирования заданий, которые могут отличаться на различных установках.

Кроме этого, у пользователя возникает задача выбора наиболее предпочтительной в данный момент вычислительной системы для запуска своих заданий. Факторами при выборе системы могут служить как текущая загруженность различных вычислительных ресурсов, так и «пригодность» системы определенного вида для запуска конкретного типа заданий. Задача выбора вычислительной системы с наиболее предпочтительной архитектурой возникает в том случае, когда доступные вычислительные ресурсы неоднородны.

Для автоматизированного решения указанных задач авторами предлагается иерархическая система управления вычислительными ресурсами высокой производительности. В основу данной системы положены протоколы взаимодействия распределенных систем, предлагаемые пакетом Globus Toolkit.

Globus Toolkit

Пакет Globus Toolkit служит для объединения различных вычислительных установок в единую вычислительную среду, так называемую «виртуальную организацию». В пакете Globus представлены средства для аутентификации пользователя на различных ресурсах и проверки его прав доступа (GSI); средства запуска задач пользователя на удаленных вычислительных установках и управления их выполнением (Resource Management); средства мониторинга состояния контролируемых ресурсов, входящих в виртуальную организацию (Information Service). Globus Toolkit является связующим компонентом, объединяющим различные части системы.

Менеджеры ресурсов

Система управления вычислительными ресурсами имеет иерархическую структуру, образуемую *менеджерами ресурсов*. Каждый менеджер отвечает за определенную часть ресурсов всей вычислительной системы. Области ответственности менеджеров формируются исходя из архитектурно-технических особенностей ВС, территориального, административного принципов деления или на основе иного подхода. Менеджеры реализуют функции мониторинга подчиненных ре-

сурсов, планирования заданий на подчиненных ресурсах и управление выполнением запущенных заданий. Принятие решений о выделении ресурсов, необходимых для выполнения задания, на основании поступивших запросов заданий также является функцией менеджеров системы планирования заданий. Менеджер либо принимает решение о выделении ресурсов, либо обращается к вышестоящему в иерархии менеджеру, если он не в состоянии принять решение, например, при отсутствии запрошенных ресурсов. Принимаемое решение о выделении ресурсов может учитывать большое число параметров и быть основанным на разных стратегиях. С точки зрения средств пакета Globus Toolkit, менеджер ресурсов сам является вычислительным ресурсом, который пользователи могут использовать для запуска своих заданий. Менеджер ресурсов реализует интерфейс, позволяющий стандартным для системы Globus способом запускать задания, контролировать их выполнение и получать информацию о самих ресурсах.

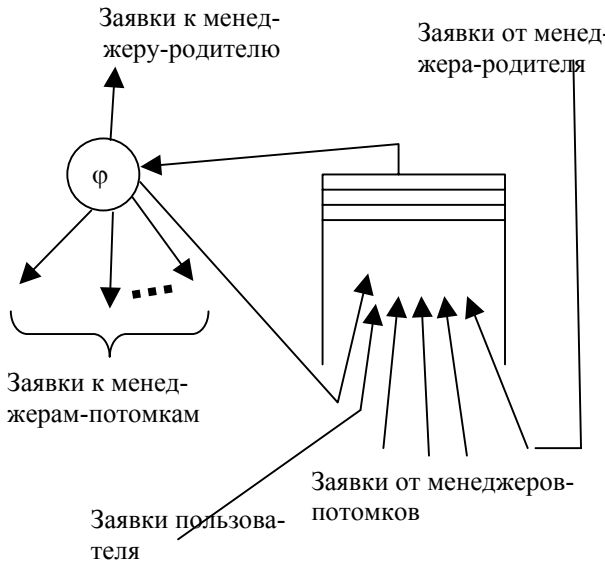
Планирование задач пользователя

Задания поступают на вход менеджера системы планирования в виде описания требований к вычислительным ресурсам и собственно параметров задания: имени исполняемого файла, параметров запуска, входных, выходных файлов. Требования могут быть сформулированы как вектор ресурсов, каждая компонента которого задает количество ресурсов определенного типа, или как указание на конкретные ресурсы (особенно это необходимо на начальном этапе). Описание задачи передается менеджеру на языке RSL, используемом в качестве языка описания запросов к ресурсам в пакете Globus Toolkit.

Источниками потока задач для менеджера являются пользователи системы, подчиненные в иерархической структуре менеджеры (менеджеры-потомки) и менеджер, находящийся в иерархии непосредственно над ним (менеджер-родитель). После получения задачи, менеджер планирует ее выполнение независимо от источника поступления задачи.

Каждый менеджер ведет собственную очередь задач. Принимая решение о назначении задачи, менеджер может оставить задачу в своей очереди, отправить менеджеру-родителю, отправить ее одному или нескольким подчиненным менеджерам. В последнем случае подчиненные менеджеры должны согласованно выделить ресурсы для исполнения задачи. Схематично менеджер системы планирования задач показан на рисунке. Все заявки поступают в очередь менеджера. Алго-

ритм планирования заявок φ выбирает заявку из очереди и определяет маршрут этой заявки: обратно в очередь, менеджеру-родителю, одному или нескольким (всем) менеджерам-потомкам. Такая архитектура представляется подходящей в силу того, что она уже используется в коммутаторах: общая очередь заявок на коммутацию для всех входов и решение, какой вход с каким выходом коммутировать.



При этом менеджер, не имеющий менеджеров-потомков, служит шлюзом в систему управления заданиями соответствующей подчиненной вычислительной системы, то есть локальную систему планирования.

При отправлении задания подчиненному менеджеру, необходимо выбрать наиболее подходящий среди всех подчиненных менеджеров для выполнения этого задания. Для принятия решения менеджер должен обладать информацией о состоянии подчиненных ресурсов. Система планирования заданий вычислительной установки может предоставить информацию о состоянии вычислительных ресурсов, такую как количество и вычислительная мощность процессоров, длина очереди, размер заданий в очереди, средняя пропускная способность. В связи с гетерогенностью установок и различными дисциплинами планирова-

ния, применяемыми в системах планирования заданий, данная информация недостаточна для выбора наиболее подходящей системы для выполнения задания. Необходим некоторый показатель, характеризующий эффективность выполнения задания пользователя на вычислительной установке. Данный показатель должен быть универсальным по отношению к гетерогенным вычислительным ресурсам и различным дисциплинам планирования, применяемым на них.

В качестве такого показателя может выступать, например, оценка времени запуска задачи на счет или завершения ее счета. Вычисление данного показателя может осуществляться либо самим ресурсом, который он характеризует, и отправляться менеджеру верхнего уровня, либо менеджером верхнего уровня на основании более простой информации, описывающей текущее состояние системы. В первом случае для оценки эффективности выполнения задания менеджер должен разослать информацию о нем на все подчиненные ресурсы с требованием оценить эффективность его выполнения, и на основании полученных результатов осуществить планирование. Во втором случае менеджер должен самостоятельно оценить данный параметр с учетом информации о состоянии подчиненного ресурса и используемой на нем дисциплины планирования.

Перемещение задания вверх по иерархии менеджеров может осуществляться как по инициативе подчиненного менеджера, так и по инициативе менеджера-родителя. Подчиненный менеджер должен направить задание вверх в случае, если он не обладает необходимыми вычислительными ресурсами, например, ни один из подчиненных ресурсов не удовлетворяет требованиям пользователя по ресурсам, необходимым для выполнения задачи или временным ограничениям, накладываемым пользователем на выполнение задачи.

Менеджер верхнего уровня может извлечь часть заданий у подчиненного менеджера и перепланировать их. Для выполнения этой операции он должен обладать некоторым показателем, характеризующим загруженность вычислительных ресурсов. Данный показатель также должен иметь универсальный характер по отношению к различным вычислительным установкам. При неравномерной загрузке подчиненных ресурсов, менеджер извлекает часть заданий из перегруженных систем и перепланирует их. При этом возникает задача определения заданий, перемещение которых обеспечит выравнивание нагрузки.

Описанные выше механизмы управления задачами реализуются алгоритмом φ, представляющим дисциплину планирования менедже-

ров. Поиск алгоритма, удовлетворяющего тем или иным требованиям, является предметом исследования.

Взаимодействие с пользователем

При направлении задания на вычисление, пользователь взаимодействует с системой через некоторый менеджер системы управления заданиями. Менеджер, получив описание задания, возвращает пользователю уникальный идентификатор задания в системе. Для дальнейшего взаимодействия пользователя с заданием используется тот менеджер, через который оно поступило в систему. Так как задание в ходе планирования может перемещаться между менеджерами, исходный менеджер задания должен отслеживать его местонахождение для предоставления пользователю возможности удалить задание, узнать его состояние. До запуска задания на счет, оно может находиться либо в очереди какого-либо менеджера, либо в очереди системы планирования заданий на вычислительной установке. После запуска задания на некоторой вычислительной установке, пользователю возвращается идентификатор задания в системе Globus. Дальнейшее взаимодействие со своим заданием пользователь осуществляет так же, как если бы он запустил это задание на данной установке непосредственно, используя средства Globus.

Дальнейшее участие менеджеров в процессе управления заданием нецелесообразно, поскольку, после запуска задания на выполнение, оно не может быть перепланировано на другую вычислительную установку и будет управляться только локальной системой планирования. То есть, после запуска задания на выполнение и пересылки пользователю его идентификатора, менеджеры ресурсов удаляют у себя все записи об этом задании и не участвуют в процессе дальнейшего управления заданием.

Разрабатываемая система управления распределенными вычислительными ресурсами высокой производительности позволяет унифицировать процесс взаимодействия пользователя с различными вычислительными ресурсами, представленными либо непосредственно вычислительными установками, либо менеджерами системы планирования, обладающими некоторыми вычислительными ресурсами. В качестве стандарта интерфейсов взаимодействия компонентов системы используется пакет Globus Toolkit. Разрабатываемая система может являться основой для исследования различных алгоритмов планирования заданий на распределенных гетерогенных вычислительных ресурсах.

ПРОБЛЕМЫ РАЗРАБОТКИ ИТЕРАЦИОННЫХ АЛГОРИТМОВ ДЛЯ ПОСТРОЕНИЯ РАСПИСАНИЙ ВЫПОЛНЕНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

В.А. Костенко

МГУ им. М.В. Ломоносова

В работе рассматриваются проблемы применения итерационных алгоритмов для решения задачи построения статических расписаний выполнения программ на многопроцессорных вычислительных системах (МВС). Задача построения расписания рассматривается в следующих двух вариантах:

- минимизируется время выполнения прикладной программы (расписания выполнения программы) при заданных ограничениях на допустимые к использованию аппаратные ресурсы МВС;
- минимизируется количество используемых процессоров МВС при заданном ограничении на время выполнения прикладной программы.

Модель прикладной программы задается набором взаимодействующих процессов. Точки взаимодействия разбивают каждый процесс на множество рабочих интервалов. Рабочие интервалы рассматриваются в качестве неделимых работ, подлежащих планированию. Каждый рабочий интервал характеризуется вычислительной сложностью. Она позволяет оценить время выполнения рабочего интервала на конкретном процессоре. Каждое взаимодействие характеризуется объемом данных обмена. Объем данных обмена позволяет оценить затраты времени на выполнение взаимодействия в конкретной коммутационной среде. Схема взаимодействия рабочих интервалов определяет ограничения на последовательность выполнения рабочих интервалов и задается отношением частичного порядка на множестве рабочих интервалов.

Расписание выполнения программы определено, если заданы: 1) множества процессоров и рабочих интервалов, 2) привязка, 3) порядок. Привязка – всюду определенная на множестве рабочих интервалов функция, которая задает распределение рабочих интервалов по процессорам. Порядок задает ограничения на последовательность выполнения рабочих интервалов и является отношением частичного порядка, удовлетворяющим условиям ацикличности и транзитивности. От-

ношение порядка на множестве рабочих интервалов распределенных на один и тот же процессор, является отношением полного порядка. Различные варианты расписаний могут быть получены изменением привязки и порядка в заданных ограничениями пределах, которые определяются требованием не нарушения заданного отношения частичного порядка в модели прикладной программы и определениями привязки и порядка.

Модель аппаратных средств определяется как совокупность исполнителей и каналов связи их объединяющих. Исполнитель обладает некоторым ресурсом: способностью выполнять заданный набор действий с определенным временем выполнения каждого действия. Канал связи может лишь обеспечивать передачу данных с некоторой задержкой. Исполнители делятся на распределенные и последовательные. Последовательный исполнитель в одно и то же время может выполнять действия, запрошенные только одним процессом. Один последовательный исполнитель может в режиме разделения времени обслуживать несколько процессов. Последовательные исполнители могут соединяться каналами связи и образовывать распределенный исполнитель. При детализации программ на уровне взаимодействия процессов мы выделяем следующие последовательные исполнители: процессоры, модули памяти, процессоры ввода/вывода и распределенный исполнитель: коммутационная среда. Коммутационная среда рассматривается как распределенный исполнитель, образованный из последовательных исполнителей – коммутационных узлов и узловых исполнителей (оконечных устройств). В качестве узловых исполнителей могут быть использованы процессоры, процессоры ввода/вывода, модули памяти и другие распределенные исполнители. Тип коммутационной среды определяется топологией связи и моделью синхронизации.

Схематично работу итерационных алгоритмов для решения задачи построения расписания можно представить следующим образом:

1) задать начальное приближение HP^0, $k = 0$;
2) вычислить целевую функцию $f(HP^k, HW)$;
3) получить $HP^{k+1} = D(\{HP^i\}, \{f(HP^i, HW)\}, Pr^k$: i $\in(1,\ldots,k))$;
4) если заданный критерий останова не достигнут, то $k = k + 1$ и перейти к п.2; в противном – завершить работу алгоритма.

Здесь, HP – расписание выполнения программы, HW – модель аппаратных средств, D – некоторая стратегия коррекции текущего рас-

писания, Pr^k – параметры стратегии (для ряда стратегий, возможно, их изменение в ходе работы алгоритма).

Данной схемой могут быть описаны генетические и эволюционные алгоритмы, алгоритмы случайного поиска (ненаправленного, направленного, направленного с самообучением), алгоритмы имитации отжига, алгоритмы алгебраической коррекции решений. Алгоритмы случайного поиска, имитации отжига и алгебраической коррекции решений на каждой итерации работают с одним решением HP, генетические и эволюционные алгоритмы – с некоторым множеством решений $\{HP\}$ (популяцией).

Одной из наиболее сложных проблем при использовании итерационных алгоритмов является проблема представления и преобразования расписания выполнения программы. Поскольку расписание является комбинаторной структурой, то возникает проблема согласованного изменения задающих расписание переменных, таким образом чтобы на всех итерациях получать корректные расписания. В генетических алгоритмах система операций преобразования решений определена: операции скрещивания, мутации и селекции. Требуется выбрать такую форму представления расписаний, чтобы операции генетического алгоритма не приводили к получению некорректных вариантов расписаний и любое корректное расписание могло быть представлено в выбранной форме представления расписаний. В алгоритмах имитации отжига, случайного поиска, алгебраической коррекции расписаний система операций преобразования расписаний вводится при разработке алгоритма. Для этих алгоритмов способ представления расписаний и система операций преобразования расписаний должны обладать следующими свойствами: применение операции должно приводить к корректному варианту расписания и система операций должна быть функционально полной.

Использование в итерационных алгоритмах системы операций преобразования расписаний, получающих некорректные расписания, приводит к необходимости введения в целевую функцию слагаемых, отвечающих за штрафы, или барьерных функций. При этом возникают проблемы оценки значений целевой функции для некорректных расписаний и искажения пространства решений.

В докладе рассмотрены непосредственные способы представления расписания (привязка и порядок задаются явно) [1] и параметрические способы представления расписания (привязка и порядок задаются значениями некоторых параметров, по которым, с использованием алго-

ритма восстановления, их значения могут быть восстановлены) [1].

Для непосредственного способа представления расписания получена функционально полная система операций преобразования расписаний и доказано, что она допускает построение итерационных алгоритмов, допускающих переход от любого корректного расписания к оптимальному за линейное число итераций относительно числа рабочих интервалов в модели поведения программы и при этом на всех итерациях алгоритм получает лишь корректные расписания [1]. Данные операции используются в алгоритмах случайного поиска, имитации отжига и алгебраической коррекции решений.

Для параметрического представления расписания доказана возможность представления любого допустимого варианта расписания и однозначность его восстановления [1]. Данная форма представления используется в генетических и эволюционных алгоритмах. Параметрическое представление допускает независимое изменение значений всех переменных в заданных интервалах, что позволяет гарантированно получать корректные варианты расписания при выполнении операций генетического алгоритма. Данная форма представления может быть использована также и в алгоритмах случайного поиска и имитации отжига.

В докладе будут приведены результаты численного исследования генетических и эволюционных алгоритмов, алгоритмов имитации отжига и алгоритмов алгебраической коррекции решений при решении задач построения расписаний.

Литература

1. *Костенко В.А*. Задача построения расписания при совместном проектировании аппаратных и программных средств // Программирование, 2002, №3. С. 64–80

ПРИМЕНЕНИЕ РАСПАРАЛЛЕЛИВАНИЯ ДЛЯ МОДИФИЦИРОВАННОГО МЕТОДА СЛУЧАЙНОГО ПОИСКА МАКСИМУМА ДЛЯ МНОГОМЕРНОЙ, МНОГОЭКСТРЕМАЛЬНОЙ ЗАДАЧИ

О.А. Кузенков, АЛ. Ирхина
Нижегородский государственный университет им. Н.И. Лобачевского

Постановка задачи

Будем рассматривать следующую задачу. Пусть $J(z)$ есть действи-

тельная, строго положительная, непрерывная функция, определенная в гиперпараллелепипеде Ω N-мерного евклидового пространства R^N то есть

$$\Omega = \{z \in R^N : a_i < z_i < b_i,\ 1 < i < N\}, \quad (1)$$

где **a, b** есть заданные векторы (**a** = $(a^1,a^2,\ldots a^N)$, **b** = $(b^1,b^2,\ldots,b^N)$), и пусть в точке z^* функция $J(z)$ достигает максимального значения на множестве Ω то есть

$$J(z^*) = \max J(z), \text{ при } z \in \Omega.$$

Требуется построить оценку $z_* \in \Omega$ точки z^*, так, чтобы значение $J(z_*)$ было бы максимально приближено к значению $J(z^*)$ на основании конечного числа n значений функции, последовательно вычисленных в выбранных точках области Ω.

При выборе точек очередного испытания (испытанием будем называть операцию вычисления значения функции в точке z_i, при этом точка z_i будет считаться лучше точки z_j, если $J(z_i) > J(z_j)$) учитываются две гипотезы:

1) вероятность осуществить поиск нового решения, опираясь на более удачное испытание, выше вероятности осуществить поиск, опираясь на менее удачное;
2) вероятность рассмотрения новой точки при улучшении уже имеющейся обратно пропорциональна степени расстояния между этими точками [1].

Математическая модель метода многоэкстремального поиска максимума и описание его модификаций

Пусть $||z_i - z||$ – величина, которая характеризует отличие точки z от точки z_i. Функция $J(z)$ – непрерывная и строго положительная, z_i – точка, рассматриваемая на i-ом шаге; $a_i f(||z - z_i||)$ – плотность вероятности рассмотреть точку z на i-ом шаге, где f удовлетворяет следующим свойствам: $f(||z - z_i||) > 0$ для всех z и z_i из Ω, f – непрерывная строго монотонно убывающая функция, $f(||z - z_i||) \to 0$ при $z \to \to \infty$, a_i – нормировочный множитель, определяемый из условия

$$\int_{\Omega} a_i f(||z - z_i||) = 1. \quad (2)$$

Каждой выбранной точке z_i на очередном $m + 1$-ом шаге (имеется в виду, что уже сделано m шагов и выбрано m точек) присваивается

вероятность осуществить поиск, равная

$$(J(z_i))^m (\sum_{j=1}^{m} (J(z_j))^m)^{-1}. \quad (3)$$

Тогда плотность p'_{m+1} – плотность вероятности выбора точки z на $m+1$-ом шаге – равна:

$$p'_{m+1}(z) = \sum_{j=1}^{m} (J(z_j))^m a_j f(\| z - z_i \|) * (\sum_{k=1}^{m} (J(z_k))^m)^{-1}. \quad (4)$$

(Первая модификация [1])

Пусть $f(0) < 1$, плотность вероятности рассмотреть проект z на $m+1$-м шаге, опираясь на z_i, равна

$$h_{im}(\| z - z_i \|) = m^{f(\|z - z_i\|)} (\int_{\Omega} m^{f(\|z - z_i\|)})^{-1}. \quad (5)$$

Очевидно, h_{im} обладает теми же свойствами, что и f; для вероятности осуществить поиск, опираясь на проект z_i при наличии m точек z_1, z_2,...,z_m, сохраним выражение (3), использованное в первой модификации. Тогда плотность на $m+1$-ом шаге равна

$$p''_{m=1}(z) = \sum_{j=1}^{m} (J(z_j))^m h_{im}(\| z - z_j \|) * (\sum_{k=1}^{m} (J(z_k))^m)^{-1}. \quad (6)$$

(Вторая модификация [1]). В [1] формулируются и доказываются теоремы о сходимости метода.

В третьей модификации для выбора новой точки на первых шагах применяется функция

$$\hat{f}(\| z - z_i \|) = 1 - f(\| z - z_i \|), \quad (7)$$

обеспечивающая достаточно широкий разброс точек по исследуемому компакту Ω. В дальнейшем метод работает аналогично второй модификации.

В четвертой – множество получаемых точек (W) делится на два подмножества – «хороших» и «плохих». В первое подмножество входит q точек с максимальными вероятностями. Во второе – все остальные. В качестве функции плотности вероятности для «хороших» точек будем использовать (5), а для «плохих» – (7). Таким образом, для поиска глобального экстремума будем использовать только «хорошие» точки, а «плохие» – для просмотра компакта Ω.

В пятой модификации для повышения информативности метода

на каждом шаге вычисляется не одна, а несколько (допустим l) новых точек. Таким образом, за n шагов будет исследовано $n*l$ точек пространства Ω. Остальное – все, как в четвертой модификации.

Описание алгоритма

Входными данными являются:

Размерность задачи – N, границы гиперпараллелепипеда Ω– **a** и **b**. Параметры метода: число шагов эксперимента – n, количество начальных шагов для просмотра области Ω – n', доля «хороших» точек – q, количество точек для одной итерации – l.

Первые l точек задаются случайным образом, каждая из последующих рассчитывается по нижеприведенной схеме, реализуемой в два этапа:

– из множества W выбирается точка z_i с вероятностью рассчитываемой по формуле (3), что согласуется с первой гипотезой;
– вычисляются l новых точек z из Ω на основании z_i, полученной на первом этапе работы.

Основой для расчета z служит функция вида (5), расчет по которой, кстати, совсем не противоречит второй гипотезе, при этом учитывается, «хорошая» это точка или «плохая» и не меньше ли номер итерации числа n'. Для каждой из l новых точек вычисляется значение функции $J(x)$. Все новые точки добавляются в множество W, и для каждой точки из W по формуле (3) пересчитывается вероятность выбора ее на следующем шаге как опорной. Дальше возвращаемся к первому этапу, и все повторяется, пока не будет сделано n шагов.

Оценка полученных результатов

На основании численных экспериментов были сделаны следующие выводы: если функция имеет много экстремумов, то стоит увеличить n' и уменьшить m; метод лучше отыскивает экстремумы пологого характера, если экстремум с пиковой структурой, рекомендуется выбрать более плоскую функцию плотности распределения. При работе метода в четвертой и в пятой модификациях скорость сходимости зависит от количества начальных точек (имеющих перевернутую функцию плотности распределения вероятности). И для пятой модификации увеличение параметра l приводит к повышению скорости сходимости метода, но значительно увеличивает время расчетов.

Для демонстрации работы метода (его четвертой модификации)

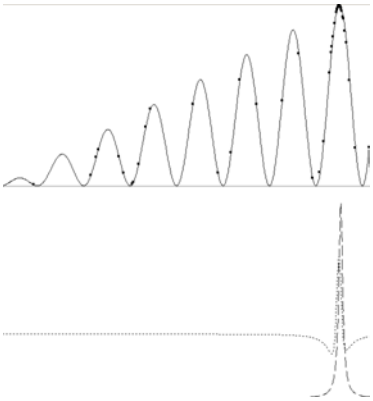


Рис. 1

были написаны программы с функциями в R^1 и в R^2. Ниже приведены результаты численных экспериментов.

Для пространства R^1 использовалась функция $J(z)=z+z\sin(z)+\varepsilon$. Пример работы программы изображен на рис. 1: сверху исследуемая функция, точки соответствуют итерациям метода, внизу сплошной линией изображена суммарная плотность распределения вероятности выбора z, пунктирной – плотность распределения вероятности выбора z для конкретной z_i.

Для 50 экспериментов с различными наборами параметров были получены следующие результаты.

Таблица 1

Параметры	Кол-во экспериментов достигших экстремума с точностью 0,0001	Кол-во экспериментов достигших экстремума с точностью 0,01	Кол-во экспериментов, не достигших экстремума
N=1, границы (0;50), n=50, n' =5, m=20; l=1; σ=0,1; ε=1	–	25	20
N=1, границы (0;50), n=200, n' =15, m=20; l=1; σ=0,1; ε=1	20	39	5
N=1, границы (0;50), n=50, n' =35, m=5; l=1; σ=0,1; ε=1	–	26	8
N=1, границы (0;50), n=150, n' =25, m=5; l=1; σ=0,1; ε=1	8	32	7
N=1, границы (0;50), n=200, n' =25, m=5; l=1; σ=1; ε=1	29	35	14

Для рассмотрения работы метода в пространстве R^2 использовалась функция

$$F(x,y)=\{(\sum_{i=1}^{7}\sum_{j=1}^{7}[A_{ij}a_{ij}(x,y)+B_{ij}b_{ij}(x,y)])^2+$$

$$+\Big(\sum_{i=1}^{7}\sum_{j=1}^{7}[C_{ij}a_{ij}(x,y)+D_{ij}b_{ij}(x,y)]\Big)^2\Big\}^{1/2}, \qquad (8)$$

где $a_{ij}(x,y)=\sin(\pi ix)\sin(\pi jy)$, $b_{ij}=\cos(\pi ix)\cos(\pi jy)$, выбор коэффициентов A_{ij}, B_{ij}, C_{ij}, D_{ij} определяется случайным (равномерно и независимо) образом на отрезке$[-1;1]$. Максимизация подобных функций возникает, например, в задаче оценки максимального напряжения (определяющего прочность) в упругой тонкой пластине при поперечной нагрузке.

На рис. 2 сверху приведен пример такой функции для очень небольшого квадрата (0; 0,6)×(0; 0,6), снизу график суммарной плотности вероятности на последнем шаге. Методом перебора с шагом 0,001 найдено значение максимума, равное 21,5306.

В табл. 2 приведены результаты работы метода при различных наборах параметров в 50 экспериментах.

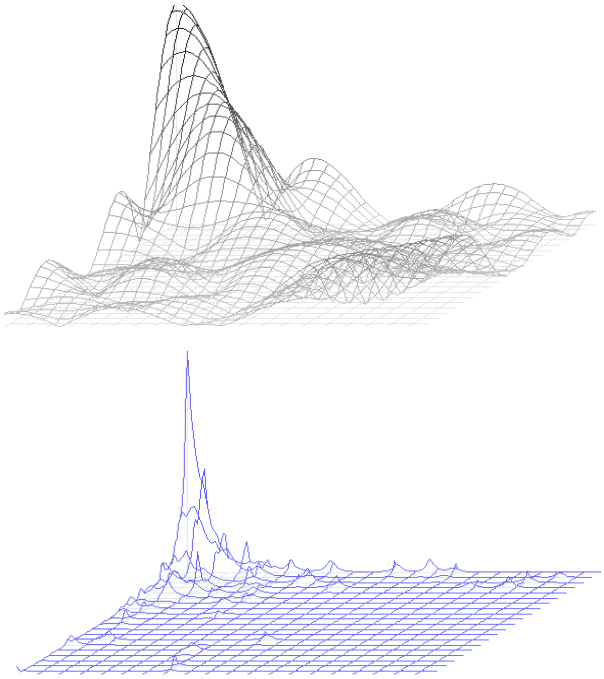


Рис. 2

Таблица 2

Параметры	Количество экспериментов, значение максимума в которых больше или равно 21,6	Количество экспериментов, значение максимума в которых больше или равно 21,5	Количество экспериментов, не достигших экстремума
(N=2, границы (0;0,6)×(0;0,6), n=150, n' =50, m=50; l=1; σ=1; ε=1)	–	50	0
(N=2, границы (0;0,6)×(0;0,6), n=150, n' =50, m=50; l=1; σ=10; ε=1)	33	45	2
(N=2, границы (0;0,6)×(0;0,6), n=150, n' =50, m=5; l=1; σ=1; ε=1)	10	47	0

Во всех случаях, за исключением двух, метод попал в окрестность точек глобального экстремума (скорее всего, из-за «удачного» подбора коэффициентов функции (8)). Возможно, при расширении границ метод даст результаты хуже, так как вид функции весьма усложняется. Результаты зависят от подбора параметров, их оценку можно производить, только выполнив серию испытаний.

Применение распараллеливания к предлагаемому методу

Исследование работы пятой версии метода будет интересно с использованием системы параллельного программирования. Посмотрим, где ее здесь можно применить.

Для каждой итерации требуется сделать следующее: из множества W выбрать опорную точку, и, опираясь на нее, выбрать l новых, подсчитать значение целевой функции для каждой, присоединить их к W, для всех из W подсчитать вероятность их выбора как опорной.

Выбирая l новых точек, программа делает l одинаковых операций, не зависящих друг от друга, и не требующих в этот момент никаких данных, кроме значения опорной точки. Положив значение параметра l равное количеству процессоров в системе и каждому передав сведения о ней (ее координаты, значение целевой функции в этой точке, «плохая» она или «хорошая» …), l действий система выполнит за один прием. Сложность каждого действия зависит от способа моделирования случайной величины (например, метод Неймана [3]), и от функции $J(z)$, подсчет значения в которой может занимать довольно много вре-

мени, как, например, для функции вида (8). Применение распараллеливания, в таком случае даст ускорение работы программы в $(l - \xi)$ раз, где ξ время обмена данными между процессорами зависящее от конкретного аппарата.

Стоит обратить внимание на выбор функции плотности вероятности, при моделировании случайной величины. Так как $z = (x^1, x^2,...,x^N)$ – вектор в N – мерном пространстве, мы имеем дело со случайными N-мерными величинами, для простоты расчетов будем считать что

$$a_i f(\| z - z_i \|) = \prod_{j=1}^{N} a_j f(| x^j - x_i^j |) .$$

Таким образом, каждую координату вектора z можно считать независимой случайной величиной. Для каждой координаты i (за x обозначим произвольную координату вектора z) берется независимая плотность распределения вероятности $f(| x^j - x_i^j |)$. Следовательно, для получения каждой новой точки надо сделать N независимых одинаковых действий. Предположим, что возможности позволяют взять количество процессоров равное $l*N$ и для каждой из l параллельно рассчитывающихся точек параллельно рассчитывать их координаты. Получим ускорение в $(l*N - \xi)$ раз.

На рис. 3 внизу представлена схема работы метода.

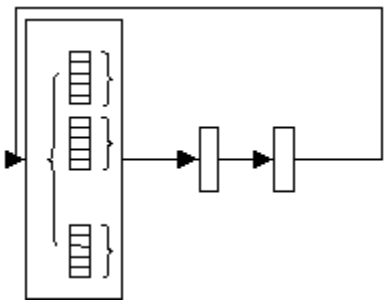


Рис. 3

Литература

1. *Кузенков О.А.* О многоэкстремальной оптимизации в математических моделях процесса оптимального проектирования // Вестник ННГУ / Н.Новгород: Изд-во ННГУ. 2000. С. 111–117.

2. *Стронгин Р.Г.* Численные методы в многоэкстремальных задачах (информационно-статические алгоритмы). М.: Наука, 1979.
3. *Соболь И.М.* Численные методы Монте-Карло. М.: Наука, 1973.
4. *Ширяев А.Н.* Вероятность. М.: Наука, 1989.

ПУТИ МОДЕРНИЗАЦИИ ПРОГРАММНЫХ И АППАРАТНЫХ КЛАСТЕРНЫХ РЕСУРСОВ ДЛЯ ЗАДАЧ ВЫЧИСЛИТЕЛЬНОЙ ХИМИИ

М.Б. Кузьминский, А.С. Мендкович, Н.А. Аникин

Институт органической химии РАН, Москва

А.М. Чернецов

Московский энергетический институт

Введение

Центр компьютерного обеспечения химических исследований (ЦКОХИ) РАН в Институте органической химии РАН является ведущим в РФ центром коллективного пользования, предоставляющим через Internet высокопроизводительные вычислительные ресурсы отечественным организациям разного ведомственного подчинения, находящихся в различных регионах страны, для проведения расчетов в области вычислительной химии.

Поскольку основными вычислительными ресурсами ЦКОХИ стали Linux-кластеры на базе x86-совместимых микропроцессоров (МП), задача модернизации кластеров с достижением наилучшего отношения стоимость/производительность выходит на первое место. Для определения оптимальных путей масштабирования производительности кластерных ресурсов необходимо использовать оценки производительности узлов кластеров и каналов связи между ними. Такие оценки не только позволяют сделать выводы о путях модернизации ресурсов ЦКОХИ, но и могут быть использованы при разработке типовых рекомендаций для построения современных высокопроизводительных вычислительных кластеров, в частности, ориентированных на задачи химии.

В связи с этим нами было проведено тестирование производительности узлов кластеров, использующих различные x86-совместимые МП в широком диапазоне производительности, на тестах

Linpack и STREAM. Полученные результаты официально «сертифицированы» (данные Linpack опубликованы, а результаты тестов STREAM будут включены в следующую версию таблицы результатов). Нами показано, что для проведения таких тестов для x86-МП необходимо использовать таймеры с высоким разрешением, основанные на счетчике тактов (RDTSC).

Авторами впервые было проведено тестирование каналов связи Gigabit Ethernet на тестах netperf для 64- и 32-разрядных шин PCI с использованием наиболее высокопроизводительных 64-разрядных плат Gigabit Ethernet фирмы Intel (Pro/1000T). Также впервые получены сертифицированные результаты тестирования производительности двухпроцессорных SMP-серверов на базе x86-совместимых МП на тестах Linpack (n = 1000). На основе полученных результатов сделан вывод о целесообразности применения Gigabit Ethernet и узлов на базе Intel Pentium 4 или AMD Athlon.

Для развернутого в ЦКОХИ кластера на базе Gigabit Ethernet разработан также новый пользовательский интерфейс прикладных программ с пакетной системой Sun Grid Engine и системой электронной почты (в настоящее время эксплуатируется с комплексом программ Gaussian-98).

Созданы базовые интерфейсы интегрированной распределенной информационной системы, включающей графический пользовательский интерфейс для интерактивного конструирования молекул, БД квантовохимических расчетов с использованием Web-интерфейса пользователя и XML/CML-интерфейсы для связи между компонентами системы. Это повышает эффективность применения кластерных ресурсов пользователями.

1. Тестирование каналов связи Gigabit Ethernet

В [1–3] проведены исследования производительности работы стека протоколов TCP/IP на тестах netperf в ОС Linux для каналов связи Fast Ethernet и Gigabit Ethernet с использованием 32- разрядных шин PCI с частотой 33 МГц. Было найдено, что на таких шинах PCI пропускная способность (ПС) на тестах TCP_STREAM и UDP_STREAM была почти в 3 раза ниже пикового значения 1 Гбит/с.

В данной работе те же сетевые платы Intel Pro1000T в 64-разрядном серверном исполнении протестированы с применением 64-разрядных шин PCI в ОС Linux 2.4.7-10 (RedHat 7.2) с использованием драйвера e1000 версии 4.0.7. В качестве узлов кластера использовались

2-процессорные платы Tyan с набором микросхем AMD760MP и МП AMD Athlon XP 1800+, а также с МП Pentium III Tualatin/1266 МГц и набором микросхем от ServerWorks. В серверах на базе AMD поддерживаются PCI-64 /33 МГц, а на базе Intel Pentium III - PCI-64/ 66 МГц. С серверами на базе AMD удалось достигнуть ПС на тестах UDP_STREAM и TCP_STREAM на уровне около 900 Мбит/с, а в тестах TCP_RR и UDP_RR производительность была на уровне 11 тысяч транзакций в секунду. Переход к PCI с частотой 66 МГц (в серверах на базе Pentium III) при работе с однопроцессорным ядром Linux несколько повышает результаты тестов UDP_STREAM и TCP_STREAM. Проведено исследование влияния установок драйвера на достигаемые результаты. Для плат на базе Pentium III найдено необходимым увеличение задержек для достижения более стабильной работы при большом трафике.

2. Тестирование производительности узлов

Было показано, что для получения достаточно точных результатов необходимо применять таймеры высокого разрешения (лучше, чем стандартные 0,01 с). При распараллеливании в кластере актуальность использования таких таймеров повышается. Примененный нами таймер использует счетчик тактов процессора (RDTSC), что для современных ПК-серверов обеспечивает разрешение меньше 1 мкс.

Ряд тестов, являющихся индустриальными стандартами, в частности, Linpack и STREAM, на современных процессорах выполняются доли секунды и требует измерения времен от порядка нескольких секунд до 10^{-5} с. В качестве конкретного примера влияния таймера на результаты измерений укажем, что ошибка стандартных таймеров в 0,01 с на тестах Linpack (n =1000) для AMD Athlon/700 МГц при использовании библиотеки Atlas приводит к ошибке производительности в 9 MFLOPS, а для Athlon XP/1533 МГц - уже в 39 MFLOPS. Очевидно, что по мере роста производительности x86-совместимых МП стандартные таймеры будут все более неприемлемыми, а RDTSC-таймеры, чье разрешение улучшается пропорционально тактовой частоте, останутся по-прежнему пригодными для измерений.

Теоретическое разрешение RDTSC-таймера равно 10^{-8} с даже для 100 МГц МП. В реальных измерениях следует учитывать поправку на время вызова подпрограммы таймера и другие факторы, в т.ч. возможные затраты времени на работу ядра, влияющие на точность измерения. Даже для самого медленного из используемых в данной работе

МП Intel Celeron/433 МГц измеренное предельное разрешение таймера равно $2x10^{-7}$ с. В тестах Linpack при n = 100 отклонения от среднего значения при 30 измерениях по порядку величины не превышают 10^{-5}с.

В тестах STREAM 5.0 производится 10 измерений производительности и считается среднее время выполнения. В наших измерениях реальное подсчитанное этим тестом разрешение таймера было лучше 1 мкс., а точность – также не хуже нескольких микросекунд. Вместе с приведенными далее результатами тестов это показывает, что применение RDTSC-таймера в рассматриваемых случаях является не только необходимым, но и, как правило, достаточным для получения приемлемой точности.

В наших тестах [4] использовано 5 различных серверов: на базе Intel Celeron/433 МГЦ с набором микросхем 440 BX и ОП типа PC100; двухпроцессорные серверы на базе Intel Pentium III/600 МГц с набором микросхем 440BX и ОП PC100; на базе AMD Athlon/700 МГц с набором микросхем VIA KX-133 и ОП PC133; двухпроцессорный сервер на базе Intel Pentium III Tualatin/1266 МГц с набором микросхем ServerSet III LE3 с ОП PC133; двухпроцессорный сервер на базе AMD Athlon XP 1800+ с набором микросхем AMD 760MP с ОП DDR266.

Результаты тестов Linpack представлены в табл.1. Они были сертифицированы и включены в официальную таблицу результатов тестов Linpack (http://performance.netlib.org/performance/html/linpack.data.col0.html). Из этих данных можно рассчитать ускорения вычислений, достигаемые в двухпроцессорных x86-серверах SMP-архитектуры (при n=1000) по сравнению с однопроцессорными. Полученные результаты достаточно высоки. Для Athlon XP при использовании более «быстрой» библиотеки Atlas 3.4.1 ускорение значительно ниже. Однако наши данные для используемой в ЦКОХИ программы Gaussian-98 показывают, что применение 2-процессорных систем SMP-архитектуры на базе таких МП на ряде задач квантовой химии позволяет получить вполне удовлетворительное ускорение.

Сопоставление с другими официальными результатами Linpack показывает [4], что МП, используемые в ПК, при n = 100 обогнали векторные процессоры, уступая им лишь при n=1000. Лидером среди МП при n = 1000 является Itanium 2, который опережает IBM Power4. В свою очередь, МП Pentium 4 и Athlon лишь немного уступают по производительности ведущим 64-разрядным МП Itanium 2 и Power4.

Таблица 1

Результаты проведенных измерений производительности на тестах Linpack

Микропроцессор/такт. частота	Число МП	Производительность			Комментарий (библ. времени выполнения)
		n = 100		n = 1000	
		lda = 201	lda = 200		
Celeron/433	1	160	160	263 248	MKL Atlas-3.4.1
Pentium III/600	1 2	113	116	410 745	MKL MKL
Athlon/700	1	295	317	552 772	MKL, станд. таймер Atlas-3.4.1
Pentium III/1266	1 2	501	503	830 1478	MKL, станд. таймер MKL, станд. таймер
Athlon XP1800+/1533	1 2	686	732	1087 1980	MKL, станд. таймер
	1 2			1623 2173(*)	Atlas-3.4.1

Примечания. Во всех измерениях, если не оговорено противное, использовался RDTSC-таймер. Для компиляции применялся транслятор ifc 5.0 с ключами -O3 -tpp6. lda – это ведущая размерность матрицы коэффициентов системы линейных уравнений.

Результаты тестов Linpack характеризуют производительность МП на задачах с плавающей запятой и практически не зависят от ПС оперативной памяти (ОП), что, как правило, не типично для современных высокопроизводительных приложений, например, в области квантовой химии. Однако нами было найдено, что производительность некоторых приложений квантовой химии, характеризующихся хорошей локализацией в кэш-памяти, в частности, метода ССП, коррелирует с данными тестов Linpack при n=100.

Результаты тестов STREAM для ПС ОП (табл. 2) дополняют данные Linpack. Сопоставление с литературными данными показывает, что по ПС ОП векторные системы далеко впереди МП, а x86-МП, как правило, превосходят RISC-процессоры [4].

Работа финансировалась в рамках проектов РФФИ 01-07-90072, 02-07-90146 и МКНТ (проект 1.2.63).

Таблица 2

Результаты тестов STREAM, Мбайт/с

Компьютер, процессор/частота, МГц	Copy	scale	add	triad
Tyan S2460, Athlon/1533	787	772	848	731
Gigabyte 7VX, Athlon/700	512	503	585	484
Tyan S2518, Pentium III Tualatin/1266	386	386	516	510
ASUS P2B-D, Pentium III/600	311	314	368	364
ASUS P2B-F, Celeron/433	204	205	246	247

Результаты тестов STREAM будут размещены официальном сайте http://www.cs.virginia.edu/stream/standart/Bandwidth.html)

Литература

1. *Кузьминский М., Мускатин А.* Открытые системы, 2001, №7–8. С. 17.
2. *Кузьминский М., Мускатин А.* 1-я национальная конференция «Информационно-вычислительные технологии в решении фундаментальных научных проблем и прикладных задач химии, биологии, фармацевтики, медицины.» Сб.тезисов, М., 2002. С. 163.
3. *Кузьминский М., Мускатин А.* Материалы международного научно-практического семинара «Высокопроизводительные вычисления на кластерных системах», 20-24 ноября 2001 г., Н.Новгород: Изд-во ННГУ, 2002. С. 92.
4. *Кузьминский М.* Открытые системы. 2002, №9. С. 10.

ВЫСОКОПРОИЗВОДИТЕЛЬНЫЙ ВЫЧИСЛИТЕЛЬНЫЙ КЛАСТЕР ВОРОНЕЖСКОГО ГОСУНИВЕРСИТЕТА

С.Д.Кургалин, В.С.Александров, С.В.Побежимов

Воронежский государственный университет, Воронеж

В 2002 г. в Воронежском государственном университете (ВГУ) создана лаборатория высокопроизводительных вычислений на основе вычислительного кластера: четырехпроцессорного сервера SUPERSERVER 8060 (4×CPU Pentium III Xeon, 700 МГц, 4 Гб SDRAM), четырех двухпроцессорных серверов SUPERSERVER 6010H

(2×CPU Pentium-III, 1,13 ГГц, 1 Гб SDRAM), восьми рабочих станций (Pentium–4, 2,0 ГГц, 512 Мб RIMM). На кластере установлена базовая операционная система Linux RedHat 8.0 и среда параллельного программирования MPI (бесплатная реализация MPICH 1.2.4).

Объединение вычислительного кластера ВГУ с действующей региональной частью Российской информационно–образовательной среды (ИОС) на основе сформированного в 2000 г. образовательного портала «Voronezh.openet.ru» – «Воронежского виртуального университета» – с его развитыми системами открытого и дистанционного обучения значительно увеличивает возможности для проведения расчетов в области фундаментального естествознания и моделирования ресурсоемких процессов. Образование лаборатории и включение ее в ИОС создает новое информационное пространство для проведения научных расчетов в Воронеже, области и регионе. Лаборатория высокопроизводительных вычислений интегрируется с лабораторией дистанционного образования ВГУ на основе высокоскоростной сети доступа к ресурсам компьютерного кластера. На базе этих лабораторий организовываются универсальные учебно-исследовательские дисплейные классы для высокопроизводительных вычислений и разрабатывается программное обеспечение для научных исследований с использованием возможностей кластера и применения систем дистанционного и открытого образования.

На основе лабораторий высокопроизводительных вычислений и дистанционного образования сформирован новый учебно-научный комплекс для обеспечения специальности «Информационные системы и технологии» факультета компьютерных наук ВГУ. Организуется раздел электронной библиотеки ВГУ для информационной и методической поддержки работы комплекса.

Работа вычислительного кластера, генерирующего большое число данных, которые необходимо переместить с сервера на клиентские рабочие станции, развитие корпоративной сети, систем дистанционного и открытого образования ВГУ требует срочного решения вопросов развития университетской сети. Ее совершенствование будет осуществляться прежде всего в направлении создания средств для использования высокопроизводительной системы в удаленном режиме. С этой целью ставятся задачи улучшения качества коммуникаций и коммуникационного оборудования и обеспечения безопасности доступа к ресурсам. Исследуется возможность образования единого университетского модемного управляемого пула и построения сервера доступа

на базе специализированных операционных систем и блоков с встроенным модемным процессором с системой управления и анализа трафика.

Включение в состав сети университета высокопроизводительного вычислительного кластера дает возможность расширить в ближайшее время круг научных, информационных, корпоративных и учебных задач. Использование распределенных вычислений при построении информационных систем, мультимедийных библиотек, систем дистанционного и открытого образования приводит к необходимости постоянного развития аппаратной и программной составляющих сетевой инфраструктуры. Имеющаяся высокоскоростная магистраль объединяет оптоволоконными каналами коммуникационные узлы корпусов ВГУ и обеспечивает подключение к ней сетей вузов, других образовательных учреждений и научно–исследовательских институтов г. Воронежа.

В лаборатории высокопроизводительных вычислений разрабатываются новые методики применения многопроцессорных систем в фундаментальных научных исследованиях. Проводится изучение новых способов и подходов к решению ресурсоемких задач, анализ возможностей модификации численных методов для их использования на параллельных компьютерных устройствах. Создаются алгоритмы оптимизации распределения потока вычислений по компонентам кластера, методы оценки риска выбора неоптимальной с точки зрения результирующей производительности и отказоустойчивости конфигурации системы. Исследуется влияние числа используемых процессоров на производительность и надежность работы кластера при решении сложных задач.

В процессе освоения кластера проводится обучение пользователей – преподавателей, аспирантов и студентов – методам эффективной разработки параллельных программ. Формирование в кластере виртуальных специализированных вычислительных подсистем, структуры которых были бы адекватны решаемым задачам, рассматривается в качестве одной из важнейших составляющих этой научно–образовательной деятельности.

Работы по созданию высокопроизводительного вычислительного кластера ВГУ выполняются в рамках гранта VZ-010 (Научно–образовательный центр «Волновые процессы») при финансовой поддержке Американского фонда гражданских исследований и развития CRDF.

КЛАСТЕРНАЯ РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА ИТЕРАЦИОННОГО РАЗМЕЩЕНИЯ ОДНОГАБАРИТНЫХ ЭЛЕМЕНТОВ

Л.С. Курилов

Пензенский государственный университет

Современные средства автоматизированного проектирования в области радиоэлектроники предъявляют повышенные требования к вычислительным ресурсам системы проектировщика. Постоянный рост сложности электронных устройств, необходимость сокращения сроков разработки и улучшения качества проектных решений объективно способствует применению технологий параллельной и распределенной обработки для увеличения производительности САПР.

Кластерные системы, благодаря оптимальному соотношению «цена / производительность», являются, безусловно, наиболее приемлемой платформой для построения относительно недорогих высокопроизводительных вычислительных комплексов, направленных на решение задач САПР.

