Альтернативные модели программирования

Корж А.А.

ОАО «НИЦЭВТ»

Начальник отдела

Высокоскоростных коммуникационных сетей

План

- Технологии программирования
- Описание библиотеки SHMEM
  - SHMEM крэш-курс
  - http://exascale.ru/shmem.pdf
- Трюки при распараллеливании UA
- Задание

Технологии параллельного программирования

Классификация

- Требуют общую память
  - Pthreads, OpenMP, TBB
- Не требуют общей памяти
  - **MPI**,PVM,HPF
  - Параллелизм по данным Charm++ итп
  - Языки класса PGAS

Двухсторонние и односторонние коммуникации

- Двусторонние коммуникации
  - Оба процесса участвуют в коммуникациях
    - Send и Recv

- Односторонние
  - Участвует только одна сторона
    - Put и Get
    - Для того чтобы сделать Put и Get надо знать адрес: как узнать адрес на другом узле?

GAS vs PGAS

- GAS – вся память общая
- PGAS – вся память общая, но у каждого есть своя локальная часть, к которой доступ гораздо быстрее

- Удобно для NUMA систем и систем с распределенной памятью

PGAS кроме SHMEM

- Global Arrays (GA)
- UPC
- CAF (Co-array Fortran)
- Titanium

- Fortress (Sun)
- Chapel (Cray)
- X10 (IBM)

Библиотека SHMEM

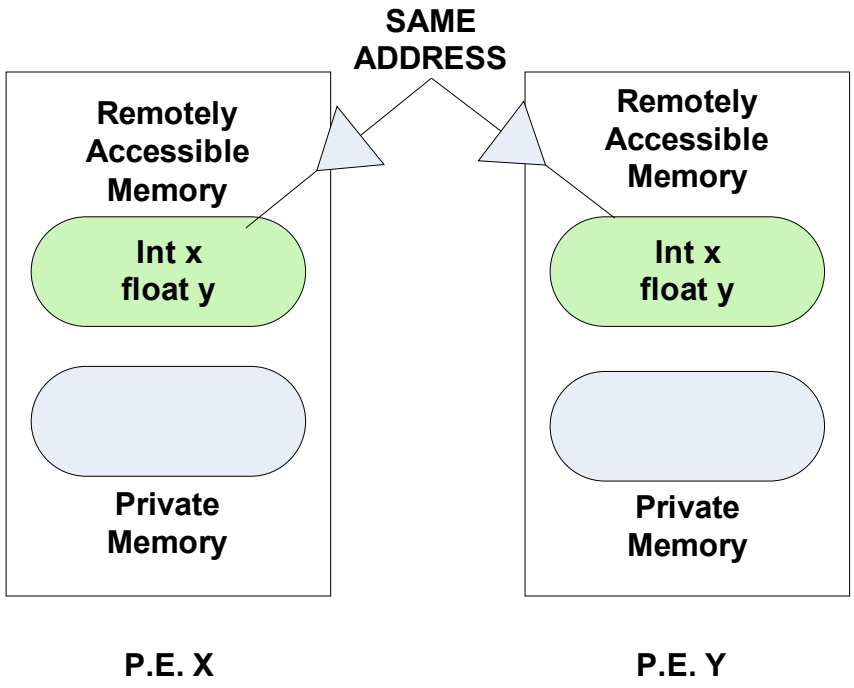
Что есть SHMEM?

- SHard MEMory library (SPMD модель)
  - Библиотека функций похожих на MPI (например *shmem_get())*
- Доступна для C / Fortran
- Гибридная message passing / shared memory модель
  - Message-passing
    - Явные коммуникации и синхронизации
    - Нужно явно указывать адрес куда отправлять данные(номер узла)
  - Shared-memory
    - Обеспечивает логически картину общей памяти
    - Не-блокирующие односторонние коммуникации
      - Позволяет любому процессору получить доступ к данным любого другого процессора, причем последний не участвует в обменах

Что есть SHMEM?

- Чтобы послать данные, нужно знать адрес куда их послать
  - Идея: один и тот же адрес на всех процессорах

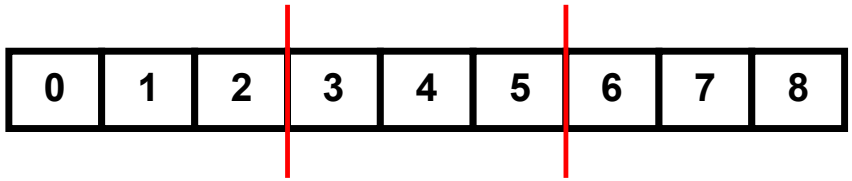
- К каким объектам можно обращаться удаленно
- (симметричный хип.)
  - Глобальные переменные
  - common-блоки
  - Указанные прагмой или директивой в Фортране
  - Выделенные с помощью shmem_alloc()



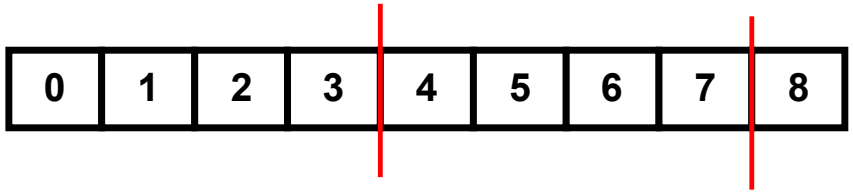
Декларация общей памяти SHMEM vs. UPC

- Пример: 1D массив из 9 элементов на системе с 3 процессорами

- Распределение 1
  - SHMEM: int x[3];
  - int *x; x = shmem_alloc(12);
  - UPC: shared [3] int x[9];



- Распределение 2
  - SHMEM: int x[4];
  - UPC: shared [4] int x[9];



Почему писать на SHMEM?

- Проще, чем MPI-1 (PVM итп)
- Можно запустить на нескольких узлах (OpenMP нельзя)

- Низкая задержка высокая пропускная **способность** (*RDMA*)

- Предоставлены **быстрые** механизмы синхронизации
  - Barrier
  - Fence, quiet

- **Коллективные операции**
  - Broadcast
  - Collection
  - Reduction

АЗЫ
Инициализация

- Заинклюдить shmem.h
- *shmem_init()* – Инициализация SHMEM
- *my_pe()* – Узнать свой процессор
- *num_pes()* – Узнать общее число процессоров
- *shmem_finalize()* – Завершение SHMEM

```
#include <stdio.h>
#include <stdlib.h>
#include <shmem.h>
int main(int argc, char **argv)
{

   shmem_init();
   printf("Hello World from process %d of %d\n",
   my_pe(), num_pes());
   shmem_finalize();
}
```

АЗЫ
Выделение памяти

- shmem_alloc(int nbytes)
  - Выделение symmetric heap
    - *void shmem_**alloc**(int bytes)*
  - Только в эту память можно читать писать с других узлов

АЗЫ
Передача данных (1)

- Put
  - Одно значение
    - *void shmem_**TYPE**_p (**TYPE** *addr, **TYPE** value, int pe)*
      - TYPE = double, float, int, long, short
  - Массив из нескольких значений
    - *void shmem_**TYPE**_put (**TYPE** *target, const **TYPE**source, size_t len, int pe)*
      - TYPE = double, int, long, ...