Описываемая реализация алгоритма автоматизации проектирования конструкторского этапа предназначена для использования при размещении компонентов на печатной плате либо ячеек СБИС на подложке кристалла и функционирует в среде распределенных вычислений под управлением специально разработанной кластерной оболочки ICEDOS. Программное обеспечение решения задачи автоматического итерационного размещения одногабаритных элементов методом парных перестановок состоит из конвертора стандартного формата PDIF во внутренний формат представления схемы и обратно, и собственно распределенного приложения, реализующего параллельный алгоритм размещения. Алгоритм, построенный на объектно-ориентированной основе, заключается в последовательности следующих шагов.

1. Чтение из файла матрицы смежности, описывающей мультиграф схемы; матрицы длин, описывающей расстояния между посадочными местами монтажно-коммутационного пространства в манхеттеновой метрике; матрицы размещения, описывающей первоначальный вариант размещения элементов (отображение множества вершин графа схемы, или элементов, во множество посадочных мест).

2. Определение конфигурации кластера, создание и перемещение объектов-вычислителей на рабочие узлы кластера.

3. Составление плана вычислений объектом-планировщиком для текущей одиночной малой итерации k (k = 1,…, n – 1) либо группы последовательных итераций $k + x$, где n – число элементов:

– разбиение вычислительной задачи текущей размерности в соответствии с количеством узлов N пропорционально прогнозируемой нормированной производительности каждого узла p:

$$Sw_k = \sum_{i=1}^{N} Sw_i = n - k\ ,\ \ Sw_i = p_i\, Sw_k$$

$$Sw = n\ (n - 1)\ /\ 2,$$

где: Sw_k – число парных перестановок для одной малой итерации, Sw – общее число перестановок для большой итерации;

– оценка степени гранулярности параллелизма, возможное совмещение нескольких малых итераций при неудовлетворительном объеме одной итерации ($k \to n$, при N – достаточно большом и малых значениях n), либо заключение о неэффективности кластеризации и расчет в одном локальном узле объекта-вычислителя;

– распределение заданий по узлам, запуск процесса вычислений.

4. Параллельный поиск наилучших вариантов перестановок элементов (i,j) объектами-вычислителями для текущей малой итерации или группы итераций:

$$\max \Delta F_{ij} = F_{ij}^{\ \text{до}} - F_{ij}^{\ \text{после}},\ \ \Delta F_{ij} > 0,$$

$$F_{ij} = \sum_{m=1}^{n} c_{im}\, d_{im} + \sum_{m=1}^{n} c_{jm}\, d_{jm},$$

где: $F_{ij}^{\ \text{до}}$ – суммарная длина связей элементов (i,j) со всеми остальными элементами до перестановки, $F_{ij}^{\ \text{после}}$ – суммарная длина связей элементов (i, j) после перестановки (элемент i занимает место элемента j и наоборот), c_{ij} – значение ячейки матрицы смежности, определяющее число связей элементов (i, j), d_{ij} – расстояние между посадочными местами элементов (i, j).

5) Сбор результатов объектом-планировщиком от всех объектов-вычислителей, анализ и выбор оптимального решения среди всех Sw_k для минимизации целевой аддитивной функции F (по критерию ми-

нимум суммарной длины связей), модификация распределенной матрицы смежности графа схемы во всех объектах.

$$\min F = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} d_{ij}.$$

В случае осуществления параллельных перестановок сразу для нескольких малых итераций, при необходимости переход к гиперграфовой модели, где смежность гиперребер в итерационных вершинах означает конфликты перестановок (т.е., в оптимальный вариант перестановок на последующих малых итерациях вовлечены вершины из оптимальных вариантов перестановок на предыдущих малых итерациях) - возможная ситуация представлена на рис. 1. Конфликты разрешаются за счет выбора одного из вариантов перестановок или исключения всех конфликтных вариантов из рассмотрения (при возможном снижении качества конечного решения).

6) Повторение шагов 3-5 большой итерации, пока $|F_i - F_{i-1}| > \varepsilon$, то есть, пока прирост сокращаемой суммарной длины связей для каждой новой большой итерации является существенным, либо повторение шагов 3-5 для заданного числа больших итераций.

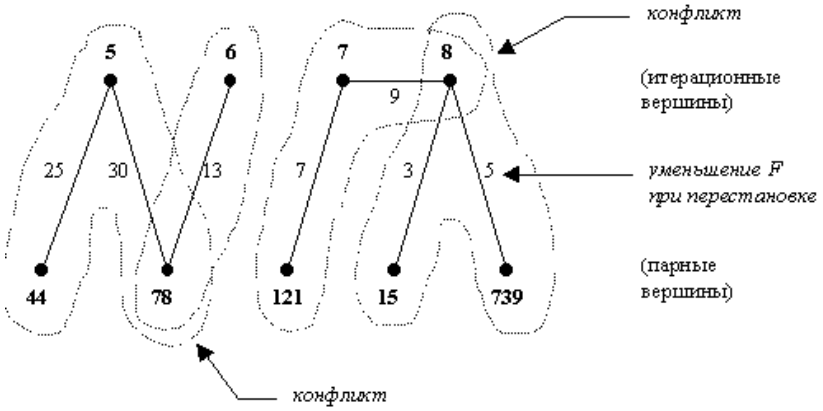


Рис. 1. Возможные конфликтные ситуации при одновременном расчете нескольких малых итераций

Эксперименты по исследованию работы параллельного алгоритма

итерационного размещения одногабаритных элементов проводились в три этапа путем оценки характеристик производительности распределенного приложения DAP (Distributed Auto Placer), реализующего алгоритм. Приложение было откомпилировано в системе Borland C++ Builder 5.0 под операционную платформу Win32. Протестированный вариант приложения для ОС Linux, откомпилированный с использованием стандартного компилятора GNU C++, показал неудовлетворительную скорость вычислений (примерно в 1,5 раза ниже, чем аналогичный код для платформы Win32), поэтому гетерогенные свойства приложения были исключены из рассмотрения.

Входными данными для программы DAP являлись несколько проектов различной размерности, представляющиеся в виде исходной схемы электрической принципиальной и первоначального варианта размещения. Монтажно-коммутационное пространство печатной платы было выбрано в виде регулярной сетки, количество посадочных мест соответствовало количеству элементов схемы. Размерность проекта определялась количеством элементов и цепей. Исследовались проекты со случайным размещением элементов и равномерным случайным распределением цепей по контактам элементов следующих размерностей (NxN – количество элементов, + C – количество цепей): 10x10=100 элементов + 500 цепей; 20x20=400 элементов + 2000 цепей; 30x30=900 элементов + 4500 цепей; 40x40=1600 элементов + 8000 цепей; 50x50=2500 элементов + 12500 цепей. В качестве элемента был выбран корпус микросхемы К155ЛА3 с типоразмером DIP14. Шаг на сетке посадочных мест был выбран равным 1 дюйм (2,54 см).

1 этап заключался в исследовании основных характеристик работы алгоритма по сравнению с аналогичным алгоритмом в известном пакете автоматизации конструкторского проектирования Accel EDA (версия системы PCAD под Windows).

Условия проведения эксперимента:

1. Техническое обеспечение – персональный компьютер с процессором K6-300 МГц, объемом ОЗУ 64 Мбайт.

2. Программное обеспечение – ОС Windows 95 (стандартная установка); пакет Accel EDA v.14.00.46, подсистема автоматического размещения и трассировки Specctra v.7.0.3, итерационное размещение компонентов методом парных перестановок Interchange; программа DAP (локальный режим) с использованием среды ICEDOS.

3. Измеряемые характеристики - время работы и качество полученного решения на фиксированном количестве больших итераций

(параметрами качества являются: процентное отношение суммарной длины связей после размещения к суммарной длине связей первоначального варианта размещения, процент трассируемости связей при последующей за размещением автоматической трассировке).

Результаты экспериментов для 1 этапа при количестве больших итераций, равном 7, выявили следующие факты. Процент трассируемости связей для вариантов размещения, полученных с помощью Accel и DAP оказался практически одинаков (трассировка производилась в подсистеме PRO Route Accel PCAD PCB). Далее, алгоритм DAP на платформе среды ICEDOS, по сравнению с аналогичным алгоритмом, реализованным в Accel EDA, обеспечивает в локальном режиме ускорение в десятки раз (10-40), в зависимости от размерности проекта, при лучшем на 2-3% качестве размещения. Эксперименты для сравнительного анализа работы алгоритмов над проектами с числом элементов больше 2500 не проводились в связи с ограничением максимального размера печатной платы в пакете Accel. (Заметим, что в DAP подобные ограничения практически отсутствуют).

2 этап имел своей целью установить характеристики масштабируемости алгоритма DAP в параллельном режиме работы.

Условия проведения эксперимента:

1. Техническое обеспечение – однородный кластер из 1…7 узлов на базе персональных компьютеров с процессором Celeron-433 МГц, объемом ОЗУ 64 Мбайт, объединенных локальной сетью Ethernet 10 Мбит/с (стандарт 10Base-T, витая пара, концентратор).

2. Программное обеспечение – ОС Windows 98 (стандартная установка); программа DAP на платформе среды ICEDOS.

3. Измеряемые характеристики – зависимость времени работы алгоритма от количества узлов кластера (распределенных ресурсов).

Результаты экспериментов для 2 этапа (число больших итераций = = 3) представлены на рис. 2 и 3.

Выявленные зависимости показывают, что алгоритм обеспечивает хорошую масштабируемость (на 3 узлах достигается ускорение примерно в 2,5 раза, на 5 узлах – в 4 раза, на 7 узлах – почти в 5 раз), и чем выше размерность проекта и чем больше распределенных ресурсов, тем более эффективен алгоритм.

Следует также отметить некоторые факторы, оказывающие влияние на изменение потенциальной производительности приложения, реализующего алгоритм. Прежде всего, возможно проявление эффекта нелинейного замедления работы с ростом размерности проекта за счет

операций свопинга на диск, связанных с увеличением требуемого объема виртуальной памяти. Эффект устраняется путем наращивания ОЗУ во всех узлах кластера. Второй фактор, заключающийся в том, что при увеличении числа активных узлов для некоторых сетевых сред уменьшается полоса пропускания, непосредственно влияет на коммуникационную составляющую функционирования приложения. Незначительное падение производительности, вызванное воздействием данных факторов, было отмечено в ходе проведения экспериментов.

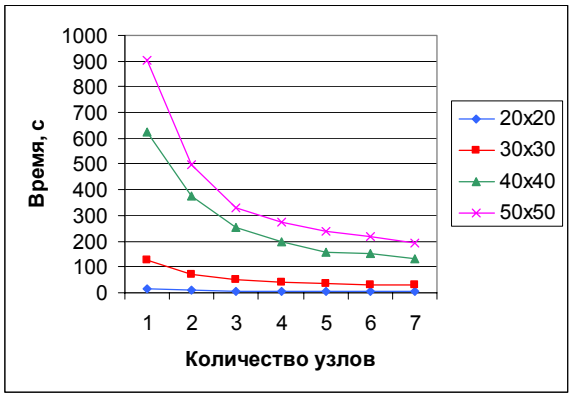


Рис. 2. График зависимости времени работы DAP от количества узлов кластера

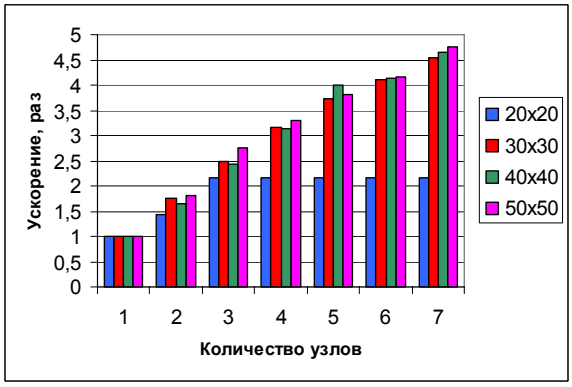


Рис. 3. Гистограмма зависимости ускорения вычислений DAP

от количества узлов кластера

3 этап был направлен на определение эффективности применения разработанной среды распределенных вычислений. Условия проведения эксперимента аналогичны предыдущему этапу, за исключением того, что исследовался один и тот же алгоритм DAP, реализованный на платформе ICEDOS и с использованием одной из библиотек стандарта MPI. В качестве конкурирующей среды был выбран пакет WMPI 09.b (версия под Win32, совместимая с MPICH 1.0.13), как наиболее известный и доступный вариант реализации MPI для Windows, обладающий хорошими скоростными характеристиками.

Для исследования были отобраны проекты средней размерности (30x30 и 40x40) вследствие наибольшей информативности, поскольку на проектах малой размерности выигрыш в скорости не заметен, а на проектах большой размерности с ростом объема вычислений алгоритма доля коммуникационной составляющей уменьшается и преимущества среды ICEDOS также не очевидны (что говорит о высокой степени локальности алгоритма DAP). Анализ полученных временных зависимостей показал, что на достаточном количестве узлов кластера время работы DAP при использовании платформы ICEDOS сокращается и обеспечивается ускорение примерно в 1,5-2 раза по сравнению с вариантом DAP на базе библиотеки передачи сообщений WMPI.

Таким образом, проведенные исследования подтверждают обоснованность и целесообразность использования систем параллельных вычислений на базе кластерных технологий для задач САПР и эффективность представленного алгоритма.

Литература

1. *Бершадский А.М.*, *Курилов Л.С.*, *Селиверстов М.Н.* Построение систем параллельной обработки на базе кластерных технологий // Теоретические и прикладные вопросы современных информационных технологий: Матер. Всероссийской конф. / Улан-Удэ. 2000. С. 40–44.
2. *Бершадский А.М.*, *Курилов Л.С.*, *Селиверстов М.Н.* Балансировка загрузки в кластерных системах // Телематика'2000. Тез. докл. Международ. науч.-методич. конф. / Санкт-Петербург. 2000. С. 124.
3. *Бершадский А.М.*, *Курилов Л.С.*, *Селиверстов М.Н.* Применение кластерных технологий в САПР // Информационные технологии. 2001, №9. С. 2–6.

4. *Курилов Л.С.* Разработка гетерогенной объектной кластерной среды для автоматизированного проектирования распределенных приложений. – Автореферат диссертации на соискание ученой степени кандидата технических наук. Воронеж, 2002.

СИСТЕМА УДАЛЕННОГО ДОСТУПА К ВЫЧИСЛИТЕЛЬНОМУ КЛАСТЕРУ (МЕНЕДЖЕР ДОСТУПА)*

Д.Ю. Лабутин

Нижегородский государственный университет им. Н.И. Лобачевского

С каждым годом увеличивается мощность компьютеров, но очевидно, что классическая фон-неймановская организация вычислений уже не обеспечивает требуемой производительности в некоторых задачах. Это обусловлено как физическими ограничениями (скорость света), так и экономическими. Поэтому стимулируется развитие параллельных вычислений. Тот факт, что параллельные вычисления до сих пор не получили должного распространения, объясняется техническими трудностями (необходимость высокоскоростных каналов связи между вычислительными комплексами или процессорами), алгоритмическими трудностями (необходимость разработки алгоритмов, учитывающих параллелизм), и, в особенности, отсутствием неких стандартных подходов и программных средств для организации параллельных вычислений.

С другой стороны, просто купить и установить вычислительный кластер вовсе не достаточно. Можно даже настроить программное обеспечение, необходимое для проведения вычислительных экспериментов. Но и при этом эффективно использовать кластер не получится. Вот, основные проблемы, которые при этом возникают:

– возможные конфликты в процессе запроса вычислительных мощностей во время проведения экспериментов (может случиться так, что во время запуска приложения на определенных компьютерах, на них уже исполняются задачи других пользова-

* Работа выполнена в рамках Федеральной Целевой Программы «Интеграция»

телей);
- необходимость личного присутствия разработчика на вычислительной площадке во время запуска экспериментов.

Для решения проблемы «конфликтов» разрабатывается центральная система управления вычислительным кластером. Благодаря этому все запросы на исполнения программ будут обрабатываться системой управления, а вычислительные мощности распределяться согласно запросам разработчиков программ и других условий.

Для решения проблемы личного присутствия разработчика на вычислительной площадке было решено использовать уже практически ставший стандартным подход – управление через Internet. Разрабатывается Web-интерфейс, который взаимодействует с описанной выше центральной системой управления вычислительным кластером. Благодаря популярному подходу к дистанционному управлению различными устройствами подключенными к всемирной компьютерной сети с помощью обычного Web-броузера, разработчик сможет производить запуск экспериментов с любого компьютера, подключенного к Internet'у, независимо от установленной на нем операционной системы.

Web-интерфейс разрабатывается как отдельный независимый модуль системы управления кластером. Взаимодействие с центральной системой производится через механизм сокетов (socket). Для этого был разработан специальный протокол обмена данными. Благодаря такому подходу можно будет разработать и stand-alone приложение, с помощью которого можно выполнять те же действия, что и через Web-интерфейс.

Основные функциональные возможности менеджера доступа

Аутентификация пользователей при работе с Web-интерфейсом управления вычислительным кластером. Каждый пользовать в начале работы должен ввести «Имя пользователя» и «пароль», известные только ему. После этого можно однозначно сопоставлять производимые действия на кластере с конкретным пользователем.

Выделение независимого файлового пространства. Каждому пользователю выделяется индивидуальное место для хранения файлов, необходимых для проведения вычислительных экспериментов. При этом накладываются определенные ограничения на хранимые файлы (максимальный суммарный размер хранимых файлов, максимальный размер каждого файла в отдельности), индивидуальные для каждого

пользователя системы.

Система задач. После того как пользователь загрузил необходимые файлы для проведения вычислительных экспериментов, он создает именованные задачи, т.е. задает необходимые параметры (имя задачи, исполняемый модуль, приоритет, имя файла с результатами, необходимое количество вычислительных мощностей, параметры командной строки для исполняемого модуля). Затем пользователь может ставить задачу в очередь на исполнения неоднократно, не утруждая себя повторным заданием параметров. В случае необходимости владелец задачи может снять ее с исполнения или убрать из очереди.

Мониторинг. После того, как одна или более задач поставлены в очередь на исполнение, пользователь может посмотреть их состояние (выполняется, ожидает выполнения в очереди, выполнилась). Так доступна информация о загруженности вычислительного кластера в целом.

Протоколирование действий. Все производимые пользователем системы действия протоколируются. Анализируя протокол можно определить, какие системные ресурсы выделяются тому или иному пользователю.

Администрирование. Так же имеется интерфейс работы с системой, предназначенный только для администраторов, где они могут производить такие действия, как добавление и удаление пользователей системы, изменение параметров, задающих ограничения на хранимые файлы, контроль хранимых пользователями файлов и т.п.

Демонстрация

На данный момент разработан пилотный вариант системы, который позволяет загружать файлы, необходимые для проведения экспериментов, на сервер и выполнять одиночные эксперименты. Данный вариант будет продемонстрирован на семинаре.

Литература

1. *Гергель В.П.*, *Стронгин Р.Г.* Высокопроизводительный вычислительный кластер Нижегородского университета. // Труды Международной научной конференции «Телематика 2001». СПб.: Изд-во СПбГИТМО, 2001. С. 160–161.
2. Общий обзор систем управления кластером. (http://nhse.cs.rice.edu/NHSEreview/CMS/).
3. *Barker, V.* (Ed.) Cluster Computing Whitepaper at http://www.dcs.port.

ac.uk/~mab/tfcc/WhitePaper/. 2000.

РАСПАРАЛЛЕЛИВАНИЕ ГЕНЕТИЧЕСКОГО АЛГОРИТМА ПРИ РЕШЕНИИ ЗАДАЧИ АВТОМАТИЧЕСКОГО ПОИСКА ПЕРСПЕКТИВНЫХ МОЛЕКУЛЯРНЫХ СТРУКТУР

В.И. Литвиненко, В.П. Кругленко, Ю.А. Бюргер

Херсонский государственный технический университет, Украина

Введение

Традиционные лазерные технологии базируются на применении в качестве активного элемента прозрачных кристаллов, где в качестве активного элемента прозрачных кристаллов, где в качестве активных центров применяются примеси из ионов различных металлов. Однако эти лазеры имеют один существенный недостаток – они не могут изменять диапазон излучения. Для преодоления этого недостатка были разработаны лазеры на жидких красителях. В них в качестве активных элементов применяются жидкие растворы красителей, что позволяет регулировать диапазон излучения. По этой причине, перед химиками стоит задача синтеза новых химических соединений с такими значениями молекулярных дескрипторов, при которых значение КПД генерации в диапазоне 500–600 нм превышает 50% [2, 3]. Ранее нами было показано [4], что совместное применение методологий генетических алгоритмов, молекулярного моделирования и нейронных сетей способно обеспечить получение молекулярных структур класса имитринов с высоким значением КПД. Была разработана гибридная система, позволяющая создавать структурные формул имитринов при помощи генетических алгоритмов и прогнозировать их КПД. Однако мы столкнулись с проблемой вычислительной сложностью в данной работе. По этой причине, целью данной работы является разработка параллельных эволюционных алгоритмов для проектирования молекулярных структур с заданными физико-химическими свойствами.

Применение генетического алгоритма для автоматического поиска перспективных молекулярных структур

Генетический алгоритм является разновидностью эволюционного эвристического алгоритма, и принцип его работы основан на моделировании таких процессов, происходящих в природе, как естественного отбор, размножение и мутация [1]. Каждая из возможных комбинаций

радикалов, однозначно определяющих структуру молекулы исследуемого класса, рассматривается алгоритмом как «особь». Структура особи, в которой «кодируется» ее форма и содержание, называется «хромосомой». На первой итерации формируется популяция особей, хромосомы которых составлены случайным образом. Каждая особь популяции получает некоторую оценку, характеризующую приспособленность особи к окружающей среде (целевая функция задачи). Затем для получения следующего поколения применяются операторы «кроссинговера» (генерация новой особи из генетического материала двух других особей) и «мутации» (случайное изменение генотипа особи). Затем оценивается приспособленность каждой новой особи. Лучшие особи отбираются для формирования следующего поколения. Далее, если не выполняется условие завершения поиска, алгоритм повторяется с фазы «воспроизводства».

Согласно поставленному условию «перспективности» исследуемых молекулярных структур, целевая функция рассматриваемой задачи должна соответствовать функции КПД лазерной генерации вещества. Формально, поиск необходимых молекулярных структур определяется как оптимизация целевой функции:

$$\max F(x_1,..,x_n), \qquad (1)$$

где $(x_1,..,x_n)$ – такие теоретические расчетные параметры молекулы, как полная энергия молекулы.

Процесс оценивания особи в рассматриваемой задачи состоит из следующих этапов:

- декодирование хромосомы особи;
- восстановление структуры молекулы;
- расчет теоретических параметров молекулы;
- вычисление функции оценки.

Применение генетического программирования для определения функции оценки на основании экспериментальных данных

Неотъемлемой частью любого генетического алгоритма является вычисление значения пригодности каждого индивидуума популяции. Пригодность содержит некоторую информацию о качестве индивидуумов. Цель алгоритма генетического программирования [5] – «выведение» программы, максимально аппроксимирующей заданную кривую. В контексте поставленной в данной работе задачи цель проектирования может быть определена как задача автоматического поиска

функции с последующей её оптимизацией.

При автоматическом поиске, в качестве критерия оценки особей может быть использовано среднее квадратичное отклонение, рассчитываемое по формуле:

$$Q = \sum_{i=1}^{N} \frac{(x_i - y_i)^2}{N}, \qquad (2)$$

где N – длина обучающей выборки; x – требуемое значение искомой функции; y – значение, рассчитанное программой.

Специфика генетических алгоритмов такова, что при их помощи можно оптимизировать многопараметрическую функцию $f(x_1,x_2,..,x_n)$ выполняя расчеты не самой функции, а некоторой $f^*(x_1,x_2,..,x_n)$, «условно параллельной» оптимизируемой функции. Условно параллельными называются такие две функции, в которых распределения элементов одной из них можно судить о характере распределения элементов другой. Для существования таких функций достаточно, чтобы выполнялись условия, при которых $f(X_i)\ R\ f(X_j)$ соответствует $f^*(X_i)\ R\ f^*(X_j)$, где X_i и X_j – различные комбинации вектора входных параметров, а символ R – отношение следования больше, меньше и равно. Легко заметить, что для таких функций линейная корреляция будет равна единице. Если в массе однородных индивидов определенному значению одного признака, рассматриваемого в качестве аргумента, соответствует не одно и тоже числовое значение, а целая гамма распределяющихся в вариационный ряд числовых значений другого признака, рассматриваемого в качестве зависимой переменной, или функции, то такого рода зависимость между переменными величинами называют корреляционной или корреляцией. Таким образом, в задачах автоматического поиска функций с целью их оптимизации можно использовать любую из условно параллельных функций, замещая ей искомый оригинал. Это существенно расширяет область приемлемых решений при поиске функции. При таком подходе в качестве критерия оценки особей в генетическом программировании можно использовать коэффициент корреляции. Осуществить полный перебор по пространству возможных решений в данной задаче не представляется возможным. Однако существует возможность значительного сокращения пространства перебора путем применения эвристик и некоторых предположений о поверхности ответа. Нами был разработан метод автоматического определения функций посредством применения частичного перебора рекурсивных композиций простейших операций и функций

из заданного множества, с эвристическим отбором на основе эволюционного подхода. Этот метод основан на предположении о существовании условно параллельных функций, степень сопряженности которых оценивается коэффициентом корреляции и характеризуется его высоким положительным значением. При этом поиск осуществляется посредством последовательного наращивания глубины дерева функции, имеющей на данный момент максимальную корреляционную оценку. Схема алгоритма аналогична схеме генетического алгоритма . На этапе инициализации происходит заполнение поколения простейшими особями, состоящими из корней деревьев функций. Каждая из таких особей рассчитывает простейшую операцию или функцию из заданного множества над входными параметрами задачи. При этом размер популяции фиксирован, и задается при конфигурировании алгоритма. На следующем шаге выполняется оценка особей при помощи коэффициента корреляции. Цель этой оценки – определить, на сколько сильно сопряжена каждая особь с искомой функцией. После оценки производится срез особей по величине корреляции. Аналогичная операция выполняется и в генетических алгоритмах при использовании элитарного отбора особей. Таким образом, корреляционный срез играет роль эвристики, позволяя сосредоточить внимание на перспективных особях. Величину порога среза следует брать равной максимальной оценке особи предыдущего поколения. Если после выполнения корреляционного среза не осталось ни одной особи, то алгоритм следует завершить, а в качестве искомой функции принять дерево особи из предыдущего поколения с максимальной оценкой. На следующем шаге выполняется генерация нового поколения и переход на этап оценки особей и выполнения корреляционного среза. При этом генерация нового поколения подразумевает последовательное применение простейших операций и функций из заданного множества ко всем отобранным особям. Таким образом, происходит наращивание глубины дерева функции. Методы генетического программирования и генетических алгоритмов наиболее подходят для решения таких задач, где обычный математический анализ не делает или не может обеспечить аналитические решения, где трудно понять взаимосвязи среди переменных, при этом приемлемо приближенное решение и существует большое количество данных, которое требует экспертизы. Особо следует выделить те оптимизационные задачи, в которых частью самой задачи является обнаружение оптимизационной функции. При этом, как было показано, существует достаточно высокая вероятность эффективного приме-

нения метода корреляционных срезов для создания популяции в генетическом программировании.

Распараллеливание генетического алгоритма

Эксперимент показал, что предложенный метод обладает достаточной эффективностью в организации автоматического поиска перспективных молекулярных структур, но требует значительной вычислительной базы. Так, при числе итераций 1 тысяча, после чего алгоритм достигает «эффективного порога», то есть вступает в фазу генерации новых молекулярных структур с прогнозируемым КПД выше лучшего из экспериментальной выборки, время работы алгоритма на компьютере Pentium-II составляет более 7 суток (размер популяции – 128 особей, каждая из которых описывается вектором из 4-х точек).

Наиболее ресурсоемкой частью рассматриваемого подхода является оценка КПД лазерного выхода молекулы. Для прогнозирования КПД молекулярной структуры используется сложная многопараметрическая функция $F(X)$, где X – множество исходных данных, определяемых на основании квантово-химических, полуэмпирических и других молекулярных расчетов.

Для проведения молекулярных расчетов используются такие программы, как HyperChem (trial-версия). Используемые пакеты позволяют выполнять расчеты в режиме «клиент-сервер», что делает возможным реализовать параллельную обработку данных, при которой происходит одновременных расчет нескольких молекулярных структур.

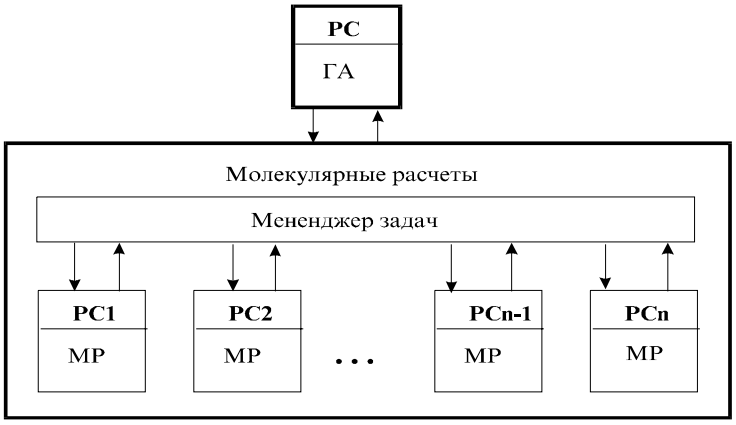


Рис. 1 Схема распараллеливания генетического алгоритма

Для реализации этой схемы необходимо $N + 1$ компьютеров, объединенных сетью. Компьютер PC занят основным циклом генетического алгоритма, где выполняется скрещивание и мутация особей, а также формирование запросов на молекулярные расчеты. Компьютеры PC_1-PC_n выполняют молекулярные расчеты. Распределением нагрузки на эти компьютеры занимается «менеджер задач», который физически относится к PC. Менеджер управляет очередью запросов на молекулярные расчеты и следит за загруженностью PC_1-PC_n. Как только один из этих компьютеров освобождается, менеджер выбирает запрос из очереди и передает его свободному компьютеру. Такое управление является целесообразным, так как временные характеристики каждого отдельного расчета молекулы индивидуальны, а значит, реализация синхронной пакетной обработки не имеет смысла.

Параллельная обработка запросов возможна только внутри одного цикла генетического алгоритма. В конце каждого такого цикла необходимо реализовать ожидание освобождения всех PC1-PCn, что соответствует завершению оценивания всех особей текущей популяции.

Распараллеливание генетического алгоритма и генетического программирования

В рассматриваемой задаче функция оценки молекул, или целевая функция генетического алгоритма, не является статической. Она определяется генетическим программированием на основании некоторой экспериментальной выборки молекул с их конфигурацией и оценкой КПД лазерной генерации (задача автоматического определения функции). Назовем эту выборку «базой знаний» алгоритма генетического программирования. Так как результат работы алгоритма генетического программирования не гарантирует абсолютное восстановление функции, то целесообразным будет реализовать параллельную работу генетического алгоритма (поиск перспективных молекул) и генетического программирования (поиск целевой функции генетического алгоритма). При этом определяющей будет возможность проведения экспериментальной оценки предложенных алгоритмом молекул и пополнения базы знаний алгоритма определения целевой функции. Это позволяет модифицировать функцию оценки молекулярных структур «на лету».

В описанную выше схему реализации параллельной работы генетического алгоритма необходимо внести некоторые изменения. В об-

щее число компьютеров, задействованных в расчетах, необходимо добавить еще один компьютер, на котором будет исполняться задача автоматического определения целевой функции генетического алгоритма. Эта задача автоматически обновляет функцию оценки молекулярных структур, которая используется задачей поиска перспективных молекул. Такая реализация позволяет инициировать поиск молекул уже на этапе формирования целевого критерия алгоритма поиска, что делает возможным более раннюю генерацию молекулярных структур, качество которых улучшается по ходу приближения оценочной функции к искомой функции КПД.

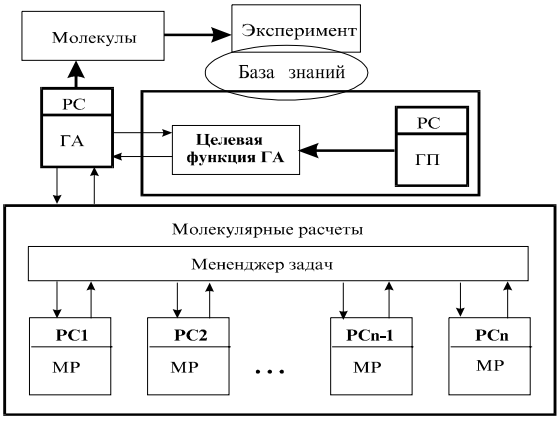


Рис.2 Распараллеливание генетического алгоритма и генетического программирования

Выводы

В данной работе разработана гибридная система на основе генетического алгоритма, генетического программирования и нейронных сетей для синтеза молекулярных структур с заданными физико-химическими свойствами. Реализована схема параллельной работы генетического алгоритма. В эксперименте было использовано 10 компьютеров, соединенных в сеть: сервер – Pentium II 300 Мгц (2 процессора), 9 машин – Celeron 400 Мгц. Выигрыш по сравнению с нераспараллеленой системой ≈ в 7 раз. Имеет смысл увеличить число компьютеров, задействованных под молекулярные расчеты вплоть до 90-95% от К, где К – длина популяции.

Литература

1. *Holland, J.H*. (1975) Adaptation in Natural and Artificial Systems. Ann Arbor, Michigan Press, 1975.
2. *Кругленко В.П.*, *Марончук И.Е.*, *Повстяной М.В.* Технология изготовления новых лазерных красителей класса имитринов с излучением в диапазоне 454-618 нм // Современные информационные электронные технологии: Тр. II международ. научн. конф. / Одесса, 28-31 мая 2001. С. 258–259.
3. *Кругленко В.П.*, *Повстяной М.В.*, *Стоилов Ю.Ю.* Имитрины VII. Лазерные красители класса имитринов работающие в диапазоне 472-581 нм с КПД генерации 30-54% / Вестник ХГТУ, №3(9) 2000. С. 69–71.
4. *Литвиненко В.И.*, *Кругленко В.П.*, *Бюргер Ю.А.* Направленный поиск химических структур с заданными свойствами при помощи генетического алгоритма / Вестник ХГТУ, №1(10). 2001. С. 53–55.
5. *Koza, J.R.* (1992), "Genetic programming: On the Programming of Computers by means of Natural Selection", Cambridge, MA: MIT Press.

РЕАЛИЗАЦИЯ ПОДСИСТЕМЫ МОНИТОРИНГА СОСТОЯНИЯ КЛАСТЕРОВ ПОД УПРАВЛЕНИЕМ ОС СЕМЕЙСТВА WINDOWS NT*

И.В. Лопатин, А.Н. Свистунов

Нижегородский государственный университет им. Н.Лобачевского

Системы управления являются критичным компонентом современных высокопроизводительных кластеров, позволяя эффективно распределять ограниченные ресурсы сети компьютеров между множеством одновременно запускаемых задач. Традиционно управляющие программы разрабатывались для векторных суперкомпьютеров, однако в дальнейшем, с ростом роли систем с распределеной памятью, появлялись новые средства мониторинга и менеджмента, рассчитанные

* Работа выполнена в рамках Федеральной Целевой Программы «Интеграция»

на использование в сетевой среде. В качестве примеров можно привести такие широко распространенные средства управления кластером как CCS, LSF, PBS и другие [1].

Типичная система мониторинга и управления состоит, как правило, из следующих частей [2]:

- компонент, взаимодействующий с пользователем (User Server). Позволяет манипулировать с очередью заданий и просматривать их статус на кластере;
- компонент, оперирующий с очередью заданий (Job Scheduler). Распределяет задания по узлам кластера и ставит их в (приоритетную) очередь;
- компонент, обеспечивающий мониторинг кластера и непосредственное выделение ресурсов (Resource Manager).

Каждый компонент, в свою очередь, может быть разбит на составные части более узкой функциональности. Например, можно разделить задачи мониторинга и выделения ресурсов на кластере.

В данной работе рассматривается возможность удаленного контроля состояния компьютеров кластера с установленной на узлах Windows 2000 и снятие с них информации о производительности. Для решения этих задач были использованы стандартные средства, интегрированные в ОС семейства Windows NT. В последних версиях этих операционных систем для контроля производительности и сбора информации о системных событиях был разработан Windows Management Interface (WMI, Интерфейс Управления Windows), реализующий усовершенствованную поддержку управления системой средствами ОС. WMI был разработан специалистами Microsoft на базе открытого стандарта управления предприятием через Web - Web-Based Enterprise Management (WBEM).

Разработчики реализовали поддержку WMI в Windows 98 и в Windows 95 OSR2 (устанавливается отдельным пакетом), сделали его доступным для NT 4.0, начиная с Service Pack 4 (SP4), и, наконец, интегрировали в Windows 2000. В основной степени WMI предназначен для использования в системах Windows 2000 и Windows XP.

Для изучения возможностей распределения задач на кластерных системах был разработан экспериментальный прототип системы управления и мониторинга, использующий WMI для отслеживания состояния узлов. Программа была опробована на вычислительном кластере нижегородского университета.

Экспериментальная система NetWeather в процессе работы произ-

водит опрос выбранных узлов кластера с задаваемой пользователем периодичностью. Тестирование показало, что для 5-6 компьютеров, соединенных сетью 100 Мбит и находящихся в одном домене, достаточно 30 секундного периода опроса. Это значение выставлено в поле по умолчанию. Как правило, задачи, запускаемые на кластерах, исполняются достаточно долго, поэтому относительно большая дискретность опроса слабо влияет на работу планировщика.

На основании информации, полученной в результате опроса сети, реализованный в системе простейший планировщик упорядочивает узлы по загрузке процессора и выбирает из них наименее загруженные. При этом учитывается наличие многопроцессорных узлов в сети и на свободные машины направляется количество потоков, пропорциональное числу процессоров. Генерируется конфигурационный файл и стандартная утилита запуска «mpirun.exe», входящая в состав дистрибутива MPI, вызывается с этим файлом в качестве параметра. Запуск каждой задачи происходит в отдельном потоке, чтобы не блокировать мониторинг кластера и не препятствовать постановке других программ на выполнение.

Несмотря на то, что многие элементы системы управления кластером на базе Windows2000 были реализованы в экспериментальной программе, остается множество отрытых вопросов и возможностей для улучшения.

Тестирование приложения на кластере Нижегородского Университета показало, что оптимальным числом узлов является 4-6. При увеличении количества опрашиваемых узлов достаточно ощутимо увеличивается как время первоначального установления соединения, так и время опроса, то есть приходится задавать более длинную паузу между вызовами опрашивающей функции. Таким образом, для получения актуальной информации необходим механизм, отличный от использованного в эксперименте. Возможным путем ускорения опроса является применение высокоскоростного интерфейса WMI (High Performance API), позволяющего пересылать данные по сети без накладных расходом на обращение с COM-объектами.

Для проверки возможностей WMI High Performance API была написана тестовая утилита, работающая в текстовом режиме и с заданной частотой пытающаяся получить данные о загрузке процессора и количестве свободной оперативной памяти с удаленного компьютера под управлением Windows2000. Эксперименты в локальной сети с пропускной способностью 100Мбит показали, что такой опрос без

проблем выполняется с частотой 0.5 секунд для одного узла. Такого периода опроса вполне хватает для отражения состояния кластера. Время первоначального соединения с удаленным компьютером также оказалось удовлетворительным (примерно 1-3 секунды для каждого узла).

В настоящее реализована и проходит тестирование специальная библиотека мониторинга, которая будет включена в разрабатываемую в Нижегородском Университете компонентную экспериментальную систему управления кластером.

Важным аспектом, на который следует обратить внимание при разработке приложений, контролирующих состояние кластера, является сохранение внутренней безопасности сети. При установках ОС по умолчанию пользователь, осуществляющий мониторинг удаленной машины, должен принадлежать к группе администраторов, что в большинстве случаев нежелательно. Однако Windows 2000 и Windows XP предоставляют возможность настройки удаленного доступа к WMI для пользователей, не являющихся администраторами системы.

Другой важной функцией компоненты-монитора в будущем также может стать предупреждение администратора о возможных неполадках на кластере и, возможно, встроенные аналитические средства, позволяющие на основе собранных статистических данных прогнозировать поведение системы на определенных типах вычислений. Также необходимо периодически проверять доступность удаленных компьютеров в сети, в противном случае планировщик заданий может попытаться разместить систему на выключенном или зависшем узле.

На основании проведенных экспериментов можно заключить, что в настоящее время системы на базе Windows имеют в своем составе развитые возможности для организации эффективного мониторинга используемых ресурсов. В комплексе со средствами поддержки параллельных вычислений (различные реализации MPI) это позволяет достаточно быстро строить простые системы управления вычислительным кластером на базе операционных систем семейства NT. Создание своих программ управления позволяет проводить исследования по оптимизации размещения задач на узлах кластера. Следует отметить, что возможность реализации таких приложений на базе Windows позволяет говорить об этой ОС как о вполне конкурентноспособной альтернативе UNIX-системам для выполнения параллельных высокопроизводительных вычислений.

Литература

1. *Baker M.*, *Fox G.*, *Yau H.W*. A Review of Commercial and Research Cluster Management Software. Cluster Computing Review, 1996.
2. *Saphir W.*, *Tanner L.A.*, *Traversat B.*. Job Management Requirements for NAS Parallel Systems and Clusters. NAS Technical Report NAS-95-006 February 95.
3. *Keller A.*, *Reinefeld A.* Anatomy of a Resource Management System for HPC Clusters. Annual Review of Scalable Computing, V. 3, 2001.
4. *Wolski R.*, *Spring N.T.*, *Hayes J.* The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing.

КОНЦЕПЦИЯ КЛАСТЕРА ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ И РЕАЛИЗАЦИЯ В СООТВЕТСТВИИ С НЕЙ VPN-ШЛЮЗА И АНАЛИЗАТОРА СЕТЕВОГО ТРАФИКА

А.П. Лубанец, В.С. Заборовский

СПбГПУ, ЦНИИ РТК

Введение

С ростом скоростей передачи данных современных телекоммуникационных технологий увеличивается актуальность обработки высокоскоростных сетевых транзакций с обязательным учетом требований по информационной безопасности. Такие классы задач как шифрование, фильтрация, маршрутизация, анализ и идентификация трафика неизбежно приводят к решению проблемы адекватности этих типов обработки скорости их совершения. Другими словами, с появлением новых технологий должна расти производительность устройств, участвующих в обработке сетевого трафика.

Наиболее емкие вычислительные задачи, такие как криптографическая обработка, анализ и идентификация могут решаться при помощи высокопроизводительных параллельных вычислительных систем – кластеров. Используя такой подход, можно получить VPN-шлюз способный обработать любое заданное количество шифрованных соединений (задача криптографической обработки), сетевой анализатор, способный выдавать состояние сильнозагруженной сети за требуемый интервал времени (задача анализа) или реализовать систему обнаружения вторжений, выявляющую не только сигнатуры атак, но и спо-

собную выявлять любые нештатные ситуации, возникающие в коммуникационных сетях (задача идентификации).

Концепция кластера информационной безопасности

Все вышеприведенные задачи логично формулируются в рамках единой концепции кластера информационной безопасности (далее КИБ). КИБ – это набор масштабируемых технических средств с определенным набором функций (шифрование, фильтрация, маршрутизация, анализ и идентификация), обеспечивающих комплексную защиту информации в компьютерных сетях.

Ключевой особенностью концепции является утверждение о том, что любая обработка сетевого трафика обладает внутренним параллелизмом и может быть решена при помощи высокопроизводительных параллельных вычислительных систем. Физически КИБ представляет собой набор устройств выполняющих те или иные задачи по обработке сетевого трафика с одинаковой скоростью. Каждое из таких устройств подразумевает настройку производительности в соответствии с предъявляемыми требованиями путем его кластеризации, которая может реализовываться двумя способами: либо посредством физического объединения двух или более (в зависимости от потребности) одинаковых устройств, либо подключение устройства, нуждающегося в вычислительных ресурсах, к выделенному вычислительному кластеру.

Все устройства, входящие в состав КИБ, могут управляться как из единого центра по выделенной для этой цели сети, так и по отдельности. Конфигурация КИБ динамична и в каждом конкретном случае ориентируется на удовлетворение потребности конечного пользователя в комплексном решении задачи обеспечения информационной безопасности в компьютерных сетях.

Базисным элементом каждого устройства КИБ является сетевой процессор, который в зависимости от выполняемой задачи (фильтрация, маршрутизация, криптографическая обработка, анализ, идентификация) имеет ту или иную конфигурацию программно-аппаратных модулей. Основной особенностью данного устройства является его прозрачность относительно внешнего окружения, т.е. данное устройство ничем не обнаруживает своего присутствия в сети.

В рамках исследования физической реализации концепции КИБ были спроектированы и реализованы пилотные образцы устройств, в задачу которых входили параллельная криптографическая обработка сетевого трафика и параллельная обработка журналов регистрации межсетевого экрана. Первое устройство является параллельным VPN-

шлюзом и представляет собой программу, работающую на MPI-кластере, второе – параллельным анализатором сетевого трафика и реализуется поверх TCP/IP как среда параллельной обработки журналов межсетевого экрана.

Параллельный VPN-шлюз

Архитектура VPN-шлюза (далее по тексту «шлюз») подразумевает второй вариант кластеризации, т.е. когда к устройству подключается внешний высокоскоростной параллельный вычислитель общего назначения, который в данном контексте удобно обозначить как «вычислитель КИБ» или ВКИБ. В его задачу входит обслуживание всех компонент КИБ, которым требуется вычислительная мощность для выполнения своего функционального назначения с заданными параметрами обслуживания потока сетевых транзакций.

Архитектурно шлюз представляет собой многокомпонентное устройство, составными частями которого служат: сетевой процессор, планировщик и ядро шлюза.

В задачу сетевого процессора входит отбор сетевых транзакций (пакетов и их последовательностей), подлежащих шифрованию и инкапсуляции с последующим туннелированием. Причем данный отбор в рамках концепции КИБ происходит совершенно прозрачно для пользователя и трафика.

Сетевой процессор обеспечивает механизм взаимодействия шлюза со средой передачи пакетов (т.к. в пилотной реализации рассматривается исключительно Ethernet, то используется понятие кадр) и реализует следующие функции:

1) извлечение кадров с канального уровня, минуя стек TCP/IP;

2) извлечение из полученного кадра IP-пакета и подготовка из него пакета для обработки (шифрация/дешифрация) в ядре шлюза;

3) подготовка кадра из обработанного в шлюзе пакета;

4) передачу на канальный уровень подготовленных кадров минуя стек TCP/IP для последующей их передачи.

Отобранные сетевым процессором кадры распределяются для дальнейшей обработки по свободным ресурсам шлюза с помощью планировщика, который реализует протокол взаимодействия с ядром шлюза, а так же управляет и планирует использование вычислительных средств в соответствии с выбранной стратегией планирования.

Планировщик состоит из следующих блоков:

1) блок взаимодействия с ядром;

2) блок взаимодействия с сетевым процессором;

3) блок планирования и контроля ресурсов в соответствии с выбранной стратегией их планирования;

4) блок управления и контроля состояния виртуальных каналов VPN и их атрибутов.

Сетевой процессор и планировщик не являются параллельно исполняемыми задачами, в то время как ядро шлюза – полностью параллельное приложение исполняемое на ВКИБ.

В качестве ВКИБ использовался высокопроизводительный вычислительный кластер с разделяемой памятью, который характеризуется использованием в качестве коммуникационной среды технологии Gigabit Ethernet, а в качестве среды для работы параллельного приложения – пакета MPICH, который представляет из себя свободно распространяемую реализацию стандарта MPI версии 1.1.

Теоретическая проработка параллельного ядра шлюза производилась исходя из теории групповых коммуникаций. Практическая реализация ядра основывается на стандартизованном программном инструментарии MPICH для обеспечения связи между ветвями параллельного приложения.

Главный упор при разработке ядра делается на то, что основной его функцией должна быть обработка пакетов определенного формата, поступающих от планировщика (приход некорректных пакетов исключается).

Ядро обеспечивает управление параллельными процессами, которые запускаются на вычислительном кластере в коммуникационной среде MPI в соответствии с определенной стратегией размещения и функциональным делением на группы:

– группа координатора обработки пакетов;

– группа сборщика обработанных пакетов;

– группа непосредственной обработки пакетов (рабочая группа).

Помимо явного параллелизма ядро высокопроизводительной криптографической системы отвечает следующим требованиям:

1) масштабируемость (статическая), которое подразумевает изменение количества процессов с помощью таких показателей, как: количество узлов кластера, количество процессоров на одном узле, коэффициент масштабирования в рамках фиксированной аппаратной конфигурации, которая задается один раз при старте узла на кластере;

2) интероперабельность и контроль доступа, т.е. возможность авторизованного взаимодействия с ядром с другой аппаратной платфор-

мы;

3) мобильность – свойство ядра, характеризующее возможность его переноса на другую платформу;

4) функциональная простота – требование, которое подразумевает под собой простую алгоритмизацию параллельной программы а также тривиальное ее расширение и адаптацию под любые другие прикладные задачи, поддающиеся распараллеливанию (то есть данное ядро может быть использовано не только в качестве ядра VPN-шлюза);

5) инвариантность по отношению к данным – требование, предъявляемое к вычислительным элементам ядра, которое подразумевай под собой такую их реализацию, что возможно функциональное расширение как типов и форматов обрабатываемых пакетов данных, так и методов их обработки.

Как уже отмечалось, основной функцией ядра, является параллельная обработка пакетов. Под обработкой в данном случае понимается следующие операции:

- применение к пакету криптографических функций (проверка целостности, шифрация, дешифрация, электронная цифровая подпись);
- проверка целостности пакетов и пометка соответствующая их маркировка (действителен/недействителен).

На пилотном образце шлюза было произведено практическое моделирование работы устройства в рамках предложенной архитектуры. Были исследованы следующие характеристики:

1) зависимость пропускной способности от вычислительной мощности ВКИБ;

2) сравнение двух подходов передачи данных: с записью их на устройства долговременного хранения (сетевая память) и без нее;

3) накладные расходы на использование параллельной среды выполнения MPICH.

Результаты исследования зависимости пропускной способности от вычислительной мощности ВКИБ таковы, что увеличение производительности линейно возрастает с ростом производительности кластера, причем рост производительности равен 50% на каждый добавленный в конфигурацию ВКИБ узел. Этот прирост может рассматриваться только как приблизительный, ввиду того, что число теоретически потери должны составлять не более 20-30%, из них 10% - это протокольная потеря «чистого» стека TCP/IP и 10-20% потери от работы протокола прикладного уровня, который, так или иначе, реализуется в коммуни-

кационных средах кластеров.

Результаты сравнения двух подходов передачи данных: с записью или без записи на устройство сетевой памяти, однозначно говорят о том, что промежуточная запись на диск возможна с небольшими временными затратами. Это подтверждает то, что реализуемое в соответствии с данным принципом вычислительное ядро КИБ, может использоваться для построения анализаторов трафика, систем обнаружения вторжений и сетевых сенсоров в соответствии с изложенной выше технологией.

Важно также то, что MPI не вносит дополнительных накладных расходов, и при подключении новых узлов к ВКИБ, производительность с увеличением числа процессоров в кластере растет линейно. Для пятиузлового ВКИБ был получен практически 300% прирост производительности по сравнению с вариантом шлюза, не подключенного к ВКИБ и не обладающего параллельным ядром.

Перейдем ко второму классу устройств, в архитектуре которых реализован альтернативный принцип кластеризации.

Параллельный анализатор сетевого трафика

Параллельный анализатор сетевого трафика (далее по тексту «анализатор»), архитектурно реализует второй подход к использованию технологий высокопроизводительных параллельных вычислений в концепции КИБ. Масштабирование данного типа устройств производиться посредством включения в параллельную работу однотипных устройств, каждое из которых в автономном режиме реализует полноценно реализует свою функцию. Полученный таким образом кластер реализует определенную параллельную обработку сетевых транзакций.

Следует отметить, что архитектура разрабатываемого в рамках концепции КИБ анализатора содержит в себе ключевую особенность: совокупность одиночных устройств, объединенных единым контуром управления образует в автоматическом режиме вычислительный кластер в котором выделяют следующие компоненты:

1) планировщик ресурсов кластера;

2) управляющий узел (с него производиться начальная конфигурация, реконфигурация и контроль работы анализатора);

3) исполнительные узлы, то есть устройства непосредственно занимающиеся обработкой журналов межсетевых экранов и занесение полученной в результате обработки информации в базу данных;

4) узел (узлы) с СУБД (параллельной СУБД), в которую заноситься вся информация о работе сети и из которой она может быть получена для последующего анализа и поддержки принятия решений администратором безопасности.

Получаемый в результате такого масштабирования кластер характеризуется как вычислительный кластер с разделяемой памятью, в качестве среды передачи у которого используется технология Gigabit Ethernet, а в качестве среды для запуска параллельных приложений – среда коммуникаций с собственным протоколам межузлового взаимодействия и планирования использования ресурсов, работающая поверх TCP/IP.

Для исследования преимуществ параллельного анализатора перед автономным устройством использовался кластер из пяти объединенных в единый контур автономных анализаторов, т.е. был образован кластер из пяти узлов, четыре из которых были чисто вычислительными, один – управляющим.

Если учесть, что средненагруженная вычислительная сеть из нескольких десятков вычислительных машин образует трафик, журнал обработки которого растет приблизительно со скоростью 100 Кбайт в секунду, то выигрыш в скорости при блочной параллельной обработке такого журнала по сравнению с последовательной обработкой очевиден.

И действительно, при испытаниях пилотного образца параллельного анализатора было получено пятнадцатикратное его превосходство по сравнению с автономным вариантом при одинаковых условиях.

Заключение

Несмотря на то, что два приведенных примера использования высокопроизводительных параллельных вычислений реализуют разные подходы к кластеризации в рамках концепции КИБ, оба устройства наглядно подтверждают предположение о внутреннем параллелизме обработки сетевого трафика и явно указывают на тот факт, что параллельные высокопроизводительные вычисления на кластерах применимы в устройствах обеспечения комплексной информационной безопасности в компьютерных сетях любого масштаба.

Две успешно решенные задачи с применением технологий высокопроизводительных параллельных вычислений: параллельная криптографическая обработка сетевых транзакций и параллельная обработка журналов межсетевых экранов являются весомым аргументом про-

должения работы в рамках изложенной концепции КИБ.

АЛГОРИТМ ПАРАЛЛЕЛЬНОГО РЕШЕНИЯ ЛИНЕЙНЫХ ЗАДАЧ МЕХАНИКИ ДЕФОРМИРУЕМОГО ТВЕРДОГО ТЕЛА С НАЧАЛЬНЫМИ НАПРЯЖЕНИЯМИ МКЭ

П.Г. Медведев, Н.В. Леонтьев

Нижегородский государственный университет им. Н.И. Лобачевского

В работе предложен алгоритм, позволяющий проводить с использованием метода конечных элементов (МКЭ) параллельное решение линейных задач механики деформируемого твердого тела с начальными напряжениями. Алгоритм основан на применении метода подконструкций. Алгоритмы распараллеливания различных задач содержатся в [1]. Описание и использование метода подконструкций изложено в [2].

Рассматриватеся линейно-упругое тело с начальными напряжениями, на которое наложены кинематические и силовые граничные условия. Разобьем тело на части (подконструкции) и рассмотрим одну из частей. Действие оставшихся частей заменим внутренними силами. Применим к рассматриваемой подконструкции МКЭ, в результате получим систему линейных алгебраических уравнений, состоящую из матрицы жесткости, искомого вектора узловых перемещений и вектора узловых сил, в который будут входить известные внешние и неизвестные внутренние силы. Методом исключения неизвестных мы можем понизить порядок линейной системы таким образом, чтобы в каждый элемент свободного вектора содержал неизвестные величины внутренних сил. Рассматривая по аналогии остальные подконструкции в итоге мы получаем совокупность систем линейных уравнений. При построении единой системы уравнений из совокупности систем неизвестные величины внутренних узловых сил исключаются. Решив систему, получаем вектор узловых перемещений, по которым определяется напряженно-деформированное состояние механической системы.

Распараллеливание вычислений для данной задачи заключается в организации системы процессов, в каждом из которых происходит построение глобальной матрицы жесткости и вектора узловых сил для подконструкции, наложение кинематических граничных условий и понижение методом исключений размерности системы. Каждый из

процессов передает полученные системы в единый процесс, где происходит сборка в систему уравнений для всей конструкции и решение задачи.

Эффективность параллельного алгоритма определяется отношением количества узлов подконструкций к узлам их границ и при увеличении этого отношения возрастает. Это связано с увеличением порядков, на которые происходит понижение размерностей систем в процессах и, соответственно, уменьшение размерностей систем, необходимых для передачи в единый процесс. Использование при численном решении модели с начальными напряжениями, позволяет применить алгоритм к итерационном решению нелинейных задач.

Литература

1. *Гергель В.П.*, *Стронгин Р.Г.* Основы параллельных вычислений для многопроцессорных вычислительных машин: Учебное пособие. Нижний Новгород: Изд-во ННГУ, 2000.
2. *Постнов В.А.*, *Тарануха Н.А.* Метод модуль элементов при расчете судовых конструкций. Л.: Судостроение, 1990.

ПРОГРАММНАЯ СРЕДА ДЛЯ ВИЗУАЛЬНОЙ РАЗРАБОТКИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

О.Г. Монахов, Э.А. Монахова

Институт вычислительной математики и математической геофизики СО РАН, Новосибирск

1. Введение

В работе представлена программная среда, позволяющая разрабатывать параллельные программы не привязываясь к конкретной вычислительной системе, использовать средства мультимедиа для придания большей ясности создаваемым с ее помощью программам и имеющая графический интерфейс для доступа через WWW. Среда представляет собой дальнейшее развитие системы автоматизации отображения параллельных программ TOPAS [1–3] и использует некоторые ее модули.

Представление параллельной программы в рассматриваемой среде основано на концепции *программного скелетона* и использует как не-

которые черты *фильмов*, предложенных в [4], так и черты описанных в [5] *скелетонов алгоритмов*. Программный скелетон объединяет в себе мультимедийное описание иерархической структуры и динамической активности алгоритма в пространственно-временных координатах. Программные скелетоны служат основой для генерации исполняемых параллельных программ на различных вычислительных системах. Для манипуляции со скелетонами были написаны редактор скелетонов, интерпретатор (необходимый для исполнения скелетона до компиляции его в исходный код), а также механизмы для подключения модулей, способных транслировать скелетон в исходный код на языке параллельной системы, на которой он должен выполняться. Кроме того, в среду включены два таких модуля: первый транслирует скелетон в программу на языке C, которая компилируется в консольное приложение Windows, второй также создает программу на C, но использует только функции стандарта ANSI и средства MPI, что позволяет применять ее на многих параллельных и кластерных системах, на которых реализован интерфейс MPI.

2. Описание скелетона

2.1. Структура скелетона

Скелетон представляет собой исполняемый модуль и обладает следующими свойствами:

- строгая древовидная структура – каждый элемент (узел дерева) скелетона имеет потомков, которые также являются элементами скелетона;
- универсальность – любой скелетон может быть транслирован в программу, пригодную к исполнению на любой вычислительной системе, для которой существует транслятор;
- простота представления и документирования – каждому элементу скелетона можно сопоставить некоторый объясняющий текст, изображение и (или) звуковой файл;
- модульность – любое поддерево скелетона может быть заменено другим поддеревом, если их вершины имеют одинаковый *тип* и *интерфейс*.

Элементы скелетона могут быть следующих типов:

Проект (Project) – корневой элемент скелетона, представляющего полностью готовую программу. Любой скелетон, готовый к компиляции, имеет корневую вершину этого типа. Эта вершина обеспечивает чтение исходных данных, и сохранение результатов вычислений. Его

потомками могут быть *библиотека типов* (*TypeLibrary*) и *вершина* (*Node*).

Библиотека типов (TypeLibrary) – элемент, потомками которого являются *типы* (*Type*). Этот узел предназначен для группировки пользовательских типов (массивы и структуры).

Тип (Type) – представляет собой тип. Существует несколько базовых предопределенных типов (char, short, int, long, float, double) и механизм для создания более сложных структур (structure и array). Каждый исполняемый элемент скелетона может иметь потомка типа TypeLibrary, который будет содержать типы, используемые его потомками. Типы идентифицируются по названию и могут перегружаться, то есть, при появлении в программе переменной какого-либо типа, описание этого типа будет взято в библиотеке типов ближайшего предка, в которой есть тип с данным именем. Такая организация позволяет сохранять части скелетона независимо друг от друга. Если тип простой, то потомков у него нет, сложный тип (structure или array) имеет потомков типа *поле* (*Field*), причем у array может быть только один такой потомок.

Поле (Field) – представляет собой поле сложного типа. Поле структуры (structure) имеет две характеристики: тип и имя, поле массива (array) – только одну, тип.

Вершина (Узел) (Node) – представляет собой последовательно исполняемую часть программы, минимальный исполняемый модуль. Ее основная характеристика – исполняемый код, который пишется на языке С. Потомками вершины могут являться элементы типа *кадр* (*shot*) и (или) *библиотека типов* (*TypeLibrary*). Кадры-потомки вызываются из кода вершины по имени как функции.

Кадр (Shot) – представляет собой блок программы, группирующий несколько параллельно исполняющихся модулей, его потомками могут быть *граф* (*graph*), *обобщенный граф* (*CommonGraph*) и (или) *библиотека типов* (*TypeLibrary*).

Граф (Graph) – предназначен для группировки и задания связей между несколькими вершинами (конкретными, статически заданными). Потомки – *вершины* (*Node*).

Обобщенный граф (CommonGraph) – предназначен для задания динамических структур и связей между вершинами. *Обобщенный граф* не может иметь произвольную структуру, его структура выбирается из конечного набора, но эта структура может иметь размерность, задаваемую переменными скелетона.

Типы обобщенных графов:

Множество (Set) – набор несвязанных узлов, количество которых задается специальной переменной обобщенного графа **N_Set.**

Кольцо (Ring) – **N_Ring** вершин, объединенных в кольцо.

Линия (Line) – **N_Line** вершин, объединенных в линию.

Двумерная решетка (2D Grid) – **Nx×Ny** вершин, объединенных в решетку.

Трехмерная решетка (3D Grid) – **Nx×Ny×Nz** вершин, объединенных в решетку.

Двоичное дерево (Binary tree) – двоичное дерево из **Nl** слоев, состоящее из 2^{Nl}-1 вершин.

Дерево (Tree) – **Nd**-арное дерево из **Nl** слоев, состоящее из Nd^{Nl}-1 вершин.

Единственный потомок *обобщенного графа* имеет тип *вершина* и представляет собой всех параллельно запускаемых потомков обобщенного графа.

Все элементы скелетона делятся на два класса: служебные и исполняемые. К служебным относятся *библиотека типов*, *тип* и *поле*, к исполняемым – *проект*, *вершина*, *кадр*, *граф* и *обобщенный граф*.

Общие характеристики всех элементов скелетона – имя и комментарий. Только поле массива не имеет имени (точнее имеет пустое имя), для всех остальных имя является необходимым. Комментарий является структурой, поясняющей данный элемент. Комментарий может содержать текст, изображение, звук, в дальнейшем возможно анимированное изображение.

Кроме того, всякий исполняемый элемент имеет набор переменных, в которых хранятся данные, локальные для него. Эти переменные делятся на четыре класса: специальные, внутренние, входные и выходные:

- набор специальных переменных постоянен – они описывают положение элемента среди его братьев (его координаты в решетке, или номер кадра в скелетоне), и представлен двумя переменными типа long: n_gid и n_pid. n_gid – номер группы, в которой находится поток, n_pid – номер потока в группе;
- внутренние переменные – это локальные переменные данного элемента, они доступны только в его коде и, как следствие, имеют смысл только для элемента типа *вершина* (*node*);
- входные переменные элемента имеют все те же свойства, что и локальные, но, кроме того, при запуске элемента на исполне-

ние они инициализируются значениями, полученными вычислением некоторых выражений в контексте элемента-предка. Набор таких выражений называется множеством *входных связей*;

– выходные переменные элемента также могут работать как локальные, но, кроме того, они фигурируют в правых частях выражений, которые вычисляются после завершения исполнения элемента и дают значения некоторым переменным из контекста элемента-предка. Набор таких выражений называется множеством *выходных связей*.

Набор связей, таким образом, обеспечивает передачу информации вниз и вверх по дереву скелетона. Но очевидно, что для параллельной программы этого мало, необходима также передача данных между параллельно исполняющимися потоками программы. Такая возможность существует и обеспечивается набором специальных функций, основанных на MPI.

2.2. Семантика исполнения скелетона

Исполнение скелетона представляет собой просто обход дерева по исполняемым вершинам. Семантика исполнения вершины зависит от ее типа, первой выполняется корневая вершина скелетона типа *проект (project)*. Единственная (и всегда корневая) вершина данного типа в скелетоне исполняется первой, в ее функции входит:

– прочитать из файла начальные значения входных переменных единственного потомка типа *вершина* (в конкретных реализациях рекомендуется производить чтение из потока стандартного входа);

– запустить на исполнение единственного потомка типа *вершина (node)* (для чего до исполнения выполняются входные связи, а после исполнения – выходные);

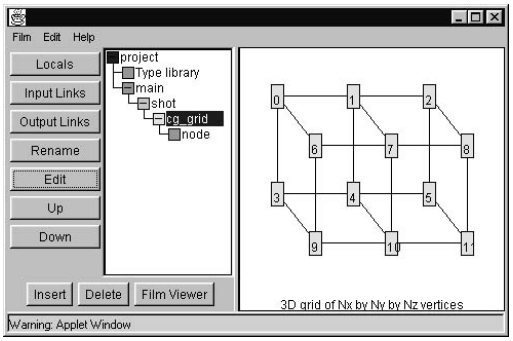
– записать в файл значения выходных переменных *вершины.*

Вершина – минимальный исполняемый элемент скелетона. *Вершина* считается выполненной после того, как выполнен содержащийся в ней код на внутреннем языке системы. Выполнение элементов следующего уровня (типа *кадр*) инициируется вызовом функции из кода вершины. Так, если вершина имеет потомка типа *кадр* с именем *shot_descendant*, то исполнится этот кадр только в том случае, если в тексте есть вызов функции *“shot_descendant();”*. Перед вызовом этой

функции будут выполнены входные связи, а после завершения выполнения соответствующего *кадра* отрабатываются выходные связи.

Кадр – элемент дерева, объединяющий несколько параллельно исполняющихся потоков. Потомками кадра могут быть элементы типов *граф* и *обобщенный граф*, которые в свою очередь обязательно имеют потомков типа вершина. Выполнение кадра состоит из следующих этапов.

1. Рассчитать входные связи для всех потомков типа обобщенный граф. Это позволит определить размер у всех динамических структур. Например, обобщенный граф типа «решетка» имеет две (и только две) входные переменные с именами «Nx» и «Ny», расчет входных связей определяет их значения и, таким образом, задает количество потоков, в которых будет выполняться единственный потомок (типа вершина) данного обобщенного графа.
2. Создать необходимое количество приостановленных потоков, в которых будут выполняться вершины-потомки графов и обобщенных графов – потомков данного кадра. В дальнейшем мы будем отождествлять вершины и потоки, в которых они исполняются. Все созданные вершины выполняются параллельно и могут обмениваться информацией с помощью посылки сообщений, в дальнейшем мы будем называть их вершинами-братьями.
3. Инициализировать служебные переменные созданных вершин. Здесь имеется в виду инициализация переменных «n_gid» и «n_pid». n_gid – номер группы, в которую входит данная вершина, n_pid – номер вершины в группе. Номер группы - это просто номер по порядку графа или обобщенного графа среди потомков кадра.
4. После того как инициализированы служебные переменные вершин, выполняются их входные связи.
5. Все вершины запускаются на параллельное исполнение.
6. Кадр ждет окончания исполнения всех вершин.
7. Исполняются выходные связи вершин.



3. Структура среды

Программная среда, позволяющая манипулировать со скелетонами, состоит из нескольких больших модулей:

- *Редактор скелетонов* – модуль, обеспечивающий создание, редактирование и сохранение скелетонов на диске. Главное окно редактора скелетонов показано на рисунке. В левой половине окна можно видеть дерево скелетона и кнопки, необходимые при его редактировании, в правой части – визуальное представление выделенного в данный момент элемента скелетона;
- *Синтаксический анализатор* – модуль, отвечающий за обработку исходного кода на внутреннем языке системы;
- *Интерпретатор скелетонов* – модуль, необходимый для исполнения скелетона до его компиляции в исходный код на языке какой-либо параллельной системы. Синтаксический анализатор и интерпретатор скелетонов вместе позволяют обнаружить большую часть синтаксических и алгоритмических ошибок еще до трансляции и запуска программы на параллельной вычислительной системе;
- *Интерфейс компилятора* – Java-интерфейс, обеспечивающий простое подключение к системе модулей, транслирующих скелетон в программу для произвольной параллельной системы. Эти модули могут выдавать как готовые к исполнению бинарные файлы, так и проекты на некотором промежуточном языке данной системы (C, Fortran и т.д.).

Языком реализации среды был выбран Java из-за своей многопоточности, модульности, универсальности и ориентированности на Internet.

4. Заключение

Целью создания данной среды является облегчение разработки и переноса вычислительных алгоритмов на различные параллельные вычислительные системы путем визуализации конструирования и автоматической генерации исполняемого кода. Скелетоны должны служить промежуточным звеном между конечным пользователем, желающим запустить свой алгоритм на высокопроизводительной системе, и параллельной вычислительной системой. В рамках данного проекта:

- предложена модель параллельной программы, названная программным скелетоном;

– разработана программная среда, предназначенная для создания, редактирования и трансляции скелетонов в исходный код для конкретных вычислительных систем;
– реализованы два модуля, транслирующих скелетоны в программы на языке C. Первый создает консольное приложение для операционных систем Windows, второй создает программу на языке C, использующую только функции стандарта ANSI и библиотеку MPI, что позволяет использовать ее на параллельных вычислительных системах.

Литература

1. *Монахов О.Г.*, *Монахова Э.А.* Параллельные системы с распределенной памятью: управление ресурсами и заданиями. Новосибирск: Изд-во ИВМ и МГ СО РАН, 2001.
2. *Монахов О.Г.*, *Монахова Э.А.* Параллельные системы с распределенной памятью: структуры и организация вычислений. Новосибирск: Изд-во СО РАН, 2000.
3. *Monakhov O.G.*, *Chunikhin O.Y.* WWW-based system for visualization, animation and investigation of mapping algorithms.// Proc. Inter. Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN'97). Taipei, Taiwan, Dec. 18-20, 1997. IEEE Press, 1997. P. 207–210.
4. Mirenkov N., VIM Language Paradigm, Parallel Processing: CONPAR'94-VAPP'IV, Lecture Notes in Computer Science, Vol.854, B.Buchberger, J.Volkert (Eds.), Springer-Verlag, 1994. P. 569–580,
5. *Cole M.* Algorithmic Skeletons: Structured Management of Parallel Computation, The MIT Press, 1989.

ОПТИМИЗИРОВАННЫЙ АЛГОРИТМ МЕДИЦИНСКОЙ ДИАГНОСТИКИ

Ф.М. Москаленко

Дальневосточный государственный университет, г. Владивосток

Введение

В настоящее время существует некоторое количество экспертных систем по медицинской диагностике, однако, большинство из них основано на довольно-таки простых онтологиях предметных областей. В

основе данной работы лежит онтология, разработанная в ИАПУ ДВО РАН к 1999 году и являющаяся одним из наиболее близких приближений к реальности.

С использованием этой онтологии в 2001 году был разработан алгоритм, решающий задачу медицинской диагностики. Этот алгоритм при хорошо наполненной базе знаний вынужден работать с большим объёмом данных (заболевания, признаки, варианты развития признаков и так далее), и при решении поставленной задачи он выполняет большое количество переборов, из-за чего время его работы (теоретически) велико.

В связи с этим цель данной работы - оптимизация алгоритма медицинской диагностики с целью ускорения его работы при постановке диагноза.

Пути оптимизация алгоритма медицинской диагностики

Для достижения поставленной цели необходимо: исследовать исходный алгоритм, найти методы его оптимизации, получить оптимизированный алгоритм и показать, что он корректен, то есть по-прежнему правильно решает задачу медицинской диагностики. Кроме того, необходимо провести сравнение исходного и оптимизированного алгоритмов и показать, что оптимизация действительно имеет место, то есть полученный алгоритм при определённых условиях работает быстрее исходного.

Так как в основе алгоритма медицинской диагностики лежат несколько (семь) вложенных циклов-переборов, то очевидно, что целью оптимизации должно быть сокращение времени выполнения этих циклов. Добиться такого результата можно двумя путями:

А) за счет сокращения переборов, производимых алгоритмом, как это уже выполнялось в некоторых системах медицинской диагностики, основанных, однако, на более простых моделях предметной области;

Б) за счет распараллеливания алгоритма для его выполнения на кластере, что является новым направлением в области оптимизации алгоритмов.

Исходный алгоритм был оптимизирован последовательно, то есть вначале были внесены дополнения, позволяющие сократить переборы (за счёт дополнительной базы данных и алгоритма, единожды формирующего её из общей базы данных), а затем оба алгоритма (дополнительный и сам алгоритм медицинской диагностики) были распараллелены (записаны на полуформальном языке, без привязки к определённой среде).

При проведении оптимизации А был модифицирован алгоритм диагностики, а также был разработан вспомогательный алгоритм получения дополнительной информации, который выполняет следующее: в исходной базе данных основными понятиями являются заболевания, от которых через признаки зависят значения признаков; алгоритм исследует все значения признаков и «переворачивает» исходную информацию, то есть в формируемой дополнительной базе данных уже от значений признаков получаем заболевания. При диагностике благодаря этой информации по значениям признаков пациента алгоритм медицинской диагностики формирует с помощью дополнительной базы знаний предполагаемые варианты диагнозов, а затем "проверяет" их методом решения обратной задачи.

Распараллеливание алгоритма диагностики

Для проведения оптимизации Б в алгоритмах, полученных после оптимизации А (усовершенствованный алгоритм диагностики и алгоритм получения дополнительной базы данных для сокращения переборов), выделены части, которые можно было бы запускать параллельно на нескольких вычислительных узлах (кластера).

Оба алгоритма разбиваются на 2 части: одна запускается на управляющем узле (сервер), «общается» с пользователем, готовит «задания» для дочерних (клиент) узлов, «раздает» их и принимает результаты; другая – выполняется в нескольких экземплярах на дочерних («клиент») узлах кластера.

Все распараллеливаемые алгоритмы (и сервер-часть и клиент-части) могут пользоваться глобальной для них Базами Знаний по Медицинской Диагностике. Кроме того, между частями алгоритмов идёт обмен данными по схеме «передача параметров – возврат результатов».

Алгоритм получения дополнительной информации предполагается запускать не часто, и для его работы можно выделить продолжительный отрезок времени. В связи с этим оптимизировать скорость его работы по максимуму не предполагается. Наипростейший способ ускорить его работу при помощи распараллеливания без существенного усложнения – распределить исследование всех признаков из базы знаний между отдельными узлами. То есть главная (серверная) часть алгоритма сформирует из названий признаков (хранящихся в БЗ) очередь заданий вида:

1) признак1;
2) признак2;
...
P) признакP.

и начнет последовательно раздавать свободным клиент-частям задания для исследования значений признаков; а каждая дочерняя часть исследует ВСЕ значения переданного ей признака и сформирует часть глобальной дополнительной базы данных.

Исследование самого алгоритма медицинской диагностики показало, что в нём целесообразно провести динамическое разделение на нескольких уровнях (в зависимости от трудоёмкости работ):

1) в общей части, выполняющейся на центральном узле, происходит общение с пользователем, идёт перебор диагнозов для проверки, а каждому дочернему узлу кластера даётся для исследования пара <диагноз – время его начала на шкале наблюдений> (выполняется при числе диагнозов для проверки большем, чем количество узлов кластера);

2) в общей части, выполняющейся на центральном узле, происходит общение с пользователем, идёт перебор диагнозов, для возможных диагнозов перебираются времена их начал, а каждому узлу кластера дается для исследования тройка <диагноз - время его начала на шкале наблюдений - введённый признак пациента> (выполняется при числе диагнозов для проверки меньшем, чем количество узлов кластера).

При распараллеливании на этих уровнях каждый узел должен иметь доступ к БЗ по медицинской диагностике и к данным, введённым пользователем. (Поэтому их удобнее сохранить в глобальной базе данных, а каждый дочерний процесс будет их читать).

Поскольку при диагностике диагноз ЗДОРОВ отделяется от других, то при необходимости его проверки сервер-часть алгоритма формирует первую очередь заданий вида <«ЗДОРОВ» – «0» – признак>, где в первом поле указано название проверяемого заболевания, во втором – время его начала на шкале наблюдений, а в третьем – введённый признак, значения которого необходимо проверить. Это распараллеливание на уровне 2.

Задания из этой очереди последовательно раздаются свободным клиент-узлам. После обработки такого задания клиент-узел возвращает положительный или отрицательный результат. Если результат отрицательный, то есть введённые значения признака невозможны при диагнозе ЗДОРОВ, то вся очередь заданий очищается, работа клиент-узлов

прерывается и алгоритм сервер-части переходит к рассмотрению других диагнозов из сформированного множества гипотез-диагнозов. Если же все все задания из первой очереди выполняются успешно, то диагноз ЗДОРОВ добавляется к результатам и дальнейший перебор диагнозов не производится.

Для перебора заболеваний отличных от ЗДОРОВ и времени их начала сервер-часть алгоритма формирует вторую очередь заданий, в которой каждое задание имеет вид <заболевание – время его начала – «все признаки»>. Здесь значение третьего поля («все признаки») говорит о том, что необходимо проверять значения всех признаков при гипотезе <заболевание – время его начала>. Это распараллеливание на уровне 1.

Задания из этой очереди последовательно раздаются свободным узлам на «проверку». При неуспешной проверке какого-то заболевания, начавшегося в какой-то момент, такая гипотезы отвергается. Если же проверка возвратила положительный результат, что рассмотренную гипотезу необходимо считать частью решения задачи диагностики, и отпадает необходимость проверять это же заболевание, начинающееся в другие моменты времени. Такие задания сразу же удаляются из очереди, а выполнение заданий находящиеся в процессе обработки этого же заболевания (но с другими моментами его начала) необходимо прервать.

По мере обработки заданий из сформированной очереди, она заканчивается, то есть наступает момент, когда все такие задания уже отданы узлам на обработку и вновь освобождающиеся узлы не могут получить задание. В этом случае необходимо прервать работу всех «занятых» узлов, при этом они должны возвратить информацию об обрабатываемых ими гипотезах вида <заболевание – его начало – признак – минимальный конец заболевания – максимальный конец заболевания>, где признак – этот тот признак, значения которого клиент-программа в момент прерывания пытались объяснить, а минимальный и максимальный конец заболевания – это временной интервал конца заболевания, который был получен при объяснении предыдущих признаков (все введённые признаки упорядочены). Из полученных пятёрок строится новая очередь мини-заданий, которые имеют вид <заболевание – его начало – признак>. Это распараллеливание на уровне 2.

Задания последовательно раздаются узлам. Проверив какой-то признак каждый узел возвращает положительный или отрицательный результат. Если обрабатывавшийся признак не смог быть объяснён при

заданном заболевании и времени его начала – необходимо удалить из очереди все задания с таким же заболеванием и временем его начала и остановить работу узлов, уже работающих с какими-то признаками при данном заболевании и времени его начала. Если же все введённые признаки при некотором заболевании и времени его начала были объяснены – к результатам работы алгоритма добавляется ещё одно заболевание.

При параллельной работе клиент-узлов, они не имеют доступа к глобальным данным, находящимся в сервер-части алгоритма (введенные пользователем данные о пациенте: значения признаков, особенностей, события). Для организации доступа к таким данным формируется третья глобальная база данных «информация для обработки». Считается, что все клиент-узлы могут параллельно читать сведения из этой базы данных.

Заключение

Все полученные алгоритмы были записаны на полуформальном языке, и была показана их корректность.

После этого было проведено исследование сложности распараллеленного алгоритма медицинской диагностики, показавшее, что при теоретически неограниченном количестве клиент-узлов время его работы приближается к времени, затрачиваемому на обработку наиболее трудоёмкого задания из очереди заданий, что при большом количестве заданий намного меньше времени работы исходного алгоритма, обрабатывающего все такие задания последовательно.

В нынешнем году предполагается приступить к реализации распараллеленных алгоритмов на кластере в ИАПУ ДВО РАН.

Мультикомпьютер, на котором предполагается реализовать систему медицинской диагностики, представляет собой кластер, состоящий из 5 узлов. На мультикомпьютере установлена операционная система Linux, для организации взаимодействия параллельных процессов используется протокол MPI (реализация LAM), для взаимодействия с базой данных установлена система Oracle.

Каждый из 5 компьютеров кластера имеет следующую конфигурацию:

– процессоры: Dual CPU P-III/800EB MHz
– кэш-память: 256 K
– частота системной шины: 133 MHz

– оперативная память: 2*SD-RAM 128Mb (PC-133) SAMSUNG– материнская плата: MB Super Micro P-III/DME <2xSlot1, i840, ATX>

Литература

1. *Каменев А.В.*, *Клещев А.С.*, *Черняховская М.Ю.* Логическая модель взаимодействия причинно-следственных отношений различных типов в области медицинской диагностики: Препринт. Владивосток: ИАПУ ДВО РАН, 1999.
2. *Москаленко Ф.М.* Разработка экспертной системы медицинской диагностики для реальной онтологии медицины, Разработка алгоритма решения задачи диагностики: Курсовая работа, 3 курс, Владивосток, 2001.
3. *Москаленко Ф.М.* Оптимизация алгоритма медицинской диагностики: Выпускная квалификационная работа бакалавра, Владивосток, 2002.

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ РАСЧЕТА НЕОКЛАССИЧЕСКОЙ МОДЕЛИ МЕЖОТРАСЛЕВОГО БАЛАНСА

И.М. Мударисов

Удмуртский госуниверситет, г. Ижевск

Описание модели

В данной экономической модели выделяется N чистых отраслей и считается, что производственные мощности каждой отрасли разделены между финансово-промышленными группами (ФПГ), число которых обозначено через A. В модель неоклассического межотраслевого баланса вводится описание расслоения отраслей по субъектам собственности (ФПГ) и описание обращения двух видов денег: суррогатных денег – неплатежей и «живых» денег.

Считается, что каждая отрасль при производстве продукта затрачивает продукты других отраслей и M видов первичных ресурсов. Выпуск продукта j-й отраслью, принадлежащей ФПГα, описывается производственной функцией $F_{j\alpha}(\mathbf{X}^{j\alpha}, \mathbf{I}^{j\alpha})$, в которой $\mathbf{X}^{j\alpha} = (X_1^{j\alpha}, \ldots,$

$X_N^{j\alpha}$) означает вектор продуктов, затрачиваемых в ФПГ α для выпуска j-го продукта, а $\mathbf{I}^{j\alpha} = (I_1^{j\alpha},\ldots,\ I_M^{j\alpha})$ – вектор первичных ресурсов, затрачиваемых в ФПГ α для выпуска j-го продукта.

Спрос населения описывается вогнутой функцией полезности (индексом потребления) $F_0(\mathbf{X}^0)$, в которой $\mathbf{X}^0 = (X_1^0,...,X_N^0)$ – вектор конечного потребления продуктов населением.

В работе [1] подробно описываются все переменные этой модели ($X_j^0, X_k^{j\alpha}, I_m^{j\alpha}, X_{j\alpha}^{i\beta}, X_{j\alpha}^0, G_{j\alpha}, Y_{j\alpha}^{i\beta}, Z_{j\alpha}^{i\beta}, p_i, \mu_m$ $i,j,k = 1,\ldots,N;$ $\alpha,\beta = 1,\ldots,A;$ $m = 1,\ldots,M$), их экономическая интерпретация и балансовые соотношения. Ставится задача вогнутого программирования о максимуме функции полезности (индекса потребления) с ограничениями типа равенств и неравенств. Эта задача, после определенных преобразований, сводится к общему виду задачи выпуклого программирования.

$$\min f(u),\ u \in \mathbf{R}^n,$$
$$g_i(u) \le 0,\ i = 1,\ \ldots,\ m,$$

где функции $f(u)$, $g_i(u)$ $(i = 1,\ \ldots,\ m)$ выпуклые.

Вектор $u = \{X_j^0, X_k^{j\alpha}, I_m^{j\alpha}, X_{j\alpha}^{i\beta}, X_{j\alpha}^0, G_{j\alpha}, Y_{j\alpha}^{i\beta}, Z_{j\alpha}^{i\beta}, p_i, \mu_m\}$,

$u \in \mathbf{R}^{\mathbf{3N^2A^2+N^2A+NMA+2NA+2N+M}}$, так как

$\{X_j^0\} \in \mathbf{R}^{\mathbf{N}}, \{X_k^{j\alpha}\} \in \mathbf{R}^{\mathbf{N^2A}}, \{I_m^{j\alpha}\} \in \mathbf{R}^{\mathbf{NMA}}, \{X_{j\alpha}^{i\beta}\} \in \mathbf{R}^{\mathbf{N^2A^2}}, \{X_{j\alpha}^0\} \in \mathbf{R}^{\mathbf{NA}},$

$\{G_{j\alpha}\} \in \mathbf{R}^{\mathbf{NA}}, \{Y_{j\alpha}^{i\beta}\} \in \mathbf{R}^{\mathbf{N^2A^2}}, \{Z_{j\alpha}^{i\beta}\} \in \mathbf{R}^{\mathbf{N^2A^2}}, \{p_i\} \in \mathbf{R}^{\mathbf{N}}, \{\mu_m\} \in \mathbf{R}^{\mathbf{M}}.$

$$f(u) = -F_0(\mathbf{X}^0),$$
$$g_1(u) = \{g_{1,j\alpha}(u)\}_{\substack{j=1,\ldots,N \\ \alpha=1,\ldots,A}} =$$

$$\left\{-F_{j\alpha}(\mathbf{X}^{j\alpha}, \mathbf{I}^{j\alpha}) + \sum_{i,\beta} X_{j\alpha}^{i\beta} + X_{j\alpha}^0 + G_{j\alpha} + \phi_{j\alpha}(X_{j\alpha}^0,\ldots,Z_{j\alpha}^{i\beta},\ldots)\right\}_{\substack{j=1,\ldots,N \\ \alpha=1,\ldots,A}},$$

$$g_2(u) = \{g_{2,jk\alpha}(u)\}_{\substack{j,k=1,\ldots,N \\ \alpha=1,\ldots,A}} = \left\{ X_k^{j\alpha} - \sum_{\beta} X_{k\beta}^{j\alpha} \right\}_{\substack{j,k=1,\ldots,N \\ \alpha=1,\ldots,A}},$$

$$g_3(u) = -g_2(u),$$

$$g_4(u) = \{g_{4,j}(u)\}_{j=1,\ldots,N} = \left\{ X_j^0 - \sum_{\alpha} X_{j\alpha}^0 \right\}_{j=1,\ldots,N},$$

$$g_5(u) = -g_4(u),$$

$$g_6(u) = \{g_{6,ij\alpha\beta}(u)\}_{\substack{i,j=1,\ldots,N \\ \alpha,\beta=1,\ldots,A}} = \left\{ X_{j\alpha}^{i\beta} - Y_{j\alpha}^{i\beta} - Z_{j\alpha}^{i\beta} \right\}_{\substack{i,j=1,\ldots,N \\ \alpha,\beta=1,\ldots,A}},$$

$$g_7(u) = -g_6(u),$$

$$g_8(u) = \{g_{8,m}(u)\}_{m=1,\ldots,M} = \left\{ \sum_{j,\alpha} I_m^{j\alpha} - I_m \right\}_{m=1,\ldots,M},$$

$$g_9(u) = \{g_{9,\alpha}(u)\}_{\alpha=1,\ldots,A} =$$

$$\left\{ \sum_{i,j,\beta} p_i Y_{i\beta}^{j\alpha} + \sum_{j,m} \mu_m I_m^{j\alpha} - \sum_{j} p_j G_{j\alpha} - \sum_{i,j,\beta} p_j Y_{j\alpha}^{i\beta} \right\}_{\alpha=1,\ldots,A},$$

$$g_{10}(u) = \{g_{10,j\alpha}(u)\}_{\substack{j=1,\ldots,N \\ \alpha=1,\ldots,A}} = \left\{ -G_{j\alpha} \right\}_{\substack{j=1,\ldots,N \\ \alpha=1,\ldots,A}},$$

$$g_{11}(u) = \{g_{11,j\alpha m}(u)\}_{\substack{j=1,\ldots,N \\ \alpha=1,\ldots,A \\ m=1,\ldots M}} = \left\{ -I_m^{j\alpha} \right\}_{\substack{j=1,\ldots,N \\ \alpha=1,\ldots,A \\ m=1,\ldots,M}},$$

$$g_{12}(u) = \{g_{12,j\alpha}(u)\}_{\substack{j=1,\ldots,N \\ \alpha=1,\ldots,A}} = \left\{ -X_{j\alpha}^0 \right\}_{\substack{j=1,\ldots,N \\ \alpha=1,\ldots,A}},$$

$$g_{13}(u) = \{g_{13,ij\alpha\beta}(u)\}_{\substack{i,j=1,\ldots,N \\ \alpha,\beta=1,\ldots,A}} = \left\{ -Y_{j\alpha}^{i\beta} \right\}_{\substack{i,j=1,\ldots,N \\ \alpha,\beta=1,\ldots,A}},$$

$$g_{14}(u) = \{g_{14,ij\alpha\beta}(u)\}_{\substack{i,j=1,\ldots,N \\ \alpha,\beta=1,\ldots,A}} = \left\{ -Z_{j\alpha}^{i\beta} \right\}_{\substack{i,j=1,\ldots,N \\ \alpha,\beta=1,\ldots,A}}.$$

Общее количество функций-ограничений равно

$$NA + 2N^2A + 2N + 2N^2A^2 + M + A + 2NA + NAM + 2N^2A^2 =$$
$$= 4N^2A^2 + 2N^2A + NAM + 3NA + 2N + M + \text{A}.$$

Методы решения

Для решения задач выпуклого программирования разработано достаточно много численных методов. Одним из таких является метод одновременного решения пары двойственных задач [2], в котором итерации по прямым и двойственным переменным ведутся следующим образом:

$$w^k = \arg\min_{w_i \geq 0, i \in I_k} \left\| \nabla f(u^k) + \sum_{i \in I_k} w_i \nabla g_i(u^k) \right\|^2 ,$$

$$I_k = \{ i : g_i(u^k) \geq -\varepsilon \} ,$$

$$u^{k+1} = u^k - \gamma_k \left(\nabla f(u^k) + \sum_{i \in I_k} w_i \nabla g_i(u^k) \right)$$

Смысл метода достаточно прозрачен: для точки u^k находятся такие приближения двойственных переменных w^k, которые минимизируют невязку в условиях экстремума. Одной из особенностей метода является то, что он сходится тем быстрее, чем больше в задаче ограничений.

При отыскании нового приближения двойственных переменных w^k необходимо решить вспомогательную задачу, которая имеет специфический вид.

$$\min f(x), x \in Q \subset R^m ,$$
$$Q = \{ x : x \geq 0 \} .$$

Это задача отыскания минимума квадратичной функции на положительном ортанте в R^m. Она успешно решается методом сопряженных градиентов [2]

$$x^{k+1} = x^k - \alpha_k p^k \ , \ \alpha^k = \arg\min_{\substack{\alpha \geq 0 \\ x^k + \alpha p^k \geq 0}} f(x^k + \alpha \ p^k) \ ,$$

$$p_i^k = \begin{cases} -\nabla f(x^k)_i + \beta_k p_i^{k-1}, i \notin I_k , \\ 0, i \in I_k , \end{cases}$$

$$\beta_k = \begin{cases} \sum_{i \notin I_k} (\nabla f(x^k)_i)^2 \ / \sum_{i \notin I_k} (\nabla f(x^{k-1})_i)^2 , \text{если} \quad I_k = I_{k-1} , \\ 0, \text{если} \quad k = 0 \quad \text{или} \, I_k \neq I_{k-1} , \end{cases}$$

$$I_k = \begin{cases} \{i : x_i^k = 0, \nabla f(x^k)_i > 0\}, \text{если } k = 0 \text{ или} \nabla f(x^k)_i = 0 \\ \qquad \text{для всех } i \notin I_{k-1}, \\ I_{k-1} \cup \{i : x_i^k = 0\}, \text{в остальных случаях..} \end{cases}$$

Программная реализация методов

Рассмотренные методы реализованы в пакете Mathematica 4.1 и на языке C++.

Реализация в пакете Mathematica 4.1 в первую очередь ценна точностью получаемых результатов. Данную реализацию можно использовать для формирования тестовых примеров. Исходные данные и все параметры описываются внутри программы.

Главной целью реализации на языке C++ является создание основы для параллельного варианта численной реализации методов. Программа на языке C++ характеризуется следующим: исходные данные хранятся на внешнем носителе в файлах, результаты каждой итерации (новые приближения прямых и двойственных переменных и значение целевых функций) включаются в файл отчета. Ведется учет времени по итерациям.

Большая размерность задачи, количество ограничений и объем требуемых вычислений приводит к идее использования высокопроизводительных компьютерных комплексов за счет распараллеливания вычислений. Средством реализации выбрано PVM. На первом шаге распараллеливание алгоритма решения осуществляется за счет множителя γ_k в методе одновременного решения пары двойственных задач. Анализ работы непараллельной реализации выявил блоки программы, которые требуют до 90% вычислительных ресурсов. Распараллеливание этих блоков – следующий этап реализации параллельного алгоритма расчета модели.

Литература

1. *Автухович Э.В.*, *Гуриев С.М.*, *Оленев Н.Н.*, *Петров А.А.*, *Поспелов И.Г.*, *Шананин А.А.*, *Чуканов С.В.* Математическая модель экономики переходного периода. Вычислительный центр РАН, Москва 1999.
2. *Поляк Б.Т.* Введение в оптимизацию, М.: Наука, 1983.

РАСЧЕТ УПРУГИХ ТЕЛ С ТОНКИМИ СЛОЯМИ И ПОКРЫТИЯМИ НА КЛАСТЕРЕ РАБОЧИХ СТАНЦИЙ

А.И. Олейников, А.О. Кузьмин

Комсомольский-на-Амуре государственный технический университет

В работе представлено описание разработки программного продукта по организации параллельных вычислений на кластере рабочих станций. В настоящее время существует много работ по численному решению фундаментальных и прикладных задач механики сплошных сред с использованием различных средств распараллеливания, наиболее распространёнными являются интерфейсы MPI, OpenMP, PVM, языки C-DVM, FORTRAN-DVM и т.д. Основные преимущества их использования состоят в возможности создания одного программного кода как для последовательной, так и для параллельной версии программы, возможность переносимости с одной платформы на другую и упрощение написания программ благодаря использования стандартных функций по посылке сообщений между процессорами. Однако, подход, использованный в данной работе, благодаря использованию низкоуровневых средств системного программирования позволил построить систему, представляющую высокопроизводительный, более надёжный и независящий от дополнительных библиотек и специализированных языков программирования пакет программ. Как и вышеуказанные средства распараллеливания, система не зависит от конфигурации сети и аппаратного обеспечения.

К числу основных проблем, решённых в процессе разработки, можно отнести: построение подсистемы рассылки сообщений, позволяющей использовать различные сетевые протоколы (в том числе TCP/IP) для обмена входными, промежуточными данными и результатами вычислений между компьютерами в сети; автоматизированной подсистемы назначения заданий между доступными компьютерами и их переназначения в случае сбоев; подсистемы восстановление расчёта после потери одного или нескольких вычислительных узлов в результате сетевого или аппаратного сбоя; подсистемы сбора статистики вычислений.

Автономность функционирования системы вкупе с пользовательским интерфейсом для постановки задач делают её удобным средством для решения инженерных задач с использование существующего парка компьютеров, причём без отрыва последних от повседневного ис-

пользования.

Программный продукт осуществляет расчёт напряжённого состояния кусочно-однородных изотропных тел на кластере рабочих станций, производимый в рамках линейной теории упругости. На практике, в качестве таких тел могут, например, выступать любые промышленные изделия, использующие нанесение многослойных покрытий.

Математическая постановка задачи сводится к следующему. Тело Ω рассматривается как состоящее из N однородных изотропных областей Ω_n, так что $\Omega = \bigcup_n \Omega_n$ и $\Sigma_{n,m} = \Omega_n \bigcap \Omega_m$ -поверхность раздела областей Ω_n и Ω_m, $n,m = 1,\ldots,N$ [1–3].

Задача сводится к определению функций $f_k^{(n)}(q_0)$, называемых фиктивными нагрузками из системы $(n,m = 1,\ldots,N)$ интегральных уравнений с выделенной сингулярностью, $k = x, y$.

$$\begin{gathered}
\frac{1}{2} f_k^{(n)}(q) + \int\limits_{\partial\Omega^{(n)}} H_{ijk}^{(n)}(q,q_0) n_j(q) f_k^{(n)}(q_0) dl_{q_0} = P_k^{(n)}(q), \\
\int\limits_{\partial\Omega^{(n)}} I_{ik}^{(n)}(q,q_0) f_k^{(n)}(q_0) dl_{q_0} = u_k^{(n)}(q), \qquad (1) \\
\frac{1}{2} f_k^{(n)}(q) + \int\limits_{\Sigma_{n,m}^{+}} H_{ijk}^{(n)}(q,q_0) n_j(q) f_k^{(n)}(q_0) dl_{q_0} - \frac{1}{2} f_k^{(m)}(q) - \\
- \int\limits_{\Sigma_{n,m}^{-}} H_{ijk}^{(m)}(q,q_0) n_j(q) f_k^{(m)}(q_0) dl_{q_0} = 0, \\
\int\limits_{\Sigma_{n,m}^{+}} I_{ik}^{(n)}(q,q_0) f_k^{(n)}(q_0) dl_{q_0} - \int\limits_{\Sigma_{n,m}^{-}} I_{ik}^{(m)}(q,q_0) f_k^{(m)}(q_0) dl_{q_0} = 0.
\end{gathered}$$

Первые два уравнения (1) отвечают корректно заданным граничным условиям $P_x^{(n)}$, $P_y^{(n)}$ и $u_x^{(n)}$, $u_y^{(n)}$ на внешней поверхности тела $\Omega^{(n)}$, а два других являются контактными условиями, $H_{ijk}(q,q_0)$ и $I_{ik}(q,q_0)$ – фундаментальные решения для плоскости.