АЗЫ
Передача данных (2)

- Get
  - Одно значение
    - *void shmem_**TYPE**_g (**TYPE** *addr, **TYPE** value, int pe)*
      - TYPE = double, float, int, long, short
  - Массивы данных
    - *void shmem_**TYPE**_get (**TYPE** *target, const **TYPE**source, size_t len, int pe)*
      - TYPE = double, int, long,....
  - Изначально функции синхронные, но есть и расширение, где они асинхронные

АЗЫ
Синхронизация

- Барьер
  - *void shmem_barrier_all()*
    - Также вызывает shmem_quiet()

- Ожидание заверения
  - shmem_quiet()

    - *Дожидается окончания всех выполненных до этого операций*

АЗЫ
Коллективные операции

- **Broadcast**
  - Один – всем, но вызывается всеми процессами
  - *void shmem_int_to_all(int *address, int pesrc)*
- **Редукция**
  - Все складывают по double и получают сумму
  - *void shmem_double_allsum(double *target, double value)*
- ***Сборка данных***
  - *void shmem_allgatherv(void *buf, int *starts,int *lens,int *blocksz);*
    - *Собирает каждому весь массив, по кусочку с одного процессора*

АЗЫ
Атомарные операции

- Арифметические
  - *void shmem_**TYPE**_add(**TYPE** *target, **TYPE** value, int pe)*
    - TYPE = int, double
    - Завершается вызовом quiet или barrier
    - Ничего не возвращает

Простейший пример (Array copy)

```
1. #include <stdio.h>
2. #include <shmem.h>
3.
4.
5. main()
6. {
7.   int me, npes, i;
8.   int *source, *dest;
9.   shmem_init();
10.  me = my_pe();
11.  npes = num_pes();
12.
13.  source = shmem_alloc(8*sizeof(int));
14.  dest = shmem_alloc(8*sizeof(int));
```

Простейший пример (Array copy)

```
15. if(me == 1) {
16. for(i=0; i<8; i++) source[i] = i+1;
17. /* put source data at PE1 to dest at PE0*/
18. shmem_int_put(dest, source, 8, 0);
19. }
20.
21. /* Make sure the transfer is complete */
22. shmem_barrier_all();
23.
24. /* Print from the receiving PE */
25. if(me == 0) {
26.   printf(" DEST ON PE 0:");
27.   for(i=0; i<8; i++)
28.   printf(" %d%c", dest[i], (i<7) ? ',' : '\n');
29. }
30. shmem_finalize();
31. }
```

Как работать с SHMEM на BGP

- Самодельная версия – есть не все функции
- Компилировать SHMEM program:
  - `/gpfs/data/anton/shmemcc -o program program.c`
  - `/gpfs/data/anton/shmemcxx -o shmemtest shmemtest.cpp`

Запускать SHMEM программу

- Точно также, как и MPI
- `LL_RES_ID=fen1.32.r mpisubmit.bg a.out`

SHMEM vs OMP

- Программист должен явно распределить данные и выделить обращения к удаленным данным

- Самый простой стиль программирования – глобальные фазы вычислений и коммуникаций разделенные барьером

Где есть SHMEM ?

- Все машины Cray
- Машины с сетью Quadrics Elan
- SGI Altix
- Макеты с сетью НИЦЭВТ
- Макет МВС-Экспресс (А.О.Лацис)
- Скиф-Аврора
- **IBM Blue Gene/P**

SHMEM версия

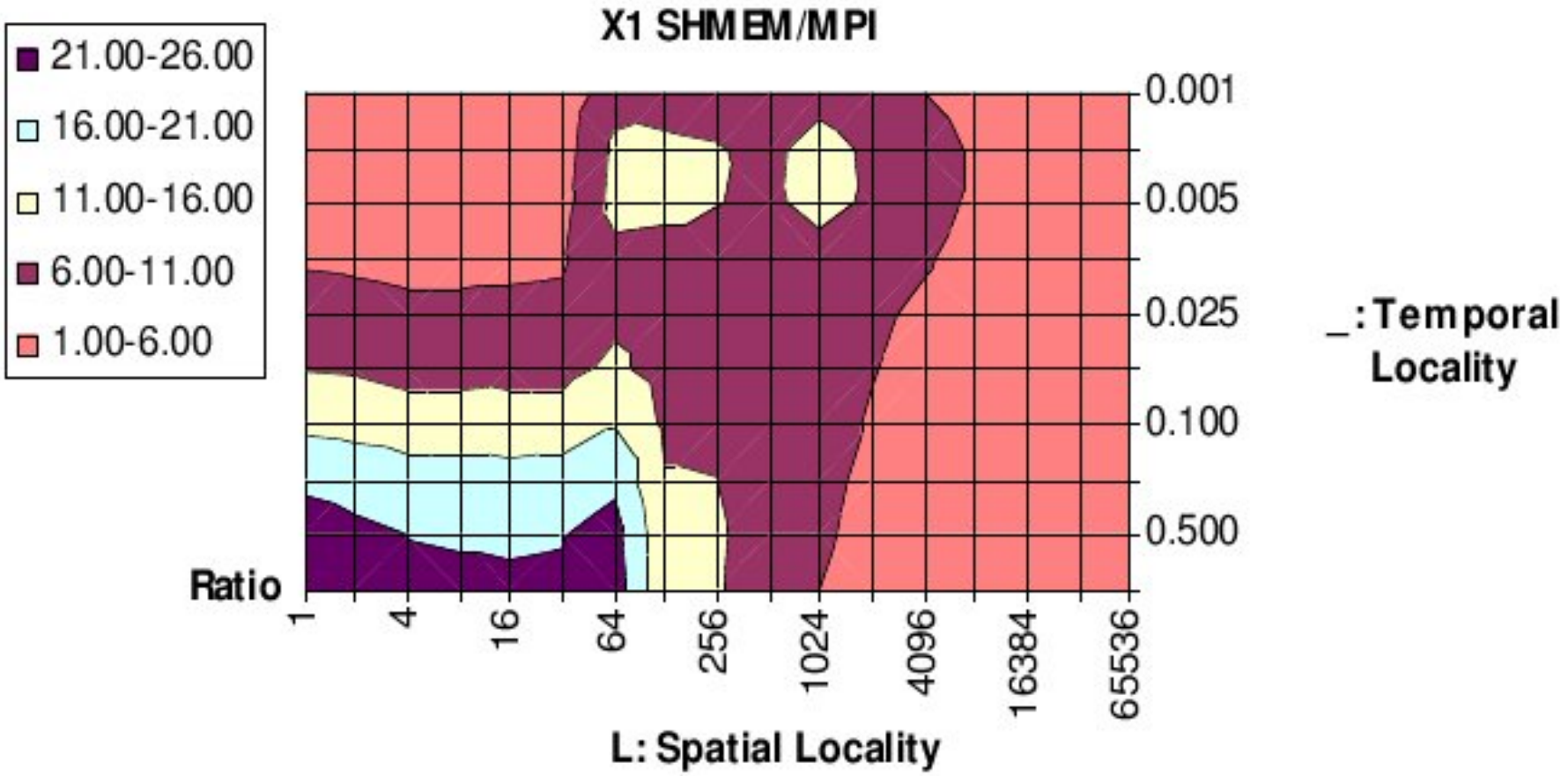
- DCMF Deep Computing Messaging Framework
  - DCMF_GlobalBarrier, DCMF_Messager_advance
  - DCMF_Put, DCMF_Get, DCMF_Send
  - DCMF_GlobalBcast, DCMF_GlobalAllreduce