При решении системы (1) методом механических квадратур из-за близкого расположения граничных элементов и использования интегральных уравнений первого рода возникает плохо обусловленная сис-

тема линейных алгебраических уравнений (СЛАУ). Методика получения устойчивого решения системы линейных уравнений основывается на методе регуляризация А.Н. Тихонова [4], сформулированном в виде вариационной задачи минимизации функционала:

$$f(x) = \|Ax - b\|^2 + \alpha\|x\|^2, \quad (2)$$

где $\alpha > 0$ параметр регуляризации, A – матрица СЛАУ, полученная методом квадратур, b – вектор граничных условий.

Минимум функционала (2) обеспечивает решение системы [5]

$$\left(A^* A + \alpha E\right)x = A^* b + \alpha x_0, \quad (3)$$

Поиск параметра α в (3) состоит из многократных формирований и решений систем линейных уравнений конечными методами. Алгоритм содержит внешний и внутренний циклы, которые обеспечивают выполнение условия, свидетельствующего о получении регуляризованного решения системы (1).

Внешний цикл формирует сходящуюся к нулю последовательность $\{\alpha_p\}$, на элементах которой производится минимизация функционала (2).

Внутренний цикл обеспечивает поиск минимума функционала (2) согласно (3) при закреплённой величине $\alpha = \alpha_p$.

В качестве вектора x_0 при $\alpha = \alpha_0$ используется вектор граничных условий b, а на каждой последующей итерации – регуляризованное приближение x, полученное на предыдущей.

Количество внутренних циклов и критерий останова могут зависеть от конкретной задачи. Одним из наиболее надёжных является критерий невязки, использующий регуляризованное приближение, получаемое на каждой итерации внутреннего цикла.

Решение системы линейных уравнений на внутреннем цикле данной задачи при описанных начальных данных проводилось методом квадратного корня, хотя, стоит заметить, что оно может быть также получено и методом сопряжённых градиентов (хотя численное решение в этом случае несколько хуже согласуется с аналитическим).

На основе данных алгоритмов разработан комплекс программ расчёта напряжённого состояния кусочно-однородных тел «PHS». Комплекс разбит на две части: переменную и постоянную. Постоянная часть представляет собой реализацию математических алгоритмов метода граничного элемента и методики получения регуляризованных

приближений. Решение системы линейных уравнений на внутреннем цикле осуществляется посредством метода квадратного корня, показавшим в отличие от метода Зейделя и метода сопряжённых градиентов свою надёжность при решении более сложных задач, а также являющимся удобным в смысле отсутствия необходимости выбора дополнительного критерия останова, как в итерационных методах и создания вариантов его реализации для параллельных вычислений.

Переменная часть представляет собой реализацию алгоритмов дискретизации исследуемых тел и подготовки всех входных данных для начала вычислений, а также организации интерактивного взаимодействия с пользователем-расчётчиком.

При проведении вычислений описанным выше методом приходится проводить большой объём вычислений, особенно это касается регуляризирующего алгоритма. В связи с этим для построенного комплекса «PHS» был разработан дополнительный уровень, реализующий алгоритм минимизации функционала (2), т.е. фактически получение фиктивных нагрузок из уравнений (1). Особенностью данного уровня является проведение вычислений на кластере рабочих станций, соединённых локальной вычислительной сетью. Данный уровень может по необходимости быть динамически состыкован с «PHS» и осуществлять в фоновом режиме все вычисления.

Математической основой, позволившей распараллелить вычисления, является принятый в алгоритме способ минимизации функционала (2) посредством формулы (3). Во-первых, постоянная часть (3), а именно произведения A^*A и A^*b, могут быть вычислены единовременно. При этом, операция умножения транспонированной матрицы на саму себя может быть легко распараллелена между процессорами (рабочими станциями) по тому же математическому алгоритму, что и на суперкомпьютерах с разделяемой памятью.

Оставшаяся часть алгоритма регуляризации, состоящая в последовательном решении линейных систем (3) с разными $\alpha = \alpha_p$, однако, может быть проведена параллельно. Это возможно благодаря использованию для решения таких систем метода квадратного корня – неитерационного метода, прямой ход которого состоит в преобразовании матрицы коэффициентов уравнений, а обратный в последовательном получении вектора-решения. Т.к. не требуется знать результатов вычислений внутреннего цикла предыдущей итерации во время прямого хода, то эти вычисления для разных $\alpha = \alpha_p$ могут проходит парал-

лельно. По окончании же преобразований одним из компьютеров матрицы для $\alpha = \alpha_p$ результат должен быть передан компьютеру, рассчитывающему итерацию с $\alpha = \alpha_{p+1}$ и т.д. Выигрыш по времени следует из того, что время, тратящееся на прямой ход, намного больше времени проведения обратного.

Для реализации описанных алгоритмов создана программная архитектура, позволяющая автоматизировать процесс параллельного расчёта на кластере, без привлечения со стороны пользователя дополнительного внимания к аспектам функционирования сети, механизмам назначения заданий компьютерам и сбора статистической информации.

Подсистема организации параллельных вычислений состоит из четырёх компонентов: главного (ГЦЗ) и подчинённого центра запуска (ПЦЗ), главного (ГММ) и подчинённого (ММ) математического модуля. Первые два реализованы в виде служб операционной системы, что позволяет им быть автоматически доступными на всём протяжении работы компьютера. Все компоненты также реализованы в виде серверов RPC, что необходимо для реализации сетевого взаимодействия между компонентами и посылки сообщений и данных между ними. «Главные» модули располагаются на машине, где установлен «PHS», «подчинённые» – тиражируются на рабочие станции.

Главный центр запуска решает задачу прозрачного взаимодействия с «PHS» для получения команд пользователя, поиска в сети компьютеров с установленным ПЦЗ и запуска процесса ГММ для организации вычислений как на своём, так и через ПЦЗ на других компьютерах.

Подчинённый центр запуска решает задачу взаимодействия с ГЦЗ для приёма команд и запуска процесса ММ на компьютере-участнике расчёта.

Главный математический модуль запускается в виде отдельного процесса операционной системы и проводит все операции по организации конкретного расчёта: рассылка данных, назначение заданий ММ, координация их работы и т.д.

Математический модуль содержит код основных математических алгоритмов, автономно выполняется на всех рабочих станциях и взаимодействует с ГММ.

Для сравнения времени работы последовательного и параллельного алгоритмов введём коэффициенты ускорения S_m и эффективно-

сти E_m :

$$S_m = \frac{T_1}{T_m}, \quad E_m = \frac{S_m}{m},$$

где T_m – время параллельного алгоритма на кластере из m рабочих станций, T_1 – время выполнения последовательного алгоритма на одной машине. T_m представляет собой совокупность чистого времени счёта и накладных расходов на подготовку и пересылку данных.

Для оценки эффективности распределённых вычислений была решена задача о распределении напряжений в режущем инструменте из твёрдого сплава ВК-6 с монопокрытием TiN толщиной 6 мкм (рис. 1).

При решении данной задачи без применения регуляризирующих алгоритмов возникает серьёзная неустойчивость решения уже при

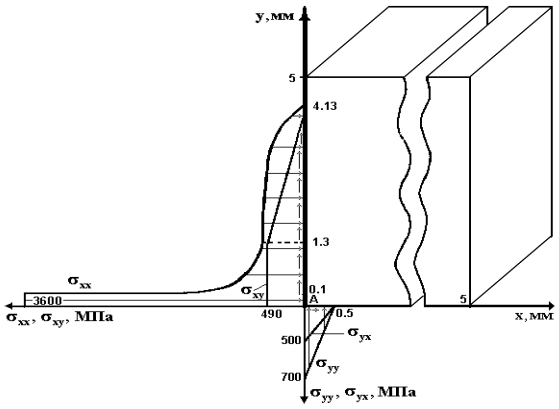


Рис. 1. Расчетная схема задачи о распределении на-

толщине монослоя 40 мкм, которую удаётся исключить путём использования представленной методики для инструментов с реальными толщинами покрытий.

Стоит заметить, что применение метода сопряженных градиентов для данной задачи оказалось невозможным из-за расходимости этого метода уже при итерации внутреннего цикла соответствующей $\alpha = 0{,}022539$. Решение, получающееся на предыдущих итерациях, не является удовлетворительным исходя из получающейся картины распределения напряжений в резце.

Хотя разница между соседними уравнениями в смещениях равна нулю, переход к двойной степени точности не приводит к изменениям в решении, сколько-нибудь меняющим характер диаграмм напряжённого состояния.

В табл. 1 приведены времена распределённого расчёта решения данной задачи, а также коэффициенты S_m и E_m при числе линейных уравнений равном 3570.

Таблица 1

m	T_m, мин	S_m	E_m
1	836	–	–
2	422	1,98	0,99
6	140	5,97	0,99
12	71	11,77	0,98

Как видно из таблицы ускорение S_m растёт практически линейно. Учитывая большую автономность выполняемых каждой машиной задач и малые накладные расходы на поддержку работы параллельных алгоритмов и обмены результатами, можно прогнозировать стабильное увеличение производительность при дальнейшем росте числа компьютеров в кластере вплоть до числа равного количеству итераций внутреннего цикла метода регуляризации.

Для проведения вычислений использовалась сеть персональных ЭВМ (Pentium III) пропускной способностью 100Мбит/с с установленной операционной системой Windows NT или Windows 2000. Для компиляции и отладки программного кода системы использовался компилятор Microsoft Visual C++ 6.0.

Литература

1. Расчет напряжений в породных массивах методом граничных интегральных уравнений /А.И. Олейников и др.: Кривой рог: НИГРИ, 1982.
2. *Олейников А.И.*, *Кузьмин А.О*. Применение численного метода граничных элементов к решению кусочно-однородных задач линейной теории упругости // Синергетика. Самоорганизующиеся процессы в системах и технологиях: Материалы международной научной конференции (г. Комсомольск-на-Амуре 21-26 сент. 2000г.). - Комсомольск-на-Амуре: Комсомольский-на-Амуре гос.

техн. ун-т. 2000. С. 122–125.
3. *Crouch S.L.*, *Starfield A.M.* Boundary element method in solid mechanics. Boston: George Allen & Unwin, 1983.
4. *Тихонов А.Н.*, *Арсенин В.Я.* Методы решения некорректных задач. Главная редакция физико-математической литературы изд-ва «Наука», М., 1974.
5. *Старостенко В.И.* Устойчивые численные методы в задачах гравиметрии. Киев: Наукова думка, 1978.

К РЕШЕНИЮ ЗАДАЧ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ СЛОЖНЫХ СИСТЕМ НА КЛАСТЕРАХ

С.И. Олзоева

Восточно-Сибирский государственный технологический университет г. Улан-Удэ

Среди иссследований в области высокопроизводительных вычислений на кластерах основное внимание уделяется методам распараллелливания задач численного моделирования, использующим парадигму параллельного программирования SPMD (Single Program – Multiple Data). При этом ускорение решения задач базируется на распараллеливании по данным. К этому направлению относятся задачи аэрогазодинамики, гидродинамики, линейной алгебры и др. Вместе с тем, судя по публикациям в области кластерных вычислений, гораздо меньше внимания уделяется задачам, решение которых предполагает использование парадигмы программирования MPMD (Multiple Program – Multiple Data). Этот вариант параллельных вычислений базируется на декомпозиции алгоритма вычислений, когда программа решения одной задачи разбивается на программы, реализующие алгоритмы решения подзадач, которые существенно отличаются и могут иметь разный объем выполняемых операций.

Для реализации на параллельных вычислительных платформах имитационного моделирования сложных систем, признанного необходимым и эффективным инструментом системного анализа, наиболее подходит именно модель вычислений MPMD по следующим причинам:

– сложные системы состоят, как правило, из параллельно функ-

ционирующих компонент на разных уровнях иерархической структуры системы. Поэтому имитационная модель (ИМ) сложной системы содержит много модулей, соответствующих различным элементам и подсистемам моделируемого объекта;

– разнородность подсистем и элементов, составляющих сложную систему, порождает и разнородность математических схем, описывающих функционирование различных элементов. Отдельные элементы и подсистемы могут быть описаны, например, обыкновенными дифференциальными уравнениями, системами массового обслуживания, конечными автоматами и т. д. Отсюда следует разнородность программной реализации модулей ИМ;
– большие вычислительные затраты (времени и памяти) в имитационном моделировании сложных систем связаны с размерностью исследуемых систем, а также объемом вычислений для получения статистически обоснованных результатов. Кроме того, в имитационном моделировании решение всегда носит частный характер, соответствующий фиксированным значениям параметров системы и начальных условий. Для анализа системы также необходимо многократно моделировать ее процесс функционирования, варьируя исходные данные задачи.

Однако, несмотря на то, что современная концепция построения ИМ сложных систем естественным образом вписывается в модель параллельных вычислений MPMD, требуется переработка принципов функционирования моделирующих программных систем для их выполнения на параллельных вычислительных системах. Для определения сути этой проблемы рассмотрим общие принципы сложившихся подходов к имитационному моделированию.

При всем многообразии и различии предметных областей имитационного моделирования, все ИМ сложных систем состоят их следующих составляющих:

– модулей, воспроизводящих поведение различных элементов и подсистем моделируемого объекта;
– управляющей программы, реализующей логику поведения модели, т.е. квазипараллельное и коррелированное течение модельных процессов;
– модулей, обслуживающих процесс моделирования, выполняющих функции сбора данных и статистической обработки результатов моделирования, ввод – вывод, планирование машин-

ного эксперимента.

Вследствие последовательного характера обработки информации в однопроцессорной машине, параллельные процессы, происходящие в модели преобразуются с помощью алгоритма управления модельным временем в последовательные. Противоречие между параллельностью модельных процессов и последовательным характером квазипараллельного процесса является причиной их неполного соответсвия [2]. Часто пренебрегают возможностью одновременного наступления двух или более событий. Если такая возможность допускается, то в управляющей программе предусматривается процедура, определяющая последовательность реакции модели на происходящие одновременные события, т.е. задается порядок их обработки. Все это предполагает предварительный теоретический анализ процесса функционирования системы. Кроме того, в управляющей программе реализованы процедуры ведения календаря событий, т. е размещения событий в списке в порядке возрастания предсказанного времени событий, что также требует значительных затрат времени для обработки списков событий, поиска места для записи, особенно с увеличением размерности моделируемого объекта. При пособытийных методах моделирования затрата времени квадратично зависит от числа наступивших событий [2].

При параллельном моделировании эти проблемы снимаются, но частично. Так как при параллельных вычислениях также необходимо программировать специальные действия по координации работы подзадач, размещенных на разных процессорах. Поскольку действия, выполняемые различными модулями ИМ зависят от действий и состояний других элементов, они должны быть скоординированы во времени или синхронизированы. Кроме того, в параллельных вычислениях на кластерах, также нужно минимизировать обмен данными между задачами, так как пересылка данных наиболее «времяемкий» процесс.

Для программ имитации характерно то, что они представляют собой сильно-связный код. Это обусловлено организацией управляющей программы имитации, осуществляющей синхронизацию во времени параллельно протекающих процессов. Здесь имеет место жесткая централизация управления и централизация базы данных моделирования, с которой работают и управляющая программа, и программы имитации процессов. Если же просто распределить выполнение модулей моделирующей программы по процессорам кластера, то многочисленные передачи информации между компонентами имитационной моде-

ли не будут способствовать эффективному ускорению процесса имитации.

В связи с этим необходима раработка новых способов построения управляющей программы, методов формирования распределенной базы моделирования, а также методов крупноблочного распараллеливания моделирующего алгоритма. Следовательно, необходима разработка средств, способных оказать исследователю помощь при распараллеливании алгоритмов имитационного моделирования, за счет автоматизации работ по крупноблочному распараллеливанию ИМ и выдачи программисту результатов измерений отдельных параметров качества, получаемого проекта распределенной (параллельной) имитационной программы. Это автоматизированное средство предназначается для отделения аспектов, связанных с разработкой программного обеспечения ИМ, от вопросов собственно организации параллельных вычислений, и должно содержать в себе научно-обоснованную методологию конструирования структуры распределенного программного обеспечения (ПО) ИМ систем. Назовем такую систему – автоматизированным средством организации распределенного имитационного моделирования (АСОРИМ).

Функционирование АСОРИМ должно состоять из следующих технологических этапов:

- анализ структуры ПО имитационной модели, выбор алгоритма распараллеливания (декомпозиции ИМ);
- декомпозиция ИМ.

Оценка качества вариантов проектов распределенной имитационной модели, определение возможных временных факторов.

Выбор окончательного проекта распределенной (параллельной) ИМ.

Исходя из общей концепции построения ИМ сложных систем, для создания ИМ систем определяются:

- базовые модели, описывающие различные аспекты функционирования подсистем системы;
- структура их взаимодействия, т.е. как эти модели соединяются друг с другом, образуя иерархическую структуру;
- алгоритм синхронизации, реализуемый управляющей программой модели.

Полученная, таким образом, составная модель определяет способ связывания отдельных составляющих моделей и несет в себе информацию:

– о совокупности входных и выходных портов;
– о соединениях, осуществляющих связь выходных портов одних компонентов с входными портами других компонентов.

Такому представлению имитационных моделей наиболее полно отвечает агрегативная модель сложных систем [1], достоинством которой являются единая стандартная форма математической модели исследуемой системы и возможность исследования систем любой сложности и размерности. Далее в качестве исследуемой системы моделирования будем рассматривать именно агрегативную модель.

Для агрегативных систем декомпозиция управляющей программы и базы данных моделирования должна основываться на результатах исследования схем сопряжения агрегатов. Каждый элемент агрегативной системы, характеризуется множеством входных контактов $[X_{ji}]$ и множеством выходных контактов $[Y_{kl}]$, где j и k – номера элементов системы, а i и l – порядковые номера входных и выходных контактов соответственно. Схемы сопряжения агрегатов задаются в виде таблицы, столбцы и строки которой, нумеруются двойными номерами (j, i) и (k, l) соответственно, а на пересечениях помещены единицы для контактов X_{ji} и Y_{kl}, соединенных элементарным каналом. Эти таблицы представляют собой матрицы смежности ориентированных графов, вершинами которых являются контакты, а ребрами элементарные каналы. Т.е. представляют собой информационный граф агрегативной системы, характеризующий взаимодействие между элементами системы. Исследуя схему сопряжения методами теории графов, можно выделить классы наиболее связанных элементов системы, с целью крупноблочного распараллеливания моделирующего алгоритма этой системы.

В такой постановке задача декомпозиции ПО ИМ представляет собой задачу разбиения графов на минимально связанные между собой подграфы. Такие задачи носят логико-комбинаторный характер, трудоемкость их решения растет комбинаторно с линейным ростом размерности задачи. Однако, существуют методы решения этих задач, ведущие к существенному сокращению перебора [5] и удобные для реализации на ЭВМ. В нашем случае, выбор конкретного метода зависит от структуры исследуемой агрегативной модели, а также от заложенных в постановку задачи определенных требований, которым должно удовлетворять искомое разбиение. Например, может быть задано число подграфов, в зависимости от предполагаемого количества

используемых процессоров вычислительной сети, либо может не существовать такого ограничения. Зависимость от структуры агрегативной модели выражается степенью связности элементов системы и видом таблицы сопряжения элементов по которой строится матрица смежности информационного графа.

В связи с этим, АСОРИМ должен располагать возможностью реализации разных подходов к исследованию графов на предмет разбиения. Выбор того или другого метода происходит на первом этапе функционирования АСОРИМ по результатам анализа структуры исследуемой ИМ.

Выходом второго этапа функционирования АСОРИМ являются варианты декомпозиции ПО ИМ, представляющие собой классы наиболее связанных элементов системы, а также сформированные таблицы сопряжения выделенных классов.

Элементы агрегативной системы, входящие в один класс, представляют единый блок, для которого формируется своя управляющая программа и база данных моделирования, включающая описание элементов класса и календарь событий для этих элементов. Затем строится главная управляющая программа со своей базой данных моделирования. Далее происходит распределение полученных блоков моделирующей программы по процессорам вычислительной сети, а главная управляющая программа синхронизирует во времени параллельно выполняющиеся вычислительные блоки. Но прежде необходимо получить возможные временные факторы распределенного имитационного моделирования с целью выбора окончательного варианта декомпозиции ИМ.

Основное внимание в области распределенного имитационного моделирования уделяется схемам синхронизации, доказательству их правильности и реализации распределенных моделей [3]. В меньшей степени исследованы вопросы анализа качества функционирования распределенной ИМ с целью определения возможных временных факторов. Временной фактор определяется как отношение времени выполнения ИМ на одном процессоре к времени выполнения на нескольких процессорах. Обычно имеется несколько вариантов декомпозиции ИМ на составные блоки. Определение наибольшего возможного временного фактора до начала процесса моделирования позволит эффективно организовать вычислительный процесс. В связи с этим предлагается способ оценки потенциального параллелизма при заданном варианте декомпозиции ИМ, позволяющий определить возможный

временной фактор. Предлагаемый способ основан на применении математического аппарата теории случайных импульсных потоков [4] и позволяет учесть характеристики неоднородности вычислительной платформы, на которой предполагается реализация распределенной ИМ. В докладе подробно излагается способ, позволяющий получить нижнюю и верхнюю оценки возможного временного фактора [7]. Эти значения позволяют однозначно выбрать вариант декомпозиции имитационной модели.

Таким образом, предлагается следующая методика организации вычислительного процесса для распределенного имитационного моделирования систем на кластерных вычислительных системах. Шаги 1, 2 выполняются АСОРИМ:

Декомпозиция имитационной модели

Входные данные: таблица сопряжения элементов моделируемой системы.

Выходные данные: классы наиболее связанных элементов системы.

Определение возможных временных факторов для вариантов декомпозиции, полученных на шаге 1.

Входные данные: параметры, полученные из таблицы сопряжения элементов.

Выходные данные: нижняя и верхняя оценки временного фактора для каждого варианта декомпозиции ИМ.

Построение иерархической управляющей программы для выбранного варианта распределенной ИМ и формирование распределенной базы данных моделирования.

Использование кластерной вычислительной системы для решения задач имитационного моделирования, требует разработки не только методов проектирования распределенных моделирующих алгоритмов, но и проведения исследований по выбору программных средств, наиболее эффективно реализующих взаимодействие параллельных блоков имитационной модели. Библиотеки интерфейса передачи сообщений MPI (Message Passing Interface) реализованы практически во всех современных кластерных системах, также могут использоваться и в локальных сетях.

Однако, использование функций MPI или PVM возможно только в программах написанных на Си или Фортране, тогда как существует множество уже отлаженных модулей имитационных моделей,

использующих и другие языки программирования, например Паскаль. В связи с этим, при построении распределенного моделирующего алгоритма, можно использовать интерфейс сокетов TCP/IP, IPX или др.

Нами были произведены исследования по использованию функций протокола IPX. Использовалась методика измерений пропускной способности двунаправленных обменов, предложенная в работе [6]. Результаты показали, что эффективность приема передачи данных составляет около 99% от пропускной способности физического канала. При этом передавалось 130 тыс. пакетов длиной 546 байт и количество неверно принятых пакетов составило – 0. Поэтому при построении сетевой имитационной модели СУРБД ЛВС [8] были использованы функции протокола IPX для обмена информацией между блоками имитационной модели.

Использование кластерных вычислительных систем, как технической базы для моделирования сложных систем, позволит решить проблемы ресурсоемкости и вычислительных затрат в имитационном моделировании.

Литература

1. *Бусленко Н.П.* Моделирование сложных систем. М.: Наука, 1978.
2. Технология системного моделирования / Под ред. С.В. Емельянова и др. М.: Машиностроение; Берлин: Техника, 1988.
3. *Райтер Р.*, *Вальран Ж.С.* Распределенное имитационное моделирование дискретно-событийных систем. М.: Мир. ТИИЭР. Т. 77. №1. 1989. С. 245–262.
4. *Лифшиц А.Р.*, *Биленко А.П.* Многоканальные асинхронные системы передачи информации. М.: Связь, 1974.
5. *Горбатов В.А.* Теория частично упорядоченных систем. М.: Советское радио, 1976.
6. *Андреев А.Н.*, *В.В. Воеводин.* Методика измерения основных характеристик программно-аппаратной среды. http://parallel.ru.
7. *Олзоева С.И.* Способ оценки потенциального параллелизма для распределенного имитационного моделирования систем / Сб. тр. Международ. конф. «Математика, ее приложения и математическое образование», Улан-Удэ, 2002. Ч. 2. С. 27 – 32.
8. *Олзоева С.И.*, *Лоскутов Р.В.* Сетевая имитационная модель системы управления распределенными базами данных. Материалы Всероссийской научно-техн. конф. «Теоретические и прикладные

вопросы современных информационных технологий», Улан-Удэ, 2000. С. 111–112.

ВОПРОСЫ ОРГАНИЗАЦИИ ПОДГОТОВКИ СПЕЦИАЛИСТОВ ПО ПАРАЛЛЕЛЬНЫМ ВЫЧИСЛЕНИЯМ

Н.Д. Пиза, Н.Н. Хохлов, Р.К. Кудерметов

Запорожский национальный технический университет, Украина

Проблемы подготовки специалистов по высокопроизводительным вычислениям становятся все более актуальными, несмотря на отсутствие массового внедрения параллельных вычислительных систем. Анализ тенденций развития вычислительной техники и роста сложности задач, подлежащих решению, показывает, что необходимо разрабатывать системное и прикладное, и, быть может, перерабатывать существующее прикладное программное обеспечение для параллельных вычислительных систем.

Исходя из этого и ориентируясь на стратегические принципы и программы, изложенные в [1], в Запорожском национальном техническом университете в рамках специальности «Компьютерные системы и сети» открыта специализация «Высокопроизводительные вычислительные системы и параллельные вычисления». По этой специализации в настоящее время разрабатывается методическое обеспечение по дисциплинам «Архитектура высокопроизводительных вычислительных систем (ВВС)», «Аппаратные средства ВВС», «Системная интеграция и оптимизация средств ВВС», «Современные кластерные и параллельные вычислительные системы», «Программирование в MPI, OpenMP и других параллельных средах», «Операционные системы и системное программирование ВВС».

В число дисциплин специальности «Компьютерные системы и сети» входит «Параллельное программирование». Дисциплина охватывает основные направления технологии распараллеливания алгоритмов и разработки параллельных программ, вместе с рассмотрением основных характеристик параллельных алгоритмов. В качестве примера языка параллельного программирования и с целью демонстрации возможных подходов к языковой реализации топологии параллельной задачи рассматривается язык Parallaxis [2] и основные функции биб-

лиотеки MPI [3].

Практические задания по дисциплине «Параллельное программирование» ориентированы на разработку параллельных алгоритмов и включают такие задачи, как сложение по методу сдваивания, известный также как каскадная схема [4], матричное умножение, вычисление числа π, решение задачи Дирихле для уравнения Пуассона, решение уравнения теплопроводности [5] и другие. На первом этапе разработки практических заданий для имитации параллельных и векторных машин использовался класс потоков (нитей), реализованный в среде Borland C++ Builder. В настоящее время разрабатываются лабораторные задания на базе функций библиотеки MPI. Разработаны магистерские дипломные проекты «Параллельная реализация быстрого преобразования Фурье», «Параллельная СУБД», «Моделирование пространственно распределенных процессов на параллельной вычислительной системе», «Параллельная вычислительная система на базе компьютерной сети с использованием библиотеки MPI».

В качестве параллельной вычислительной среды используются локальные компьютерные сети, в частности, практические занятия студентов специальности «Компьютерные системы и сети» проводятся в четырех компьютерных классах, каждый из которых включает по 25 рабочих станций Pentium III/866 MHz и Pentium IV/1.7 GHz. В настоящее время ведется подготовка к установке кластера, с использованием коммуникационного интерфейса SCI (Scalable Coherent Interface).

В рамках научной работы по параллельным системам и программированию проводятся исследования в области параллельных численных методов интегрирования обыкновенных дифференциальных уравнений для моделирования одного класса динамических систем, визуализации параллельных вычислений, разрабатывается оболочка для организации параллельных вычислений в компьютерной сети и система распределения параллельных заданий в компьютерной сети университета.

Литература

1. *Домрачев В.Г.*, *Ретинская И.В.*, *Скуратов А.К.* Стратегия и практические пути подготовки специалистов по высокопроизводительным вычислениям // Информационные технологии, 2001. №7. С. 28–35.
2. *Бройнль Т.* Паралельне програмування: Початковий курс: Навч. посібник / Втуп. слово А. Ройтера; Пер. з нім. В.А. Святного. Киев:

Вища школа, 1997.
3. *Корнеев В.В.* Параллельные вычислительные системы. М.: Нолидж, 1999.
4. *Гергель В.П., Стронгин Р.Г.* Основы параллельных вычислений для многопроцессорных вычислительных систем: Учеб. пособие. Нижний Новгород: Изд-во ННГУ, 2000.
5. *Ортега Дж.* Введение в параллельные и векторные методы решения линейных систем. М.: Мир, 1991.

ПРОГРАММИРОВАНИЕ НА JAVA ДЛЯ МНОГОПРОЦЕССОРНЫХ СИСТЕМ

К.Г. Попов

Отдел математики (*филиал г. Сыктывкар*),
Институт Математики и Механики. УрО РАН

Многопроцессорные вычислительные системы дают возможность не только увеличить скорость обработки информации и надежность ее хранения, но и более последовательно реализовать основные достоинства Объектно Ориентированного программирования (ООП). Основная парадигма этого подхода к программированию может быть образно представлена как пространство (космос), в котором локализованы экземпляры классов (изолированные объекты), обменивающиеся сообщениями в процессе выполнения программы. В таких языках высокого уровня как Java или C++ существует достаточно ресурсов и для размещения своих объектов на любом наборе связанных процессоров, и для организации информационных потоков между ними. Однако, на практике используются специальные расширения языков, например, MPI (интерфейс обмена сообщениями), которые решают те же задачи, но с помощью средств низкого уровня. Это приводит к существенному ограничению возможностей языков высокого уровня, поскольку программирование возвращается к своему процедурному прошлому. Такая цена скорости счета и простоты использования недопустима или чрезмерна. Дело в том, что ориентированный на пользователя (а не на машину) язык дает возможность в процессе программирования мыслить понятийно, абстракциями высокого уровня. Это более адекватно мыслительному процессу человека, чем что-либо явно опирающееся

на бинарный код. Часто ООП это единственный путь, позволяющий моделировать природные явления в виртуальной реальности. Несомненно, оптимальные технологии использования многопроцессорных систем, свободных от описанных недостатков, будут созданы в ближайшем будущем.

В качестве иллюстрации вышесказанного мы предлагаем рассмотреть компьютерный эксперимент из области статистической геометрии, описанный в [1]. Суть его состоит в следующем: плоскость покрывается без разрывов равносторонними шестиугольниками, углы которых случайным образом берутся из резервуара большой емкости, где они распределены по известному закону. После накопления статистически значимого числа построенных шестиугольников анализируется функция распределения углов. Оказывается, что она не зависит от функции распределения углов, использованных для строительства этого кривого паркета. Описанный эксперимент был реализован в стандартном сетевом компьютерном классе. Один процессор выступал в роли клиента, а десять других – в роли удаленных серверов. Все программы были реализованы на языке Java. Хотя существует возможность решения данной задачи в технологии CORBA или апплет-серверного приложения, мы ограничились более простым решением, применив инструментарий RMI (вызов удаленного метода). Сервера, помимо реализации интерфейса источников удаленных сообщений, инкапсулировали в себе объекты класса-генератора шестиугольников с требуемыми свойствами. Клиент занимался непосредственной сборкой паркета, связывая нитями легковесных процессов точки размещения очередных шестиугольников с удаленными «фабриками» (серверами), где они производятся. Кроме прозрачности реализации замысла, возможности построить интерфейс визуализации и достаточной скорости счета, мы имеем возможность в реальном времени анализировать происходящие события.

Литература

1. *Займан Дж.* Модели беспорядка. М. Мир. 1982.

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ РАСПРЕДЕЛЕНИЯ РЕСУРСОВ В ИЕРАРХИЧЕСКИХ СИСТЕМАХ С ЛЕКСИКОГРАФИЧЕСКИМ

УПОРЯДОЧЕНИЕМ ЭЛЕМЕНТОВ

М.Х. Прилуцкий, Л.Г. Афраймович
Нижегородский государственный университет им. Н.И. Лобачевского,

Общая математическая модель

Пусть $G = (V,A)$ – многоуровневая иерархическая система порядка N, $A \subseteq V^2$, где V – множество узлов системы и соответственно $|V| = N$. В системе G распределяется однородный ресурс. Элементы множества A характеризуют возможные направления распределения ресурса. Так, если $(i,j) \in A$, то это означает, что ресурс, находящийся в узле i, может быть переправлен в узел j, $i \in V, j \in V$.

Все множество узлов системы разбивается на три подмножества, соответствующих источникам ресурса V_s, коммуникационным узлам V_k, потребителям ресурса V_u.

На каждый из узлов системы наложены ограничения, связанные с возможными режимами их работы. Для узлов из множества V_s это ограничения на объемы ресурса, которые поставляют (производят) узлы; для узлов множества V_u это ограничения на объемы ресурса, которые получают узлы; для узлов множества V_k это ограничения на объемы ресурса, которые передают (ретранслируют) эти узлы.

Обозначим через $Q_\nu = \{i|(\nu,i) \in A\}$ – множество узлов системы, непосредственно следующих после узла ν, $R_\nu = \{i|(i,\nu) \in A\}$ – множество узлов системы, непосредственно предшествующих узлу ν, $\nu \in V$, причем $R_j = \varnothing, j \in V_s$, $Q_j = \varnothing, j \in V_u$.

Допустимым вариантом распределения ресурсов (допустимым решением) называется неотрицательная действительная функция $f(i,j)$, заданная на множестве A, для которой выполняются условия:

$$A_i \le \sum_{j \in Q_i} f(i,j) \le B_i \ , \ i \in V_s, \qquad (1)$$

где $[A_i, B_i]$ – сегмент, соответствующий ресурсным ограничениям узла i, $0 \le A_i \le B_i < \infty$, $i \in V$.

Исследование математической модели

В работе [1] исследована частная модель системы распределения однородного ресурса в сильносвязных иерархических системах. Для проверки совместности системы ограничений частной модели в работе [1] разработан метод приведенных границ, имеющий линейную временную оценку. В работе [2] предложены методы решения задач распределения однородного ресурса в более общей постановке (иерархия системы произвольная), для корректной работы которых требуется определять совместность систем линейных ограничений транспортного типа. Для проверки совместности систем линейных алгебраических неравенств могут быть использованы классические результаты общей теории линейных неравенств (см. например, [3]). Поскольку универсальные алгоритмы проверки на совместность систем линейных ограничений имеют экспоненциальную трудоемкость (поиск ненулевого минора в матрице ограничений системы), то они не могут быть использованы для исследования большеразмерных систем. С другой стороны, существующие полиномиальные алгоритмы решения задач линейного программирования (см. например, метод эллипсоидов, предложенный в работе [4]) позволили надеяться и в нашем случае найти эффективные (полиномиальные) методы решения поставленных задач. В основу этих методов положена схема эквивалентного представления многоуровневых иерархических систем в виде сетевых потоковых моделей с ограниченными пропускными способностями дуг. Использование для решения потоковых задач известных методов (например, модифицированный метод расстановки пометок для задачи о максимальном потоке [5]) дало возможность построить процедуру решения задачи распределения однородного ресурса в многоуровневых иерархических системах произвольной структуры, временная сложность которой равна $0(n^3)$.

Лексикографическое упорядочение для контролируемых элементов

Среди множества узлов $V_s \cup V_u$ выделим подмножество контролируемых узлов – K, $| K | = k_0$, узлов, которые определяют эффективность функционирования системы. Каждый из контролируемых узлов i задает на соответствующем ему сегменте $[A_i, B_i]$ бинарное

отношение, отражающее его предпочтение относительно объема ресурса, который он будет предоставлять (для $i \in V_s$) или получать (для $i \in V_u$).

Для каждого контролируемого элемента ν введем совокупность из p+1 вложенных друг в друга сегментов $S_\nu^{(t)}$, $t \in \{0,1,..., p\}$, $S_\nu^{(t)} \subseteq S_\nu^{(t+1)}$, $S_\nu^{(p)} = [A_\nu, B_\nu]$.

Рассмотрим функцию

$$\psi_\nu(y_\nu, S_\nu^{(0)}, S_\nu^{(1)},..., S_\nu^{(p)}),$$

где

$$y_v = \begin{cases} \sum_{i \in Q_v} f(v,i), \text{ при } \nu \in V_s \\ \sum_{i \in R_v} f(i,v), \text{ при } \nu \in V_u \end{cases},$$

принимающую значение *t*, если $y_\nu \in S_\nu^{(t)}$, $y_\nu \notin S_\nu^{(t-1)}$, $t \in \{0,1,..., p\}$.

Пусть система совместна, тогда требуется найти такой допустимый вариант распределения ресурсов (такое допустимое решение системы (1)–(3)), при котором достигают экстремальные значения частные критерии оптимальности, формализованные в виде функций, определенных для контролируемых узлов:

$$\psi_\nu(y_\nu, S_\nu^{(0)}, S_\nu^{(1)},..., S_\nu^{(p)}) \to \min, \quad \nu \in K. \tag{4}$$

Поставленная задача является задачей многокритериальной оптимизации, поэтому для ее решения необходимо выбрать схему компромисса, в качестве которой предлагается лексикографическое упорядочивание частных критериев оптимальности.

Введем, как и в [2], (p+1)-ичную k_0-мерную решетку, на которой определим порядок Π. Каждому узлу решетки $\mathbf{r}$ поставим в соответствие систему $C(\mathbf{r})$, содержащую не зависящие от узла решетки ограничения (1)–(3) для всех $i \in V \setminus K$, и ограничения для контролируемых узлов, зависящие от координат узла $\mathbf{r}$ следующим образом: для $K = \{i_1, i_2,..., i_{k_0}\}$, если $r_j = t$, то $y_{i_j} \in S_{i_j}^{(t)}$. На множестве узлов решетки зададим двузначную функцию $g(\mathbf{r})$, принимаю-

щую значение 1, если соответствующая узлу $\mathbf{r}$ система $C(\mathbf{r})$ совместна, и 0 в противном случае. В качестве порядка Π рассмотрим лексикографическое отношение порядка: $\mathbf{r}^1 \Pi \mathbf{r}^2$ тогда и только тогда, когда для некоторого s, $s = \overline{1, k_0}$, $r_s^1 < r_s^2$ и одновременно $r_i^1 = r_i^2$ для $i = \overline{1, s-1}$. Тогда задача заключается в определении оптимального узла решетки $\mathbf{r}^0$, такого, для которого $g(\mathbf{r}^0) = 1$, и выполняются условия: $\mathbf{r}^0 \Pi \mathbf{r}$ для всех узлов решетки, для которых $g(\mathbf{r}) = 1$. Так как $S_\nu^{(t)} \subseteq S_\nu^{(t+1)}$, $t = \overline{0, p}$, $\nu = \overline{1, k_0}$, то, если $\mathbf{r}^1 \le \mathbf{r}^2$, то $g(\mathbf{r}^1) \le g(\mathbf{r}^2)$, отсюда следует, что введенная функция $g(\mathbf{r})$ является монотонной. Монотонность функции $g(\mathbf{r})$ позволяет предложить алгоритм поиска оптимального узла решетки, который заключается в последовательном вычислении координат узла решетки, осуществляемом с помощью бинарного поиска, на каждом шаге которого определяется значение функции $g(\mathbf{r})$, т.е. проверяется на совместность система типа (1)–(3).

Таким образом, предложенный алгоритм решения задачи будет включать в себя порядка $k_0 \log_2(p+1)$ проверок совместности систем типа (1)–(3). Для проверки совместности системы типа (1)–(3) предложена схема ее эквивалентного представления в виде сетевой потоковой модели с ограниченными пропускными способностями дуг и решения для этой модели задачи поиска максимального потока, например, модифицированным методом расстановки пометок (см.[5]), временная сложность которого равна $0(n^3)$. Поэтому, при большой размерности решение поставленной задачи требует значительных временных затрат, что обуславливает поиск альтернативных подходов к ее решению. Примерами таких подходов могут являться параллельные вычисления ([6]).

Параллельные алгоритмы распределения ресурсов

Использование многопроцессорных систем для решения задач распределения ресурсов позволяет за счет распараллеливания алгоритмов существенно уменьшать время решения задач.

Пусть h_0 – количество параллельно работающих процессоров. Тогда, если $h_0 \geq p+1$, то поиск оптимального узла решетки может быть осуществлен за k_0 шагов, на каждом шаге проверяя на совместность одновременно $p+1$ систем типа (1)–(3), тогда время решения задачи сокращается в $\log_2(p+1)$ раз. Если $h_0 < p+1$, то на каждом шаге достаточно разбить множество $\{0,1,...,p\}$ на h_0 равномощных подмножеств, осуществлять двоичный поиск внутри каждого подмножеста, а затем выбрать оптимальную координату узла, и так для каждой координаты. В этом случае время решения задачи сокращается в $\log_2(p+1)/\log_2((p+1)/h_0)$ раз.

При нахождение оптимального узла решетки возможен поиск вспомогательных величин, которые могли бы сократить временные затраты. Если $h_0 \geq k_0$, тогда возможен параллельный поиск величин $x_i \in \{0,1,...,p\}$, $i = \overline{1,k_0}$, таких, что $g(p,p,..,p,x_i,p,p,...,p) = 0$, и x_i принимает наибольшее из возможных значений (при совместности соответствующих систем для всех значений элементов множества $\{0,1,...,p\}$, значение величины x_i полагается равным –1). Тогда при нахождение i-ой координаты оптимального узла решетки $\mathbf{r}^0$ поиск достаточно осуществлять на множестве $\{x_i+1, x_i+2,..., p\}$. Это возможно в силу монотонности функции $g(\mathbf{r})$. Если $h_0 < k_0$, то достаточно разбить множество $\{1,2,...,k_0\}$ на h_0 равномощных подмножеств для каждого из которых последовательно осуществлять поиск соответствующих вспомогательных величин x_i. Предложенная процедура не всегда приводит к уменьшению времени решения задачи (например, в случае, когда $\mathbf{r}^0 = (0, 0, \ldots, 0)$), хотя очевидно, что во многих реальных задачах, предложенная процедура сокращает временные затраты.

В случае, когда $h_0 = 2$, предложенная процедура может быть модифицирована. Пусть на одном процессоре, используя двоичный поиск по значениям из множества $\{0, 1, \ldots, p\}$, определяется i-я координат ν_i оптимального узла решетки $\mathbf{r}^0$, проверяя на каждом шаге поиска совместность системы типа (1)-(3). Тогда на другом процессоре может

параллельно определяться максимальное значение вспомогательной величины $x \in \{0,1,...,p\}$ такой, что $g(\nu_1, \nu_2, ..., \nu_{i-1}, p, x, p, p, ..., p) = 0$, где ν_1, ν_2, …, ν_{i-1} – значения уже найденных на предыдущих шагах координат оптимального узла $\mathbf{r}^0$. Если $x = x_0$ (как и в предыдущем случае величина x_0 может быть равна –1), то монотонность функции $g(\mathbf{r})$ позволяет осуществлять поиск $i + 1$ координаты оптимального узла решетки на множестве $\{x_0 + 1, x_0 + 2, ..., p\}$.

При поиске максимального потока в сети, которая моделирует систему ограничений (1)–(3), возможен одновременный поиск не более чем h_0 различных направленных маршрутов, увеличивающих поток. Тогда параллельно можно искать различные маршруты, пропускать поток по одному из них, а затем, если какой-либо из найденных маршрутов все еще увеличивает поток, то пропускать поток и через него, так просматривая все найденные маршруты.

Заключение

Многие реальные объекты, для которых актуальны задачи, описанные в данной работе, имеют большую размерность, и их решение с использованием однопроцессорных вычислительных систем требует значительных временных затрат. Поэтому целесообразно для решения таких большеразмерных задач использовать параллельные вычисления с их реализацией на многопроцессорных вычислительных системах.

Литература

1. *Прилуцкий М.Х.* Распределение однородного ресурса в иерархических системах древовидной структуры. Труды международной конференции «Идентификация систем и задачи управления SICPRO'2000». Москва , 26-28 сентября 2000. М.: Институт проблем управления им. В.А.Трапезникова РАН , 2000. С. 2038–2049.
2. *Прилуцкий М.Х.* Многокритериальное распределение однородного ресурса в иерархических системах // Автоматика и телемеханика. 1996. №2. С. 24–29.
3. *Черников С.Н.* Линейные неравенства. М.: Наука, 1968.
4. *Хачиян Л.Г.* Полиномиальные алгоритмы в линейном программировании // ДАН СССР. Т. 244. Вып. 5. 1979. С. 1033–1096.

5. *Пападимитриу Х.*, *Стайглиц К.* Комбинаторная оптимизация. Алгоритмы и сложность. М.:Мир, 1985.
6. *Воеводин В.В.* Математические модели и методы в параллельных процессах. М.: Наука, 1986.

БАЗИСНЫЕ ФУНКЦИИ ОТЛАДЧИКА ПАРАЛЛЕЛЬНЫХ ПРОГРАММ GEPARD И ИХ РЕАЛИЗАЦИЯ ДЛЯ МВС-1000/М

А.А. Романенко, В.Э. Малышкин

Новосибирский государственный университет

Введение

В последние годы большое распространение получили параллельные ЭВМ типа мультикомпьютер. Ежегодно появляются все новые мультикомпьтеры, способные решать все более сложные и ресурсоемкие задачи. В Новосибирском Академгородке в Сибирском Суперкомпьютерном Центре (ССЦ) [1] установлен один из таких мультикомпьютеров – МВС-1000/М. Это вычислительный комплекс кластерной архитектуры. Каждый узел содержит по два процессора Alpha-21264, 2ГБ оперативной памяти. Все узлы соединены между собой высокопроизводительной сетью Myrinet. Мультикомпьютер работает под управлением операционной системы Linux RedHat-6.2. Сейчас МВС-1000М насчитывает 8 узлов. В течение года число узлов достигнет 32. ССЦ имеет неплохой канал до московского Межведомственного Суперкомпьютерного Центра [2], что позволяет, отладив программу в Новосибирске, эксплуатировать ее в Москве. Для разработки программ на управляющем узле установлены компиляторы C/C++, Fortran, для передачи сообщений используется MPI.

При разработке параллельных программ необходимо также иметь специальные средства их отладки. К сожалению, на текущий момент времени на МВС-1000/М установлены только средства отладки последовательных программ и есть острая потребность в инструменте отладки параллельных программ.

Основные отладчики параллельных программ

Множество всех типов отладчиков можно разбить на 4 группы [3]:

1. Системы мониторинга. Они применяются, когда надо минимизировать влияние отладчика на ход выполнения программы, либо нет другого способа получения информации о поведении программы. Системы мониторинга ведут сбор информации об отлаживаемой программе с целью ее дальнейшего анализа.

2. Системы мониторинга с возможностью псевдовыполнения. Позволяют повторно (многократно) выполнять программы на основе собранной предварительно информации – трассе. При этом с некоторой вероятностью можно гарантировать повторяемость последовательности исполнения программы.

3. Активные отладчики реального времени. Эти отладчики позволяет пользователю вмешиваться в ход выполнения программы (пускать, останавливать процессы, управлять очередями сообщений).

4. Интерактивные отладчики. Этот класс отладчиков объединяет возможности двух предыдущих. По сути, отладчики этого типа должны поддерживать все функции, которые реализованы в отладчиках для последовательных программ.

Область параллельного программирования существует давно и за это время разработаны инструменты отладки для многих систем.

1. TotalView [4].

Коммерческий продукт из Германии. Он предназначен для интерактивной отладки параллельных программ. Поддерживаются как системы с разделяемой памятью, так и с общей памятью.

2. RAMPA (Институт прикладной математики им. М.В.Келдыша РАН) [5].

Эта система предназначена для разработки и отладки параллельных программ. Основными языками являются Fortran DVM и НОРМА. В дальнейшем разработчики планируют расширить свою систему с целью поддержки Fortran GNS, Fortran 77, C.

3. AIMS – Automated Instrumentation and Monitoring System [6].

Некоммерческий продукт, разрабатывается в NASA Ames Research

Center в рамках программы High Performance Computing and Communication Program. Программа предназначена для построения трасс и их визуализации. Кроме того, возможен сбор статистики по вызовам процедур. Поддерживаемые языки программирования – Fortran 77, HPF, C. Библиотеки передачи сообщений: MPI, PVM, NX. Поддерживаемые платформы IBM RS/6000 SP, рабочие станции Sun и SGI, Cray T3D/T3E.