SHMEM vs MPI2

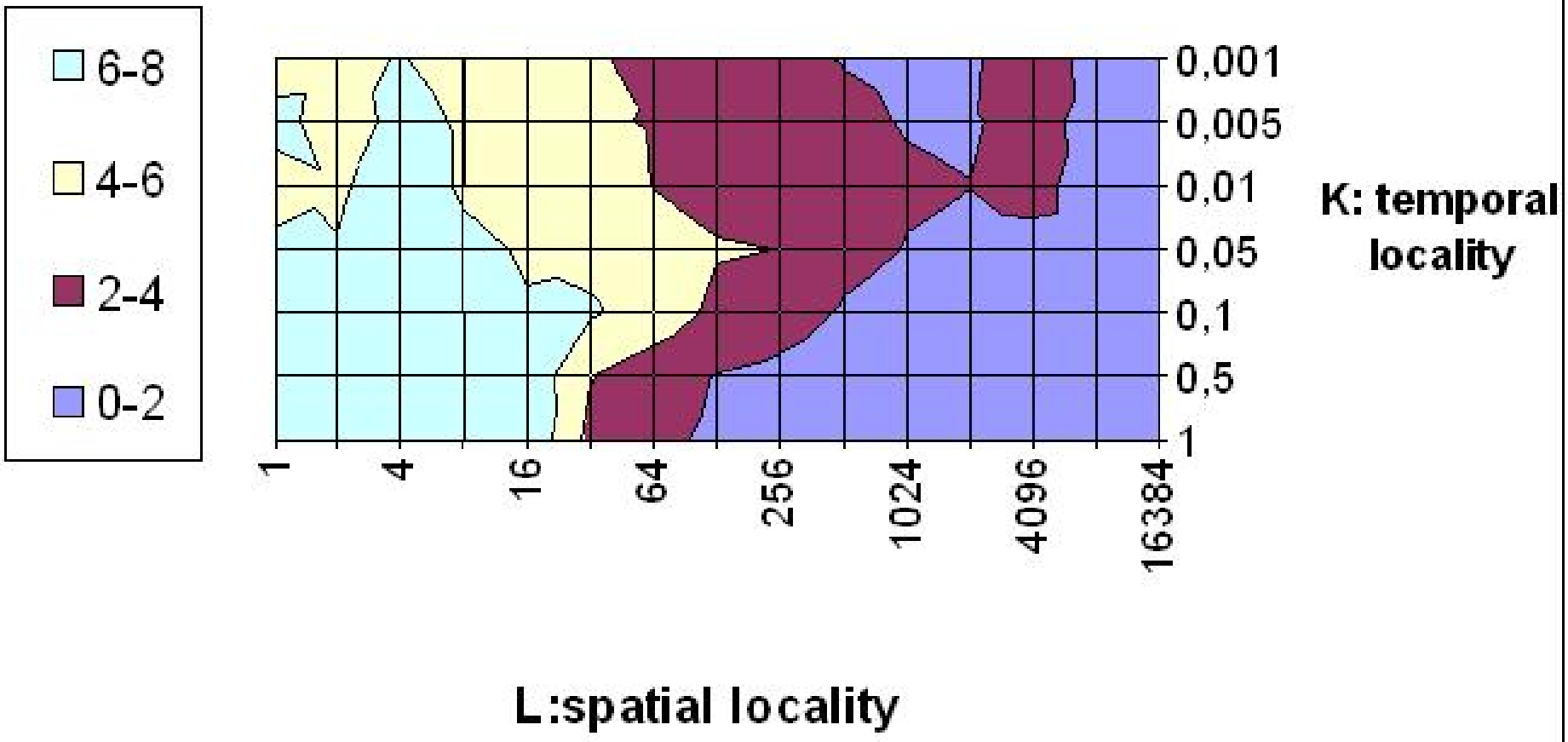
```
int MPI_Put(
  void *origin_addr,
  int origin_count,
  MPI_Datatype
    origin_datatype,
  int target_rank,
  MPI_Aint target_disp,
  int target_count,
  MPI_Datatype
    target_datatype,
  MPI_Win win
);
```

```
void shmem_double_put(
    double *target,
    double *source,
    unsigned  len,
    unsigned  pe
);
```

```
DCMF_Result DCMF_Put
(   DCMF_Protocol_t *          registration,
        DCMF_Request_t *             request,
        DCMF_Callback_t   cb_done,
        DCMF_Consistency
    consistency,
        size_t             rank,
        size_t             bytes,
        DCMF_Memregion_t *
    src_memregion,
        DCMF_Memregion_t *
    dst_memregion,
        size_t             src_offset,
        size_t             dst_offset,
        DCMF_Callback_t   cb_ack
    )
```


X1 SHMEM/MPI
21.00-26.00
16.00-21.00
11.00-16.00
6.00-11.00
1.00-6.00
Ratio
0.001
0.005
0.025
0.100
0.500
_: Temporal Locality
1
4
16
64
256
1024
4096
16384
65536
L: Spatial Locality

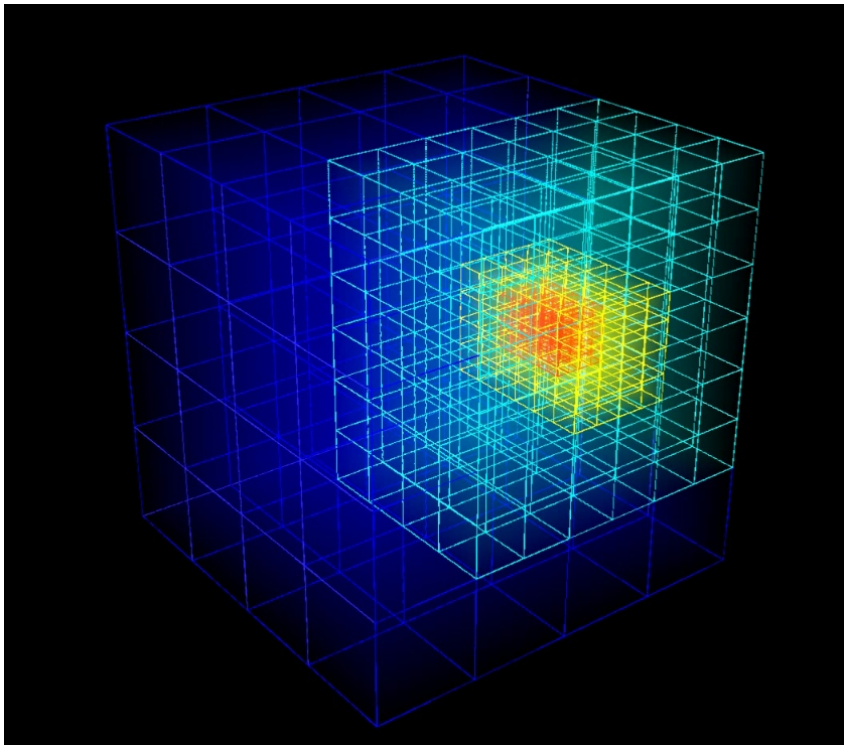
Blue Gene /P, SHMEM/MPI



Задача:
Unstructured Adaptive

Описание бенчмарка NPB UA

- Уравнение теплопереноса в кубической области (конвекция + диффузия)
- Источник тепла –
- Движущийся шар
- Нерегулярная сетка
- Каждый 5й шаг – сетка меняется
- Используется параллельный 3D Mortar method



Основной цикл

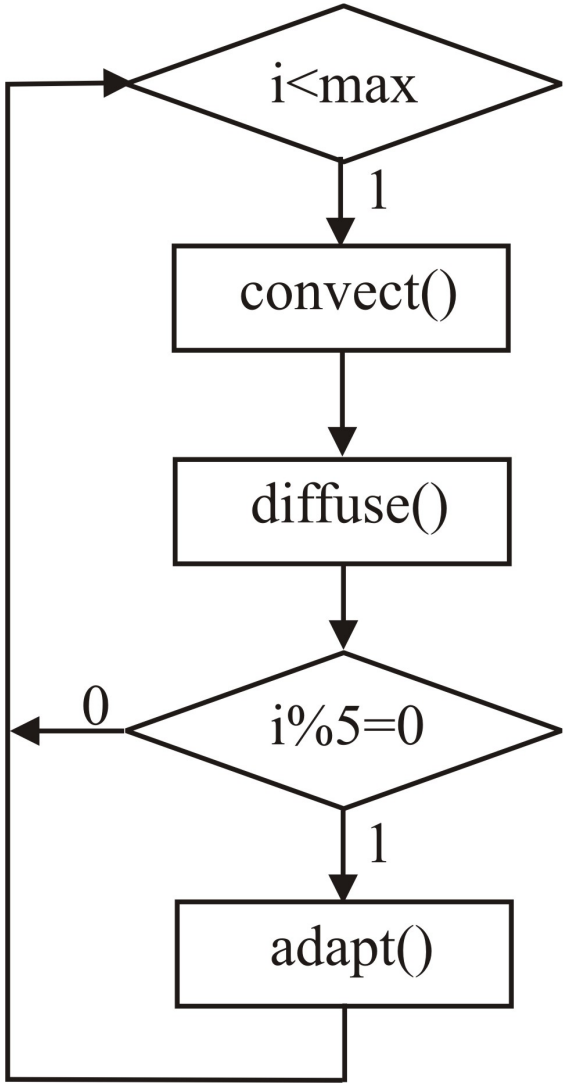
- convect – продвижение во времени конвекционного члена исходного уравнения

$$\hat{T}^{n+1} = \mathrm{RK}_4\left(-\mathbf{v}\nabla T^n + S(\mathbf{x}, t)\right)$$

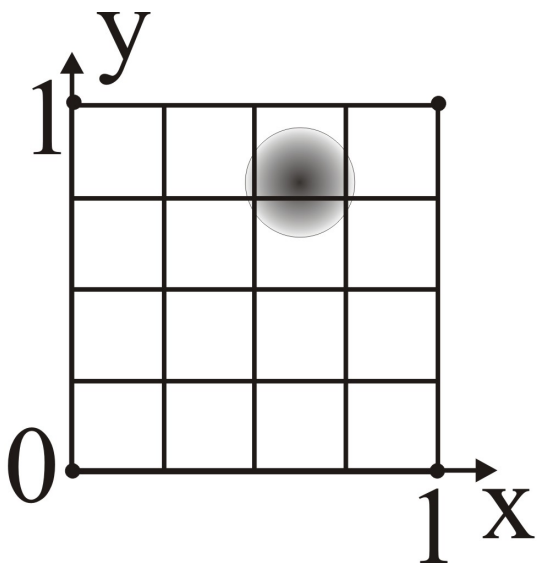
- diffuse – продвижение вперёд диффузионного члена уравнения методом сходящегося градиента

$$\frac{T^{n+1} - \hat{T}^{n+1}}{\Delta t} = \varepsilon \nabla^2 T^{n+1}$$

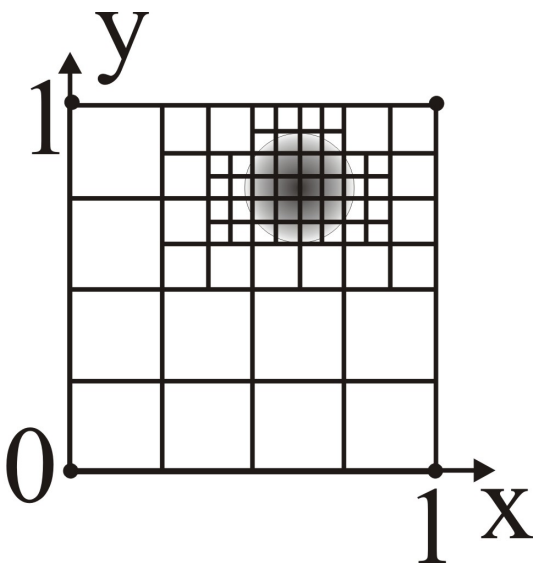
- adapt – изменение сетки



Нерегулярность сетки



равномерная сетка

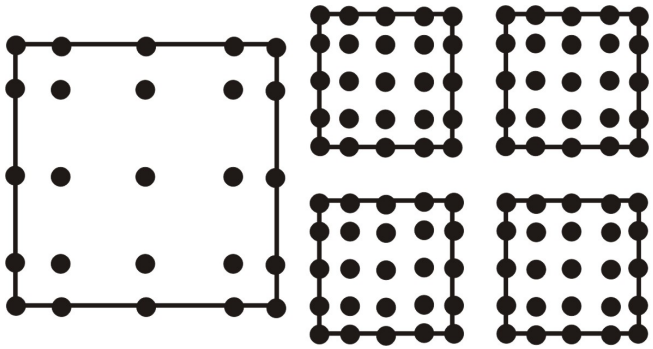


нерегулярная сетка

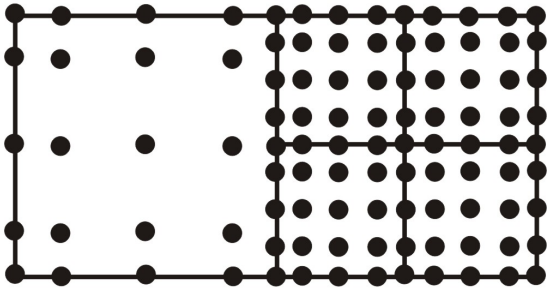
Пространственная дискретизация

Spectral Element Method – SEM:

- Каждый элемент сетки покрывается декартовым произведением $(N+1)^3$ точек коллокации, где N=4



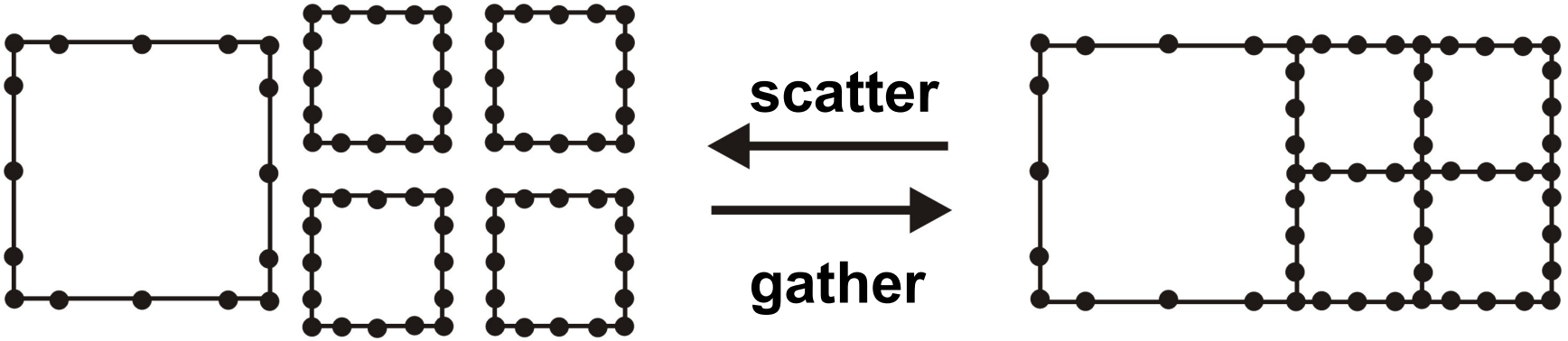
Точки коллокации



Точки сетки

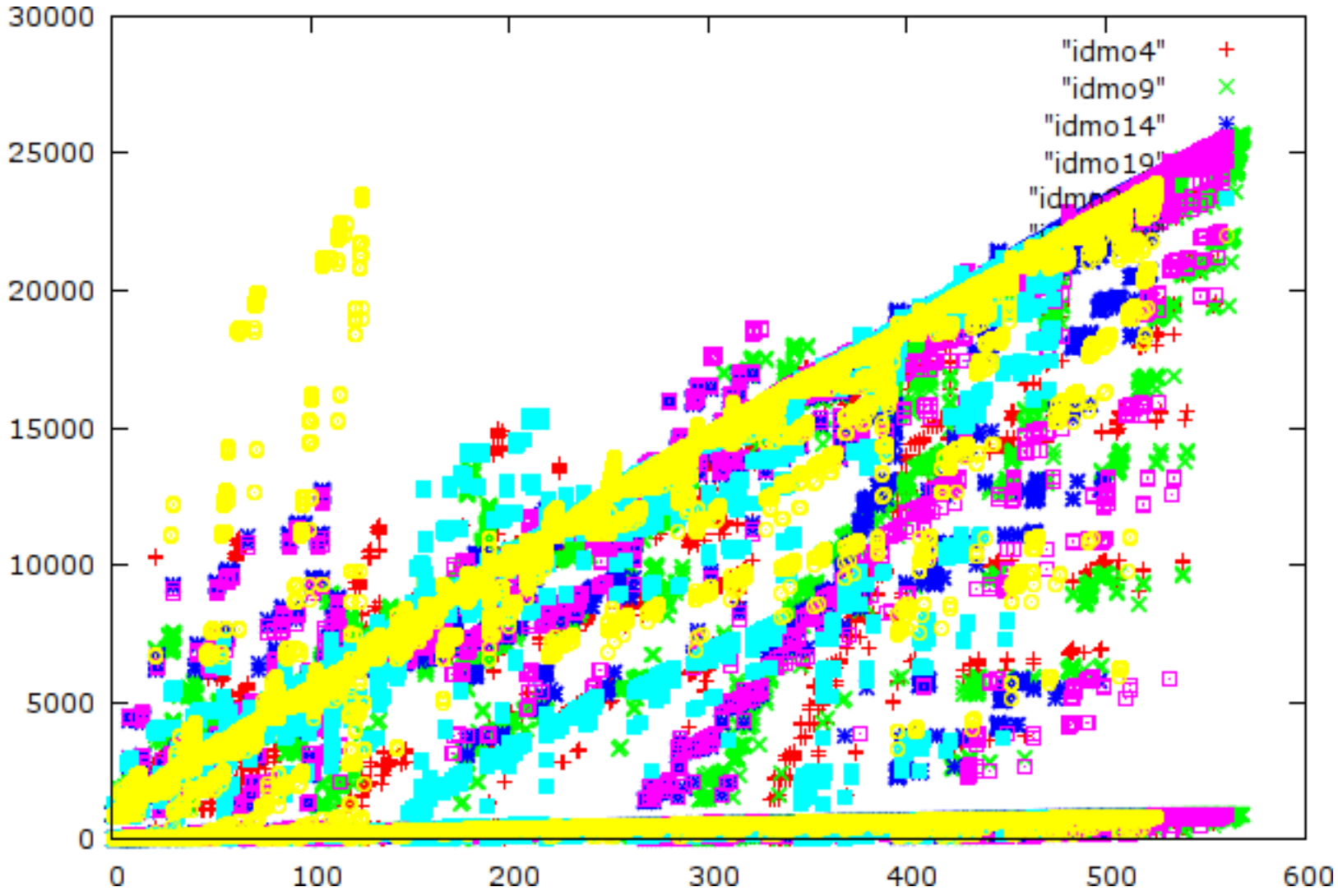
- Положение точек коллокации внутри элемента соответствует квадратурной формуле Гаусса-Лобатто-Лежандра