4. Vampir [7].

Коммерческий продукт, разработка компании Pallas (Германия). Программа предназначена для построения трасс и их визуализации. Поддерживаемые языки программирования – Fortran, C. Библиотека передачи сообщений - MPI. Этот продукт поддерживает большое число платформ как с общей, так и разделяемой памятью.

5. Jumpshot [8].

Некоммерческое средство, разработано в Аргоннской национальной лаборатории. Распространяется вместе с пакетом MPICH и предназначено только для визуализации потоков данных. К существенным недостаткам этого продукта можно отнести невозможность задания рамок, в которых производится отладка и невозможность сбора никакой информации кроме как об операциях коммуникации.

6. Pablo Performance Analysis Toolkit Software [9].

Некоммерческий пакет, разработанный в университете штата Иллинойс. Пакет состоит из программ и библиотек сбора статистики, трасс и визуализации. Пакет ориентирован на языки ANSI C, Fortran 77, Fortran 90 (с ограничениями), HPF (Portland Group) и разрабатывался для платформ с общей памятью (Sun Solaris, RS6000, SP2, Intel Paragon, Convex Exemplar, SGI IRIX).

7. Paradyn [10].

Некоммерческое средство, разрабатываемое в университете Висконсина. Предназначено для онлайн-анализа параллельных программ на языках Fortran, Fortran 90, C, C++. Поддерживаемые библиотеки передачи сообщений – MPI, PVM. Поддерживаемые платформы (операционные системы):

– **Sun SPARC (только PVM);**
– **Windows NT на x86;**
– **IBM RS/6000 (AIX 4.1 или старше).**

Базовые функции отладчика

В мультикомпьютере каждый узел – это последовательный процессор, а параллельная программа для мультикомпьютера – это система асинхронных последовательных взаимодействующих процессов [12], [13]. Потому написанию параллельной программы предшествует этап разработки последовательных фрагментов алгоритма и их отладка. Далее отладка параллельной программы состоит в отладке межпроцессных взаимодействий, а для этой цели средства интерактивной отладки (TotalView) не подходят. Они вносят слишком большие искажения в реальное поведение программы, которыми при отладке последовательных программ можно было пренебречь. Еще одна проблема состоит в том, что в каждом узле мультикомпьютера течет свое время, заметно отличающееся от времени соседних узлов и непросто определить, например, в каком порядке процессы стартуют или завершаются.

При отладке параллельной программы возникает необходимость следить не только за операциями коммуникации (Jumpshot), но и получать некую дополнительную информацию, например, распределение загрузки по узлам вычислительного комплекса. Это необходимо для оптимального распараллеливания программы.

Сбор отладочной информации ведется либо методом вставки в код программы дополнительных инструкций, либо программа исполняется под управлением внешнего трассировщика, что вносит искажения в реальное поведение программы. Эти искажения легко могут быть учтены и компенсированы на системах с общей памятью (программа *pc* в пакете AIMS), однако для мультикомпьютера это сложная задача и пользователю приходится учитывать такие особенности отладчика.

Отладчик GEPARD

К сожалению, все рассмотренные отладчики имеют существенные ограничения реализации, что делает невозможным их использова-

ние для отладки параллельных программ для МВС-1000/М. С учетом вышеописанных требований, не удалось найти ни одного инструмента отладки параллельных программ под заданную архитектуру и операционную систему.

В виду вышесказанного, была поставлена задача разработать отладчик параллельных программ для мультикомпьютера, который бы:

- **оказывал минимальное влияние на поведение программы и мог учитывать это влияние при анализе полученного результата;**
- **имел гибкую систему сбора и анализа отладочной информации;**
- **удовлетворял всем требованиям эксплуатации МВС-1000/М.**

Как говорилось выше, в параллельной программе в отладке нуждаются в основном взаимодействия системы процессов, и поэтому должно быть минимизировано влияние отладчика на поведение программы, а этого можно достичь только используя систему мониторинга. Это соображение определило выбор типа отладчика – система мониторинга.

В любом инструменте отладки можно выделить три подсистемы:

- **предварительной обработки;**
- **сбора отладочной информации;**
- **анализа и представления собранной информации.**

Отладка начинается с того, что пользователь задает программу отладки на языке отладчика, указывает отладчику, какая отладочная информация ему необходима, на каких участках кода и как ее собирать, что с нею делать. Это может приводить как к изменениям правил компиляции и сборки программы (добавление опций, подмена/добавление библиотек), так и к модификациям исходного текста отлаживаемой программы. Одновременно может производиться сохранение данных для обеспечения информационного контекста – отождествление инструкций бинарного кода с позицией в исходном коде программы.

Подсистема сбора отладочной информации – основная часть любого инструмента отладки. Именно ее реализацией определяет класс, к которому относится отладчик (интерактивный, система монито-

ринга ...). Сбор информации может осуществляться либо на основе анализа исходного текста программы (интерпретаторы), либо в процессе ее исполнения.

На основе собранной информации подсистема анализа и представления может выдавать пользователю рекомендации по оптимизации программы и устранению ошибок. Анализом собранной информации может заниматься и сам пользователь, для чего нужна ее визуализация.

В случае параллельной программы в основном отлаживается система взаимодействий процессов. Для этой цели удобным является представление процесса исполнения программ графом событий. Для построения этого графа как минимум необходимо отслеживать все случаи пересылки данных между процессами. Для более детального анализа исполнения программы необходима дополнительная информация (статистика по вызовам функций, распределение процессов по узлам мультикомпьютера…). Однако чем больше информации собирается о программе, тем больше искажений в исходное поведение вносится системой мониторинга. Поэтому в параллельном отладчике важно иметь возможность последовательно отключать сбор ненужной информации, точно локализовать области сбора отладочной информации с целью уменьшения влияния отладчика на поведение отлаживаемой программы, указывать характеристики взаимодействий. Был выбран следующий метод сбора подобной информации.

После запуска потока исполнения программы на узле вычислительного комплекса создается дополнительный процесс – mondc, связанный с потоком исполнения программы через неименованный программный канал. Процесс создается через системный вызов fork(2). Информация, переданная через программный канал, сохраняется в буфере mondc и по мере заполнения буфера, или после специальной команды передается на сервер. Наличие на каждом узле, где исполняется программа дополнительного процесса, дает возможность возложить на него обязанность по сбору информации о среде исполнения, например, для слежения за равномерностью загрузки узлов вычислительного комплекса.

На сервере, который может также быть одним из узлов вычислительного комплекса, исполняется программа monsrv. Назначение

этой программы – получение отладочной информации с узлов вычислительного комплекса, где исполняется отлаживаемая программа и сохранение ее в файл отчета.

Такой подход к реализации сбора отладочной информации позволяет уменьшить нагрузку на сеть – пользователь сам может решать, когда необходимо сохранять данные на сервер, и минимизирует паразитное влияние отладчика на исполнение отлаживаемой параллельной программы – пересылкой информации и отчасти ее сбором занимается процесс mondc, а не отлаживаемая программа.

Сценарий работы пользователя

Сценарий отладки с помощью GEPARD может быть следующим. После того, как на стадии тестирования обнаружена ошибка, пользователь с помощью специального языка отладки (вручную, или используя графический интерфейс) вставляет в код программы инструкции, которыми указывает информацию, которая должна быть собрана для анализа. Затем, после обработки кода программы препроцессором, программа должна быть откомпилирована и собрана в исполняемый модуль. По мере исполнения программы собранная информация помещается в файл отчета. С помощью системы визуализации и анализа полученная информация представляется в удобном для пользователя виде. Для более точного понимания поведения программы пользователь может менять представление отображаемой информации, например, применять фильтры. В процессе анализа пользователю могут даваться советы по поиску ошибок и оптимизации программы. Если в результате анализа не удалось обнаружить ошибку, пользователю следует вернуться на шаг назад и дать указания отладчику по сбору дополнительной отладочной информации.

Состояние проекта

На текущий момент проведен анализ существующих средств отладки параллельных программ, сформулированы общие требования, предъявляемые к отладчику, построена и проанализирована модель создаваемого отладчика и, согласно этой модели, создан прототип отладчика GEPARD. С его помощью проведен тест ряда

функций MPI, полученные результаты представлены на web сервере ССЦ [1].

В ходе тестов функций MPI было обнаружено, что завершение посылки сообщения, с помощью блокирующих функций передачи данных, может происходить раньше, чем начнется их прием. Это говорит о том, что посылающий процесс не может быть уверен в доставке посланного сообщения. Также было установлено, что все процессы выходят из функции MPI_Init не одновременно.

Сейчас ведется детальная проработка языка отладки, наполнение отладчика дополнительными функциями по сбору, анализу и представлению отладочной информации.

Планируется, что в ближайший месяц появится первая работоспособная версия отладчика, которой будут пользоваться не только сотрудники институтов СО РАН, но и студенты НГУ в учебном процессе на лабораторных занятиях по курсу «параллельное программирование».

Литература

1. *Kranzlmüeller.* Clustering computer review. Ph.D.Thesis, http://www.npac.syr.edu/techreports.
2. http://ssd2-new.sscc.ru.
3. http://www.jscc.ru.
4. *Петренко А.К.* Методы отладки и мониторинга параллельных программ (обзор). // Программирование. 1994. №3. С. 39–63.
5. http://www.uni-karlsruhe.de/~SP.
6. http://www.applmat.ru.
7. http://science.nas.nasa.gov.
8. http://www.pallas.de.
9. http://www-unix.mcs.anl.gov.
10. http://vibes.cs.uiuc.edu.
11. http://www.cs.wisc.edu.
12. http://parallel.ru/v-ray.
13. *Вальковский В.А.*, *Малышкин В.Э.* Синтез параллельных программ и систем на вычислительных моделях. Новосибирск: Наука, 1988.
14. *Хоар Ч.* Взаимодействующие последовательные процессы. М: Мир, 1989.

ОРГАНИЗАЦИЯ ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ ОБРАБОТКИ БОЛЬШИХ МАССИВОВ ДАННЫХ ДЛЯ МАССИВНО-ПАРАЛЛЕЛЬНЫХ ПЛАТФОРМ

О.В. Савцов, В.А. Григорьев

Тверской государственный технический университет

Создание программного обеспечения для обработки больших массивов данных в многопроцессорных вычислительных системах с массивно-параллельной архитектурой является в настоящее время одним из актуальных направлений системного программирования.

Анализ задач, связанных с обработкой больших массивов данных и решаемых на вычислительных системах с массовым параллелизмом [2–4], показывает, что имеется большой класс задач, при распараллеливании которых прикладными программистами используется следующий подход. Вычислительная задача разбивается на подзадачи. Для каждой подзадачи в процессорном массиве системы выделяются непересекающиеся группы процессорных узлов, обычно называемые полями. Каждая подзадача, в свою очередь, разбивается на процессы, которые запускаются на отдельных процессорах в пределах поля, назначенного данной подзадаче.

При реализации задач указанного класса прикладные программисты при разбиении множества процессоров на поля, как правило, не учитывают физическую топологию системы. При этом для реализации межпроцессорных коммуникаций прикладные программисты используют системные сервисы общего назначения (ОС Router для МВС-100/1000), которые часто оказываются недостаточно эффективными для задач с интенсивными обменами данными между процессорами.

Предлагается использовать оригинальную подходы к разработке программного комплекса для распределенной обработки больших массивов данных на отечественной многопроцессорной вычислительной системе МВС-100/1000 [1].

Программный комплекс предоставляет системные сервисы, которые могут использоваться прикладными программистами при реализации задач указанного класса. Комплекс рассчитан на использование в массивно-параллельных системах с архитектурой CD2 [5].

Данная архитектура предполагает разбиение множества процессорных узлов вычислительной системы на непересекающиеся группы процессоров, называемых процессорными кластерами. Процессорный кластер состоит из небольшого числа процессорных узлов и имеет фиксированную топологию межпроцессорных соединений. CD2-система представляет собой множество процессорных кластеров, объединенных высокоскоростной соединительной сетью. На топологию процессорных кластеров накладываются определенные ограничения (например, длина кратчайшего пути между узлами не превышает 2), что позволяет реализовать межпроцессорное взаимодействие более эффективно, чем с помощью сервисов, предоставляемых операционной системой Router для МВС-100/1000.

Программный комплекс предоставляет системные сервисы, позволяющие обрабатывать данные различной природы. В зависимости от объема, времени жизни и частоты обращения к ним, данные могут быть разделены на следующие три категории: персистентные данные, временные данные и сообщения.

Персистентные данные имеют большой объем (не помещаются в оперативную память процессорного узла), характеризуются многократными к ним обращениями и существуют до и после выполнения задачи.

Временные данные имеют большой объем (не помещаются в оперативную память процессорного узла), характеризуются многократными к ним обращениями и существуют только в период выполнения задачи.

Сообщения имеют небольшой объем (помещаются в оперативную память одного процессорного узла), характеризуются однократными к ним обращениями и существуют только в период выполнения задачи.

В соответствии с данной классификацией предлагаемая реализация программного комплекса включает в себя систему передачи сообщений и систему хранения данных.

Система передачи сообщений обеспечивает средства асинхронного обмена данными между любыми двумя процессорными узлами вычислительной системы. В соответствии с принципами CD2-архитектуры система передачи сообщений состоит из двух подсис-

тем:

маршрутизатора и кондуктора. Маршрутизатор обеспечивает асинхронную передачу сообщений между узлами, принадлежащими различным процессорным кластерам. Кондуктор обеспечивает средства для асинхронной передачи сообщений в пределах одного процессорного кластера. Подсистемы имеют сходный интерфейс, но их реализация использует принципиально различные протоколы обмена, учитывающие топологию процессорных кластеров.

Система хранения данных обеспечивает унифицированный интерфейс обработки персистентных и временных данных на базе следующих устройств хранения: жесткий диск host-компьютера, дисковая подсистема вычислительной системы и электронная дисковая подсистема.

Система хранения данных строится на базе архитектуры клиент-сервер. В соответствии с этим среди процессорных узлов выделяются *узлы-клиенты* и *узлы-серверы*. Серверная часть системы хранения запускается на узле-сервере и обслуживает запросы клиентских частей, запускаемых на узлах-клиентах.

Система хранения предполагает две реализации узла-сервера: аппаратную и программную. Аппаратная реализация узла-сервера строится на базе модуля дисковой подсистемы: серверная часть запускается на модуле дисковой подсистемы и использует драйвер дисковой подсистемы. Программная реализация узла-сервера строится на базе стандартного процессорного модуля: серверная часть запускается на процессорном модуле, оперативная память которого используется в качестве хранилища данных других процессорных модулей.

Программная структура система хранения данных включает в себя электронную дисковую подсистему и систему управления файлами.

Электронная дисковая подсистема (ЭДП) реализует виртуальный модуль дисковой подсистемы с устройством хранения данных в оперативной памяти процессорного узла и предоставляет соответствующие низкоуровневые сервисы для чтения-записи данных (драйвер ЭДП).

Система управления файлами (СУФ*)* поддерживает понятие файла, как именованного набора записей фиксированной длины. Файлы

используются для унифицированного представления и обработки персистентных и временных данных. СУФ представлена иерархией следующих подсистем: менеджер дисков, менеджер наборов и менеджер файлов. Менеджер дисков обеспечивает страничную организацию диска и предоставляет средства для асинхронного чтения и записи указанной страницы диска. Менеджер дисков состоит из клиентской и серверной части. Менеджер наборов обеспечивает представление данных, хранящихся на диске, в виде совокупности наборов (связных списков) страниц. Менеджер наборов обеспечивает буферизацию данных на основе единого буферного пула. Менеджер файлов обеспечивает представление данных в виде файлов – именованных множеств неструктурированных записей одинаковой длины и обеспечивает стандартные функции для работы с файлами.

Клиент менеджера дисков, менеджер наборов и менеджер файлов составляют клиентскую часть системы хранения данных. Серверную часть системы хранения составляют сервер менеджера дисков и драйвер реальной либо электронной дисковой подсистемы.

При интенсивных обменах данными между узлами-клиентами и узлами-серверами пропускная способность линков становится узким местом. Данная проблема решается с помощью буферизации данных. Буферизация заключается в выделении буферного пула в основной памяти узла-клиента. Если запрашиваемая страница данных уже находится в буфере клиента (ситуация попадания), то повторного считывания страницы не происходит, что позволяет значительно сократить объем данных, передаваемых от сервера к клиенту.

При обработке данных большого объема, как правило, не удается обеспечить буферный пул, вмещающий весь массив данных, необходимых задаче. В этом случае приходится вытеснять из буферного пула страницы, которые в данный момент не используются. При использовании вытеснения страниц возможна ситуация неудачи, когда повторно запрашиваемая страница данных отсутствует в буфере. Отсюда возникает проблема подбора эффективной стратегии вытеснения страниц, которая минимизировала бы число неудач.

В настоящее время известно несколько стратегий вытеснения, например, FIFO, LIFO, LRU, CLOCK и другие [5]. Стратегии вытес-

нения обычно использую «возраст» страниц, находящихся в буферном пуле или частоту их использования для предсказания будущего поведения операций с диском.

Хотя некоторые из них показывают в среднем удовлетворительную производительность, они могут быть наихудшими стратегиями в отдельных случаях [6].

Попытка улучшить данные алгоритмы приводит к новым, концептуально сложным алгоритмам, работа которых зависят от большого числа параметров. Предлагается использовать другой подход, когда из фиксированного множества стратегий вытеснения динамически выбирается оптимальная для заданного набора страниц.

Ключевую роль в данном подходе играет избыточный индекс буферного пула. Размер этого индекса определяется как $k*M$, где M – размер буферного пула в страницах, а целое $k >> 1$. Избыточный индекс хранит историю загрузки страниц в буфер. Элемент избыточного индекса имеет статические атрибуты для реализации общих стратегий вытеснения: счетчик обращений к странице, время последнего доступа к странице, очередность помещения в буфер и т.п.

Менеджер буферного пула динамически анализирует эти атрибуты, чтобы найти циклы подкачки страниц. Если такой цикл имеет место, то менеджер буферного пула выбирает стратегию вытеснения, оптимальную для данного набора страниц и вида доступа к нему.

Классическим примером, описанным в [6], является порядок вытеснения страниц для обеспечения восстановления базы данных после сбоя. Страницы, содержащие журнал транзакций, должны вытесняться на диск строго после страниц, которые потерпели изменения в ходе транзакций.

Для предотвращения подобных ситуаций предлагается использовать технику вытеснения, основанную на концепции статических и динамических рейтингов страниц. Каждый элемент избыточного индекса страниц имеет статический рейтинг. Это целое число из диапазона [0;20]. Статический рейтинг назначается при помещении страницы в буфер. Статический рейтинг страницы остается постоянным все время нахождения страницы в буферном пуле и теряется после ее вытеснения.

Динамический рейтинг – это функция от вышеупомянутых статических атрибутов страницы, возвращающая вещественное число из интервала [0;1[. Динамический рейтинг изменяется в процессе работы системы.

Суммарный рейтинг подсчитывается как сумма статического и динамического рейтингов. При необходимости освободить место в буферном пуле вытесняется страница с минимальным суммарным рейтингом.

Механизм статических рейтингов обеспечивает системные приоритеты вытеснения страниц из буферного пула. Страницы, имеющие большой статический рейтинг, будут оставаться в буфере независимо от выбранной стратегии вытеснения.

Механизм динамических рейтингов позволяет имитировать практически любую стратегию вытеснения страниц из буферного пула.

В работе описан подход к реализации комплекса системных программ для распределенной обработки данных в вычислительных системах с массовым параллелизмом. Основными компонентами комплекса являются система передачи сообщений и система хранения данных. Комплекс может быть использован в качестве системного окружения в задачах, требующих интенсивной обработки больших массивов данных.

Предложен оригинальный метод вытеснения страниц из буферного пула, основанный на использовании избыточного индекса буферизованных страниц и введении статических и динамических рейтингов страниц.

Литература

1. ***Левин В.К.*** **Отечественные суперкомпьютеры семейства МВС. http://parallel.ru/mvs/levin.html.**
2. ***Коковихина О.В.*** **Распараллеливание алгоритма решения задачи о распространении акустических колебаний в газовых потоках // Алгоритмы и программные средства параллельных вычислений. Сб. науч. тр. Екатеринбург. УрО РАН. 1995. С. 79–85.**
3. ***Мельникова Л.А.*****,** ***Розенберг В.Л.*** **Численное моделирование динамики блоковой структуры на МВС // Алгоритмы и про-**

граммные средства параллельных вычислений. Сб. науч. тр. Екатеринбург. УрО РАН. 1998. С. 221–235.

4. ***Цепелев И.А., Короткий А.И. и др.*** **Параллельные алгоритмы решения задачи моделирования высоковязких течений в верхней мантии // Алгоритмы и программные средства параллельных вычислений. Сб. науч. тр. Екатеринбург. УрО РАН. 1998. С. 301–317.**

5. ***Effelsberg W., Harder T.*** **Principles of Database Buffer Management // ACM Trans. on Database Systems. Dec. 1984. V. 9. No.4. P. 560–595.**

6. ***Stonebraker M.*** **Operating System Support for Database Management // Communications of the ACM (CACM). July 1981. Vol. 24. No.7. P. 412–418.**

ПРОТОТИП СИСТЕМЫ АВТОМАТИЗИРОВАННОГО СОЗДАНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ «PARUS»

А.Н. Сальников, А.Н. Сазонов, М.В. Карев

МГУ им. М.В. Ломоносова

Введение

Современные вычислительные комплексы достигают высокой производительности за счет распараллеливания вычислений. Архитектура таких систем весьма разнообразна, и на сегодняшний день не существует классификации, достаточно полно описывающей все особенности архитектур вычислительных систем. Программы, написанные для машин последовательной архитектуры напрямую невозможно использовать на машинах с параллельной архитектурой, не сводя эффективность исполняемого кода программы практически в ноль. Более того, для каждой параллельной архитектуры существует своя специфика написания эффективных параллельных программ, так что программы, эффективно исполняющиеся на одной параллельной вычислительной системе, в подавляющем большинстве случаев полностью непригодны, а если и пригодны, то неэффективны на другой. Таким образом, решение сложных задач на вычислительной системе с параллельной архитектурой связано с необходимостью предварительного анализа алгоритма и выбора оптимального способа организации параллельных вычислений.

В системе предлагается несколько способов создания параллельной программы:

1. Автоматическое построение по последовательному коду.

2. Ручное написание программы, путем декларации узлов графа программы и связей между ними.

В будущем планируется, создание программы, по декларированным зависимостям по данным для пакетов функций обработки данных, таких, как Mathlab.

По последовательному коду программы написанной на языке программирования С строится граф алгоритма, или же граф алгоритма строится вручную, вершины графа символизируют действия, а дуги зависимость по данным. Производится тестирование многопроцессорной системы на скорость передачи сообщений между отдельными процессорами многопроцессорной системы, а так же тестирование производительности процессоров. Граф преобразуется в параллельную программу, например программу, написанную на языке C++ с MPI вызовами. Строится расписание выполнения полученной параллельной программы на многопроцессорной системе.

Преобразователь последовательного кода программы в граф алгоритма

Создаваемая система получает на вход последовательную программу в виде исходного текста и строит по нему *взвешенный граф алгоритма*. Под *взвешенным графом алгоритма* понимается граф удовлетворяющий следующим требованиям.

1) Ориентированный граф, с пометками на дугах и вершинах без ориентированных циклов. Направление дуги задает порядок следования операторов, метка – передаваемые данные.

2) Отсутствуют дуги между вершинами одного яруса.

3) Отсутствуют дуги «снизу-вверх» – от вершин в ярусе с большим номером к вершинам яруса с меньшим номером. Ярусы нумеруются по возрастанию. Метки (операторы) вершин, расположенных в одном ярусе могут быть выполнены одновременно. Ярусы нумеруются естественным образом от вершин источников, вершин в кото-

рые не входит ни одной дуги, к вершинам стокам, вершин из которых не исходит ни одной дуги.

Вершины графа — вычисления, дуги — информационные зависимости между ними по данным. При этом дуга направлена от вершины источника данных, к вершине приемнику данных. Зависимости по данным фиксируют необходимый порядок выполнения операций. Существует три типа таких зависимостей:

Прямая зависимость «out-in»
Обратная зависимость «in-out»
Зависимость по выходам «out-out»

Параллельная программа в форме графа алгоритма

Граф алгоритма преобразуется в параллельную программу. Операторы, приписанные вершинам графа, преобразуются в функции и могут быть выполнены на произвольном процессоре многопроцессорной системы. Дуги становятся вызовами функций передачи сообщений.

Для эффективного выполнения параллельной программы на многопроцессорной системе граф алгоритма должен обладать следующими дополнительными свойствами.

1. Узел графа алгоритма сложный, в вычислительном смысле, набор операторов. Задача выбора правильного размера узла графа алгоритма, чрезвычайно важна. Слишком легкий узел графа алгоритма приведет к тому, что накладные расходы при вызове его на конкретном процессоре, перекроют по времени тот полезный код, который выполняется в данном узле графа.

2. Операторы узла графа алгоритма явно обращаются только к локальной памяти процессора многопроцессорной системы. Данное свойство является следствием того, что рассматриваются многопроцессорные вычислительные системы с распределенной памятью.

3. Операторы узла графа алгоритма выполняются только в тот момент времени, когда получены все данные, входящие в вершину графа по дугам графа. Данные по дугам графа алгоритма могут приходить в произвольном порядке, асинхронно.

Построение расписания

По полученной на вход программе в форме графа алгоритма и графу многопроцессорной вычислительной системы строится статическое расписание исполнения программы на многопроцессорной системе. При этом строится близкое к оптимальному по времени расписание выполнения параллельной программы на данной вычислительной системе. Программа представлена в форме графа алгоритма.

Задача решается при помощи генетического алгоритма, схема кодирования которого не допускает задания невыполнимых расписаний, например, таких, у которых нарушается условие частичной упорядоченности узлов графа алгоритма. Для подсчета функции качества по генетическому коду особи строится расписание, длина которого, предположительное время работы программы, становится значением функции качества. Таким образом, цель программы построения расписания – минимизировать функцию качества. Желательно найти глобальный минимум, однако это не обязательно, а в случае сложных входных данных и маловероятно. От требуемой точности решения сильно зависит время работы построителя расписания, чем выше точность, тем дольше работает построитель.

Перед тем, как строить расписание выполнения параллельной программы и выполнением самой параллельной программы необходимо получить сведения о производительности многопроцессорной системы. Тесты запускаются в режиме, монопольного захвата процессора.

Процесс исполнения полученной параллельной программы

Для исполнения полученной параллельной программы в многопроцессорной системе выделяется один специальный управляющий процессор, при помощи которого осуществляется управление деятельностью других процессоров. Управляющий процессор определяет последовательность вызовов узлов графа алгоритма, интерпретируемых как функции, определяет в какой последовательности необходимо передавать данные по ребрам графа алгоритма, учитывает производительность сети и процессоров при планировании, учитывает статическое расписание, полученное для данного графа алгоритма заранее. Остальные процессоры обмениваются данными, передаваемыми через ребра графа алгоритма, вызывают узлы графа

алгоритма и выполняют приписанный им код, накапливают наработанные узлами графа алгоритма данные.

Узлы графа алгоритма располагаются по уровням, и по мере загрузки уровня на процессор следующий уровень добавляется в список узлов графа заявленных на выполнение. Среди всех узлов заявленных на выполнение производится поиск узлов с минимальным временем выполнения на процессоре. Если минимум достигается для нескольких узлов графа одновременно, то выбирается тот, который заявлен в расписании на выполнение на данном процессоре.

Заключение

Несмотря на то, что система является только прототипом и не пригодна к промышленному написанию параллельных программ для многопроцессорных систем с распределенной памятью, получена рабочая версия, которая позволяет строить параллельную программу по графу алгоритма. Граф алгоритма можно получить как по последовательной программе написанной на языке C, так и вручную, путем создания или редактирования текстового файла (подробнее в Материалах Международного научно-практического семинара «Высокопроизводительные параллельные вычисления на кластерных системах» Нижний Новгород 20-24 ноября 2001, С. 149–167.)

Разработан набор документации, описывающий работу компонент системы, формат входных данных для компонентов системы, в том числе формат представления графа алгоритма в виде текстового файла, а также производится тестирование системы.

Разработчиками прочувствованны проблемы, связанные с таким подходом строительства параллельных программ, а также пути дальнейшего развития системы с целью сделать ее более пригодной к промышленному программированию. Планируется подключение к системе библиотек подобных Mathlab, планируется существенно усложнить модель графа алгоритма, добавив возможность динамического строительства самого графа алгоритма в процессе исполнения параллельной программы.

Данная работа выполняется на факультете Вычислительной Математики и Кибернетики МГУ им. М.В. Ломоносова, в рамках студенческой лаборатории Intel МГУ. При поддержке гранта РФФИ

02-07-90130 и гранта РФФИ 02-07-06104.

Авторы приносят благодарности члену корреспонденту РАН, профессору Льву Николаевичу Королеву и доктору физ. мат. наук, доценту Нине Николаевне Поповой за консультации при создании данной системы.

НОВЫЙ ДИАГОНАЛЬНЫЙ АЛГОРИТМ ГЛОБАЛЬНОЙ МИНИМИЗАЦИИ [*]

Я.Д. Сергеев, Д.Е. Квасов

Нижегородский государственный университет им. Н.И.Лобачевского,
Калабрийский университет, Козенца, Италия

Рассматривается задача поиска глобального минимума многомерной многоэкстремальной функции, удовлетворяющей в допустимой области $D \subset R^N$ условию Липшица с константой Липшица $0 < L < \infty$:

$$\min f(x),\ x \in D, \tag{1}$$

$$D = \{x \in R^N : a_i \le x_i \le b_i,\ i = 1,\ldots,N\}, \tag{2}$$

$$|f(x') - f(x'')| \le L\,\|x' - x''\|,\quad x', x'' \in D, \tag{3}$$

где $\|\cdot\|$ есть евклидова норма.

Целевая функция $f(x)$ педполагается недифференцируемой и, следовательно, только значения $f(x)$ в точках $x \in D$ могут быть сосчитаны в ходе решения (1)–(3), причем операция вычисления значения функции в точке считается требующей больших затрат времени. Такая ситуация часто встречается в практических задачах (см. [1–4]).

В литературе рассматривается множество различных алгоритмов решения задачи (1)–(3) (см. [1–4]). Одним из подходов к решению данной задачи является следующий. На каждой итерации

[*] Работа выполнена в рамках научной программы «Университеты России» Министерства образования РФ

алгоритма область D разбивается на несколько подобластей (интервалов) D_i. В каждом из интервалов D_i определяются точки, в которых вычисляется функция $f(x)$. Например, $f(x)$ может быть вычислена в центральных точках интервалов (см. [5]) или на концах главных диагоналей интервалов D_i, как в диагональных алгоритмах [4, 6]. На основе полученной информации выбираются один или несколько интервалов, пригодных для дальнейшего разбиения. Выбранные интервалы разбиваются и процесс повторяется до тех пор, пока не будет выполнено заданное пользователем условие остановки, например, пока диагональ выбранного на текущей итерации алгоритма интервала не станет меньше определенного параметра $\varepsilon > 0$.

Алгоритмы решения задачи (1)–(3) могут быть разделены на четыре группы в зависимости от способа получения оценки константы Липшица L из (3). К первой группе относятся алгоритмы, в которых применяется оценка L, заданная априори (см., например, [3, 7]). Вторая группа включает алгоритмы, адаптивно оценивающие в ходе поиска глобальную константу Липшица во всей области D (адаптивная глобальная оценка, см. [1, 2]). Алгоритмы с использованием адаптивных локальных оценок L_i в каждом интервале D_i (см. [2,8]) входят в третью группу. Наконец, к четвертой группе принадлежат алгоритмы, в которых оценка константы Липшица выбирается из некоторого множества возможных значений (см. [5]).

К последней группе относится и предлагаемый в данной работе алгоритм решения задачи (1)–(3). В отличие от схемы, принятой в алгоритме из [5], в новом алгоритме используется диагональный подход [4]. Разбиение области поиска в его работе производится при помощи эффективной стратегии [6], применяемой при построении адаптивных диагональных кривых (см. [6, 9]). Как было показано в [6], данная стратегия позволяет избежать генерирования избыточных точек вычисления $f(x)$, а также сэкономить машинную память, требуемую для хранения информации о ходе поиска глобального минимума.

В работе исследуются свойства нового алгоритма, в частности, его поведение при изменении условий остановки. С этой целью предлагается использовать генератор классов многомерных

тестовых функций из [10]. Обсуждаются вопросы параллельной реализации алгоритма на многопроцессорной (32 процессора Intel Pentium III) системе Icarus института ICAR–CNR (г.Козенца, Италия).

Литература

1. ***Стронгин Р.Г.*** **Численные методы в многоэкстремальных задачах. М.: Наука, 1978.**
2. ***Strongin R.G. and Sergeyev Ya.D.*** **Global Optimization with Non-Convex Constraints: Sequential and Parallel Algorithms. Dordrecht: Kluwer Academic Publishers, 2000.**
3. **Handbook of Global Optimization: Horst R., Pardalos P.M. (editors). Dordrecht: Kluwer Academic Publishers, 1995.**
4. ***Pintér J.*** **Global Optimization in Action. Dordrecht: Kluwer Academic Publishers, 1996.**
5. ***Jones D.R.***, ***Perttunen C.D.***, ***Stuckman B.E.*** **Lipschitzian optimization without the Lipschitz constant //J. Optimizat.Theory Appl. 1993. Vol. 79. №1. P. 157–181.**
6. ***Sergeyev Ya.D.*** **An efficient strategy for adaptive partition of N-dimensional intervals in the framework of diagonal algorithms //J.Optimizat.Theory Appl. 2000. Vol. 107. N 1. P. 145–168.**
7. ***Пиявский С.А.*** **Один алгоритм отыскания абсолютного экстремума функции //Ж.вычисл.матем. и матем.физ. 1972. Т. 12. N 4. С. 888–896.**
8. ***Sergeyev Ya.D.*** **An information global optimization algorithm with local tuning // SIAM J. Optimizat. 1995. Vol. 5. N 4. P. 858–870.**
9. ***Сергеев Я.Д., Квасов Д.Е.*** **Адаптивные диагональные кривые и их программная реализация //Вестник ННГУ. Математическое моделирование и оптимальное управление. Н.Новгород: Изд-во ННГУ. 2001. Вып. 2(24). С. 300–317.**
10. ***Gaviano M., Kvasov D.E., Lera D., Sergeyev Ya.D.*** **Software for generation of classes of test functions with known local and global minima for global optimization //Submitted.**

РЕШЕНИЕ ЗАДАЧ ГРАВИТАЦИОННОЙ ДИНАМИКИ НА

МНОГОПРОЦЕССОРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

А.В. Снытников

ИВМиМГ СО РАН, Новосибирск

Описывается модель самогравитирующей системы. Приводится обзор методов и программ параллельной реализации такой модели. Рассмотрен численный алгоритм и методика распараллеливания. Описываются тестовые расчеты, параметры которых соответсвуют требованиям задачи о моделировании протопланетного диска.

Введение

Эволюция самогравитирующих систем таких, как галактики или протопланетные диски имеет большой интерес для астрофизики Образование структуры диска представляет собой задачу многих тел в самосогласованном гравитационном поле. Хорошим приближением для моделирования самогравитирующего диска является кинетическое уравнение Власова–Лиувилля. Вместе с уравнением Пуассона для самосогласованного гравитационного поля они образуют систему уравнений звездной динамики [1]:

$$\begin{cases} \Delta\Phi = 4\pi G\rho, \\ \dfrac{\partial f}{\partial t} + \vec{v}\nabla f - \nabla\Phi\dfrac{\partial f}{\partial \vec{v}} = 0. \end{cases}$$

Здесь диск галактики считается плоским и бесстолкновительным, и используется приближение дальнодействующего поля. Оценка ресурсов ЭВМ [2], требуемых для моделирования эволюции протопланетного диска, дает сетку 1000x1000x500 узлов и порядка 100 млн. частиц. Для этого требуется 8 Gb оперативной памяти, таким образом, данная задача должна решаться на суперкомпьютерах.

Обзор программ для решения задач гравитационной динамики

Существует несколько методов численного решения уравнений звездной динамики. Первый из них – непосредственное вычисление взаимодействия частиц, так называемый метод «частица-частица», или P^2. Этот метод является наиболее точным, но также наиболее

трудоемким, его сложность $O(N^2)$. Для расчета движения частицы в этом методе необходимо знание координат всех остальных частиц, таким образом моделирование с использованием более чем 10^5 частиц невозможно даже на суперкомпьютерах.

Модификацией P^2-метода является так называемый древесный алгоритм (англ. treecode), в котором близко расположенные частицы объединяются в группы так, чтобы можно было вычислить силу притяжения группы частиц как одной частицы той же массы. Если такая аппроксимация силы оказывается неточной, то группа разбивается на более мелкие подгруппы. Разбиение частиц на группы имеет вид восьмеричного дерева, отсюда название метода. Этот метод работает существенно быстрее, нежели P^2, его сложность $O(N\log N)$, но менее точно. Распределение загрузки процессоров выполняется во время построения дерева [3].

В методе «частица-сетка», или PM в пространстве моделирования вводится сетка, на которой по координатам частиц вычисляется плотность вещества. Далее, путем решения уравнения Пуассона по плотности вычисляется гравитационный потенциал, как правило, при этом используется метод быстрого преобразования Фурье. Для каждой из частиц сила вычисляется интерполяцией значений, вычисленных на сетке, в координату данной частицы. PM-метод – наиболее быстрый из существующих, его сложность $O(N\log M)$, где M – число узлов сетки, однако он работает достаточно точно только для однородных систем – в противном случае требуется чрезвычайно большое количество частиц. Основную сложность представляет распараллеливание решения уравнения Пуассона. Если большую часть времени занимает расчет движения частиц, то необходимо использовать динамическую балансировку загрузки [4]. В работе [5] предложен алгоритм динамической балансировки загрузки для PM-метода, основанный на разделении задачи на множество небольших подзадач, которые назначаются на процессоры по мере их освобождения.

Существует большое количество программ для космологического моделирования, реализующих P^3M-метод. Этот метод представляет собой комбинацию P^2 и PM-методов; непосредственно вычисляется притяжение близко расположенных частиц, обычно внутри одной ячейки или соседних ячеек, в то время как сила, действующая

со стороны удаленных частиц вычисляется через уравнение Пуассона. Применение P^3M-метода к задачам космологии можно объяснить тем, что в этом случае вещество разбито на компактные изолированные группы (галактики). При распараллеливании P^3M-алгоритма в [6] используются две различных схемы декомпозиции области, одна для расчета взаимодействия частиц (P^2-часть), другая для решения уравнения Пуассона (PM-часть). Число операций в этом методе меняется от $O(N\log M)$ до $O(N^2)$.

Сочетанием всех названных методов является алгоритм «дерево-частицы-сетка», TPM (Treecode-Particle-Mesh) [7]. Данный алгоритм основан на том, что поле плотности может быть разбито на множество изолированных плотных областей. В каждой из таких областей гравитационный потенциал рассчитывается по древесному алгоритму, для остальной части объема потенциал и силы вычисляются с помощью PM–метода. Так как каждая подобласть может рассматриваться независимо, алгоритм допускает очень эффективное распараллеливание. Деревья, соответвующие подобластям, распределяются по процессорам так, чтобы уравнять их загрузку. Сравнение показало, что TPM- алгоритм считает по крайней мере с той же точностью, что и P^3M, но гораздо быстрее. Эффективность распараллеливания зависит от степени кластеризации вещества, то есть от количества и плотности деревьев, аналогично предыдущему сило операций в TPM-методе меняется от $O(\mathrm{Nlog}\ M)$ до $O(\mathrm{Nlog}\ N)$.

Сравнение параметров реализации перечисленных выше методов приведено в следующей таблице. Эффективность распараллеливания здесь – это отношение ускорения расчета к увеличению числа процессоров.

Метод	**Размер сетки**	**Число процессоров**	**Эффективность распараллеливания**	**Число частиц**	**Статья**
P^2	**–**	**112**	**–**	**30 тыс.**	**[8]**
PM	**256^3**	**64**	**85%**	**16.7 млн.**	**[5]**
Tree-code	**–**	**16**	**93%**	**100 тыс.**	**[3]**
P^3M	**1024^3**		**–**	**1 млрд.**	**[6]**

TPM	**512^3**	**128**	**90%**	**134 млн.**	**[7]**

Описание численного алогоритма

Основную трудность в задаче моделирования эволюции протопланетного диска представляет расчет потенциала самосогласованного гравитационного поля, который определяется из уравнения Пуассона.

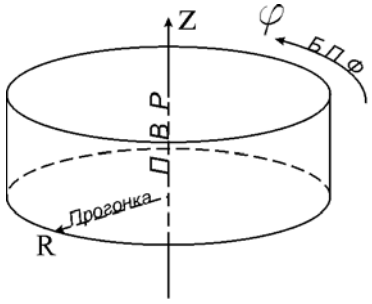
Для того, чтобы получить быстрый алгоритм решения уравнения Пуассона, необходимо учесть особенности задачи и сделать алгоритм хорошо распараллеливаемым, то есть допускающим применение произвольного числа процессоров.

Рассмотрим особенности рассматриваемой задачи. Во-первых, она является нестационарной. Это означает, что физические величины - гравитационный потенциал, плотность и поле скоростей не могут изменяться скачкообразно. То есть, значение потенциала на соседних временных шагах не могут заметно отличаться. Следовательно, необходим метод решения, способный использовать значение потенциала, полученное на предыдущем временном шаге. Этому требованию соответствуют итерационные методы. Однако они являются недостаточно быстродействующими, поэтому предлагается комбинированный алгоритм, являющийся сочетанием прямых и итерационных методов.

Во-вторых, протопланетный диск можно считать плоским, то есть толщина диска много меньше его радиуса, и плотность не равна 0 только на поверхности диска. Это означает, что изменение плотности вследствие сдвига частиц на некотором временном шаге приводит к заметному изменению потенциала только вблизи поверхности диска. Напротив, в узлах, удаленных от диска, эти изменения незначительны и условие сходимости выполняется гораздо быстрее. Таким образом, время счета значительно сокращается за счет того, что, удаленные от диска узлы не участвуют в расчете.

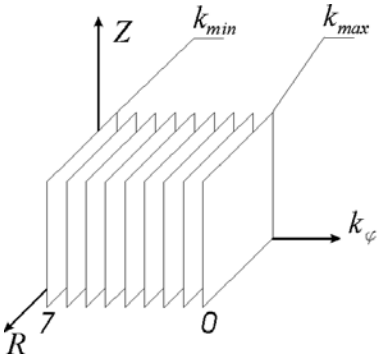
Уравнение Пуассона решается на сетке в цилиндрической системе координат для того, чтобы учесть симметрию диска и избежать появления нефизических структур, которые возникают при расчете в прямоугольных декартовых координатах.

Общая структура метода решения показана на рисунке. Первый шаг – быстрое преобразование Фурье по угловой кординате, в результате чего получается система линейных алгебраических уравнений на фурье-гармоники потенциала, каждое из которых независимо друг от друга. Далее двумерное уравнение для каждой гармоники решается методом верхней релаксации с прогонкой по радиальной координате. Общая структура метода показана на следующем рисунке. После окончания итерационного процесса выполняется обратное преобразование Фурье для значений потенциала в плоскости диска.



Распараллеливание

Одной из основных задач распараллеливания является минимизация обмена данными между процессорами. В предлагаемом методе решения уравнения Пуассона их удается полностью исключить за счет того, что уравнения для Фурье-гармоник потенциала не зависят друг от друга. Таким образом, появляется возможность разделить область решения на полностью независимые подобласти по угловым волновым числам, как показано на рисунке. Разделение области равномерное, каждый процессор получает одинаковое количество гармоник.



Частицы разделяются между процессорами равномерно, без учета их расположения в пространстве. Это означает, что для расчета движения частиц каждый процессор должен знать распределение потенциала во всей плоскости диска.

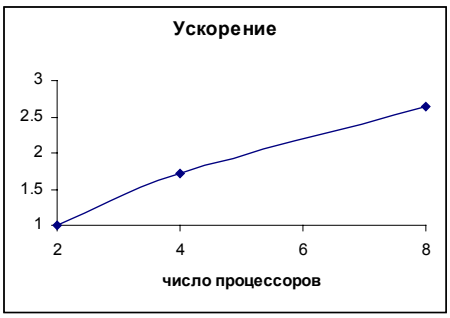
Межпроцессорные коммуникации в программе реализованы в помощью коллективных операций библиотеки MPI. На каждом временном шаге процесса моделирования обмен данными между процессорами выполняется дважды. Во-первых, после завершения итераций выполняется сборка гармоник потенциала в плоскости диска для обратного преобразования Фурье. Затем, после вычисления плотности в каждом процессоре их вклады суммируются.

Тестовые расчеты

Все тестовые расчеты проводились на суперкомпьютере МВС-1000М в ССКЦ, г. Новосибирск (до 8 процессоров) и в МСЦ, г.Москва. Отладка программы проводилась на рабочей станции с двумя процессорами PentiumIII под управлением ОС Linux. Параметры некотрорых расчетов приведены в следующей таблице:

Число процессоров	Процессор	Сетка	Число частиц	Время расчета одного шага,с
1	2	3	4	5
2	PentiumIII	120x128x150	120 тыс.	4
2	Alpha21264	400x512x200	20 млн.	19
1	2	3	4	5
4	Alpha21264	400x512x200	20 млн.	11.1
8	Alpha21264	400x512x200	20 млн.	7.2
8	Alpha21264	400x512x200	100 млн.	28
64	Alpha21264	1000x1024x1000	1 млрд.	141
128	Alpha21264	500x1024x400	100 млн.	204
64	Alpha21264	1000x2048x800	400 млн.	229.8
128	Alpha21264	1000x2048x800	400 млн.	180
256	Alpha21264	1000x2048x800	400 млн.	177.6

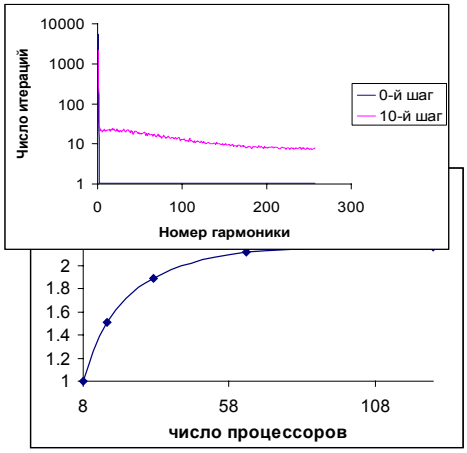
На следующих графиках приведено ускорение расчета на отладоч-



ной сетке 400x512x200 с 20 млн. частиц.

Таким образом, для небольшого числа процессоров эффективность распараллеливания составила 85%. При увеличении числа процессоров ускорение ухудшается:

и свыше 128 процессоров практически отсутствует. Причину этого можно прояснить с помощью следующего графика, на котором показана зависимость числа итераций, требуемое для достижения сходимости для каждой гармоники потенциала.



Распределение плотности изначально симметрично по угловой координате. Поэтому после пребразования Фурье только нулевая гармоника не равна 0, и сходимость для всех остальных гармоник достигается за 1 итерацию. Очевидно, на первом шаге работает только один процессор, содержащий эту гармонику. В дальнейшем картина изменяется (на графике показан 10-й временной шаг процесса моделирования), однако все равно расчет гармоник с малыми номерами занимает больше времени. Таким образом, процессоры во время решения уравнения Пуассона оказываются неравномерно загруженными, причем неравномерность усиливается с увеличением числа процессоров.

Важно отметить, что для различных сеток значения ускорения отличаются. При переходе с 64 процессоров на 128 на графике время расчета уменьшилось только на 2%. При увеличении сетки время

расчета растет быстрее, чем время на коммуникации. Так, для сетки 1000x2048x800 (см. таблицу) при переходе с 64 процессоров на 128 расчетное время уменьшилось на 10%.

Выводы

Проведены расчеты, параметры которых соответсвуют требованиям задачи о моделировании протопланетного диска, и сравнимые с зарубежными разработками. Эффективность распараллеливания 75% до 32 процессоров, что связано с неравномерной загрузкой процессоров.