Согласование решения



- Метод Mortar Element Method (MEM) основывается на введении пространства согласования, состоящее из точек сетки, на границе элементов

Шаблон доступа в операциях Scatter/Gather



В итоге: факты об NPB UA

- Нерегулярный шаблон доступа к памяти
- Динамически изменяемая сетка

- Нет MPI версии
- OMP версия так себе масштабируется
- На кластере не запустить вообще
- 8000 строк кода на Fortran77
- Класс D не везде запустить (> 8 GB)

Что было сделано?

- OMP версия трансформирована в SHMEM+OMP
  - Данные вручную распределены по узлам
  - Итерации OMP циклов распределены по узлам
  - Зависимости по данным заменены на вызовы SHMEM
  - Инкрементальное распараллеливание (часть кода все еще последовательная)

- SHMEM реализован на Blue Gene/P
  - Через библиотеку MPI-2
  - Через DCMF (низкоуровенная комм. библиотека BG/P)
  - Добавлены active messages в SHMEM (DCMF только)

Распределяем массивы блочно

- Циклы полностью параллельные по данным

```
do iel=1,nelt
 call laplacian(tx(..,..,..,iel)
end do
```

трансформируем в

```
do iel=startelt,endelt
 call laplacian(tx(..,..,..,iel)
end do
```

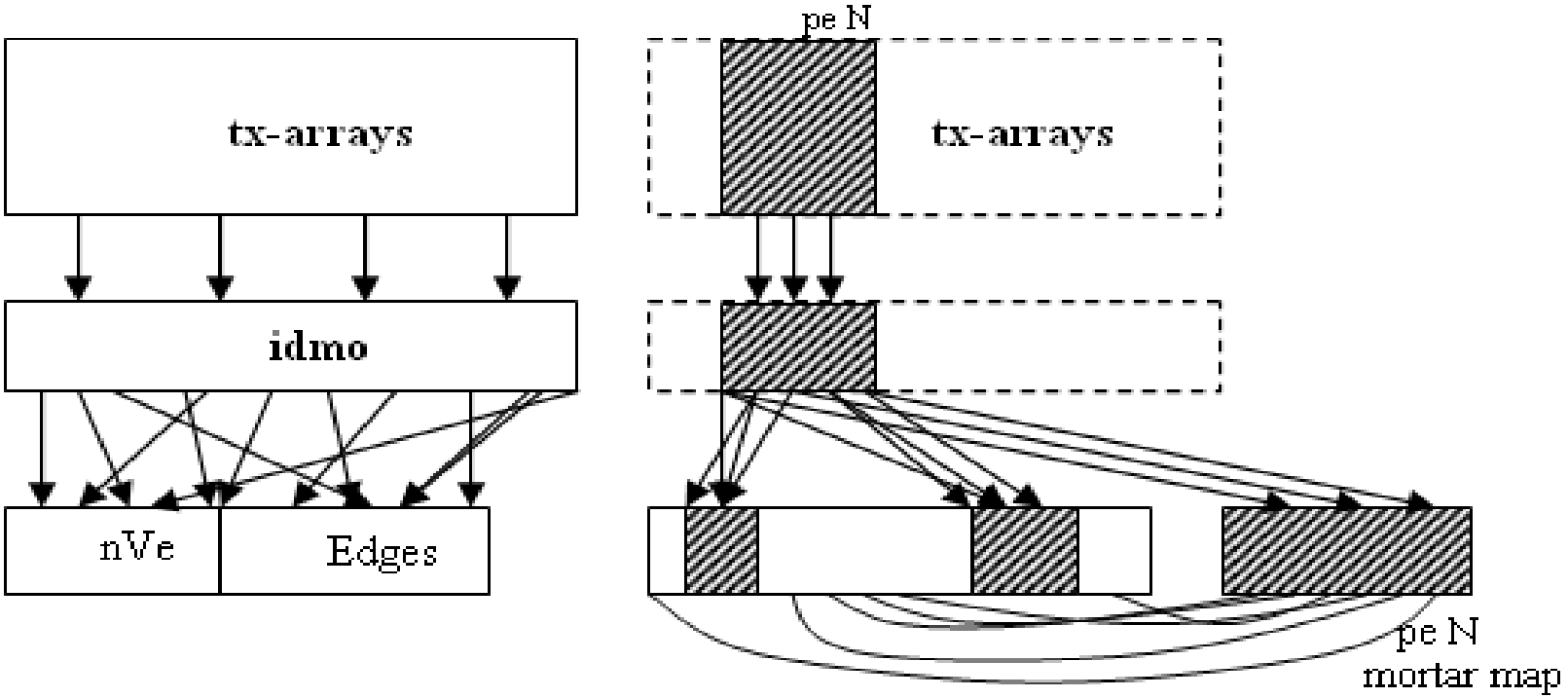
Не все OMP циклы такие…

- Чтения только локальных элементов, записи могут быть удаленные, между итерациями нет зависимостей
- Есть небольшое число нелокальных чтений (может быть большой блок – граница)
- Большое число удаленных чтений

Gather

```
c$OMP PARALLEL DO
do iel=1,nelt
    do j=1,25
     tx (j,iel) =a(j,1)*tmor(idmo(f(j,1),iel))
&                 +a(j,2)*tmor(idmo(g(j,1),iel))+...
    end do
end do
c$OMP END PARALLEL DO
```

Модификация индекс-вектора



serial

distributed

Распределение не только блочное

- Циклическое
- Блочно-циклическое итп

- Любое заданное пользователем

Задать функции

makedistribution
islocal_elt
pe_elt
local_elt

Gather

```
call shmem_barrier_all
 do i = 1,mormap_size
    pe = pemor(mormap(i))
    call shmem_double_g (morloc(i),
&                        tmor(localmor(mormap(i))), pe)
 end do
call shmem_quiet
```

```
c$OMP PARALLEL DO
    do iel=startelt, endelt
      do j=1,25
        tx(j,iel) =  a(1,j) * readmor ( xidmo(f(j,1),iel) ) +
 &                  +b(2,j) * readmor ( xidmo(g(j,1),iel) )+ ….
      end do
    end do
c$OMP END PARALLEL DO
```

PUT вместо GET

```
do i=0,revmap_size
    dst = revmap_dst(i)
    src = revmap_src(i)
    pe = revmap_pe(i)
    call
   shmem_double_p(morloc(dst),tmor(src),pe)
end do
call shmem_barrier_all
```