Литература

1. ***Хокни Р.*, *Иствуд Дж.* Численное моделирование методом частиц. М.: Мир, 1987.**
2. ***Вшивков В.А.*, *Малышкин В.Э.*, *Снытников А.В.*, *Снытников В.Н.* Численное моделирование гравитационной динамики многих тел методом частиц в ячейках: параллельная реализация. СибЖВМ (в печати).**
3. ***Miocchi P.*, *Capuzzo-Dolcetta R.*, An efficient parallel tree-code for the simulation of self-gravitating systems. A&A, http://xxx.lanl.gov/astro-ph/0104152, 2001.**
4. ***Краева М.А.*, *Малышкин В.Э.* Динамическая балансировка загрузки в реализации PIC-метода на MIMD мультикомпьютерах // Программирование. № 1. 1999.**
5. ***Caretti E.*, *Messina A.* Dynamic Work Dustribution for PM Algorithm. http://xxx.lanl.gov/astro-ph/0005512, 2000.**
6. ***MacFarland T.*, *Couchman H.M.P.*, *Pearce F.R.*, *Pilchmeier J.* A New Parallel P^3M Code for Very Large-Scale Cosmological Simulations. New Astronomy, http://xxx.lanl.gov/astro-ph/9805096, 1998.**
7. ***Bode P.*, *Ostriker J.*, *Xu G.* The Tree-Particle-Mesh N-body Gravity Solver, ApJS, http://xxx.lanl.gov/astro-ph/9912541, 1999.**
8. ***Griv E.*, *Gedalin M.*, *Liverts E.*, *Eichler D.*, *Kimhi Ye.*, *Chi Yuan* Particle modeling of disk-shaped galaxies of stars on nowadays concurrent supercomputers. NATO ASI The Restless Universe, http://xxx.lanl.gov/astro-ph/0011445, 2000.**

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ВЫДЕЛЕНИЯ ДВИЖУЩИХСЯ ОБЪЕКТОВ ПО ПОСЛЕДОВАТЕЛЬНОСТИ КАДРОВ

Р.В. Сологуб, Л.К. Бабенко, О.Б. Макаревич, А.Г. Чефранов

УНЦ СИБ,
Таганрогский государственный радиотехнический университет

Введение

В ряде технических приложений существует задача выделения движущихся объектов и анализа характеристик их движения. К таким системам относятся, например, системы дорожного мониторинга, системы слежения, охранные системы. Данные для анализа поступают в виде видеоизображений, получаемых от портативных видеокамер, закрепленных над наблюдаемым объектом. Основная трудность заключается в низком качестве входного видеоизображения, малой размерности выделяемых объектов, сильной зашумленности отдельных кадров. Кроме того, специфика задач требует функционирования системы в реальном масштабе времени.

Предлагаемый подход заключается в сравнительном анализе кадров видеоизображения и формирования изображения неподвижной части сцены на основании статистических данных о яркостных характеристиках изображения. Алгоритм рассматривается на примере системы дорожного мониторинга. Задача алгоритма – выделение движущихся транспортных средств и анализ таких характеристик как количество транспортных средств, скорость и направление их движения. Данные для анализа поступают от неподвижно закрепленной над проезжей частью видеокамеры.

Основные алгоритмы и определения

Будем называть *первичным изображением* область P на плоскости, в каждой точке которой задана *яркость* $f(x,y)$, являющаяся функцией координат (x,y) на плоскости. Поскольку обработка ведется на ЭВМ, функции яркости являются дискретными на плоскости и квантованными по яркости.

Предположим, что имеется серия изображений, состоящая из последовательности нескольких кадров, полученных при наблюдении

системой слежения за движущимся объектом. Для обнаружения целеподобных объектов необходимо провести анализ двух последовательных кадров из такой серии.

Опишем кадр, полученный в момент времени t_i, с помощью множества векторов $X = \{(i, j, f(i, j))\}$, а кадр, полученный в момент времени t_j, в виде $Y = \{(i, j, g(i, j))\}$. Тогда функция яркости $h(x,y)$ разностного полутонового кадра $H = \{(i, j, h(i, j))\}$ определяется следующим образом:

$$h(i, y) = \begin{cases} |f(i, j) - g(x, y)|, \text{если } |f(i, j) - g(x, y)| \geq \delta, \\ 0, \text{в противном случае.} \end{cases}$$

Здесь δ – значение некоторого порогового уровня, обусловленного влиянием шума, возникающего из-за помех в видеоканале и вибрации видеокамеры. Как показали практические исследования, значение δ необходимо выбирать таким, чтобы отделить элементы изображения, в которых отличие значений функции яркости обусловлено движением распознаваемых объектов, от точек, в которых за период времени $t_i - t_j$ произошли незначительные шумовые перепады яркости.

Функция яркости полученного разностного кадра является дискриминантной функцией, так как она принимает свои наибольшие значения в точках, движущихся с максимальной относительной скоростью от кадра к кадру. Области разностного кадра с большими значениями функции яркости принадлежат целеподобным объектам и подлежат дальнейшему анализу.

Для повышения качества распознавания необходимо увеличить площадь, занимаемую на разностном кадре целеподобными объектами. Один из способов достижения этого – рассматривать кадры, отстоящие друг от друга на большем временном интервале. В частности, предлагается рассматривать разность текущего кадра с двумя предыдущими кадрами с последующим объединением результатов. Дальнейшее повышение количества кадров ведет к повышению зашумленности разностного кадра, поскольку помехи также взаимно усиливаются.

Другим подходом к повышению качества распознавания является накопление статистики о фоновом изображении в течении длительного количества времени (20 – 50 и более кадров) и последующем формировании фонового изображения. Если имеется фоновое изображение, можно построить разностный кадр между фоновым изображением и текущим кадром. В этом случае площадь занимаемая целеподобными объектами будет максимальна. Однако при большом усреднении фонового изображения также повышается зашумленность разностного кадра. Кроме того, формирование фонового изображения по результатам статистических данных – довольно дорогая с точки зрения трудоемкости операция.

Полученное разностное изображение, как правило, содержит значительное количество помех, обусловленных различными факторами. К таким факторам относятся дрожь видеокамеры из-за движения транспортных средств, движения мелких элементов фона. Поэтому полученное изображение необходимо дополнительно обработать различными фильтрами. В первую очередь это пороговая фильтрация, которая введена в алгоритм вычисления разностного кадра. Кроме того, для эффективного снижения количества шумов можно использовать алгоритм медианной фильтрации, заключающийся в сортировке значений яркости точек примыкания и выборе в качестве нового значения точки среднего элемента выборки. Под точками примыкания здесь понимается множество точек, координаты которых удовлетворяют условию

$$|x - x_0| < r, \quad |y - y_0| < r,$$

где r – радиус фильтрации.

Данный алгоритм позволяет эффективно удалять малоразмерные помехи, однако имеет достаточно большую трудоемкость и плохо подвергается распараллеливанию. Для устранения этих недостатков предлагается использовать следующий алгоритм, базирующийся на алгоритме медианной фильтрации. Для данной точки вычисляется значение, как в алгоритме пространственной регуляризации [1] по формуле

$$f^*(x_0, y_0) = \frac{1}{r^2} \sum_{x=x_0-r}^{x_0+r} \sum_{y=y_0-r}^{y_0+r} f(x, y).$$

Далее, значения яркостей точек окрестности сравниваются с по-

лученным значением и в качестве нового значения яркости данной точки принимается ближайшее к нему. Если размер помех не превышает $r^2/2$ точек, фильтр эффективно их удаляет. Данный алгоритм хорошо распараллеливается, поскольку вместо задачи сортировки приходится выполнять r^2 сравнений.

Полученный после фильтрации разностный кадр содержит информацию о целеподобных объектах. Для определения количества объектов и их пространственного положения изображения необходимо кластеризовать. Найденные центры кластеров можно считать позициями объектов. Для подсчета количества кластеров используется алгоритм Боннера. Зададим максимальный размер кластера. Центром первого кластера считается произвольная точка изображения. Далее полагаем, что центром следующего кластера могут быть только точки, не принадлежащие ни одному из уже найденных кластеров. Если это условие для рассматриваемой точки выполняется, она считается центром очередного кластера. Таким образом, за один проход по всем точкам изображения возможно посчитать количество кластеров. Далее оставшиеся точки изображения относятся к одному из найденных центров кластеров согласно критерию расстояния, например

$$d = (x - x_0)^2 + (y - y_0)^2.$$

На последнем этапе кластеризации необходимо отбросить кластера, размер которых меньше допустимого минимального размера кластера, что позволяет отбросить помехи, обусловленные движениями мелких объектов, не подлежащих анализу, и скорректировать положение центра кластера по принципу геометрического положения. Полученные центры кластеров пригодны для анализа параметров движения объектов.

Анализ характеристик движения

Для вычисления характеристик движения объектов необходимы данные о положении центров кластеров для последовательности кадров. Предположим, что имеется два массива точек, представляющих центры кластеров в последовательные моменты времени:

$$C_{t_1} = \{(x_i, y_i),\ i = \overline{1, N}, \quad C_{t_2} = \{(x_j, y_j),\ j = \overline{1, N},$$

где N – количество кластеров, обнаруженных в момент времени t_1,

***M* – количество кластеров для момента времени t_2.**

Будем считать, что следующим положением *i*-го кластера из множества C_{t_1} является ближайший к нему кластер из множества C_{t_2}. Тогда движение движущегося объекта описывается вектором с координатами $(x_{t_2} - x_{t_1}, y_{t_2} - y_{t_1})$. Ориентация этого вектора определяет направление движения, а его длина – относительную скорость движения.

Если расстояние от кластера из множества C_{t_1} до любого кластера из множества C_{t_2} превышает некоторое пороговое значение, то считаем, что объект, описываемый этим кластером, покинул поле наблюдения и далее не учитывается.

Параллельная реализация алгоритма

Одним из требований, предъявляемых системе дорожного мониторинга, является функционирование в реальном масштабе времени. Поскольку изображение состоит из множества одинаковых элементов, которые на некоторых этапах алгоритма не зависят от соседних элементов, их можно обрабатывать параллельно. Для повышения эффективности параллельной обработки предлагается использовать специализированный процессор NeuroMatrix NM6403 НТЦ «Модуль». Этот RISC-процессор, ориентированный на обработку целочисленных матриц программируемой разрядности. Суммарный объем обрабатываемой матрицы составляет 32×64 бита, что составляет 256 точек изображения при 8-битном задании одной точки. Над данной матрицей могут выполняться операции суммирования/вычитания, логические операции а также вычисляться пороговые функции активации для каждого элемента матрицы. С учетом специфики функционирования процессора, можно предложить эффективный алгоритм нахождения разностного кадра и наложения пороговой фильтрации. При этом скорость обработки удается повысить до 1.5 тактов на каждую точку изображения. Алгоритм параллельного построения разностного кадра имеет следующий вид:

– в процессор NM6403 загружаются два массива точек – фрагмент анализируемого кадра и соответствующий ему фрагмент предыдущего кадра или фонового изображения;

– ыычисляется разность между этими матрицами;

– для матрицы результата вычисляется пороговая функция, которая устанавливается в качестве маски для векторного узла процессора;

– немаскированные элементы инвертируются. Последние две операции позволяют выполнить операцию инвертирования отрицательных чисел;

– из полученной матрицы вычитается значение порога;

– вычисляется пороговая функция;

– полученная функция накладывается на исходную матрицу по закону XOR.

В результате этого получается фрагмент разностного кадра изображения. Поскольку все операции над рабочей матрицей выполняются за 32 такта и одновременно обрабатывается до 256 точек изображения, время обработки составляет 1 такт на точку без учета пересылки данных или ~1.5 такта с учетом обмена данными между NM6403 и хост-машиной.

В настоящее время ведется работа по повышению качества распознавания в условиях изменяющейся освещенности и плохих погодных условиях.

Работа ведется при поддержке РФФИ (№00-07-90300, 01-07-90211, 02-07-06057, 02-07-06056).

Литература

1. ***Галуев Г.А.*** **Параллельные цифровые нейрокомпьютерные системы и нейросетевые процессоры обработки и распознавания зрительных образов. Таганрог: Сфинкс, 1997.**
2. ***Дюран Б.*****,** ***Оделл П.*** **Кластерный анализ. М.: Статистика, 1977.**
3. ***Ту Дж., Гонсалес Р.*** **Принципы распознавание образов. М.: Статистика. 1979.**

АЛГОРИТМЫ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ СТЕПЕНИ ИНВАРИАНТНОСТИ КРИСТАЛЛИЧЕСКИХ СТРУКТУР*

Н.В. Сомов, С.С. Носов, Е.В. Чупрунов

Нижегородский государственный университет им. Н. И. Лобачевского

Как известно структура кристалла во многом определяет возможный спектр физических свойств кристалла. При этом геометрия кристаллической структуры может определять наличие определенных свойств кристалла безотносительно к его химическому составу. Одной из таких геометрических характеристик является псевдосимметрия, т.е. инвариантность части функции электронной плотности относительно надгруппы группы симметрии кристалла [1]. Вычисление псевдосимметрии представляет собой достаточно ресурсоемкий процесс.

Величина псевдосимметрии определяется функционалом [2]:

$$\eta_t\left[\rho(\mathbf{x})\right] = \frac{\int_V \rho(\mathbf{x})\cdot\rho(\hat{t}\mathbf{x})dV}{\int_V \rho^2(\mathbf{x})dV},$$

где $\rho(\mathrm{x})$ – функция электронной плотности, $\hat{t} = \{q, \boldsymbol{\tau}\}$ – операция симметрии состоящая из q – матрица обобщенного поворота, τ – вектор трансляции. Вычисления данного функционала удобно проводить в обратном пространстве:

$$\eta_{\hat{g}}\left[\rho(\mathbf{x})\right] = \frac{\sum_{\bar{H}}\left[F(\mathbf{H})F(-q\mathbf{H})\exp\{-2\pi i(\mathbf{H}\boldsymbol{\tau})\}\right]}{\sum_{\bar{H}}\left|F(\mathbf{H})\right|^2},$$

где $F(\mathbf{H})$ – Фурье образ функции электронной плотности в обратном

* Работа выполнена при поддержке Фонда содействия развитию малых форм предприятий в научно-технической сфере

пространстве, $\boldsymbol{\tau}$ – трансляционная компонента, $F(\mathbf{H})$ определяется выражением: $F(\mathbf{H}) = \sum_j f_j \exp[2\pi i(\mathbf{H}\mathbf{x}_j)]$, где f_j – представляет собой атомный фактор, который является экспериментальной величиной и содержится в таблицах, для получения непрерывной зависимости нами применялась линейная интерполяция по двум ближайшим узлам.

Сложность задачи заключается в следующем, нам необходимо провести расчет псевдосимметрии для 96000 кристаллических структур Кембриджского банка данных, относительно всех возможных операций симметрии. Притом что, расчет одной структуры занимает время от несколько минут до нескольких часов, если использовать один компьютер средней производительности.

Весь спектр задач по изучению псевдосимметрии кристаллических структур можно разбить на две группы. Первая группа задач представляет собой вычисление псевдосимметрии относительно некоторого небольшого набора операций симметрии у большого числа кристаллических структур, т.е. проведение статистического анализа группы структур. Ко второй группе задач относится детальное изучение псевдосимметрических особенностей конкретной кристаллической структуры. Для проведения такого анализа необходимо вычисление псевдосимметрии для большого числа операций симметрии.

Для решения задач из каждой группы нами был разработан особый алгоритм, позволяющий более рационально использовать ресурс кластера. Рассмотрим решение первой группы задач:

- **на стадии инициализации нулевой процесс рассылает таблицу атомных факторов, набор операций симметрии остальным процессам. Каждый процесс также получает информацию о кристаллической структуре, расчет для которой предстоит провести;**
- **получив все необходимые данные, процессы начинают вычисления;**
- **после завершения вычисления процесс сообщает об этом нулевому процессу и передает ему результат вычислений;**
- **нулевой процесс, приняв результат, вычисление передает информацию о следующей структуре освободившемуся про-**

цессу, предварительно сохранив полученный результат вычислений.

Таким образом, нулевой процесс занимается исключительно распределением данных и сохранением результатов. Остальные процессы занимаются непосредственно вычислениями степени инвариантности кристаллической структуры.

Для такого алгоритма легко записать время работы в случае N процессов и n структур ($n > N$):

$$t_{nap} = \tau_i + a\overline{T} + \tau_{IO}(a + b), \quad (1)$$

$$a = n \text{ div } (N - 1), \quad (2)$$

$$b = n \bmod (N - 1), \quad (3)$$

где div – целочисленное деление, mod – остаток деления, $\overline{T}$ – среднее время расчета структуры, τ_{IO} – время обмена данными с нулевым процессом, τ_i – время инициализации нулевого процесса. Более наглядно смысл этой формулы можно проиллюстрировать на графике (рис. 1) построенном для случая с четырьмя процессами.

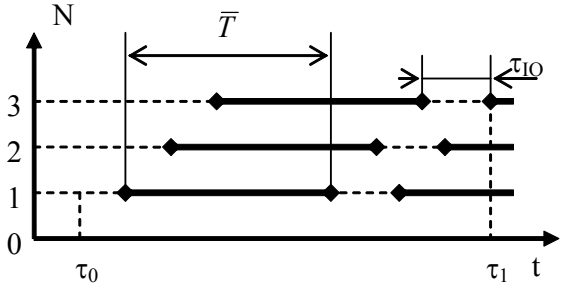


Рис. 1. График работы процессов.

На рис. 1 по оси ординат отложены процессы, а по оси абсцисс время работы. Отрезками показаны периоды времени занятые непосредственно вычислениями. Момент времени τ_0 соответствует окончанию периода инициализации, τ_1 – время окончания одного цикла вычислений, τ_{IO} – время обмена данными.

Выражение (1) при увеличении числа процессов до определенного значения дает снижение времени вычислений (за счет убывания ве-

личины a) затем начинается рост времени расчетов за счет повышающегося вклада времени обмена данными $(a+b)\tau_{IO}$. Очевидно, что влияние этого члена на время расчетов будет расти меньше с увеличением N в случае, когда мало время обмена данными τ_{IO}. Таким образом необходим подбор оптимального количества процессов вычисления. Эффективность работы алгоритма можно записать выражением:

$$S = \frac{n\overline{T}}{\tau_i + a\overline{T} + \tau_{IO}(a+b)}. \qquad (4)$$

Решение задач второй группы сводиться к распределению процесса вычисления псевдосимметрии для одной структуры между несколькими процессами. Наиболее рационально проводить распараллеливание внешнего цикла вычислений, поскольку для эффективной работы кластера необходимо, чтобы время обмена данными было много меньше времени обработки данных процессами.

Рассмотрим более подробно алгоритм решения задач относящихся ко второй группе:

- **нулевой процесс на стадии инициализации рассылает таблицу атомных факторов, набор операций симметрии и исследуемую структуру (каждый процесс получает одну и туже структуру);**
- **все процессы кроме нулевого занимается вычислением собственного участка внешнего цикла;**
- **по окончанию вычислений процессы передают результат своих расчетов нулевому процессу, который, собрав результаты со всех задействованных процессов, формирует результат вычислений;**
- **сохранив результат вычислений, нулевой процесс рассылает всем очередную структуру.**

Время расчета одной структуры запишется в виде:

$$t_2 = \tau_i + T + (N-1)\tau_{IO}, \qquad (5)$$

где T – время вычисления одной части внешнего цикла процессом, N – число процессов, τ_i – время инициализации нулевого процесса, τ_{IO} – время обмена данными. Как и в первом случае, наи-

большая эффективность работы растет с увеличением числа процессов, а затем падает за счет накопления бездействия системы на операциях обмена данными. Эффективность алгоритма будет иметь вид:

$$S_2 = \frac{(N-1)T + \tau_i}{\tau_i + T + (N-1)\tau_{IO}}. \qquad (6)$$

Практическая реализация первого алгоритма показала следующие результаты. Тестовые замеры времени для случая со 100 структурами и четырьмя процессами, представлены в таблице.

Таблица 1

Время работы процессов

τ_i, с	τ_{IO}, с	$\overline{T}$, с	$t_{пар}$, с	$t_{посл}$, с	S
0,2819	0,0177	6,3099	215,4203	631,2719	2,9304

где $t_{пар}$ – время расчета 100 кристаллических структур в параллельном режиме, $t_{посл}$ – время расчета в последовательном режиме, т.е. вычисления, проводились на одном компьютере, S – эффективность вычислений.

Эксперимент показал, что в случае четырех процессов скорость вычислений увеличивается почти в три раза по сравнению с последовательным алгоритмом вычислений, это является хорошим показателем параллельного алгоритма.

Литература

1. *Чупрунов Е.В.*, *Тархова Т Н.*, *Белов Н.В.* // ДАН. 1988. Т. 303, №1. С. 105.
2. *Чупрунов Е.В.*, *Солдатов Е.А.*, *Тархова Т.Н.* // Кристаллография. 1988. Т. 33. №3. С. 753.

АБСОЛЮТ ЭКСПЕРТ – ПРОГРАММНЫЙ КОМПЛЕКС ПАРАЛЛЕЛЬНОГО РЕШЕНИЯ ЗАДАЧ МНОГОМЕРНОЙ МНОГОКРИТЕРИАЛЬНОЙ ОПТИМИЗАЦИИ*

* Работа выполнена в рамках Федеральной Целевой Программы «Интеграция»

А.В. Сысоев, В.П. Гергель

Нижегородский государственный университет им. Н. И. Лобачевского

В настоящей работе рассматривается реализация программного комплекса Абсолют Эксперт, позволяющего исследовать класс задач многомерной многокритериальной оптимизации, характеризующихся существенной сложностью вычислений и большим объемом накапливаемой в ходе работы поисковой информации.

Класс задач и модель вычислений

Рассматривается класс задач выбора оптимального варианта, сводящихся к следующей математической модели. Пусть объект исследования характеризуется набором параметров $\bar{y} = (y,u) = (y_1,...,y_N;\ u_1,...,u_L)$ и вектор-функцией характеристик $w(\bar{y}) = (w_1(\bar{y}),....,\ w_n(\bar{y}))$. Вектор *y* может непрерывно изменяться в гиперинтервале $D = \{y \in R^N: a_i \le y_i \le b_i,\ 1 \le i \le N\}$, кортеж *u* принимает значения в виде набора дискретных параметров из некоторого множества. В отношении части характеристик $w_i(\bar{y})$ ставится условие уменьшения их значений до некоторых заданных допусков, часть характеристик рассматривается как векторный критерий эффективности. Конкретная характеристика может принадлежать обеим частям одновременно. При осуществлении оптимального выбора предполагается упорядоченность частных критериев эффективности, составляющих векторный критерий, по важности. Минимизируется первый по важности частный критерий, назначается величина допустимого увеличения его значения. С учетом этого условия ищется минимальное значение второго критерия и т.д. Получаемое в итоге решение считается оптимальным.

Оценка оптимального выбора исходной многомерной многокритериальной задачи с упорядоченными критериями и существенно нелинейными, невыпуклыми ограничениями, зависящей также и от дискретных параметров, проводится с использованием последовательного редуцирования этой задачи к семейству скалярных одномерных, однокритериальных задач. Понижение размерности осу-

ществляется с помощью разверток. Для сохранения информации о близости точек в многомерном пространстве используется множественная развертка [1], приводящая к появлению семейства из $L + 1$ одномерной задачи оптимизации, каждая из которых эквивалентна исходной N-мерной постановке и определена на интервале [0, 1]. Полученные одномерные задачи обладают важным свойством, состоящим в том, что любое испытание (вычисление функционалов задачи) в D, выполняемое для решения отдельной одномерной задачи, при некоторых условиях может быть интерпретировано как испытание, выполняемое для каждой задачи из семейства. Этот факт позволяет осуществить одновременное (параллельное) решение всех $L + 1$ одномерных задач, используя для каждой задачи отдельный процессор и обеспечив обмен результатами испытаний.

Параллельная схема вычислений

Важным свойством рассматриваемого класса задач в совокупности с предлагаемой схемой решения является то, что времена выполнения испытаний могут существенно различаться как от задачи к задаче, так и от точки к точке. Вследствие этого эффективная схема параллельного решения должна предполагать асинхронность выполнения расчетов отдельными процессорами, а значит, допускается разнотипность процессоров и возможность их отличия по вычислительным характеристикам.

Пусть для решения каждой одномерной задачи используются отдельный процессор. Необходимо обеспечить межпроцессорную передачу поисковой информации, складывающейся из результатов всех выполненных в процессе поиска испытаний. Предлагаемый подход ([2]) состоит в следующем.

В схеме выполнения итерации поиска выделяются три ключевых этапа:

- **поиск следующей точки испытания;**
- **выполнение испытания;**
- **обновление поисковой информации.**

Рассмотрим прохождение каждого этапа конкретным процессором. Перед началом поиска очередной точки испытания процессор пытается получить всю информацию, которая могла быть послана

другими процессорами. Найдя очередную точку испытания, процессор инициирует передачу остальным процессорам информации, содержащей точку испытания (в редуцированном виде $x^k \in [0, 1]$) и некоторый признак, показывающий отсутствие результата вычислений. Цель этого сообщения – исключить дублирование точек выполняемых итераций поиска. Выполнив испытание, процессор обновляет массив собственной поисковой информации и инициирует повторную рассылку сообщения, на этот раз заполненного полностью.

Изложенная схема решает заявленную выше задачу обеспечения асинхронности работы каждого процессора.

Программная реализация

Существенная сложность вычислений и большой объем накапливаемой в ходе работы поисковой информации, характеризующие рассматриваемый класс задач, а также необходимость обеспечения отказоустойчивости, возможности остановки вычислений с сохранением текущего состояния и продолжения счета в дальнейшем, привели к необходимости выделения в функциональной структуре (рис. 1) комплекса следующих подсистем:

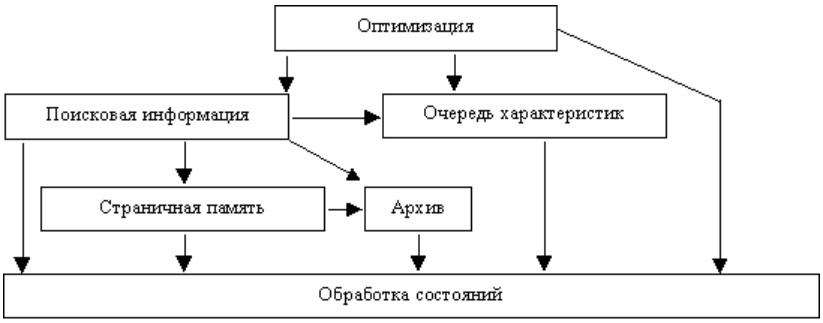
- **подсистема *Оптимизация* – отвечает за построение объекта оптимизации, постановку задач, определение и настройку методов, назначение заданий и выполнение процесса поиска оптимального варианта в рамках изложенной выше модели;**
- **подсистема *Поисковая информация* – реализует структуры данных и методы, обеспечивающие получение, хранение и анализ информации, генерируемой в ходе итераций поиска;**
- **подсистема *Страничная память* – содержит структуры данных и методы для организации страничного представления оперативной памяти, используемой для обработки поисковых данных;**
- **подсистема *Архив* – предназначена для организации хранения поисковой информации во внешней памяти;**
- **подсистема *Очередь характеристик* – ускоряет работу алгоритмов при больших объемах поисковой информации**

(случай высокой точности вычислений или большая размерность исходной постановки);

- **подсистема *Обработка состояний* – имеет целью создание в рамках программного комплекса унифицированной схемы контроля и наблюдения за состояниями системы в целом.**

Подсистема *Оптимизация* является ключевым элементом комплекса. Предлагаемый подход по ее реализации основан на введении в подсистеме иерархической структуры, на вершине которой находится Процесс оптимизации, оперирующий набором заданий. Каждое задание есть объединение семейства совместных (в смысле накапливаемой в процессе поиска оптимума информации) задач, метода их решения, параметров метода. Задачи формируются на основе Объекта оптимизации, параметрами которого являются лишь наиболее общие характеристики: размерность, максимальная область поиска, функционалы. При определении оптимизационной задачи может быть изменена область поиска, часть конструктивных параметров может быть зафиксирована или сведена к дискретному набору параметров. При формировании задания определяются функционалы-ограничения и допуска для них, а также критерии эффективности, единые для входящего в задание семейства задач.

Предложенная структура подсистемы обеспечивает реализацию



требования максимально эффективного использования накапливаемой поисковой информации, предоставляя возможность обмена ею между задачами; позволяет организовать распределенную схему вычислений; предоставляет возможность поэтапной разработки. В настоящий момент существует версия, обладающей всей заявленной функциональностью в однопроцессорном варианте. Ведутся работы по созданию распределенного варианта.

В реализации подсистем Оптимизация и Очередь характеристик принимали участие студенты 5-го курса факультета ВМК ННГУ Веденеева М., Никитина А., Стройков Н.

Литература

1. *Strongin R.G., Sergeev Ya.D*. (2000). Global optimization with non-convex constraints: Sequential and parallel algorithms. Kluwer Academic Publisher, Dordrecht.
2. *Гергель В.П.*, *Стронгин Р.Г.* (2001). Параллельные вычисления в задачах выбора глобально-оптимальных решений для многопроцессорных кластерных систем. // Современные методы математического моделирования: Сб. лекций Всероссийской молодежной школы международной конференции «Математическое моделирование». Самара. С. 46–55.
3. *Gergel V.P*. A software system for multiextremal optimization // European Journal of Operation Research. V. 65. N 3. P. 305–313, 1993

РЕАЛИЗАЦИЯ ПРОТОТИПА КОНФЛЮЭНТНОЙ СИСТЕМЫ ПРОДУКЦИЙ НА МНОГОПРОЦЕССОРНОЙ ЭВМ

М.Б. Тютюнник
ДВГУ, г. Владивосток

Введение

Системы продукций (СП) получили широкое распространение как средство представления знаний в экспертных системах. При использовании систем продукций знания представляются в виде совокупности правил, описывающих взаимосвязи типа «если-то» между данными предметной области.

Одним из классов СП являются конфлюэнтные продукции, основное достоинство которых состоит в том, что для них результат процесса логического вывода не зависит от порядка выполнения правил.

Система продукций РЕПРО относится к классу конфлюэнтных продукций [1]. При реализации системы продукций РЕПРО на од-

нопроцессорной ЭВМ выполняется компиляция правил в подпрограммы на алгоритмическом языке.

Конфлюэнтные продукции обладают еще одним свойством, позволяющим рассматривать их реализацию на параллельной машине: все правила являются независимыми друг от друга, изначально параллельными.

Вычислительная среда

Мультикомпьютер, на котором предполагается реализовать систему продукций РЕПРО, представляет собой кластер, состоящий из 5 узлов. На мультикомпьютере установлена операционная система Linux, для организации взаимодействия параллельных процессов используется протокол MPI (реализация LAM). Кластер имеет следующую конфигурацию:

– процессоры: Dual CPU P-III/800EB MHz;
– кэш-память: 256 K;
– частота системной шины: 133 MHz;оперативная память: 2*SDRAM 128Mb SAMSUNG;
– материнская плата: MB Super Micro P-III/DME <2xSlot1, i840, ATX>

Кластер состоит из 10 процессоров.

Система продукций РЕПРО

Язык представления знаний (ЯПЗ) РЕПРО является универсальным ЯПЗ продукционного типа [2]. Объекты ЯПЗ РЕПРО моделируют понятия предметной области в виде индивидов и отношений. Знания на ЯПЗ РЕПРО представляются с помощью правил. Язык поддерживает создание модульных баз правил. Взаимодействие между модулями организует управляющий модуль, представляющий собой программу (или подпрограмму) на алгоритмическом языке. Процесс решения задачи в экспертной системе является процессом выполнения логических модулей в последовательности, определяемой модулем управления. При необходимости могут быть использованы процедурные модули, представляющие собой подпрограммы на алгоритмическом языке.

Отношения РЕПРО могут быть точными либо недоопределенными. Значения точных отношений задаются в исходных данных и не меняются в процессе логического вывода. Значения недоопределенных отношений в процессе логического вывода уточняются. Для точных отношений задаются наборы кортежей значений аргументов, на которых точное отношение имеет значение «истина». Для тех наборов значений аргументов, которые не заданы для точного отношения, отношение имеет значение «ложь». Для недоопределенных отношений используется трехзначная логика: отношение может иметь значение «истина», «ложь» либо «не определено» для наборов аргументов. В процессе уточнения значение «не определено» может измениться на «истина» либо «ложь».

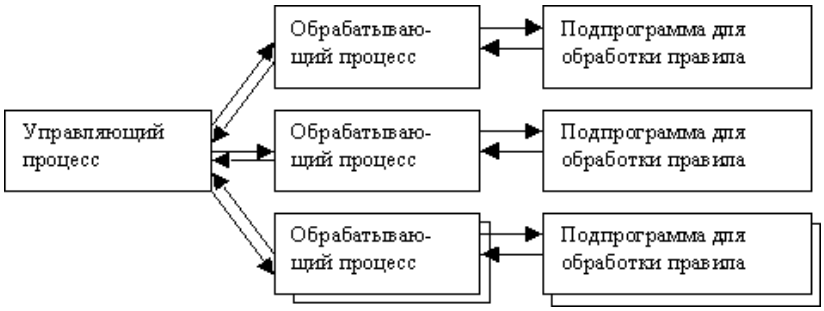
Процесс логического вывода

Процесс логического вывода (ПЛВ) определяется как процесс, состоящий из последовательности шагов. На каждом шаге ПЛВ в СП решаются следующие подзадачи: выборка правил базы правил и формирование множества активных правил (АП); выбор правила из АП и нахождения для него любых означиваний; организация выполнения действий, определяемых следствием выбранного правила. В результате выполнения действий изменяется состояние рабочей среды: состояние ПЛВ и множество АП.

В РЕПРО каждое правило компилируется в подпрограммы на алгоритмическом языке. Каждая подпрограмма выполняет поиск всех означиваний для правила и выполняет действия, определяемые следствием правила. Кроме подпрограмм, соответствующих правилам, РЕПРО формирует управляющую подпрограмму, которая организует выбор правила из множества активных правил. В состав РЕПРО также входят подпрограммы среды поддержки логического вывода: их назначение – поддержка множества активных правил, ввод исходных данных и вывод результата.

Распараллеливание

Так как система продукций, рассматриваемая в данной работе, является конфлюэнтной, и вычислительная система построена на основе многопроцессорной ЭВМ, можно использовать параллельные



вычисления для обработки правил при всех существующих означиваниях.

Рис. 1. Параллельная обработка правил

В отличие от ПЛВ, реализованного на однопроцессорном компьютере, где каждое правило обрабатывается одно за другим, многопроцессорная машина позволяет построить работу программной системы, реализующей ПЛВ, как набор процессов, выполняемых разными узлами компьютера. Одному из процессов предложена роль диспетчера, управляющего работой по подготовке правил для обработки, в то время как остальные процессы выполняют обработку каждого правила независимо друг от друга. Такое параллельное выполнение вычислений позволяет уменьшить время работы ПЛВ по сравнению с аналогичными вычислениями на однопроцессорной ЭВМ.

Управляющий процесс координирует работу с данными и правилами и организовывает запуск процессов-обработчиков. Первоначально управляющий процесс производит выборку правил базы правил и формирует множество активных правил. Затем определяется количество свободных процессов-обработчиков, которым и передаются данные, необходимые для обработки каждого правила из множества активных правил. Наконец, данные, полученные от каждого отработавшего процесса, синхронизируются с оригинальными данными. Это значит, что результат выполнения действий, описанных в следствии правила, добавляется к имеющимся данным, а само правило записывается в множество АП, если для него возможно появление новых означиваний (которые обычно появляются в том случае, когда текущее состояние ПЛВ расширяется новыми фактами).

Процесс-обработчик работает в режиме ожидания данных от управляющего процесса и ничего «не знает» о деятельности других процессов. Данные, которые он получает от управляющего процесса, несут информацию только об одном правиле и о тех объектах, которые используются в правиле. После получения данных проверяется условие применимости правила, и формируются действия, выполняемые в результате применения этого правила. Затем все выходные данные пересылаются обратно управляющему процессу.

Рабочая память

Кластер характеризуется разделенной памятью, вследствие чего протокол MPI не позволяет использовать память как единое целое, поэтому программная система (ПС) не может использовать глобальные данные. Обычно применяется следующий метод: в одном из процессов (назовем его главным) заводится оригинал данных, с которых снимаются копии и пересылаются зависимым процессам. В свою очередь, данные, полученные от зависимых процессов, синхронизируются с оригиналом.

Применительно к ПС, объекты и их значения будут храниться в памяти, используемой управляющим процессом. Для передачи данных процессам-обработчикам управляющий процесс снимает копию той части памяти, где находятся необходимые данные. Результаты же обработки правил, полученные от процесса-обработчика, сравниваются с оригинальными, хранящимися в памяти управляющего процесса, и дополняют их при необходимости.

Сравнение времени выполнения системы продукций на однопроцессорной и многопроцессорной ЭВМ

Для получения оценки эффективности работы схемы распараллеливания, следует сравнить ее со схемой организации процесса логического вывода на однопроцессорной ЭВМ. Если одновременно на нескольких процессорах работают несколько процессов, то время работы определяется наибольшим временем выполнения одного из запущенных процессов. Если предположить, что процессоров хватает для выполнения всех правил одновременно, то время выполнения процесса логического вывода совпадает с максимальным

временем обработки правил. Если же процессоров не хватает, то время выполнения процесса логического вывода будет, очевидно, суммой таких времен.

При выполнении процесса логического вывода на однопроцессорной ЭВМ время выполнения есть сумма времен обработки всех правил. Очевидно, что распараллеливание дает выигрыш по времени выполнения процесса логического вывода. Можно предположить, что чем большее число правил содержит база правил, тем больше будет выигрыш.

Другие схемы распараллеливания

Схема распараллеливания, рассмотренная выше, не является единственной для системы конфлюэнтных продукций. Одной из альтернативных схем организации процесса логического вывода является схема, в которой учитываются связи между правилами по передаче данных; эти связи отражаются в графе передачи данных. Если при учете связей между правилами может быть построена последовательность правил, тогда при организации работы на параллельной системе распараллеливается процесс выполнения одного правила. Если же в графе существует несколько ветвей, то распараллеливание состоит в параллельном выполнении веток. В дальнейшей работе предполагается исследовать все возможные схемы распараллеливания, получить оценки времени выполнения для каждой из схем и выбрать более оптимальную схему для реализации системы конфлюэнтных продукций на многопроцессорной ЭВМ.

Литература

1. *Артемьева И.Л., Клещев А.С.* Вывод в системах продукций с недоопределенными объектами. Процесс логического вывода. Препр. Владивосток: ИАПУ ДВО РАН , 1992. 41 с.
2. *Артемьева И.Л., Клещев А.С.* Расширенная модель декларативных продукций. Препр. Владивосток: ИАПУ ДВО АН СССР, 1991. 36 с.

ЭФФЕКТИВНОСТЬ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ ВЛОЖЕННЫХ МЕТОДОВ РУНГЕ–КУТТА ПРИ МОДЕЛИРОВАНИИ СЛОЖНЫХ ДИНАМИЧЕСКИХ СИСТЕМ

Л.П. Фельдман, И.А. Назарова
Донецкий национальный технический университет, Украина

Введение

Моделирование сложных динамических систем с сосредоточенными параметрами требует решения систем обыкновенных дифференциальных уравнений (СОДУ) большой размерности. Эффективное решение таких систем возможно лишь с использованием высокопроизводительных параллельных вычислительных средств. Одна из основных проблем, связанных с применением методов Рунге–Кутта, состоит в получении надежных и, в то же время простых и эффективных оценок погрешности решения задачи Коши. В решении данной проблемы значительные успехи достигнуты за счет введения вложенных методов Рунге-Кутты (ВМРК)[1, 2]. В докладе рассматриваются параллельные алгоритмы вложенных методов Рунге-Кутты для процессоров SIMD структуры при использовании квадратной решетки или ее замкнутого эквивалента – тора.

1. Общая характеристика вложенных методов Рунге–Кутта

Численно решается задача Коши, уравнение или система обыкновенных дифференциальных уравнений (ОДУ) с известными начальными условиями :

$$\begin{cases} y' = f(x, y); \\ y(x_0) = y_0. \end{cases} \quad (1$$

Общая схема методов Рунге–Кутта с вложенным решением имеет вид:

$$\begin{cases} y^{n+1} = y^n + h_n \cdot \sum_{l=1}^{s} b_l \cdot k_l; \\ \widehat{y}^{n+1} = y^n + h_n \cdot \sum_{l=1}^{s'} \widehat{b}_l \cdot k_l; \\ k_l = f(x_n + c_l \cdot h_n; y_l + h_n \cdot \sum_{i=1}^{l-1} a_{li} \cdot k_i; l = 1,...,s; \\ d^{n+1} = \left| (\widehat{y}^{n+1} - y^{n+1}) \right|. \end{cases} \quad (2)$$

Для определения локальной погрешности менее точного результата и управления величиной шага интегрирования используется величина пропорциональная d^{n+1}. Вложенный метод Рунге-Кутты $r(\widehat{r})$ – это схема, в которой метод $\widehat{r}$ -го порядка получается, как побочный продукт метода r-го порядка. Порядок аппроксимаций: y^n и $\widehat{y}^n$ обычно отличаются на 1, т.е. $r = \widehat{r} + 1$ или $r = \widehat{r} - 1$.

2. Вычислительные схемы вложенного метода Кутта–Мерсона

Идея вложенных форм для методов Рунге–Кутта реализована в различных вычислительных схемах той или иной эффективности. В докладе рассмотрены параллельные вложенные методы Кутта–Мерсона, Фельберга и Дормана–Принса [1,2]. В силу громоздкости приводится схема и оценки качества параллельного метода Кутта–Мерсона (5) для решения линейных однородных СОДУ с постоянными коэффициентами в силу того, что лучшие результаты схема дает именно в этом случае. Численное решение таких систем пошагово можно получить по следующей схеме Кутта–Мерсона:

$$\begin{cases} \bar{k}_1^n = A\bar{y}^n; \\ \bar{k}_2^n = A[\bar{y}^n + h\bar{k}_1^n/3]; \\ \bar{k}_3^n = A[\bar{y}^n + h\bar{k}_1^n/6 + h\bar{k}_2^n/6]; \\ \bar{k}_4^n = A[\bar{y}^n + \bar{k}_1^n h/8 + 3\bar{k}_2^n h/8]; \\ \bar{k}_5^n = A[\bar{y}^n + \bar{k}_1^n h/2 - 3\bar{k}_2^n{}_h/2 + 2h\bar{k}_4^n]; \\ \bar{y}^{n+1} = \bar{y}^n + h/2(\bar{k}_1^n - 3\bar{k}_3^n + 4\bar{k}_4^n); \\ \bar{\bar{y}}^{n+1} = \bar{y}^n + h/2(\bar{k}_1^n + 4\bar{k}_4^n + \bar{k}_5^n), \end{cases} \quad (3)$$

где $\bar{y}$ – вектор неизвестных; $\bar{y}_0$ – вектор начальных условий; A – матрица коэффициентов линейной системы. Выполним элементарные преобразования системы (3) для оценки времени последовательной реализации одного шага интегрирования:

$$\begin{cases} \bar{k}_1^n = A\bar{y}^n; \\ \bar{k}_2^n = \bar{k}_1^n + A\bar{k}_1^n h/3; \\ \bar{k}_3^n = \bar{k}_1^n + A\bar{k}_1^n h/6 + A\bar{k}_2^n h/6; \\ \bar{k}_4^n = \bar{k}_1^n + A\bar{k}_1^n h/8 + 3A\bar{k}_2^n h/8; \\ \bar{k}_5^n = \bar{k}_1^n + A\bar{k}_1^n h/2 - 3A\bar{k}_2^n h/2 + 2hA\bar{k}_4^n. \end{cases} \quad (4)$$

Время, затраченное на выполнение последовательного вычисления шаговых коэффициентов по схеме (4):

$$T_{\bar{k}} = 4t_{A\times\bar{y}} + 8t_{C\times\bar{y}} + 8t_{\bar{y}+\bar{y}}, \quad (5)$$

где $t_{A\times\bar{y}}$ – время умножения матрицы на вектор; $t_{C\times\bar{y}}$ – время умножения вектора на скаляр; $t_{\bar{y}+\bar{y}}$ – время выполнения сложения векторов. Общее время, затрачиваемое на реализацию одного шага интегрирования, по схеме Кутта–Мерсона (4) включает время вычисления шаговых коэффициентов, обеих аппроксимаций и оценки локальной погрешности:

$$T_{1шага} = 4t_{A\times\bar{y}} + 13t_{C\times\bar{y}} + 15t_{\bar{y}+\bar{y}}. \quad (6)$$

Предложенная схема имеет достаточно высокую вычислительную сложность. Для сокращения вычислений представим величины $\bar{y}$,

$\bar{\bar{y}}$ и $\bar{d}$ на каждом шаге, как функции матрицы коэффициентов системы и шага интегрирования.

После упрощающих преобразований получим:

$$\begin{cases} \bar{\bar{y}}^{n+1} = [E + hA + \dfrac{h^2}{2}A^2 + \dfrac{h^3}{6}A^3 + \dfrac{h^4}{24}A^4] \cdot \bar{y}^n; \\ \bar{y}^{n+1} = [E + hA + \dfrac{h^2}{2}A^2 + \dfrac{h^3}{6}A^3] \cdot \bar{y}^n; \\ \left| \bar{\bar{y}}^{n+1} - \bar{y}^{n+1} \right| = \dfrac{h^4}{24}A^4 \cdot \bar{y}^n. \end{cases} \quad (7)$$

Вычислительная схема (7) позволяет уменьшить время каждого шага интегрирования за счет создания подготовительного этапа, который будет выполняться до начала основного счета и содержать операции, не связанные непосредственно с номером шага. Основными операциями подготовительного этапа будут: вычисление степеней матрицы коэффициентов системы A^2, A^3, A^4 и умножение матриц на скаляры. Заметим, что эти операции являлись наиболее трудоемкими в схеме (7). Время подготовительного этапа вычисляется, как:

$$T_{подг.этапа} = 3t_{A\times A} + 3t_{C\times A},$$

где $t_{A\times A}$ – время вычисления скалярного произведения матриц; $t_{C\times A}$ – время умножения матрицу на скаляр. Определим время выполнения одного шага интегрирования по схеме (7) с учетом того факта, что нет необходимости в вычислении обоих приближений, достаточно знать $\bar{y}^{n+1}$ и $\bar{d}^{n+1}$:

$$T'_{1шага} = 2t_{A\times\bar{y}} + 4t_{C\times A} + 3t_{умн} + 3t_{A+A}. \quad (8)$$

3. Параллельная реализация вложенных схем Кутта–Мерсона

Рассмотрим отображение алгоритмических схем Кутта–Мерсона на многопроцессорные вычислительные системы SIMD-структуры с распределенной памятью. Конфигурацию системы считаем фиксированной: число процессорных элементов и схема их соединения не изменяются в процессе счета. Каждый процессор может выпол-

нить любую арифметическую операцию за один такт, временные затраты, связанные с обращением к памяти отсутствуют.

Параллельная реализация метода Кутта–Мерсона требует распараллеливания следующих базовых операций: матричное и векторное умножение и сложение, а также умножение вектора и матрицы на скаляр. Из предыдущего анализа вычислительных свойств метода можно предположить, что одно из наиболеее оптимальных топологических решений – это квадратная сетка $m \times m$ или ее замкнутый эквивалент – тор. На такой топологической схеме достаточно эффективно выполняются матричные операции, составляющие основу преобразованных вложенных форм. Вычисление матричного умножения и умножения матрицы на вектор может быть выполнено по систолическому алгоритму, который является наиболее эффективным для SIMD-систем [3]:

$$T_{AxA}^{SYS} = mt_{ум} + (m-1)t_{сл} + 3(m-1)t_{сд}, \qquad (9)$$

где $t_{ум}, t_{сл}, t_{сд}$ – времена выполнения одиночных операций умножения, сложения и сдвига. Вычисление систолического умножения матрицы на вектор на базе алгоритма сдваивания требует следующего времени выполнения[4]:

$$T_{AxY}^{SYS} = t_{ум} + (m + \hat{m} - 2)t_{сд} + \log_2 \hat{m} t_{сл}, \qquad (10)$$

где $\hat{m} = 2^l - 1$ и l = [$\log_2 m$]. Тогда, время решения задачи за n шагов интегрирования по схеме (4) равно:

$$T_{KM} = (17nt_{умн} + (4Log_2\hat{m} + 15)nt_{сл} + [4n(m + \hat{m} - 2) + (m-1)]t_{сд}.$$

Для преобразованной схемы (7) общее время выполнения равно:

$$T'_{KM} = (9n + 3m + 3)t_{умн} + (nLog_2\hat{m} + 3n + 3m - 3)t_{сл} +$$

$$+ [n(m + \hat{m} - 2) + 9(m-1)]t_{сд},$$

в том числе время подготовительного этапа составляет:

$$T_{подг.этапа} = (3m + 3)t_{умн} + (3m - 3)t_{сл} + 9(m-1)]t_{сд}..$$

Анализ полученных результатов позволяет сделать следующие выводы:

- подготовительный этап схемы (7) имеет большую вычислительную и обменную сложность, что не является недостатком в силу того, что эти операции выполняются один раз до начала основного счета;
- вычислительная сложность преобразованной схемы Кутта–Мерсона меньше, чем схемы без преобразований;
- количество обменных операций в схеме (4) приблизительно в четыре раза больше, чем в схеме (7);
- обе вычислительные схемы имеют больший удельный вес операций обмена, чем арифметических в общем объеме вычислений.

5. Оценка качества параллельных схем Кутта–Мерсона

Для сравнения и оценки качества предложенных параллельных вычислительных схем алгоритма Кутта–Мерсона используются следующие наиболее распространенные критерии: коэффициет ускорения S и эффективности E. При этом считается, что последовательные варианты этих схем реализованы на однопроцессорной ВС с быстродействием арифметического процессора и объемом ОЗУ равным суммарному объему всех ЗУ арифметических процессоров и с необходимым числом внешних устройств, имеющих скорости обмена такие же, что и МПВС типа SIMD.

Приведем характеристики потенциального параллелизма схем Кутта–Мерсона (4) и (7), а также оценку качества алгоритмов с учетом операций обмена.

Для вычислительной схемы (4) время реализации на однопроцессорной ВС составляет для n точек интегрирования:

$$T_{посл} = (4m^2 + 13m)t_{умн} + (4m^2+15m)t_{сд},$$

тогда коэффициент ускорения без учета обменных операций равен

$$S_{пот.КМ} = \frac{n(4m^2 + 13m)t_{умн} + n(4m^2 + 15m)t_{сд}}{17nt_{\text{умн}} + (4\log_2 \widehat{m} + 15)\text{nt}_{\text{сл}}},$$

а с учетом таковых:

$$S_{КМ} = \frac{n(4m^2 + 13m)t_{умн} + n(4m^2 + 15m)t_{сд}}{17nt_{\text{умн}} + (4\log_2 \widehat{m} + 15)nt_{\text{сл}} + [4n(m + \widehat{m} - 2) + (m - 1)]t_{\text{сд}}}.$$

Качество вычислительной схемы (7) может быть оценено следующим образом:

$$S'_{пот.КМ} = \frac{(3m^3 + 3m^2 + 6nm)t_{\text{умн}} + (3m^3 + 5nm)\text{t}_{\text{сл}}}{(9n + 3m + 3)t_{\text{умн}} + (n\log_2 \widehat{m} + 3n + 3m - 3)t_{\text{сл}}},$$

$$S'_{КМ} = [(3m^3 + 3m^2 + 6nm)t_{\text{умн}} + (3m^3 + 5nm)\text{t}_{\text{сл}}]\{(9n + 3m + 3)t_{\text{умн}} +$$

$$+ (n\log_2 \widehat{m} + 3n + 3m - 3)t_{\text{сл}} + [n(m + \widehat{m} - 2) + 9(m - 1)]t_{\text{сд}}\}^{-1}.$$

Коэффициент эффективности может быть получен из коэффициента ускорения путем деления, в данном случае на m^2. Определение характеристик параллелизма осуществлялось с помощью пакета *Mathematica®* (Wolfram Research Inc.).

Заключение

Проведенный сравнительный анализ различных схем вложенных методов Рунге–Куттf показал существенное различие в эффективности их отображения на параллельную SIMD-систему с матричной топологией. Модифицированный метод с вычислительной схемой (2) продемонстрировал улучшение характеристик потенциального параллелизма и параллелизма реального, учитывающего, кроме арифметических, еще и обменные операции, которые могут непосредственно влиять на сложность алгоритма в целом.

Таким образом, сокращение времени решения и, следовательно, времени моделирования сложных технических систем с изменяющимися во времени параметрами, возможно при использовании многопроцессорных вычислительных систем, эффективность работы которых существенно зависит от характеристик применяемых параллельных алгоритмов.

Литература

1. *Хайрер Э., Нерсетт С., Ваннер Г.* Решение обыкновенных дифференциальных уравнений. Нежесткие задачи: Пер. с англ. М.: Мир, 1990.
2. *Bergman S.*, *Rauber T.*, *Runger G*. Parallel execution of embedded Runge-Kutta methods.International Journal of Supercomputer Applications, 10(1):62-90, 1996.
3. *Бройнль Т.* Паралельне програмування: Початковий курс: Навч. посібник / Вступ.слово А.Ройтера; Пер. з нім. В.А. Святного. Киев: Вища школа, 1997.
4. *Фельдман Л.П.* Параллельные алгоритмы моделирования динамических систем, описываемых обыкновенными дифференциальными уравнениями // Научн. тр. ДонГТУ. Серия:Информатика, кибернетика и вычислительная техника, (ИКВТ-99). Вып. 6: Донецк: Изд-во ДонГТУ, 1999. С. 24–29.

ВЕРОЯТНОСТНЫЕ МОДЕЛИ АНАЛИЗА ОЦЕНКИ ЭФФЕКТИВНОСТИ КЛАСТЕРНЫХ СИСТЕМ

Л.П. Фельдман, Т.В. Михайлова

Донецкий национальный технический университет, Украина

Одной из систем классификации кластерных систем является классификация по совместному использованию одних и тех же дисков отдельными узлами: кластеры с совместным и раздельным использованием дискового пространства [1].

Одной из больших проблем при построении таких вычислительных систем (ВС) с распределенной обработкой является выработка рекомендации для рационального использования ресурсов этой сети. Эта задача возникает как при эксплуатации, так и при проектировании сети.

Модели кластеров с совместным использованием дискового пространства

Упрощенная модель кластера с совместным использованием дискового пространства приведена на рис. 1, в которой пренебрегли

мультиплексором и контроллерами (шиной). В ней M рабочих станций пользователей, N1 однотипных кластеров, выполняющие одинаковые функции (например, распределенная СУБД).

Каждый из M пользователей посылает с вероятностью p_{12} посылает запрос к одному из N1 серверов, которые, в свою очередь, обрабатывая этот запрос обращаются к одному из N2 дисков с вероятностью p_{23}.

Функционирование рассматриваемой системы можно представить замкнутой стохастической сетью [2], содержащей три системы массового обслуживания (СМО), в которой циркулирует M заявок. Граф передач этой сети изображен на рис.2, на котором введены следующие обозначения:

S_1, S_2, S_3 – СМО, соответствующие клиентам, серверам и дисковым пространствам, соответственно.

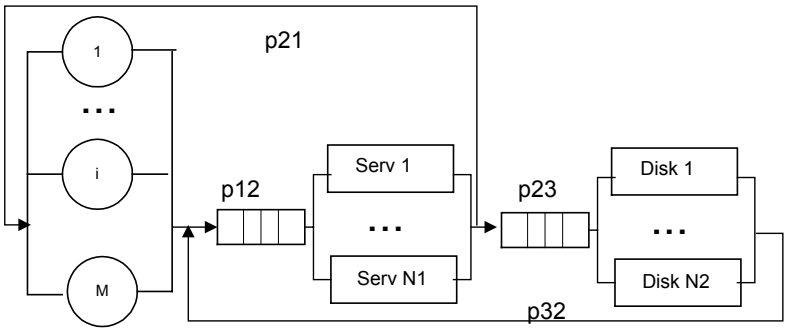


Рис. 1. Структурная схема кластера с совместным использованием дисков

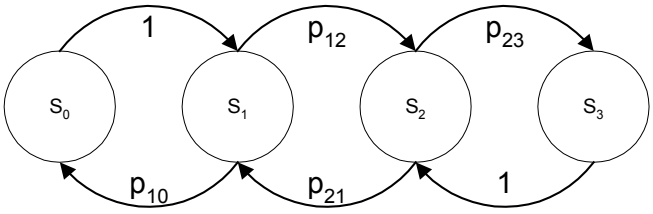


Рис. 2. Граф передач

По графу передач можно определить коэффициенты посещений каждой СМО, решая систему уравнений (1).

$$\begin{aligned} \alpha_1 &= 1 + p_{21} * \alpha_2, \\ \alpha_2 &= p_{12} * \alpha_1 + p_{32} * \alpha_3, \\ \alpha_3 &= p_{23} * \alpha_2. \end{aligned} \qquad (1)$$

Если за состояние системы принять распределение заявок по СМО $\overline{m} = (m_1, m_2, m_3)$, где $m_1+m_2+m_3 = M$, то по теореме Джексона [3] получается следующее выражение для стационарных вероятностей:

$$\pi(\overline{m}) = \frac{1}{Q(M,N)} \prod_{j=1}^{N} R_j(m_j)(\alpha_j \nu_j)^{m_j}, \qquad (2)$$

где

$$Q(M,N) = \sum_{повсемсостояниям} \prod_{j=1}^{N} R_j(m_j)(\alpha_j \nu_j)^{m_j}, \qquad (3)$$

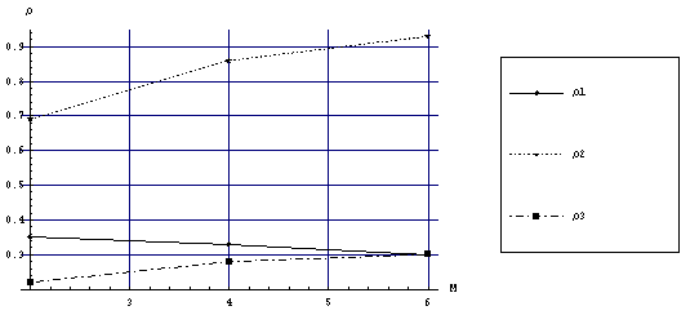
$$R_j(m_j) = \begin{cases} 1/m_j!, & m_j \le k_j, \\ 1/(k_j! k_j^{m_j - k_j}), & m_j > k_j, \end{cases} \qquad (4)$$

$$\mu_i(m_i) = \begin{cases} m_i / \nu_i, & m_i \le k_i, \\ k_i / \nu_i, & m_i > k_i. \end{cases} \qquad (5)$$

Основные характеристики кластера выражаются через стационарные вероятности состояний [4].

Задав класс решаемых задач переходными вероятностям (например, $p_{10} = 0{,}05$, $p_{12} = 0{,}95$, $p_{21} = 0{,}75$, $p_{23} = 0{,}25$), варьируя количеством клиентов M (начиная с M=1), можно проанализировать эффективность такого кластера (например, нагрузку серверов).

Для данного класса задач и конфигурации кластера «узким» местом в сети является сервер (рис. 3), загрузка которого за пределом по-



роговых значений уже для шести клиентов.

Рис. 3. Зависимость нагрузки узлов сети (ρ)
от количества клиентов (M)

Для уменьшения загрузки надо ввести еще один сервер с аналогичными техническими характеристиками и функциональными возможностями (рис. 4).

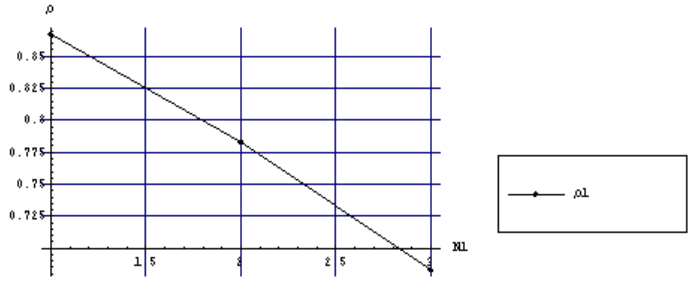


Рис. 4. Зависимость нагрузки сервера (ρ) от количества серверов (N1)

Для класса задач с большой вероятностью обращения к серверу (например, $p_{12} = 0{,}95$; $p_{12} = 0{,}75$) и относительно маленькой вероятностью обращения к диску (например, $p_{23} = 0{,}25$) эффективность кластера повышается за счет увеличения количества серверов (рис. 6).

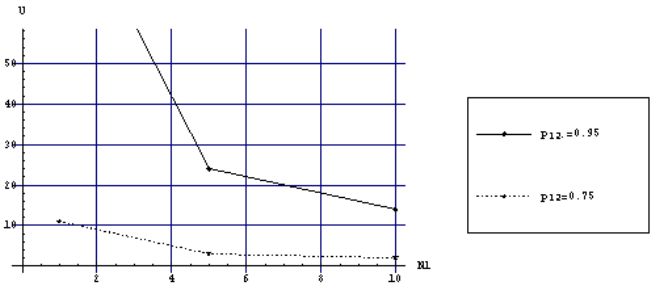


Рис. 6. Зависимость времени отклика от количества серверов

(N1- количество серверов, U - время отклика)

Производительность таких кластеров увеличивается за счет увеличения количества серверов или количества дисков, или того и другого. При этом время отклика уменьшается, однако, увеличивается стоимость кластера Выигрыш, полученный за счет малого времени отклика, должен соизмеряться с ценой потерь, возникающих из-за недогрузки оборудования, что характеризуется критерием сбалансированности [3].

Узкоспециализированные кластеры устроены таким образом, что каждый сервер выполняет свое приложение. Модель такого кластера с совместным использованием дискового пространства (если сделать те же упрощения, что и в предыдущей модели) можно представить так как на рис. 7. В ней M рабочих станций пользователей, N1 серверов, выполняющие разные приложения, N2 диска.

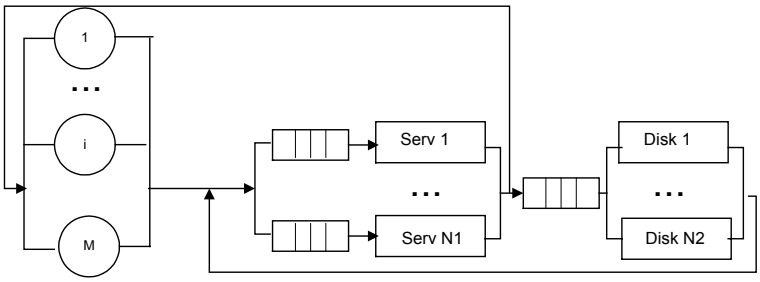


Рис. 7. Модель кластера с совместным использованием дискового пространства

Каждый из M пользователей посылает с вероятностью $p_{N1+2,i}$ запрос к i-му серверу, который, в свою очередь, обрабатывая этот запрос обращаются к одному из N2 дисков с вероятностью $p_{i,N1+1}$.

Функционирование рассматриваемой системы можно представить замкнутой стохастической сетью, содержащей N1+2 системы массового обслуживания (СМО), в которой циркулирует M заявок. Граф передач этой сети изображен на рис. 8

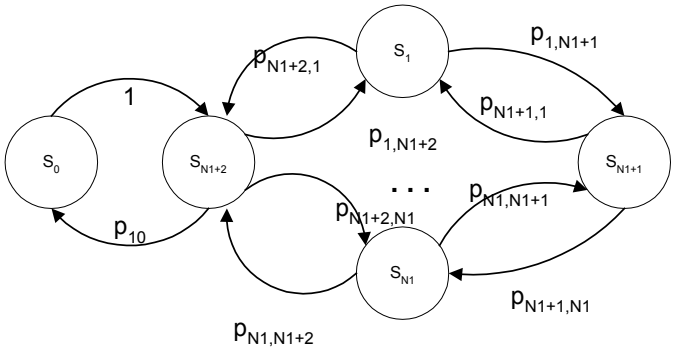


Рис. 8. Граф передач сети

На этом рисунке введены следующие обозначения: $S_{N1+2,1}$ – СМО, соответствующая клиентам, $S_1,\ldots,S_{N1}$ – СМО, соответствующие серверам; S_{N1+1} – СМО, соответствующая дисковым пространствам.

По графу передач можно определить коэффициенты посещений каждой СМО, решая систему уравнений (6).

$$\begin{aligned}
\alpha_{N1+2} &= 1 + p_{1,N1+2}*\alpha_1 + \ldots + p_{N1,N1+2}*\alpha_{N1},\\
\alpha_{N1+1} &= p_{1,N1+1}*\alpha_1 + \ldots + p_{N1,N1+1}*\alpha_{N1},\\
\alpha_1 &= p_{N1+2,1}*\alpha_{N1+2} + p_{N1+1,1}*\alpha_{N1+1},\\
&\ldots\ldots\ldots\ldots\\
\alpha_i &= p_{N1+2,i}*\alpha_{N1+2} + p_{N1+1,i}*\alpha_{N1+1},\\
\alpha_{N1} &= p_{N1+2,N1}*\alpha_{N1+2} + p_{N1+1,N1}*\alpha_{N1+1}.
\end{aligned} \qquad (6)$$

Такие кластеры практически не масштабируются (что видно из рис. 9): время отклика слабо изменяется в зависимости от добавления

новых устройств

Рис. 9.

Поэтому изначально надо как можно «идеальней» проектировать такие кластеры или использовать другую (более дорогую) структуру кластера, а именно, увеличивать количество серверов, выполняющих одни и те же приложения, или оптимизировать состав и структуру такого кластера [4].

Аналогично строится модель кластера без предоставления доступа к ресурсам с целью анализа его эффективности.

С помощью сетевых моделей вычислительных систем с распределенной обработкой данных различных топологий можно сделать сравнительный анализ этих структур, в зависимости от класса решаемых на этих структурах задач, определить их достоинства и недостатки относительно класса решаемых задач на этих структурах.

Литература

1. *Спортак М.*, *Франк Ч.*, *Паппас Ч. и др*. Высокопроизводительные сети. Энциклопедия пользователя. Киев: ДиаСофт, 1998.
2. Основы теории вычислительных систем / С.А.Майоров, Г.И.Новиков, Т.И.Алиев и др. М.: Высшая школа, 1978.
3. *Фельдман Л.П., Михайлова Т.В.* Оценка эффективности кластерных систем с использованием моделей Маркова / Тезисы докл. Международной научно-технической конференции «КОМТЕХ 2001». Таганрог: Изд-во ТРТУ, 2001(в печати).
4. *Фельдман Л.П., Михайлова Т.В.* Оптимизация состава и структуры высокопроизводительных вычислительных систем с использованием метода средних / Научн.тр. ДонНТУ. Серия: Информатика, кибернетика и вычислительная техника. Донецк: ДонНТУ, 2002 (в печати).

ПАРАЛЛЕЛЬНАЯ ФИЛЬТРАЦИЯ ИЗОБРАЖЕНИЙ

В.А. Фурсов, С.Б. Попов

Институт систем обработки изображений РАН, Самара

Обсуждаются особенности научных исследований, связанные с развитием высокопроизводительных вычислительных систем. Рассматривается задача компьютерной обработки изображений, требующая применения многопроцессорных систем. Обсуждаются особенности постановок задач и связанные с ними проблемы распараллеливания, приводятся результаты.

Введение

В последние годы усиливается внимание к использованию высокопроизводительной вычислительной техники в научных исследова-

ниях. Связано это с тем, что во многих областях знаний фундаментальные научные исследования связаны с необходимостью проведения масштабных численных экспериментов, реализация которых возможна только на современных суперкомпьютерах. Общая тенденция состоит в том, что происходит концентрация мощных вычислительных ресурсов в центрах коллективного пользования и развитие инфраструктуры удаленного доступа с использованием средств телекоммуникаций.

Создание центров коллективного пользования явилось предпосылкой для формулировки новых задач научных исследований во многих областях знаний. Оказалось, что построение параллельных вычислительных процессов, обеспечивающих достижение максимальной производительности, является важной самостоятельной проблемой. Известно, что перенос обычной программы на многопроцессорную вычислительную систему может не дать ожидаемого выигрыша в производительности. Более того, в результате такого переноса программа может работать медленнее, чем на компьютере с традиционной архитектурой. В то же время простым изменением первоначальной математической формулировки задачи часто удается добиться значительного эффекта. По-видимому, «развитие математики в определенный период времени отражает природу вычислительных средств, доступных в этот момент, в значительно большей степени, чем можно было бы предположить» [1, стр.152].

В настоящей работе пойдет речь об одной из задач, которая решалась в рамках гранта № 01-01-00097 «Разработка и исследование методов и алгоритмов обработки изображений, распознавания образов и компьютерной оптики, реализуемых на многопроцессорных вычислительных системах». В частности, о задаче параллельной фильтрации больших изображений.

Кластерная технология параллельной фильтрации больших изображений

В настоящее время во многих прикладных исследованиях используются изображения, полученные при дистанционном зондировании Земли, с разрешением около 1 м и широкой полосой захвата. На этапе формирования таких изображений по данным аэрокосмического мониторинга осуществляют их фильтрацию с целью улуч-

шения качества. Размеры регистрируемых изображений обычно таковы, что их обработка возможна лишь с применением многопроцессорных систем. Как известно, эффективность параллельных программ напрямую зависит от соответствия алгоритма архитектуре вычислительной системы. Ниже рассматривается реализованная на кластере технология оценки и коррекции искажений изображений, обусловленных влиянием оптической системы, ПЗС-приемников и динамики процесса видеоизмерений.

В работе исследовалась возможность использования для обработки больших изображений линейного фильтра с бесконечной импульсной характеристикой (БИХ-фильтра):

$$g(n_1,n_2) = -\sum_{(m_1,m_2)\in Q_g}\sum a_{m_1,m_2}\, g(n_1-m_1,n_2-m_2) + \\ + \sum_{(m_1,m_2)\in Q_f}\sum b_{m_1,m_2}\, f(n_1-m_1,n_2-m_2),, \qquad (1)$$

где $f(n_1,n_2)$, $g(n_1,n_2)$ – отсчеты входного и выходного изображений соответственно, а a_{m_1,m_2}, b_{m_1,m_2} – коэффициенты фильтра.

Предполагается, что параметры фильтра определяются по малым фрагмента изображений так, как описано в работе [2]. На этом этапе решается также задача обеспечения устойчивости фильтра (1).

В случае использования БИХ-фильтра при фиксированной расфокусировке можно существенно уменьшить размер опорной области по сравнению со случаем использования фильтра с конечной импульсной характеристикой (КИХ-фильтра). Это позволяет существенно снизить вычислительную сложность алгоритма на одной итерации. Однако, возникают проблемы реализуемости БИХ-фильтра, связанные с тем, что для вычисления выходных отсчетов должны использоваться отсчеты выходного изображения, которые для наиболее часто используемых форм опорной области, как правило, еще не определены. Для преодоления этого затруднения применяют итерационную схему вычисления отсчетов выходного изображения [3]

$$y(n_1,n_2) = a(n_1,n_2) ** x(n_1,n_2) + c(n_1,n_2) ** y(n_1,n_2), \qquad (2)$$

где звездочки означают двумерную свертку, а $c(n_1,n_2) = 1 - b(n_1,n_2)$.

В частотной области итерационная схема по аналогии описывается соотношением

$$Y(e^{\omega_1}, e^{\omega_2}) = A(e^{\omega_1}, e^{\omega_2})X(e^{\omega_1}, e^{\omega_2}) + C(e^{\omega_1}, e^{\omega_2})Y(e^{\omega_1}, e^{\omega_2}), \quad (3)$$

где $C(e^{\omega_1}, e^{\omega_2}) = 1 - B(e^{\omega_1}, e^{\omega_2})$..

Для обеспечения устойчивости итерационной схемы (2), (3) вводится параметр λ, удовлетворяющий неравенствам

$$0 < \lambda < 2 / \max_{(\omega_1, \omega_2)} B(e^{\omega_1}, e^{\omega_2}) \quad (4)$$

так, что частотный отклик фильтра принимает вид

$$Y_i(e^{\omega_1}, e^{\omega_2}) = \lambda A(e^{\omega_1}, e^{\omega_2})X(e^{\omega_1}, e^{\omega_2}) + C(e^{\omega_1}, e^{\omega_2})Y(e^{\omega_1}, e^{\omega_2}), \quad (5)$$

где $C(e^{\omega_1}, e^{\omega_2}) = 1 - \lambda B(e^{\omega_1}, e^{\omega_2})$.

Имеет место связь точности восстановления с величиной параметра λ и числом итераций. Ниже приводится пример выбора этих параметров, обеспечивающих компромисс вычислительной эффективности и качества восстановления.

Параллельная фильтрация изображений строится с использованием декомпозиции по данным, учитывающей специфику и формат обрабатываемой информации. Параллельно обрабатываются неперекрывающиеся полосы изображения. При этом задача заключается в выборе размеров и количества полос, при которых обеспечивается равномерная загрузка всех процессоров, участвующих в вычислениях. Если используемый кластер позволяет использовать достаточно большое число процессоров, то целесообразно размеры фрагментов (полос) выбирать такими, чтобы данные полностью размещались в оперативной памяти. Это позволяет избежать затрат дополнительного времени на буферизацию изображения в памяти постоянного хранения. Однако при этом, все равно, необходимо на каждом шаге итерационного процесса осуществлять обмен крайними строками между процессами, обрабатывающими соседние фрагменты изображения.

Такая схема слабо чувствительна к скорости коммуникационной среды, однако очень чувствительна к правильной реализации обмена данными. Заметим, что это преимущество связано с использованием БИХ-фильтров. Применение КИХ-фильтров приводит к существенному увеличению объемов данных, которыми должны об-

мениваться процессоры.

В качестве примера приведем схему реализации параллельной фильтрации [4] с использованием БИХ-фильтра, описываемого передаточной функцией

$$\hat{G}(z_1, z_2) = \hat{A}(z_1, z_2) / \hat{B}(z_1, z_2), \qquad (6)$$

где

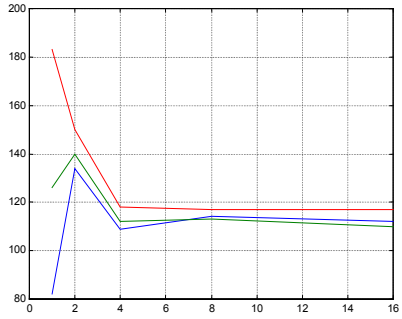
$$\hat{A}(z_1, z_2) = \hat{a}_0[1 + \hat{a}_1(z_1 + z_1^{-1} + z_2 + z_2^{-1}) + \\ + \hat{a}_2(z_1 z_2 + z_1^{-1} z_2^{-1} + z_1 z_2^{-1} + z_1^{-1} z_2)],$$

$$\hat{B}(z_1, z_2) = 1 + \hat{b}_1(z_1 + z_1^{-1} + z_2 + z_2^{-1}) + \\ + \hat{b}_2(z_1 z_2 + z_1^{-1} z_2^{-1} + z_1 z_2^{-1} + z_1^{-1} z_2).$$

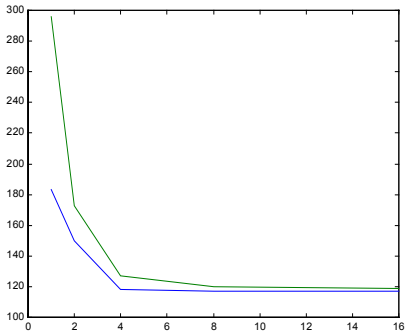
Параметры фильтра – $\hat{a}_0 = 35{,}2857$, $\hat{a}_1 = -1{,}2$, $\hat{a}_2 = 0{,}27$, $\hat{b}_1 = 1{,}2$, $\hat{b}_2 = 0{,}27$ **задавались исходя из требования обеспечения качества восстановления полученного моделированием исходного изображения при условии устойчивости БИХ-фильтра.**

Передаточной функции (6) соответствует опорная область 3×3. Следовательно, для обеспечения работы описанного выше параллельного алгоритма после каждой итерации процессы, соответствующие соседним полосам изображения, должны обмениваться одной строкой. Параметр λ итерационной схемы (5) принимался равным 0,01. При таком значении качество восстановления изображения получается вполне приемлемым при количестве итераций 5-10.

На рис. 1,*а* приведены графики изменения времени обработки изображения размером 4096×4096 в зависимости от числа используемых процессоров для 1, 5, 10 итераций (нижняя кривая соответствует 1 итерации, верхняя – 10). Одному процессору соответствует обычный, то есть последовательный алгоритм. На рис. 1,*б* приведены графики, полученные при тех же условиях для 10 и 20 итераций.



а)



б)

Рис. 1. Зависимость времени выполнения программы от количества процессоров для 1, 5, 10 итераций

Из графика видно, что при двадцати итерациях, обработка ускоряется в два с половиной раза. Однако ускорение имеет место лишь при увеличении числа процессов до восьми. При дальнейшем увеличении числа процессоров время обработки практически не изменяется. Это означает, что увеличение времени, затрачиваемого на обмен данными, происходит также быстро, как и сокращение времени вычислений за счет увеличения числа процессоров.

Эффективность распараллеливания зависит от размеров обрабатываемых изображений. Прогонка на кластере тестовых задач фильтрации изображений различной размерности позволяет

рации изображений различной размерности позволяет установить предельное число процессоров для каждой заданной структуры БИХ-фильтра, при которых еще имеет место ускорение. Результаты этих предварительных испытаний далее используются для управления вычислительной эффективностью в процессе обработки данных мониторинга.

На рис. 2 приведены исходное и обработанное изображения, полученные при значении параметра $\lambda = 0{,}01$ и числе итераций 10.

Рис.2. Пример обработки участка изображения (использован демонстрационный снимок с www.sovinformsputnik.com)

Работа выполнена при частичной поддержке РФФИ (гранты № 01-01-00097, 00-01-05001).

Литература

1. ***Тоффоли Т.*, *Марголус Н.* Машины клеточных автоматов. М.: Мир, 1991.**
2. ***Сойфер В.А.*, *Котляр В.В.*, *Фурсов В.А.* Построение алгоритмов оперативной коррекции искажений на изображениях в оптико-**

электронных системах наведения и целеуказания // В сб. Современные методы проектирования и отработки ракетно-артиллерийского вооружения. Саров, ВНИИЭФ, 2000. С. 340–345.

3. Методы компьютерной обработки изображений / Под ред. Сойфера В.А., М.: Физматлит, 2001.

4. *Попов С.Б.*, *Сойфер В.А.*, *Тараканов А.А.*, *Фурсов В.А.* Кластерная технология формирования и параллельной фильтрации больших изображений // Компьютерная оптика. №23. 2002. С. 75–78.

ОБЩИЕ ПРИНЦИПЫ ДЕЯТЕЛЬНОСТИ НИЖЕГОРОДСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА ПО РАЗВИТИЮ РАБОТ В ОБЛАСТИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

А.Ф. Хохлов, Р.Г. Стронгин, В.П. Гергель, В.И. Швецов

Нижегородский государственный университет им. Н.И. Лобачевского

Развитие аппаратно-технической базы параллельных вычислений

ННГУ уже достаточно давно ведет планомерную работу по развитию аппаратно-технической базы параллельных вычислений. Первый комплекс для апробации параллельных вычислений был смоделирован на базе двух ЭВМ Электроника-100/25 в 1988 г. Кроме этого, разработка новых параллельных алгоритмов апробировалась на многопроцессорном транспьютерном комплексе в Техническом университете Дании (г. Копенгаген, Дания, 1989) и на параллельной вычислительной системе Alliant в университете Калабрия (г. Козенца, Италия, 1990). В 1995 г. ННГУ при долевом участии администрации Н.Новгорода приобрел четырехпроцессорный комплекс Parsytec с суммарной производительностью 320 млн. операций/с. Работа велась в рамках выполняемой под руководством академика Самарского А.А. программы создания российских высокопроизводительных вычислительных центров на базе многопроцессорной транспьютерной техники. В 2001 г. Нижегородский университет получил в рамках сотрудничества с корпорацией Интел быстродействующий вычислительный кластер. Кластер включает:

– 2 вычислительных сервера, каждый из которых имеет 4 процессора Intel Pentium III 700 Мгц, 512 MB RAM, 10 GB HDD, 1 Гбит Ethernet card;

– 12 вычислительных серверов, каждый из которых имеет 2 процессора Intel Pentium III 1000 Мгц, 256 MB RAM, 10 GB HDD, 1 Гбит Ethernet card;

– 12 рабочих станций на базе процессора Intel Pentium 4 1300 Мгц, 256 MB RAM, 10 GB HDD, CD-ROM, монитор 15", 10/100 Fast Etherrnet card.

Следует отметить, что в результате передачи подобного оборудования Нижегородский госуниверситет оказался первым вузом в Восточной Европе, оснащенным ПК на базе новейшего процессора INTEL®PENTIUM®4. Пиковая суммарная производительность кластера составляет 50 млрд. операций с плавающей запятой/с.

Важной отличительной особенностью кластера на основе подобного оборудования является его многоплановость. В состав кластера входят рабочие места, оснащенные новейшими процессорами Intel Pentium 4 и соединенными относительно медленной сетью (100 Мбит), и вычислительные 2 и 4-процессорные сервера, передача данных между которыми выполняется при помощи быстрых каналов передачи данных (1000 Мбит). В результате, кластер может использоваться не только для решения сложных вычислительно-трудоемких задач, но также и для проведения различных экспериментов по исследованию многопроцессорных кластерных систем и параллельных методов решения научно-технических задач.

Создание телекоммуникационной сети с высокоскоростным каналом связи

Нижегородский государственный университет является одним из исполнителей системного проекта Миннауки и Минобразования РФ «Создание национальной сети компьютерных технологий для науки и высшей школы».

Университетом создан и поддерживается образовательный сегмент телекоммуникационной сети Н. Новгорода. ННГУ имеет 2 канала выхода в Интернет (RbNet – 10 Мбит/с, Трансгелеком – 2Мбит/с). ННГУ имеет развитую внутреннюю сеть, соединяющую десять

корпусов университета с пропускной способностью от 2 до 10 Мбит/с. Общее число компьютеров ННГУ, подключенных к сети – около 1000.

К сети ННГУ подключен ряд образовательных и научных учреждений региона, в том числе Вятский технический университет, Академия госслужбы, Институт физики микроструктур РАН и другие (более 50 организаций).

ННГУ имеет опорную точку на Междугородной телефонной станции МГТС, что позволяет расширять круг подключаемых абонентов. ННГУ имеет возможность расширения имеющихся внешних каналов (на коммерческой основе). ННГУ подключен к узлу обмена внутрирегиональным трафиком (NN Internet Exchange), что позволяет иметь внутригородскую скорость обмена трафиком 2 Мбит/с.

ННГУ имеет лицензию на услуги передачи данных и услуги телематических служб.

В 2002 году в рамках ФЦП «Интеграция науки и высшего образования России на 2002-2006 г.» совместно с Институтом прикладной физики РАН ННГУ выполняет по направлению 4.16 «Развитие интегрированной сети с высокоскоростными телекоммуникационными каналами» проект «Развитие скоростной широкополосной мультисервисной телекоммуникационной сети организаций науки и образования Нижнего Новгорода».

В результате выполнения проекта предполагается:

- завершение полной реконструкции сети науки и образования Нижего новгорода с использованием оптоволоконных линий связи и оборудования АТМ;
- обеспечение удаленного доступа и загрузка Нижегородского распределенного Центра коллективного пользования для суперкомпьютерных вычислений ИПФ РАН и ННГУ;
- расширение возможностей практического применения суперкомпьютерных вычислительных технологий для решения актуальных фундаментальных и прикладных научно-технических проблем по тематике научных исследований коллективов-исполнителей проекта;
- создание коллективной системы информационный безопасности и антивирусной поддержки организаций науки и образова-

ния;

– внедрение технологий использования современных телекоммуникационных широкополосных видов сервисов, таких как видеоконференции, IP-телефония.

Развитие научных исследований

Одним из основных направлений научных исследований в области параллельных вычислений в ННГУ является разработка параллельных методов глобальной оптимизации (первые публикации по данной тематике появились в 1985 г.). Последние результаты работ отражены в монографии Strongin R.G., SergeyevYa.D. Global Optimization with Non-Convex Constraints. Sequential and Parallel Algorithms. Kluwer Academic Publishers., 2000, 728 pp.

Выполнены исследовательские работы по проекту «Оптимизация вычислений в кластерных компьютерных системах на примере библиотеки типа MKL», поддержанном компанией Intel в рамках программы финансирования перспективных научных исследований.

Организованы работы по проекту «Формирование комплексной системы подготовки кадров в области суперкомпьютерных технологий для вузов и научных организаций Нижегородского региона» в рамках Федеральной целевой программы «Государственная поддержка интеграции высшего образования и фундаментальной науки на 2002-2006 годы» по направлению 3.13 «Совместная разработка и адаптация вузами и исследовательскими организациями программ научно-методического обеспечения подготовки кадров в области суперкомпьютерных, информационных и наукоемких технологий».

Сформирована начальная программа учебно-исследовательских и научных работ по применению высокопроизводительных вычислительных систем в учебном процессе и в научных исследованиях в ННГУ на 2002 г.

– использование методов параллельного программирования для моделирования нелинейной динамики дискретных активных нейроноподобных решеточных сред (руководитель – зав. кафедрой теории колебаний, профессор, д.ф.-м.н. Шалфеев В.Д.);

– применение параллельных вычислений для расчета псевдосимметрии (руководитель – декан физического факультета, профессор, д.ф.-м.н. Чупрунов Е.В.);

– применение параллельных вычислений для исследования физических свойств высокотемпературных сверхпроводников (руководитель – доцент, к.ф.-м.н. Максимов И.Л.);

– разработка эффективных параллельных методов для выбора глобально-оптимальных решений на многопроцессорных кластерных системах (руководитель – профессор, д.ф.-м.н. Стронгин Р.Г.);

– разработка параллельных схем вычислений для алгоритма Моцкина–Бургера нахождения общего решения системы линейных неравенств (руководитель – доцент, к.ф.-м.н. Золотых Н.Ю.)

Деятельность Нижегородского университета в области высокопроизводительных параллельных вычислений поддерживается рядом организаций информационной индустрии. Так, часть работ, выполняемых совместно с малым предприятием Иннотех, были поддержаны в 2002 г. Фондом содействия малых форм предприятий в научно-технической сфере в рамках проекта «Разработка компонентов научного, учебно-методического и программного обеспечения высокопроизводительных кластерных вычислительных систем для вузов и научных организаций Нижегородского региона».

Подготовка и переподготовка кадров в области параллельных вычислений

Проводится летняя учебно-исследовательская практика студентов для выполнения работ в области параллельного программирования.

Разработан оригинальный учебный курс «Многопроцессорные вычислительные системы и параллельное программирование» для студентов физико-математических факультетов.

Проведены занятия по учебному курсу «Многопроцессорные вычислительные системы и параллельное программирование» для студентов факультета Нижегородского технического университета.

Подготовлена программа и проведена подготовка в области

параллельных вычислений для работников компании Интел (в рамках совместной образовательной программы).
Подготовлена и проведена программа повышения квалификации для сотрудников Института прикладной физики РАН.

Организационная деятельность

Для координации работ в области высокопроизводительных вычислений в ННГУ создан Центр компьютерного моделирования.

Проведен Международный семинар и Молодежная школа «Высокопроизводительные параллельные вычисления на кластерных системах».

Проведен конкурс научных проектов сотрудников ННГУ по использованию высокопроизводительной вычислительной техники для анализа и исследования важных научно-технических проблем с высокой вычислительной трудоемкостью. Победители конкурса получили финансовую поддержку для проведения работ из средств ННГУ (приказ НИЧ ННГУ №14-151 от 24.04.2002).

Издательская деятельность

Опубликовано учебное пособие «Основы параллельных вычислений для многопроцессорных вычислительных систем» (Стронгин Р.Г., Гергель В.П., 2001).

Опубликованы материалы Международного семинара «Высокопроизводительные параллельные вычисления на кластерных системах».

Подготовлена первая версия Интернет-сайта Центра компьютерного моделирования http://www.software.unn.ac.ru/ccam/ (январь 2002).

МЕТОД ПОДКОНСТРУКЦИЙ – ЭФФЕКТИВНЫЙ ИНСТРУМЕНТ РАСПАРАЛЛЕЛИВАНИЯ АЛГОРИТМОВ В МЕХАНИКЕ

С.К. Черников

Казанский физико-технический институт

Главным препятствием широкого использования вычислительных кластеров или метакомпьютеров для анализа механических процессов в сложных системах методом конечных элементов (МКЭ) является отсутствие параллельных алгоритмов и программ, реализующих особенности конкретной архитектуры компьютера и оптимизирующих пересылки информации между вычислительными узлами. В контексте сказанного весьма эффективным инструментом может стать метод подконструкций, естественно ориентированный на параллельную обработку данных о сложных объектах и обладающий рядом свойств, полезных при реализации на распределенных вычислительных устройствах.

Будем полагать, что сложная конструкция может быть расчленена на несколько подконструкций, связанных между собой (рисунок 1), и рассмотрим схему распараллеливания вычислений с конденсацией матицы жесткости подконструкции для различных задач анализа. Рассматриваемая техника основана на разделении основных неизвестных подконструкции q на «внутренние» q_i и «внешние» q_s (внутренние неизвестные связаны с теми узлами сетки, которые не стыкуются с другими подконструкциями).

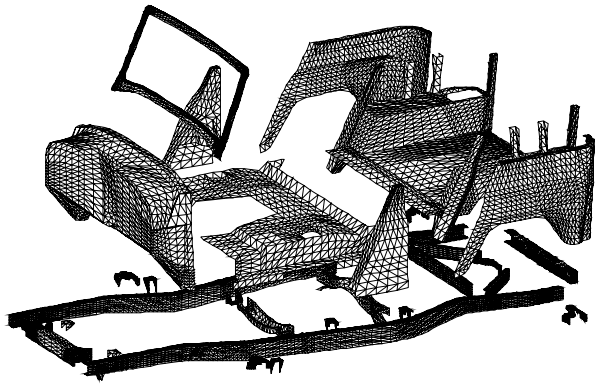


Рис. 1

Линейная задача статики. При использовании МКЭ в варианте метода перемещений решение линейной задачи статики сводится к отысканию из решения системы алгебраических уравнений $Kq = P$ вектора обобщенных перемещений q и вычисления вторичных не-

известных (напряжений, интенсивностей и т.п.), определяемых этим вектором. Схема алгоритма распараллеливания решения этой задачи с конденсацией, показанная на рис. 2.

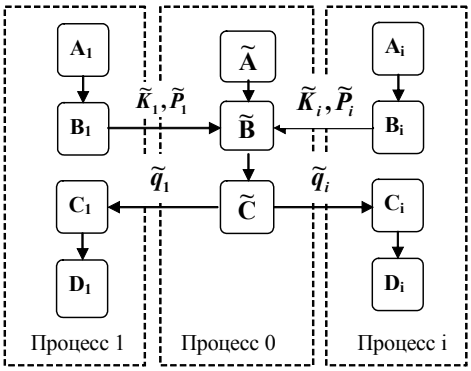


Рис. 2

Алгоритм представляет собой следующую последовательность действий.

1. В отдельном вычислительном процессе для каждой подконструкции формируются матрицы K и P (блок A_i на схеме).
2. Для построения коэффициентов конденсированных матриц подкострукции, имеющей n «внешних» степеней свободы, однократно решается система уравнений $[K + \alpha E][Q^1 Q^p] = [\alpha\tilde{E} \mid P]$ с $n + 1$ правой частью [1]. Здесь: E = diag [1 0 1 .. 1] – диагональная матрица, в которой «единицы» стоят на местах, соответствующих внешним степеням свободы, $\tilde{E}$ представляет собой матрицу E с вычеркнутыми нулевыми столбцами, Q^1, Q^p – матрицы перемещений от единичных загружений и внешней нагрузки соответственно. Коэффициенты $\tilde{k}_{ij}$ и $\tilde{p}_i$ конденсированных матриц подкострукции вычисляются по соотношениям:

$$\widetilde{k}_{ij} = \begin{cases} -\alpha(Q^1_{ind(i)j} - 1) \text{ если } i = j, \\ -\alpha Q^1_{ind(i)j} \text{ если } i \neq j, \end{cases} \quad (1)$$

$$\widetilde{p}_i = -\alpha Q^p_i, \quad (2)$$

в которых *ind* – матрица индексов, содержащая глобальные номера внешних неизвестных.

3. Конденсированные матрицы $\widetilde{K}$ и $\widetilde{P}$ передаются ведущему процессу для формирования разрешающих уравнений структуры для «внешних» неизвестных $[\sum \widetilde{K}^i + \sum k]\widetilde{q}_s = \sum \widetilde{P}^i$ (Блок $\widetilde{\mathrm{B}}$). В этом выражении k – матрица жесткости «элемента стыковки» подконструкций [1], которая для каждой пары стыкуемых узлов вычисляется по соотношению $[k] = \alpha \begin{bmatrix} e & -e \\ -e & e \end{bmatrix}$ в блоке $\widetilde{\mathrm{A}}$.
4. Далее в ведущем процессе определяется вектор «внешних» неизвестных структуры $\widetilde{q}$ (Блок $\widetilde{\mathrm{C}}$), и его соответствующие компоненты $\widetilde{q}_i$ передаются процессам, обрабатывающим информацию о подконструкциях.
5. Для каждой подконструкции в блоке C_i вычисляются основные неизвестные в узлах $\{q\} = Q^1\{q_s\} + Q^p$, а в блоке D_i – вторичные результанты (напряжения, интенсивности и т.п.)

Отметим, что в описанном алгоритме объем информации, передаваемой между вычислительными процессами, определяется только количеством внешних неизвестных.

Определение динамической реакции. Для обеспечения безусловной устойчивости процесса интегрирования уравнений $m\ddot{q} + Kq = P(t)$ обычно используются неявные методы интегрирования, такие, например, как метод Ньюмарка [2]. Процедура распараллеливания этого метода для реализации на вычислительных системах с рас-

пределенной памятью, схема которой приведена на рис. 3, может быть реализована в виде следующей последовательности действий.

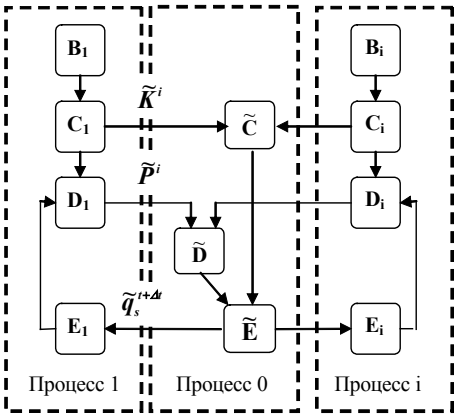


Рис. 3

1. Для каждой подконструкции в отдельном вычислительном процессе задаются начальные условия и вычисляются коэффициенты a_i (Блок B_i).

2. Формируется матрица $\overline{K}^i = K + a_0 M + \alpha E$ (Блок C_i), и, после решения системы уравнений $[\overline{K}^i][Q^1] = [\alpha \widetilde{E}]$ с n правыми частями, по формулам (1) производится вычисление коэффициентов конденсированной матрицы $\widetilde{K}^i$. После этого конденсированная матрица подконструкции $\widetilde{K}^i$ передается ведущему процессу.

3. В блоке D_i по соотношению $\overline{P}^i_{t+\Delta t} = P_{t+\Delta t} + M(a_0 q_t + a_2 \dot{q}_t + \ + a_3 \ddot{q}_t)$ для подконструкции определяется вектор $P^i_{t+\Delta t}$. Коэффициенты $\widetilde{p}_i = -\alpha Q_i^p$ конденсированного вектора $\widetilde{P}^i_{t+\Delta t}$ вычисляется после решения уравнения $[\overline{K}^i][Q^p] = [\overline{P}^i]$. Отметим, что при

использовании метода LDL^T -факторизации при определении Q^1, эта процедура будет достаточно быстрой, т.к. может использовать уже готовые компоненты разложения матрицы $\overline{K}^i$. Вычисленный таким образом конденсированный вектор $\widetilde{P}^i_{t+\Delta t}$ передается ведущему процессу.

4. В ведущем процессе для конструкции производится вычисление конденсированной матрицы жесткости $\sum \widetilde{K}^i$ (Блок $\widetilde{\mathrm{C}}$), конденсированного вектора $\sum \widetilde{P}^i_{t+\Delta t}$ (Блок $\widetilde{\mathrm{D}}$) и определяются внешние неизвестные $\widetilde{q}^{t+\Delta t}_s$ из уравнения $[\sum \widetilde{K}^i]\widetilde{q}^{t+\Delta t}_s = \sum \widetilde{P}^i_{t+\Delta t}$ (Блок $\widetilde{\mathrm{E}}$). Соответствующие компоненты вектора $\widetilde{q}^{t+\Delta t}_s$ передаются в процессы, обрабатывающие информацию о подконструкциях.

5. Далее, в соответствующих процессах, обрабатывающих информацию о подконструкциях, производится определение перемещений $\{q^{t+\Delta t}\} = Q^1\{q^{t+\Delta t}_s\} + Q^p$, скоростей $\dot{q}^{t+\Delta t} = \dot{q}^t + a_6\ddot{q}^t + \ + a_7\ddot{q}^{t+\Delta t}$ и ускорений $\ddot{q}^{t+\Delta t} = a_0(q^{t+\Delta t} - q^t) - a_2\dot{q}^t - a_3\ddot{q}^t$ для *i*-й подконструкции (Блок E_i).

6. После этого вычисления повторяются, начиная с позиции 3, пока текущее значение времени интегрирования находится внутри исследуемого интервала.

Отметим, что, как и в предыдущем алгоритме, объем информации, передаваемой между вычислительными процессами, определяется только количеством внешних неизвестных.

Модальный анализ. При решении задачи модального анализа для больших систем широко используется метод итераций в подпространстве [2]. Рассмотрим процедуру распараллеливания этого метода с использованием подконструкций (рис. 4)

1. В отдельном для каждой подконструкции процессе выбирается начальный базис R_0 (Блок A_i) и вычисляются компактные матрицы $K_i^* = R_i^T K_i R_i$ и $M_i^* = R_i^T M_i R_i$ (Блок B_i), которые передаются ведущему процессу.

2. В ведущем процессе производятся вычисление компактных матриц для конструкции $K^* = \sum_{i=1}^{n} K_i^* + \tilde{R}_s^T \sum k\tilde{R}_s$, $M^* = \sum_{i=1}^{n} M_i^*$ (Блок $\tilde{B}$) и решение системы $K^*\{\beta\} - \lambda M^*\{\beta\} = 0$ методом Якоби (Блок $\tilde{C}$). После этого вектор собственных чисел $\{\beta\}$ передается в процессы, обрабатывающие информацию о подконструкциях.