revmap_* – обратная карта доступа

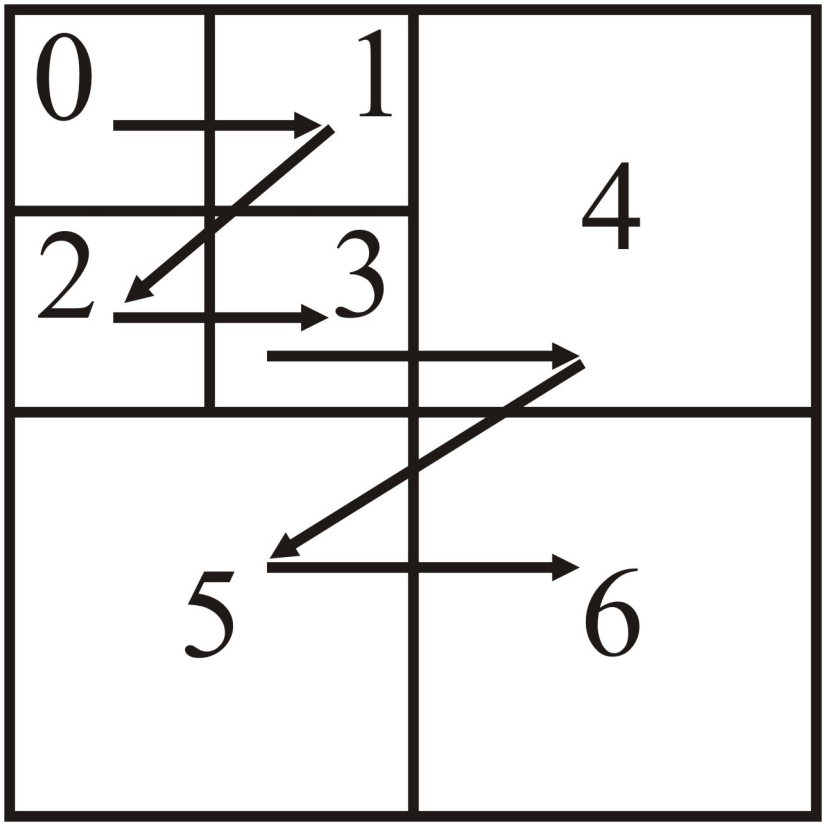
Трюки в стиле PGAS

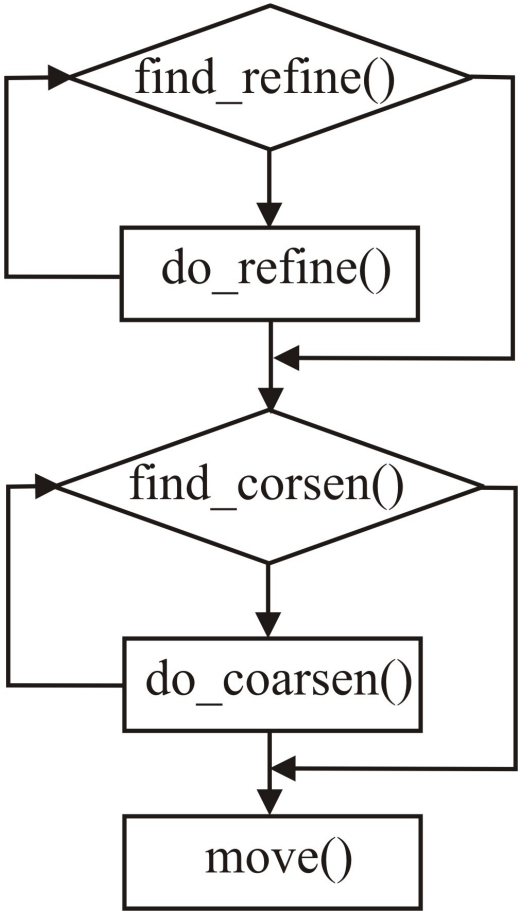
- Вставить цикл prefetch перед циклом с множеством удаленных запросов (DAE)
- Заменить GET на PUT

- Группировать атомарные локально
- Посылать вычисления к данным (active messages или RPC)