3. Далее, в соответствующих процессах для каждой из подконструкций в блоках C_i вычисляются значения $\{q_i^n\} = \sum_{j=1}^{m} \beta_j^{(n)} \{R_i^j\}$, а в блоках D_i вычисляется матрица $[P] = M_i^* \{q_i\}$, решается система $[K + \alpha E][Q^1 Q^p] = [\alpha\tilde{E} \mid P]$ и по формулам (1), (2) находятся коэффициенты конденсированных матриц $\tilde{K}$ и $\tilde{P}$, которые передаются в ведущий процесс.

4. В ведущем процессе вычисляются матрицы жесткости $\tilde{K}$ и правых частей $\tilde{P}$ конструкции в целом (Блок $\tilde{D}$), и из решения системы $[\sum \tilde{K}^i + \sum k]\tilde{R}_s = \sum \tilde{P}^i$ (Блок $\tilde{E}$) находятся внешние компоненты базиса $\tilde{R}_s$. Соответствующие сегменты $\tilde{R}_s$ передаются процессам, обрабатывающим данные о подконструкциях.

5. В соответствующих процессах для каждой из подконструкций в блоке E_i вычисляются уточненные компоненты базиса подконструкций $\{R\} = Q^1\{\tilde{R}_s\} + Q^p$, и вычисления повторяются, начиная с действий, определенных в блоках B_i, пока не будет достигнута необходимая точность вычислений собственных

значений исходной задачи.

Отметим, что в описанном алгоритме объем информации передаваемой между процессами определяется количеством внешних неизвестных и форм колебаний.

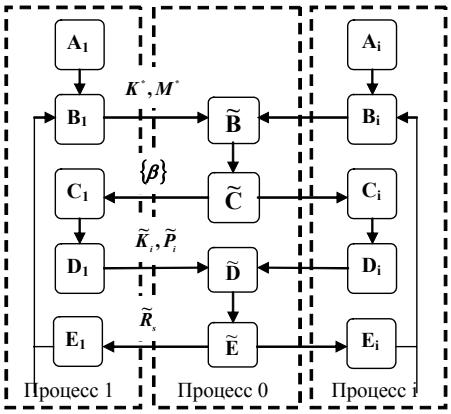


Рис. 4

Нелинейные задачи. Рассмотрим технологию распараллеливания на примере геометрически нелинейной задачи с небольшой нелинейностью. Решение такой задачи часто сводится к итерационной процедуре $Kq^{n+1} = P + F(q^n)$, в которой вектор $F(q^n)$ включает в себя нелинейные члены. Схема этой процедуры, приведенная на рис. 5, включает в себя следующие операции.

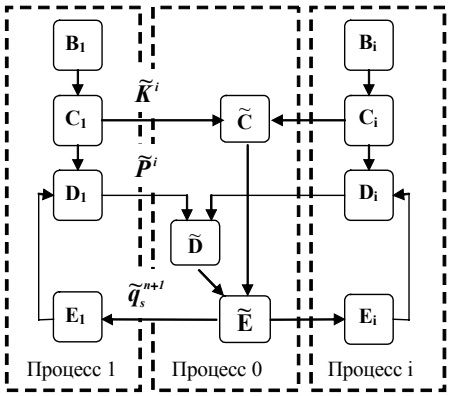


Рис. 5

1. В отдельном для каждой подконструкции процессе выбирается начальное приближение q_0 (Блок B_i), формируется матрица K и, используя результаты решения системы $[K^i + \alpha E][Q^1] = [\alpha \widetilde{E}]$, по формуле (1) вычисляются коэффициенты конденсированной матрицы $\widetilde{K}^i$ (Блок C_i). После этого матрица $\widetilde{K}^i$ передается ведущему процессу, в котором происходит вычисление конденсированной матрицы жесткости $\sum \widetilde{K}^i$ для всей конструкции (Блок $\widetilde{C}$).

2. В процессах, обрабатывающих данные подконструкций, формируются векторы $\overline{P}_n^i$ и, на основании результатов решения систем $[\overline{K}^i + \alpha E][Q^p] = [\overline{P}_n^i = P^i + F(q^n)]$, по соотношению (2) вычисляются коэффициенты конденсированных матриц $\widetilde{P}_n^i$ (Блоки D_i).

3. Эти матрицы передаются ведущему процессу, в котором для конструкции в блоке $\widetilde{D}$ происходит вычисление конденсированного вектора $\sum \widetilde{P}_n^i$, а в блоке $\widetilde{E}$ – решается система

уравнений $[\sum \widetilde{K}^i]\widetilde{q}_s^{n+1} = \sum \widetilde{P}_n^i$, и определяются внешние неизвестные $\widetilde{q}_s^{n+1}$.

4. Соответствующие компоненты вектора $\widetilde{q}_s^{n+1}$ передаются в процессы, обрабатывающие данные подконструкций, где по соотношениям $\{q^{n+1}\} = Q^1\{q_s^{n+1}\} + Q^p$ вычисляются неизвестные q^{n+1} для каждой подконструкции (Блок E_i). При необходимости продолжения итераций вычисления повторяются, начиная с пункта 2.

Литература

1. ***Черников С.К., Ашихмин А.Н.*** **Программная реализация метода подконструкций для анализа сложных структур с распараллеливанием вычислений. Математическое моделирование в механике сплошных сред на основе методов граничных и конечных элементов: Тр. XIX междунар. конф. Т. 3. СПб.: Изд-во НИИХ СпбГУ, 2001. С. 220–224.**
2. ***Бате К., Вилсон Е.*** **Численные методы анализа и метод конечных элементов. М.: Стройиздат, 1982.**

ПОДГОТОВКА СПЕЦИАЛИСТОВ В ОБЛАСТИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

Л.П. Чернышева, Ф.Н. Ясинский

Ивановский государственный энергетический университет

Введение

В Ивановском государственном энергетическом университете на кафедре Высокопроизводительных вычислительных систем, начиная с 1996 года, введется подготовка специалистов в области параллельных вычислений. Читаются курсы:

– «Параллельные вычисления» студентам IV курса по специальности «Прикладная математика»;

– «Параллельные вычисления и нейрокомпьютерные системы» студентам V курса по специальностям «САПР» и «Программное обеспечение компьютерных систем»;

– «Использование высокопроизводительной вычислительной техники» студентам V курса по специальности «Электрические системы»;

– «Компьютерное моделирование электромеханических систем» магистрам II курса по специальностям «Электропривод» и «Электромеханика».

Обучение ведется на многопроцессорной вычислительной системе Parsytec Power X'plorer. Система включает в себя 8 процессоров, каждый из которых имеет собственную память по 32 Mbyte, и транспьютеры с оперативной памятью по 4 Mbyte. В связи с возможностью выхода через Internet на МВС-1000М ведется обучение параллельному программированию на MPI. Разработана методика преподавания курсов и набор лабораторных работ по каждому курсу.

Содержание курса

Каждый из четырех читаемых курсов имеет свои особенности. Рассмотрим содержание курса «Параллельные вычисления и нейрокомпьютерные системы».

В первой части курса рассматриваются современные многопроцессорные вычислительные системы с общей и разделенной памятью, коммуникационные среды, транспьютеры, кластеры и многопроцессорная вычислительная система Parsytec Power X'plorer, приводится схема процессора, физическое соединение процессоров. Изучаются топологии на множестве процессоров, обмен данными между процессорами, функции параллельной обработки, процессы, потоки, сокеты и семафоры. Студенты знакомятся с базовыми понятиями MPI (процессы, группы процессов, коммуникаторы) и параллельными функциями MPI. Студенты изучают стандартные топологии и учатся создавать новые топологии, определяемые спецификой решаемой задачи, особенности синхронного и асинхронного обмена данными, изучают параллельные функции.

Вторая часть курса посвящена математическому моделированию на многопроцессорных вычислительных системах. Рассматриваются виды параллелизма, эффективность и ускорение вычислений на МВС. Излагаются алгоритмы распараллеливания при решение систем обыкновенных дифференциальных уравнений, уравнений в частных производных, систем линейных и нелинейных уравнений. Рассматриваются параллельные алгоритмы интерполирования, численного интегрирования, численных методов линейной алгебры, оптимизации, быстрого преобразования Фурье. Рассматривается распараллеливание методов прогонки и векторной прогонки. Студенты учатся применять численные методы в параллельном программировании, разрабатывать и отлаживать параллельные программы.

Третья часть курса посвящена изучению нейрокомпьютерных систем и реализации их на многопроцессорных вычислительных системах. Рассматривается история возникновения и развития искусственных нейронных систем, строение и работа биологического прототипа (мозга, нейрона), изучается формальный нейрон, его свойства, сети из формальных нейронов. Изучаются алгоритмы обучения, однослойный перцептрон, многослойный перцептрон. Изучаются способы прогнозирования природных процессов с помощью нейронных сетей. Студенты учатся разрабатывать программы, реализующие искусственные нейронные сети.

Организация учебного процесса

Курс «Параллельные вычисления и нейрокомпьютерные системы» состоит из лекций (34 часа), лабораторных занятий (72 часа) и практических занятий (14 часов). Студенты выполняют задания, результатом их работы являются параллельные программы и отчеты. Для самостоятельной работы студентам выдаются программа, имитирующая работу многопроцессорной вычислительной системы Parsytec Power X’plorer, и библиотека MPI.

Первое задание посвящено решению задач по параллельному программированию. Это задание проверяет знание топологий, параллельных функций, умение правильно организовать синхронный и асинхронный обмен данными, умение организовать работу нескольких процессов, умение создавать свои собственные тополо-

гии. Результатом работы являются программа и отчет, в котором приводится задача, метод ее решения и листинг программы.

Второе задание посвящено математическому моделированию. Студенты учатся применять численные методы в параллельном программировании, разрабатывать оптимальные алгоритмы распараллеливания численных методов. Студенты выполняют задания на Parsytec Power X'plorer, результатом их работы являются параллельная программа и отчет, в котором приводятся постановка задачи, расчетные формулы, алгоритм распараллеливания, листинг программы, приводится анализ выбранного алгоритма распараллеливания, вычисляется ускорение и указывается, где имеют место потери времени.

Третье задание посвящено программированию на MPI. Выдается задание по математическому моделированию, но программа должна быть написана на MPI. Студенты учатся использовать функции MPI, разрабатывать параллельные алгоритмы. Результатом работы являются параллельная программа и отчет.

Четвертое задание посвящено нейрокомпьютерам. Студенты создают программы, реализующие искусственные нейронные сети с различными алгоритмами обучения, функциями активации. Результатом работы является программа и отчет, в котором приводится задача, которую решает эта сеть, способ организации сети, алгоритм обучения, функция активации, методы, используемые в созданной сети.

Заключение

В Ивановском государственном энергетическом университете несколько лет выпускаются студенты, знающие параллельное программирование, умеющие работать на многопроцессорной вычислительной системе Parsytec Power X' plorer, получившие навыки распараллеливания сложных задач и применения численных методов для параллельных вычислений, умеющие программировать на MPI. Каждый год выполняются дипломные работы по указанным выше специальностям, вычисления для которых осуществляются на многопроцессорной вычислительной системе. С удовлетворением можно отметить, что курсы по параллельным вычислениям и

использованию многопроцессорных вычислительных систем читаются не только для студентов факультета информатики и вычислительной техники, но и магистрам электромеханического факультета и студентам электроэнергетического факультета. Надо отметить, что появилось понимание необходимости обучение студентов инженерных специальностей умению использования многопроцессорных вычислительных систем для решения научно-исследовательских и инженерно-технических задач в электроэнергетике.

Изданы учебные пособия: Ф.Н. Ясинский, Л.П. Чернышева «Многопроцессорные вычислительные системы. Архитектура. Математическое моделирование», Ф.Н. Ясинский, Л.П. Чернышева, В.В. Пекунов «Математическое моделирование с помощью компьютерных систем». Подготовлено к изданию учебное пособие Ф.Н. Ясинский, Л.П. Чернышева «Параллельные алгоритмы и комплексы программ».

КОМПЛЕКСНЫЙ ПОДХОД К ПОСТРОЕНИЮ И ОПТИМИЗАЦИИ КЛАСТЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СЕТЕЙ

А.Ю. Чесалов

Тверской Государственный Технический Университет

На сегодняшний день задача проектирования распределенных кластерных вычислительных сетей (КВС) стоит достаточно остро. Ее решение, по мнению автора, должно производиться не с рассмотрения отдельного сервера или кластера в отдельности, а с построения и анализа моделей центров обработки информации, которые представляют собой сложные аппаратно-программные комплексы и обрабатывают запросы, решают задачи, поступающие от других центров в режимах непосредственного доступа, пакетном режиме в реальном масштабе времени. При проектировании необходимо также учитывать, что управление каналами передачи данных сетевого аппаратно-программного комплекса КВС на протокольном уровне осуществляется как в режиме нормальной процедуры обслуживания, так и асинхронном, а также в режимах селективного или группового отказов.

При построении кластерной вычислительной сети одной из основных задач является моделирование структуры, обеспечивающей передачу заданных потоков информации по всем направлениям информационного обмена в приемлемое время. В процессе ее решения необходимо учитывать значительное множество характеристик сети, таких как пропускная способность, задержки, надежность, стоимость, а также ограничения, накладываемые быстродействием кластеров, пропускной способностью каналов, характеристиками трафика и т.д.

Сложность синтеза КВС с учетом всего диапазона взаимосвязанных вопросов такова, что оптимизация по одному общему (комплексному) критерию практически не возможна или приводит к неоправданным вычислениям и временным затратам из-за ограниченных возможностей достоверного определения необходимых объемов исходных данных, динамически меняющихся в ходе эксплуатации сети.

Поэтому, при проектировании КВС предлагается производить оптимизацию не по комплексному, а по частным критериям оптимальности системы, таким как производительность и оптимальная пропускная способность каналов связи, среднее время задержки в сети, надежность, минимальная стоимость сети с помощью комплекса взаимосвязанных частных моделей, реализованных в виде единого программного комплекса, подсистемы которого представлены на рис. 1.

Полученные, с использованием таких моделей частные решения позволяют находить квазиоптимальные решения, последовательно корректируя структуру и характеристики сети.

Совокупность моделей носит многоуровневый, иерархический характер, и позволяет учитывать на каждом уровне новые факторы в сравнении с предыдущими, при фиксации параметров, определенных ранее.

Если полученные на каком-либо этапе параметры КВС не удовлетворяют заказчика, то осуществляется повторный итерационный расчет с изменениями ранее принятых исходных данных и ограничений.

Результаты исследований позволили выделить основные этапы, разработать алгоритмы и методику проектирования КВС (рис. 2).

Так, методика предполагает на первом этапе построение сети минимальной стоимости, обеспечивающей передачу всех потоков информации при использовании минимальной пропускной способности каналов, при условии передачи информации в каждый момент времени только между двумя узлами сети (свойство двухсвязности) и ограничении на количество транзитных узлов коммутации на пути от источника до адресата. Затем определяются задержки в сети при одновременной передаче всего заданного трафика, находятся оптимальные пропускные способности каналов, обеспечивающие такие задержки. Определяется необходимость создания новых каналов связи или использования некоторой избыточности на тех или иных маршрутах.

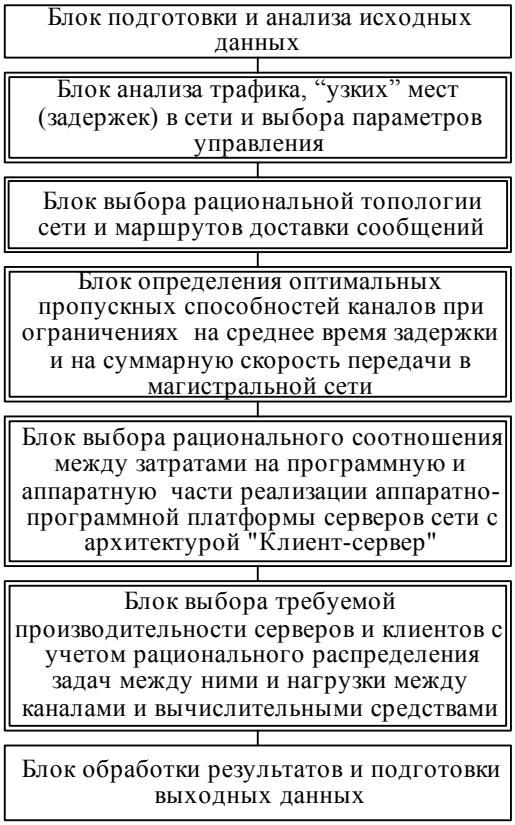


Рис. 1. Подсистемы программного комплекса проектирование КВС

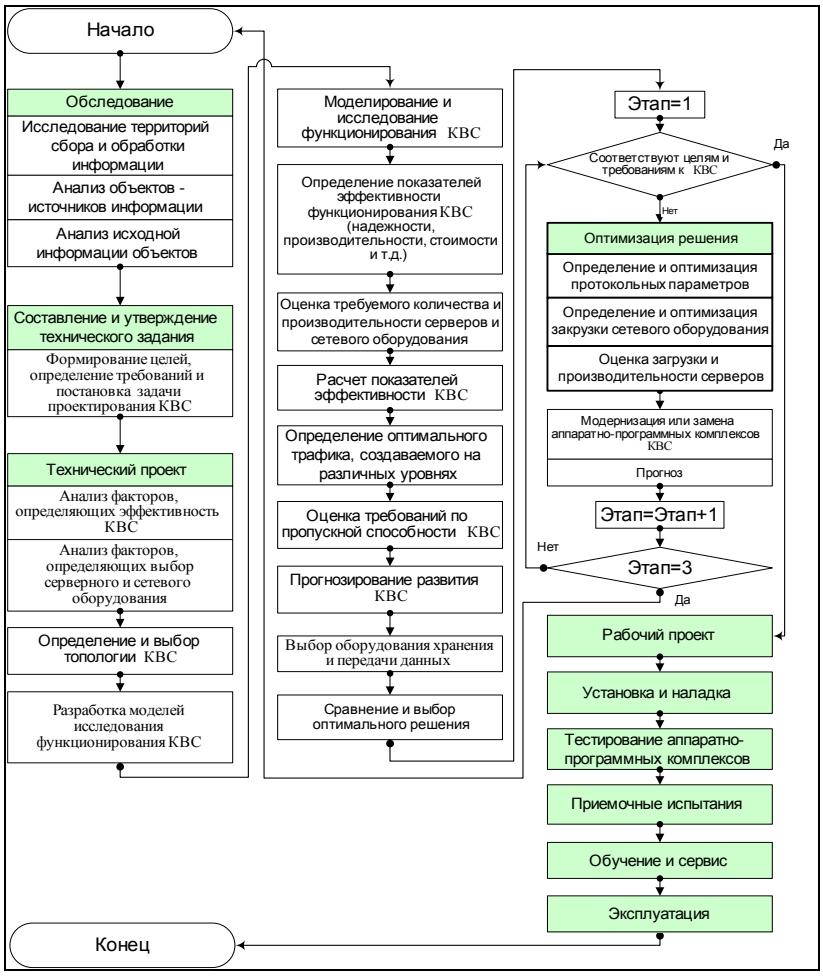


Рис. 2. Основные этапы и алгоритм проектирования КВС

После этого определяются требования к производительности серверов, входящих в кластеры, и если они оказываются труднореализуемыми или экономически не целесообразными, то решается вопрос компромиссного перераспределения требований между кластером (по производительности) и каналами (по пропускной спо-

собности). Так же рассматривается возможность большего совмещения вычислительных и связных операций. При отрицательном результате осуществляется возврат в блок подготовки и анализа исходных данных и их корректировка. Например, уменьшается количество клиентов, обслуживаемых одним кластером, или уменьшается максимальное количество переприемов при сбоях в передаче и т.п.

Математический аппарат рассматриваемой методики базируется, прежде всего, на основе теории массового обслуживания, на вероятностно-статистических методах, теории надежности, анализа структуры системы и обработки данных о производительности системы и ее элементов, и значительно облегчает и ускоряет процесс выбора рациональных структур и характеристик КВС.

С другой стороны, используемые математические методы, разработанной методики, позволяют произвести оптимизацию не только по критерию производительности, но и по критерию затрат. При проектировании или оптимизации систем необходимо не только удовлетворить требования к заданным показателям производительности, но и выполнить эти требования при минимальных затратах. Эта задача может формулироваться двояко: при минимальных затратах удовлетворить требования к заданным показателям производительности; при затратах, не превышающих заданной величины, максимизировать показатель производительности. Под затратами могут пониматься не только стоимость, но и, например, масса или габариты.

МЕТОДЫ ВЫБОРА ВЫЧИСЛИТЕЛЬНЫХ СРЕДСТВ ПРИ ПРОЕКТИРОВАНИИ КЛАСТЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СЕТЕЙ

А.Ю. Чесалов

Тверской государственный технический университет

Один из основных вопросов, возникающих в любой организации при проектировании или реорганизации распределенных кластерных вычислительных сетей (КВС) состоит в определении требова-

ний к производительности вычислительных средств, обеспечивающих оптимальное время отклика на запросы.

Решение этих задач должно обеспечить эффективное и экономичное функционирование КВС за счет выбора оптимального варианта построения системы, рационального распределения задач между серверами и учетом многих других факторов.

Учет этих и других факторов в ходе проведения специальных тестирований может значительно усложнить и увеличить время их проведения. Поэтому желательно иметь аналитическую модель работы КВС, позволяющую гарантировать ориентировочные оценки в широком диапазоне изменения всех основных факторов, которые можно было бы использовать для целенаправленного выбора различных вариантов построения системы.

В качестве базовой выбрана модель, учитывающая следующий перечень основных характеристик серверов (кластеров) КВС:

- число задач N – решаемых ИС; число запросов – N_{p3}; число обработанных документов $N_{\text{род}}$;
- периодичность решения τ_i или интенсивность поступления в сервер на обработку задачи $\lambda_{\text{ср}}$;
- требуемые времена отклика (ответа) – T;
- среднее процессорное время решения одной задачи (обработка запроса) $T_{\text{п}}$ или число машинных операций (ввода-вывода) на одну задачу (запрос) – $S_{\text{п}}$;
- среднее число запросов к данным на удаленной базе данных, и собственной локальной базе, приходящиеся на 1 задачу – D_i;
- средний объем задач (программ) в оперативной памяти сервера – J;
- средний объем рабочей зоны оперативной памяти – J_p;
- число серверов-клиентов, подключенных к серверу – m;

В качестве выходных основных показателей КВС в такой модели выбраны средние времена ожидания и обслуживания заявок (вре-

мена отклика), загрузка КВС и ее отдельных вычислительных ресурсов.

Необходимые исходные данные берутся из технического задания на систему или по результатам ее обследования, остальные исходные данные могут быть получены в результате анализа алгоритмов решаемых задач и особенностей структурного построения КВС.

При определении интенсивности поступления в сервер «средней» задачи $\lambda_{ср}$ необходимо предварительно выполнить расчет усредненных характеристик комплекса решаемых задач, для чего по каждой задаче вычисляется стохастический коэффициент повторяемости. Все характеристики для комплекса задач усредняются.

Для серверов, на которых решаются задачи в режимах запросов и обработки документов рассчитывается загрузка ресурсов и время выполнения задач в режиме запросов, а так же быстродействие процессора для задач обработки данных.

Постановка задачи выбора типа и количества серверов для КВС по результатам оценки с использованием данной модели может быть сформулирована следующим образом.

Система должна обеспечить решение N задач интенсивностью решения каждой из них λ задач в единицу времени при времени ответа, не превышающем заданного значения.

При решении задачи приняты следующие допущения:

- поступление запросов и время их обслуживания в каждом ресурсе системы распределены по экспоненциальному закону;
- в любой момент времени количество запросов на обслуживание в КВС не может быть больше числа задач в системе N;
- каналы обслуживания в каждом ресурсе (работа с удаленной базой данных, с собственной локальной базой, работа самого процессора) одинаковы и могут обслужить любой запрос к ресурсу.

Ожидаемое время ответа для такой системы определяется по формуле:

$$T = 1/(\mu_2/D_2 - \lambda) + 1/(\mu_3/D_3 - \lambda) + 1/(\mu_1/D_1 - \lambda).$$

Требуемое процессорное время решения задачи для обеспечения заданного времени ответа определяется по формуле:

$$T_{п1} = (1 - K_c)/(\lambda + [T - 1/(\mu_2 - \lambda) - 1/(\mu_1 - \lambda)]^{-1}),$$

где K_c = (0,1:0,2) – коэффициент системного времени, зависящий от характеристик прерывания, частоты обращения к ресурсу и интенсивности запросов, тогда требуемое быстродействие процессора можно найти по формуле:

$$W_{mp} = W^*(T_п/T_{п1}) = (W \Sigma\delta_i T_{пi})/T_{п1},$$

где W – быстродействие процессора, для которого задавалось время $T_п$.

Количество процессоров, необходимое для обеспечения заданного времени ответа определяется по формуле:

$$C = [W_{mp}/W_p],$$

где W_p - производительность процессора для сервера рассматриваемого типа.

С использованием данной математической модели была проведена сравнительная оценка серверов NCR, Tandem и Sequent, подвергавшихся тестированию на предмет проверки требований к их производительности.

Получено, что среднее время ответа по каждому документу 60 с не будет превышено, если за 1 се будет обрабатываться не более 200 документов в сервере NCR и не более 250 документов в Sequent и Tandem. Поскольку средняя расчетная интенсивность поступления на обработку составляет 33,3 документа в секунду, а вероятность превышения этих величин при экспоненциальном законе распределения времени поступления документов пренебрежимо мала.

В то же время выявлено, что неоправданно большое значение допустимого времени ответа (60 с), заданное заказчиком привело к необоснованно завышенной частоте поступления запросов на обработку (120 тысяч и 2 млн. запросов в час), принятой при тестировании, и как следствие к тому, что серверы при испытаниях показали

заниженное среднее время обработки запросов, так как работали в области перегрузки с коэффициентом загрузки, близким к единице.

Для обеспечения соответствующего уровня надежности и производительности КВС большое значение имеет количество (K) и расположение узлов сети и замыкаемых на них центров обработки информации (ЦОИ) ($N_k, k = \overline{1, K}$), которые представляют собой сложные аппаратно-программные комплексы, определяемые в ходе выбора топологической структуры.

Каждый узел сети обладает такими параметрами, как класс C_A, объем суточной входящей V' и исходящей V'' информации, $n_{\text{см}}$ – количество сеансов связи в сутки, а также расстояния между узлами L_i.

Соответственно годовой объем обрабатываемой информации для ξ-сервера (кластера) рассчитывается как

$$V_{\xi} = \sum_{i=1}^{T} n_{ci} (V_{ci}^{'} + V_{ci}^{''}) * G\mu,$$

где T – продолжительность рабочего дня в часах; n_c – количество сеансов связи в час ; V_c' – объем входящей информации за сеанс связи (например, бит/с); V_c'' – объем исходящей информации за сеанс связи; G – количество рабочих дней в году; μ – коэффициент увеличения объема информации зависящий от помехозащищенных кодов и от программного контроля достоверности информации.

Если ЦОИ обслуживает p серверов в году, то общий объем обрабатываемой информации КВС составляет:

$$W = \sum_{\xi=1}^{p} V_{\xi} .$$

Оборудование выбирается из множества типов оборудования D, рекомендованного для каждого i-го класса.

$$D^l = \{d_\alpha, d_\beta, ..., d_\omega\},\ l = \overline{1, Z},$$

где Z – количество классов.

Каждый тип оборудования обладает набором следующих характеристик: максимальная скорость передачи данных по каналу связи ($V_{\max}$), стоимость (C), занимаемая площадь (γ), количество устройств ввода-вывода (ν), количество обслуживающего инженерно-технического персонала (λ), потребляемая мощность (ω);

$$d_j = (V_{\max}, C, \gamma, \nu, \lambda, \omega),\ j = \overline{1, M},$$

где M – количество типов оборудования.

При выборе оборудования необходимо учитывать также тарифы на аренду каналов связи, стоимость квадратного метра занимаемой оборудованием площади и единицы потребляемой им энергии, заработную плату обслуживающего персонала. Задача заключается в выборе из множества оборудования передачи данных такое и в таком количестве, чтобы обеспечить своевременную передачу объемов входящей и исходящей информации каждому ЦОИ и при этом суммарные приведенные затраты на приобретение и эксплуатацию оборудования были бы минимальными:

$$C_{np} = \sum_{K=1}^{K} \sum_{i=1}^{N_K} C_i^K * x_j,\ i = \overline{1, N_k},\ j = \overline{1, M},$$

$$t_{ij} = f(V', V'', V_{\max}, n_{cm}) < T_i^{доп},$$

где x_j – количество оборудования j-го типа; t_{ij} – время, затрачиваемое j-м оборудованием і-го сервера на передачу требуемых объемов информации; $T_i^{доп}$ – допустимое время на передачу для i-го сервера; C_i^K – приведенные затраты на единицу оборудования i-го сервера, подключенного к K-му узлу.

Определение необходимого количества анализируемого оборудования x_j осуществляется в зависимости от классов решаемых задач и режимов работы.

$$C_i^K = K_z * E_A + Z_{\text{экспл.}},$$

где $K_Z = f(D)$ – капитальные затраты; E_A – нормативный коэффициент капитальных вложений; $Z_{\text{экспл.}}$ – эксплуатационные затраты за год, определяемые как сумма амортизационных затрат на обслуживание каналов вязи.

В такой постановке задача решается как задача целочисленного программирования и для ее формализованного решения производится последовательный перебор каждого типа оборудования из подмножества рекомендуемого с выбором оптимального по приведенным затратам.

ОРГАНИЗАЦИЯ РАСПРЕДЕЛЕННОЙ ВЫЧИСЛИТЕЛЬНОЙ СЕТИ ЦЕНТРА ВЫСОКОПРОИЗВОДИТЕЛЬНОЙ ОБРАБОТКИ ИНФОРМАЦИИ КАЗАНСКОГО НЦ РАН

Г. Шамов, М. Астафьев

Отдел информационных технологий
Казанского Научного Центра РАН

В 2000 г., в рамках ФЦП «Интеграция» при Казанском Научном Центре РАН был создан Центр высокопроизводительной обработки информации. В настоящее время он предоставляет вычислительные ресурсы пользователям ВУЗов и институтов РАН г. Казани, обслуживая около 20-ти исследовательских групп. Практически все задачи, решаемые пользователями на кластерах Центра, относятся к весьма ресурсоемким задачам квантовой химии и молекулярной динамики. Первоначально в качестве вычислителя использовался созданный в 2000 г. вычислительный кластер КазНЦ РАН на основе рабочих станций Alpha, объединяющий 8 станций DS10L и одну двухпроцессорную DS20E, с Fast Ethernet в качестве коммуникационной среды.

Впоследствии, в 2001м-2002м гг. были созданы еще два вычислительных кластера на базе ПК, в Казанском Госуниверситете (6 AMD Athlon 900MHz,) и Казанском Государственном Технологическом Университете (11 AMD Athlon 1200MHz). Ограниченный

бюджет проекта не позволил нам использовать для этих новых кластеров дорогие решения, такие как Myrinet или SCI, а малая пропускная способность Fast Ethernet уже не могла нас удовлетворить. Одним из очевидных путей повышения пропускной способности являлся переход на Gigabit Ethernet. Другим, выглядевшим привлекательно, прежде всего в силу своей меньшей стоимости, путем было использовании технологии объединения нескольких сетей Fast Ethernet в одну логическую сеть с большей пропускной способностью, т.н. «channel bonding». Для кластера КГТУ было выбрано первое решение, для кластера КГУ – второе. Все три кластера работают под управлением ОС Linux.

Результаты тестовых расчетов показали, что объединения трех каналов Fast Ethernet дает максимальную пропускную способность сравнимую с таковой для PCI-32 Gigabit Ethernet (270 и 330 Мбит/с, соотв.), а латентности для него оказываются даже меньшими. Недостатками «channel bonding» являются сложность конфигурирования и обслуживания, а также большая загрузка CPU узлов кластера по сравнению с Gigabit Ethernet.

Тесты, основанные на реальных квантовохимических задачах, показали улучшение масштабируемости по сравнению с Fast Ethernet для задач использующих модель распределенной общей памяти, таких как DDI-реализация метода MP2 в программе GAMESS [1]. Нужно также отметить, что масштабируемость GAMESS вообще оставляет желать лучшего, и для реализованных в модели реплицируемой памяти (и наиболее часто применяемых на практике) вычислений SCF не улучшается с увеличением пропускной способности коммуникационной среды.

Важнейшей задачей при эксплуатации вычислительного центра коллективного пользования является правильная организация проведения вычислений. Для эффективной работы в многопользовательской среде необходимо так организовать выполнение заданий, чтобы предотвратить неконтролируемую конкуренцию процессов пользователей за ресурсы вычислительных узлов, обеспечить справедливое и надежное обслуживание задач пользователей, предоставить им удобный интерфейс для управления заданиями. Такая организация невозможна без внедрения системы пакетной работы (batch queuing system).

На каждом из наших кластеров управление заданиями осуществляется при помощи системы PBS Pro 5.1.[2] Для кластера КазНЦ после запуска PBS загрузка кластера весьма сильно выросла и достигла значений 70-90%; задачи пользователей вынуждены ожидать своего запуска в очередях PBS достаточно длительное время, 0.5 -- 1.2 суток. Для кластеров КГТУ и КГУ, построенных на базе учебных классов ПК, режим вычислительного кластера для проведения расчетов на них устанавливается средствами PBS, в свободное от учебных занятий время. В учебное время они используются как компьютерные классы в самых разнообразных целях. Но нужно отметить, что во время, выделенное для расчетов, загрузка этих кластеров также велика.

За время использования кластеров Казанского НЦ РАН накоплен некоторый опыт их использования в условиях многопользовательской среды, который обобщается в настоящем сообщениии. Большое количество пользователей и высокая загруженность кластеров остро ставит вопросы выбора наилучших политик планирования. Перегруженность кластеров большим количеством залдач препятствует выполнению очень больших задач и задач, запрашивающих на большое количество ресурсов. Нагрузка по кластерам распределяется неравномерно. Использование нескольких кластеров для пользователя оказывается неудобно, т.к. ему необходимо помнить о различиях в их конфигурациях. Особенно это касается кластеров КГТУ и КГУ, где пользователю нужно самому координировать запросы своих задач с сеткой расписания учебного процесса.

В этой связи является актуальным создание распределенной вычислительной сети, в которой координировалась бы постановка задач в очереди отдельных кластеров, так, чтобы загрузка каждого кластера была максимальной, а среднее время ожидания задачи в очередь -- минимальным. Сеть должна также предоставить пользователю единый, прозрачный и безопасный доступ ко всем доступным вычислительным ресурсам.

Создаваемая сеть должна включать следующие компоненты метапланировщик, управляющий метаочередью заданий, общей для всех кластеров входящих в нее, безопасное соединение кластеров по обычной, открытой сети Интернет, системы аутентификации пользователей, репликации пользовательских каталогов, и др.

В настоящее время мы работаем над основным компонентом распределенной сети: созданием метаочереди заданий используя метапланировщик Silver[3], действующий по принципу резервирования. Silver способен управлять любыми кластерами, на которых установлен локальный планировщик Maui [3].

Стендовые испытания метапланировщика Silver и планировщика Maui, проведенные нами на кластерах КазНЦ и КГТУ, продемонстрировали работоспособность такого решения.

Литература

1. **GAMESS: M.W.Schmidt et.al. // J. Comput. Chem. (1993) 14, P. 1347–1363.**
2. **PBS Pro: http://www.pbspro.com/.**
3. **Maui и Silver: http://www.supercluster.org.**

АВТОМАТИЗАЦИЯ РАЗРАБОТКИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ В ТЕХНОЛОГИИ ГРАФО-СИМВОЛИЧЕСКОГО ПРОГРАММИРОВАНИЯ

В.В. Жидченко

Самарский государственный аэрокосмический университет

Введение

Быстрый рост сфер использования, сложности и ответственности функций, выполняемых современным программным обеспечением, резко повысил в последнее время требования к качеству и надежности его функционирования. Методы обеспечения заданного уровня надежности программ можно разделить на две группы: методы безошибочного проектирования и методы обнаружения и устранения ошибок. Одним из средств, реализующих методы первой группы, является технология графо-символического программирования (ГСП) [1]. Она позволяет избежать множества ошибок при проектировании за счет образного представления программы в виде графа, описывающего ход вычислительного процесса, а также за счет автоматизации многих этапов проектирования. Графический язык технологии ГСП предоставляет также удобные средства для описания параллельных программ, а принципы, заложенные в

основу технологии, делают процесс создания таких программ простым и наглядным. Исходя из этого, было принято решение дополнить технологию ГСП методами и средствами написания параллельных граф-программ.

Технология графо-символического программирования

Программирование в технологии ГСП начинается с изучения предметной области (ПО) - набора данных, над которыми нужно произвести вычисления (словаря данных), и набора программных модулей, осуществляющих эти вычисления. Программные модули в технологии ГСП называют акторами.

В отличие от традиционного текста программы на алгоритмическом языке, в технологии ГСП создается графический образ программы. В специализированном редакторе рисуется граф (обозначим его G), в вершинах которого лежат акторы, а дуги графа определяют передачу управления от одной вершины к другой. Каждой дуге приписывается логическая функция над данными ПО, истинное значение которой разрешает переход по дуге, а ложное – запрещает. Эту функцию называют предикатом. Созданный граф, называемый агрегатом, может входить в качестве вершины в другие графы. Текстовое описание применяется лишь для акторов и предикатов, которые записываются на одном из алгоритмических языков. В качестве примера на рис. 1 приведена граф-программа, предназначенная для численного решения дифференциального уравнения в частных производных.

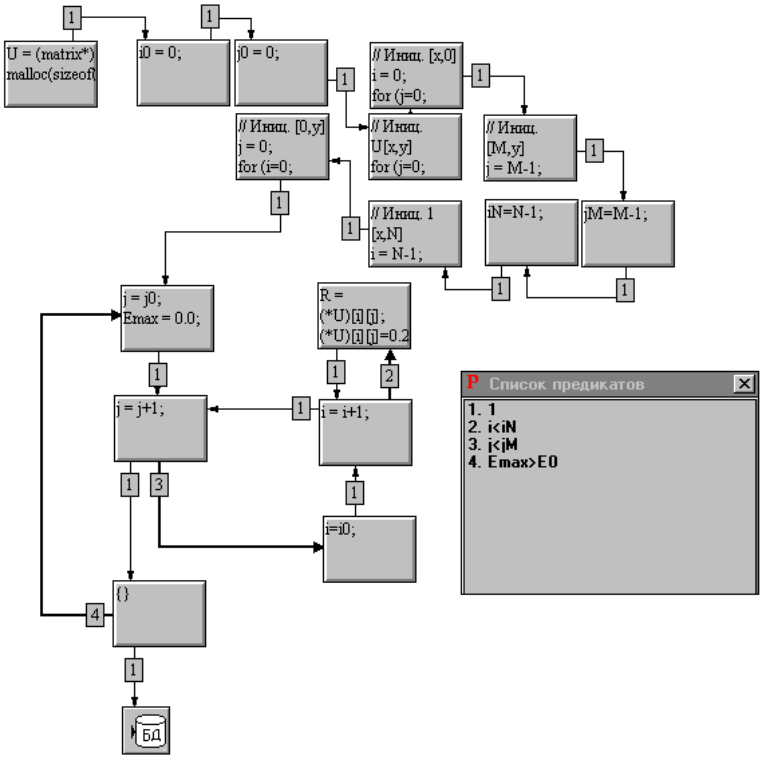


Рис.1

Технология ГСП позволяет сократить сроки разработки сложных программных продуктов на традиционных языках программирования C, Pascal и повысить их надежность за счет автоматизации проектирования программного продукта, уменьшить сложность и повысить комфортность труда программиста за счет образного графового стиля программирования, а также упростить режим отладки и модификации программ.

Созданная на кафедре информационных систем и технологий СГАУ система программирования с использованием технологии ГСП позволяет транслировать графический образ программы в выполнимые коды ЭВМ.

Создание параллельных программ в технологии ГСП

Рост сложности решаемых задач и удешевление аппаратной части современных компьютеров привели к активному использованию методов параллельного программирования при создании современных сложных программных комплексов. Использование технологии ГСП особенно эффективно при разработке сложных программ, поэтому наличие в технологии средств параллельного программирования стало острой необходимостью.

Параллельные программы содержат фрагменты кода, выполняемые одновременно на нескольких процессорах. Эти фрагменты называются параллельными ветвями. Процессоры могут принадлежать одной многопроцессорной ЭВМ или нескольким ЭВМ, объединенным сетью передачи данных.

Зачем понадобилось создавать параллельные программы?

Параллельные программы обладают важными преимуществами перед традиционными последовательными программами:

– одновременное выполнение различных функций позволяет значительно сократить время решения многих задач – повышается быстродействие программы. Особенно актуальным данный подход становится в последнее время, когда сложность решаемых задач не позволяет традиционным последовательным программам найти решение за приемлемое время;

– большая размерность современных задач приводит к необходимости оперировать огромными объемами данных, а значит, увеличивать сложность и стоимость аппаратных средств. Применяя методы параллельного программирования, можно обрабатывать различные части области данных одновременно на нескольких недорогих ЭВМ, связанных коммуникационной средой;

– многие задачи содержат в себе естественный параллелизм, и представление алгоритмов их решения в параллельной форме позволяет упростить восприятие алгоритмов человеком, а значит ускорить процесс решения задачи и сделать его более надежным за счет лучшего понимания работы программы челове-

ком.

Однако за эти преимущества приходится платить. Каковы основные трудности, возникающие при создании параллельных программ?

1) Способ записи программы. При последовательном программировании операторы программы выполняются последовательно, и принятый способ описания алгоритма в виде последовательности функций, выполняемых одна за другой, является достаточно простым для записи его в текстовом виде. При записи параллельной программы сложно изобразить одновременное выполнение различных функций в текстовом виде, особенно при большом объеме кода, содержащегося в этих функциях.

2) Описание синхронизации во времени между параллельными ветвями. Необходим механизм согласования параллельных ветвей программы во времени, а также наглядное его описание.

3) Написание надежных программ. При параллельном программировании к ошибкам, которые могут возникнуть в последовательной программе (деление на нуль, зацикливание, передача управления на несуществующий адрес и т.п.), добавляются новые типы ошибок, связанные с синхронизацией во времени и с совместным доступом к данным. Контроль правильности программы осложняется еще и тем, что на данные, которыми она оперирует, воздействуют внешние факторы – параллельные подпрограммы. При последовательном программировании мы уверены в том, что записанное в ячейку памяти значение не изменится до следующего обращения к нему. В параллельной программе это значение может быть изменено одной из подпрограмм, выполняемых одновременно.

Таким образом, при написании параллельных программ значительно увеличивается количество факторов, которые необходимо учитывать программисту для создания работоспособной, надежной программы. Следовательно, сделать это становится труднее.

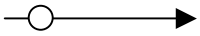
Принятый в технологии ГСП графический способ представления программы позволяет преодолеть традиционные трудности параллельного программирования:

– программа становится наглядной и легко модифицируемой;

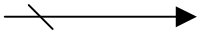
– упрощается описание механизма синхронизации параллельных процессов;

В конечном итоге повышается надежность программы за счет упрощения и автоматизации многих этапов ее создания.

Для описания параллельной программы графический язык технологии ГСП расширяется новыми символами: параллельная дуга (определяет начало параллельной ветви алгоритма) и терминирующая дуга (обозначает конец параллельной ветви), приведенными на рис.



параллельная дуга



терминирующая дуга

2.

Рис. 2

Проектируемая граф-программа разбивается на несколько ветвей, выполняемых параллельно. Число ветвей неограниченно. Каждая ветвь начинается параллельной дугой и заканчивается терминирующей. В предлагаемой модели запрещена передача управления из вершины одной параллельной ветви в вершину другой ветви.

На рис. 3 приведен пример параллельной граф-программы, реализующей решение дифференциального уравнения в частных производных.

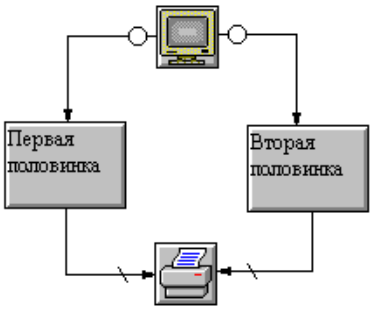


Рис. 3

Увеличение производительности достигается в ней за счет одновременной обработки двух частей матрицы системы конечно-разностных уравнений (вершины «Первая половинка» и «Вторая половинка»).

В технологии ГСП все акторы работают с одним множеством данных - одной предметной областью. Поэтому при разработке параллельных программ в технологии ГСП используется модель с общей памятью. Для разрешения конфликтов по данным между параллельными ветвями разработаны эффективные и удобные в использовании методы синхронизации. При разработке программ для распределенных систем (например, для все более популярных кластеров) каждая параллельная ветвь может иметь локальные данные, которые после окончания обработки пересылаются основной программе, запустившей эти ветви.

Синхронизация во времени между параллельными ветвями выполняется с помощью механизма передачи сообщений. Для этого вводится граф синхронизации Gs, получаемый из графа программы (рис. 4). В нем дуга, соединяющая две вершины, означает, что запуск на выполнение вершины, в которую входит дуга, зависит от того, выполнилась или нет вершина, из которой исходит дуга.

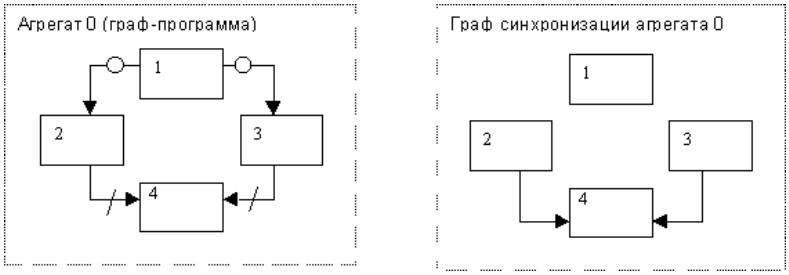


Рис. 4

Вводится список сообщений:

$$L_c = [C_{i_0 j_0}, C_{i_1 j_1}, \ldots C_{i_m j_r}],$$

где C_{ij} – сообщение, посылаемое вершиной i вершине j. Наличие сообщения в списке говорит о том, что вершина i завершила работу и хочет сообщить об этом вершине j.

Запуск вершины j зависит от значения семафорного предиката $R_j = f(C_{i_0 j}, C_{i_1 j}, \ldots C_{i_m j})$ – логической функции, определенной на множестве сообщений, посылаемых данной вершине. Если $R_j = 1$, запуск вершины разрешается, в противном случае вершина ждет, когда семафорный предикат R_j станет истинным.

Дуги на графе синхронизации изображают передачу сообщений между вершинами агрегата.

Графический способ записи программы облегчает описание синхронизации пользователю технологии – для него этот процесс заключается в рисовании дуг на графе синхронизации.

Разработаны методы проверки корректности семафорных предикатов (предотвращения блокировок), а также автоматического поиска критических секций – участков программы, на которых возникает конфликт совместного использования данных параллельными ветвями.

Рассмотрим процесс создания параллельной программы в технологии ГСП. Он состоит из следующих этапов:

1) Создание словаря данных.

2) Создание акторов – небольших программ на одном из традиционных языков программирования. Акторы выполняются последовательно и описывают простейшие вычисления над переменными ПО.

3) Рисование граф-программы.

4) Создание предикатов – условий передачи управления от одной вершины другой. Предикаты записываются на одном из традиционных языков программирования и выглядят как логические выражения над переменными программы.

5) Рисование графа синхронизации.

6) Создание семафорных предикатов для синхронизируемых вершин.

7) Компиляция и запуск программы.

Разработана система, реализующая параллельную граф-программу средствами технологии MPI (Message Passing Interface) [2]. Каждой ветви соответствует отдельный процесс MPI. Процесс выполняется на отдельном компьютере (или процессоре в многопроцессорной системе), при этом на одном компьютере одновременно может выполняться несколько процессов. Распределение процессов по компьютерам задается жестко путем конфигурирования системы MPI. Разработчику граф-программы не требуется изучать MPI, знать ее особенности или синтаксис. Все преобразования выполняются автоматически и скрыты от пользователя системы.

Литература

1. ***Коварцев А.Н.*** **Автоматизация разработки и тестирования программных средств. Самара: Изд-во СГАУ, 1999.**
2. ***Snir M.*****,** ***Otto S.*****,** ***Huss-Lederman S.*****,** ***Walker D.*****,** ***Dongarra J.*** **MPI: The Complete Reference. MIT Press, 1998.**

СОДЕРЖАНИЕ

Высокопроизводительные параллельные вычисления на кластерных системах

Материалы Второго международного научно-практического семинара
26–29 ноября 2002 г.

Под ред. **Р.Г. Стронгина**
Отв. за выпуск **В.А. Гришагин**

Формат 60×84 1/16 .Бумага офсетная. Печать офсетная.
Гарнитура Таймс. Усл. печ. л. 21,5. Уч.-изд. 21,75.
Тираж 200 экз. Заказ .

Типография ННГУ. 603000, Н. Новгород, ул. Б. Покровская, 37

366