Особенности процедуры адаптации

- Нумерация (обход) по Мортону – Z--нумерация



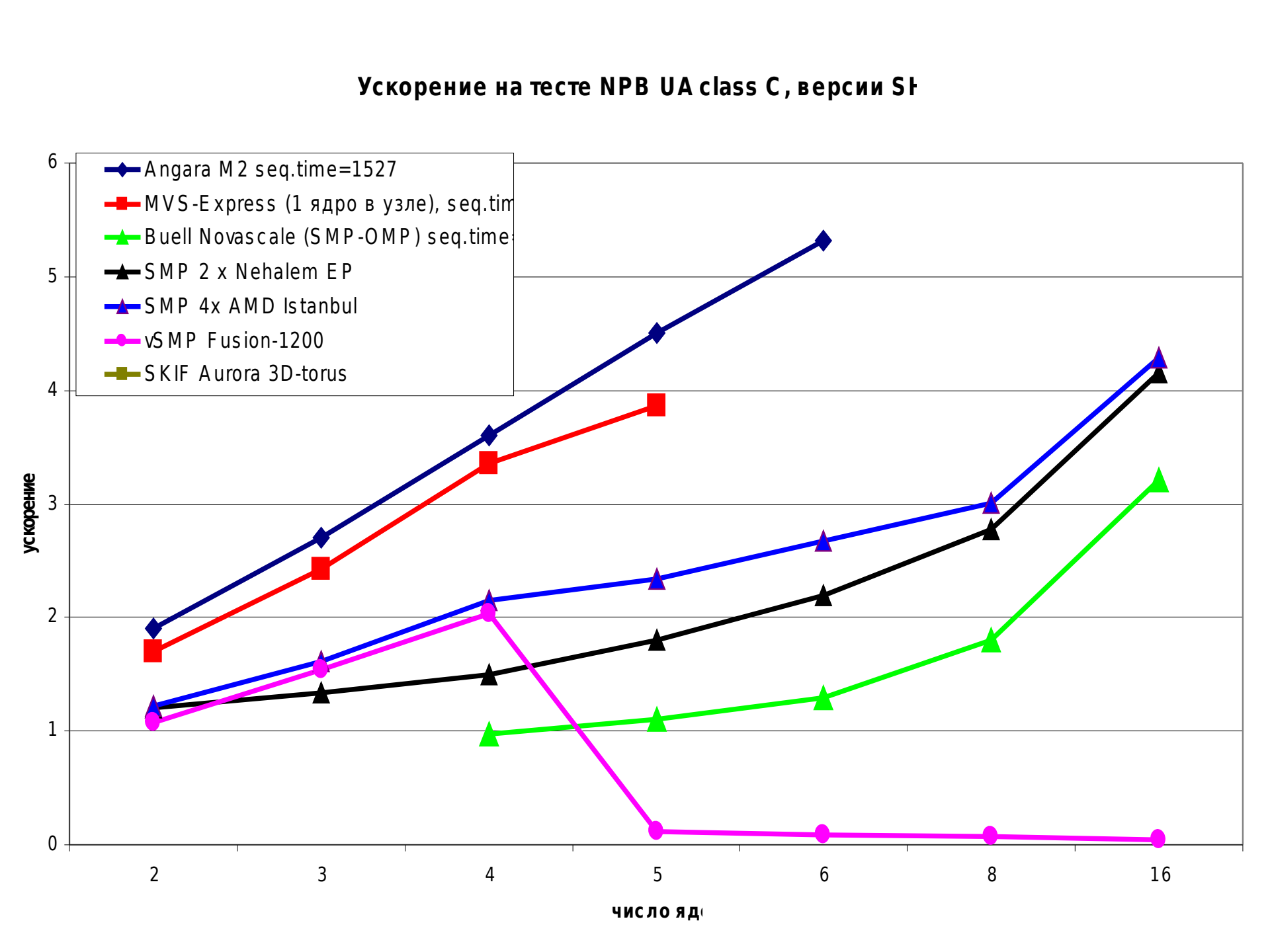


Инкрементальное распараллеливание

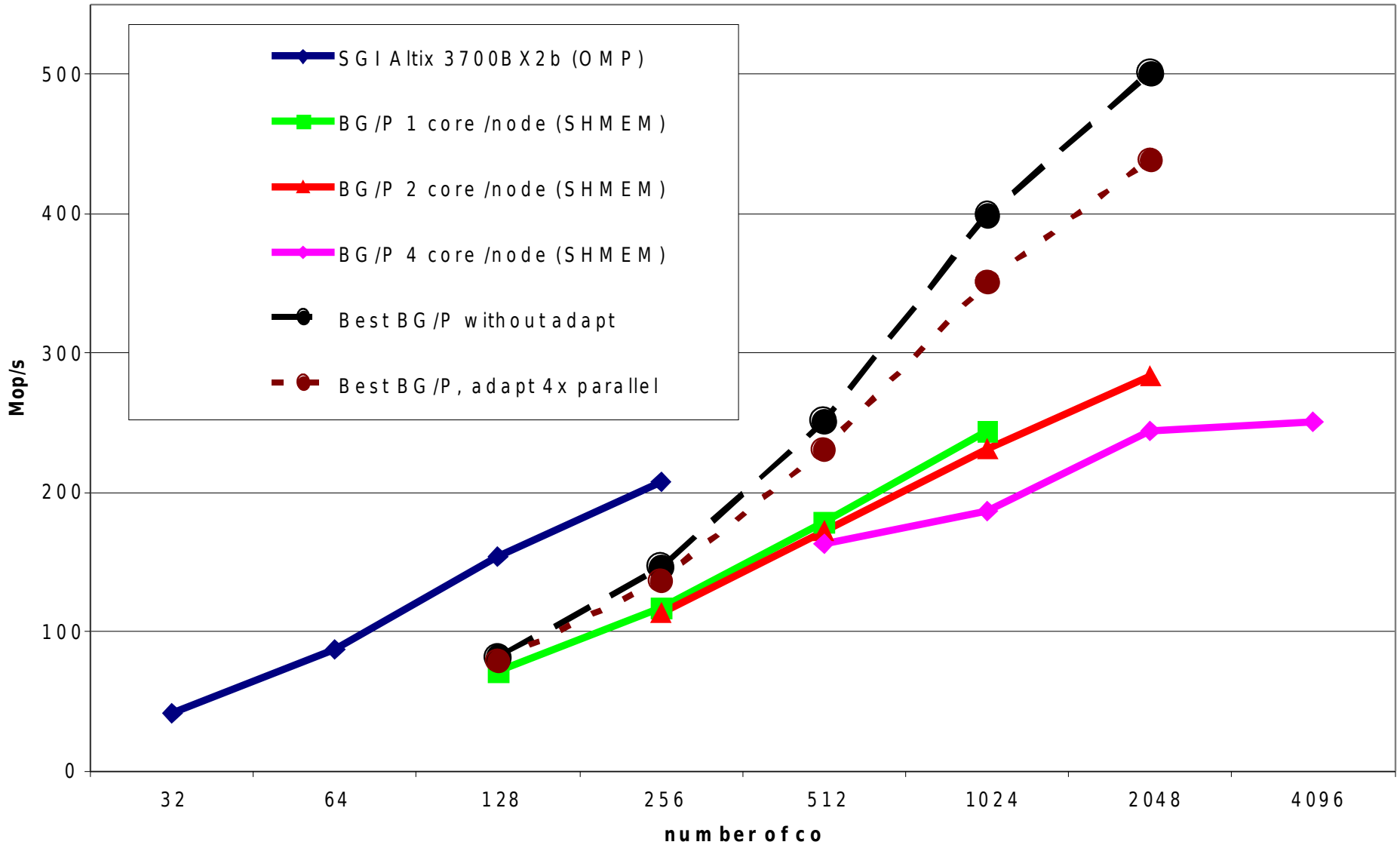
```
do step=0,nstep
c.........parallel code
      call convect_parallel
      call diffuse_parallel

      call shmem_allgatherv(t, t_big,starts,lens)
c.........serial code
      call adapt_serial
end do
```

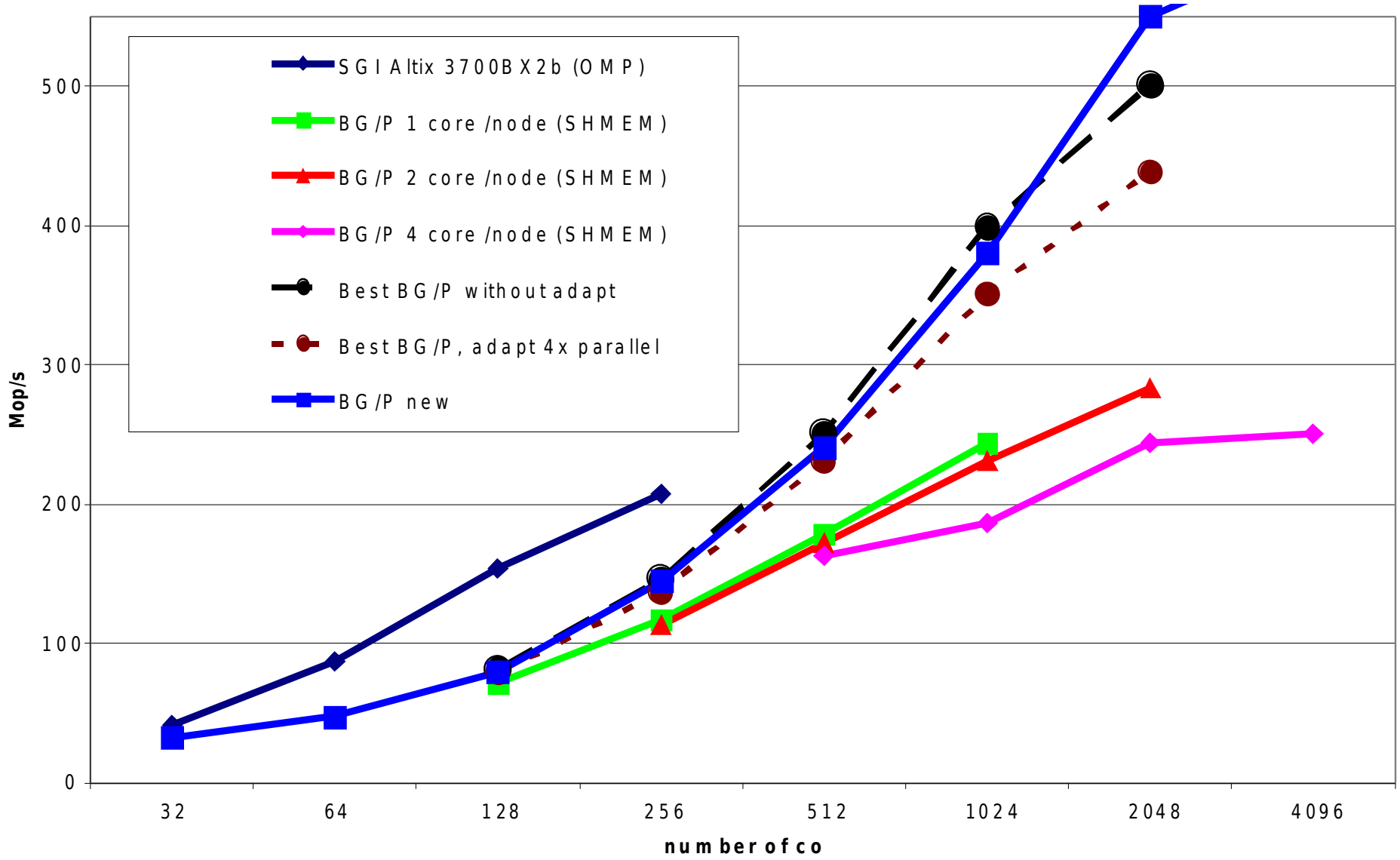
Результаты


Ускорение на тесте NPB UA class C, версии SH
Angara M2 seq.time=1527
MVS-Express (1 ядро в узле), seq.tim
Buell Novascale (SMP-OMP) seq.time
SMP 2 x Nehalem EP
SMP 4x AMD Istanbul
vSMP Fusion-1200
SKIF Aurora 3D-torus
ускорение
0
1
2
3
4
5
6
2
3
4
5
6
8
16
число яд

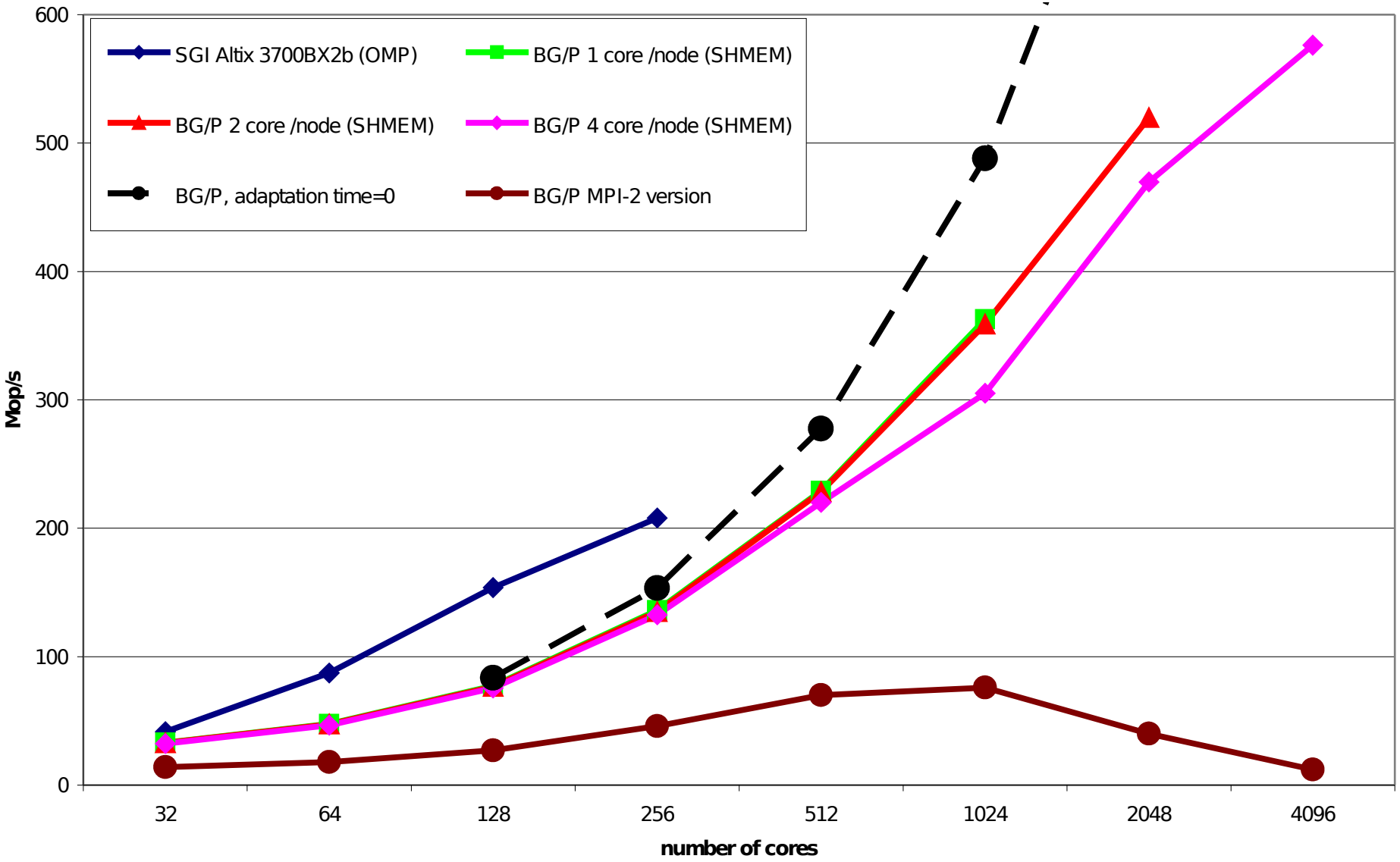
Если изменение сетки последовательно



Если изменение сетки частично распараллелено



Сравнение с MPI-2



Производительность на Классе C, millions coll. points advanced /s

	1	32	64	128	256	512	1K	2K	4K	8K
Altix* Itanium	5.9	42	87	154	207					
BG/P DCMF	2.5	33	48	80	142	242	393	588	777	769
BG/P MPI-2	2.5	14	18	27	46	70	76	40	12	3

* Благодарность за данные по Altix Jin. Haoqiang из NASA NAS Подразделения

Анализ масштабирования

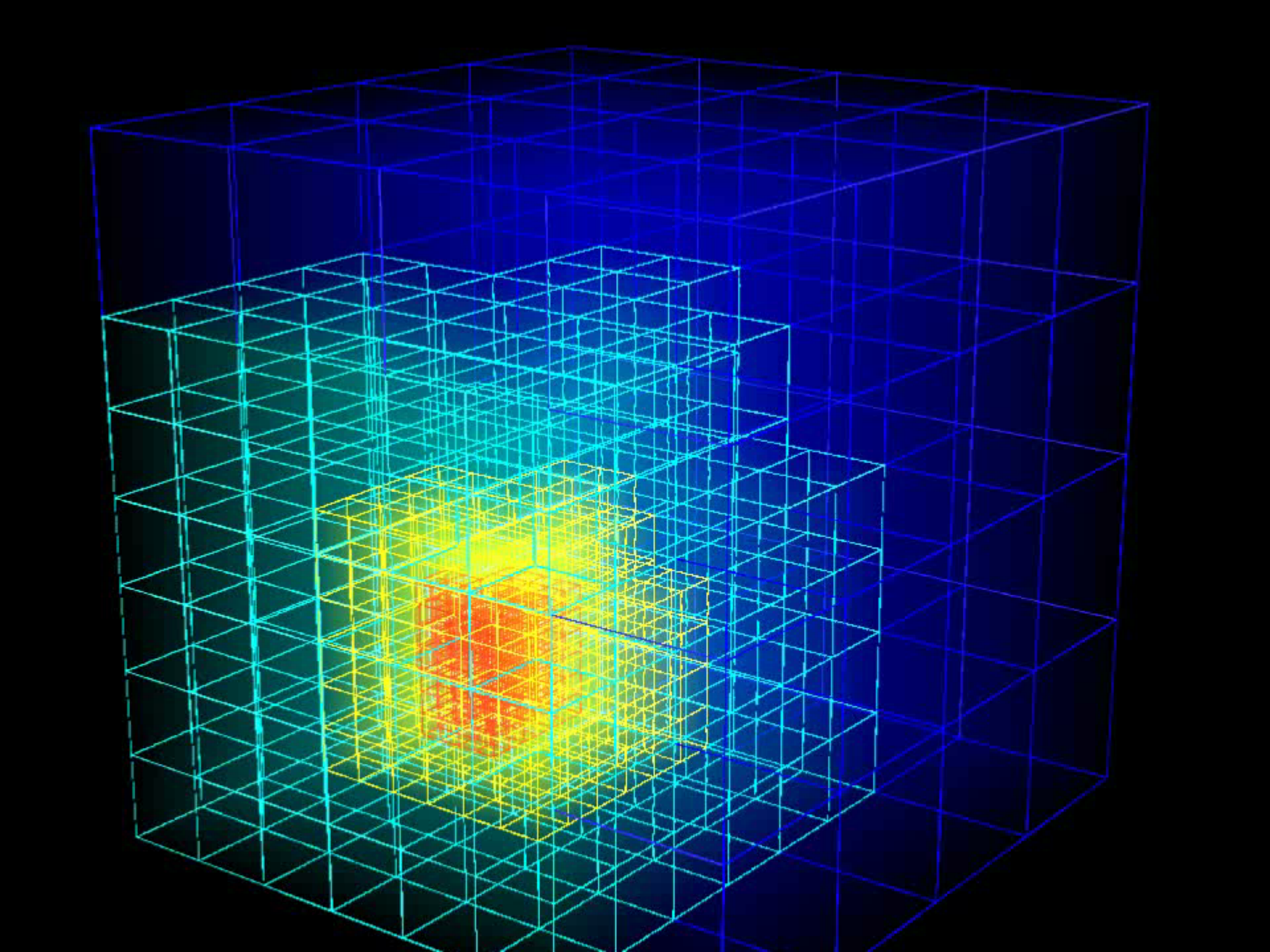
	serial	1	2	4	8	16	32	64	128	256	512	1 K	2 K*	4 K*
Total time, seconds	**3480**	**3784**	**2440**	**1200**	**700**	**410**	**266**	**185**	**113**	**65**	**38**	**24**	**17**	**14**
Mesh Adaptation + Overhead, %	0.4	2	2	3	3	4	4	5	7	11	16	25	35	43
Scatter time, %	12	14	18	20	22	26	30	34	36	37	35	33	29	21
Gather time, %	9	15	19	21	24	27	31	34	34	32	31	27	24	28
Rest computations time, %	78	69	61	56	51	43	35	27	23	20	18	15	13	8
Parallel efficiency, %	100	92	71	73	62	53	41	29	31	21	18	14	11	6

Результаты для Class D

- 64 ядер Novascale (Itanium 1.6) : **32** MOp/s
  нет false scaling как на C классе

	64	128	256	512	1K	2K	4K	8K*
BG/P SHMEM DCMF	83	126	190	345	620	1160	1880	2204

* Запущено в VN режиме (4 ядра на узел). DUAL и SMP режимы дадут лучшие результаты



Задание

- Переписать cpi-rma.c на SHMEM
  - Один в один
  - С помощью коллективных вызовов
- Пример программы на SHMEM
  - /gpfs/data/anton/shmemtest.cpp
  - /gpfs/data/anton/shmemcxx –o test shmemtest.cpp