Stochastic Allocation and Scheduling for Conditional Task Graphs in MPSoCs

Michele Lombardi and Michela Milano

(1) DEIS, University of Bologna
V.le Risorgimento 2, 40136, Bologna, Italy

Abstract. This paper describes a complete and efficient solution to the stochastic allocation and scheduling for Multi-Processor System-on-Chip (MPSoC). Given a conditional task graph characterizing a target application and a target architecture with alternative memory and computation resources, we compute an allocation and schedule minimizing the expected value of communication cost, being the communication resources one of the major bottlenecks in modern MPSoCs. Our approach is based on the Logic Based Benders decomposition where the stochastic allocation is solved through an Integer Programming solver, while the scheduling problem with conditional activities is faced with Constraint Programming. The two solvers interact through no-goods. The original contributions of the approach appear both in the allocation and in the scheduling part. For the first, we propose an exact analytic formulation of the stochastic objective function based on the task graph analysis, while for the scheduling part we extend the timetable constraint for conditional activities. Experimental results show the effectiveness of the approach.

1 Introduction

The increasing levels of system integration in Multi-Processor Systems on Chips (MPSoCs) emphasize the need for new design flows for efficient mapping of multi-task applications onto hardware platforms. The problem of allocating and scheduling conditional, precedence-constrained tasks on processors in a distributed realtime system is NP-hard. As such, it has been traditionally tackled by means of heuristics, which provide only approximate or near-optimal solutions, see for example [1], [2], [3].

In a typical embedded system design scenario, the platform always runs the same application. Thus, extensive analysis and optimization can be performed at design time. This paper proposes a complete decomposition approach to the allocation and scheduling of a conditional multi-task application on a multi-processor system-on-chip (MPSoCs) [4]. The target application is pre-characterized and abstracted as a Conditional Task Graph (CTG). The task graph is annotated with computation (e.g., execution time), communication (e.g., number of bits to be communicated between tasks), storage (e.g., size of data and instruction memory required to execute the task) requirements. However, not all tasks will run on the target platform: in fact, the application contains conditional branches (like if-then-else control structures). Therefore, after an accurate application profiling step, we have a probability distribution on each conditional branch that intuitively gives the probability of choosing that branch during execution.

This paper proposes a non trivial extension to [5] that used Logic Based Benders decomposition [6] for resource assignment and scheduling in MPSoCs. In that paper, however, task graphs did not contain conditional activities. Allocation and scheduling were therefore deterministic. The introduction of stochastic elements complicates the problem.

We propose two main contributions: the first concerns the allocation component. The objective function we consider depends on the allocation variables. Clearly, having conditional tasks, the exact value of the communication cost cannot be computed. Therefore our objective function is the expected value of the communication cost. We propose here to identify an analytic approximation of this value. The approximation is based on the Conditional Task Graph analysis for identifying two data structures: the activation set of a node and the coexistence set of two nodes. The approximation turns out to be exact and polynomial.

The second contribution concerns scheduling. We propose an extension of the timetable constraint for cumulative resources, taking into account conditional activities. Its deterministic version [7] is available in ILOG Scheduler. The use of the so called *optional activities* (what we call conditional tasks) has been taken into account in [8] for filtering purposes into the precedence graph, originally introduced by Laborie in [9]. To the best of our knowledge, only disjunctive constraints have been defined in presence of conditional activities in [10].

In the system design community, this problem is extremely important and many researchers have worked extensively on it, mainly with incomplete approaches: for instance in [1] a genetic algorithm is devised on the basis of a conditional scheduling table whose (exponential number of) columns represent the combination of conditions in the CTG and whose rows are the starting times of activities that appear in the scenario. The number of columns is indeed reasonable in real applications. The same structure is used in [10], which is the only approach that uses Constraint Programming for modelling the allocation and scheduling problem. Indeed the solving algorithm used is complete only for small task graphs (up to 10 activities).

Besides related literature for similar problems, the Operations Research community has extensively studied stochastic optimization in general. The main approaches are: sampling [11] consisting in approximating the expected value with its average value over a given sample; the *l-shaped* method [12] which faces two phase problems and is based on Benders Decomposition [13]. The master problem is a deterministic problem for computing the first phase decision variables. The subproblem is a stochastic problem that assigns the second phase decision variables minimizing the average value of the objective function. A different method is based on the branch and bound extended for dealing with stochastic variables, [14].

The CP community has recently faced stochastic problems: in [15] stochastic constraint programming is formally introduced and the concept of solution is replaced with the one of *policy*. In the same paper, two algorithms have been proposed based on backtrack search. This work has been extended in [16] where an algorithm based on the concept of scenarios is proposed. In particular, the paper shows how to reduce the number of scenarios, maintaining a good expressiveness.

This paper is organized as follows: in section 2 we present the architecture and the target application considered. In section 3 we present the allocation and scheduling models used. Experimental results are shown in section 4.

2 Problem description

2.1 The architecture

Multi Processor Systems on Chips (MPSoCs) are multi core architectures developed on a single chip. They are finding widespread application in embedded systems (such as cellular phones, automotive control engines, etc.). Once deployed in field, these devices always run the same application, in a well-characterized context. It is therefore possible to spend a large amount of time for finding an optimal allocation and scheduling off-line and then deploy it on the field, instead of using on-line, dynamic (sub-optimal) schedulers [17, 18].

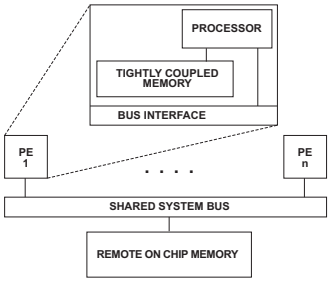

Fig. 1. Single chip multi-processor architecture.

The multi-processor system we consider consists of a pre-defined number of distributed Processing Elements (PE) as depicted in Figure 1. All nodes are assumed to be homogeneous and composed by a processing core and by a low-access-cost local scratchpad memory. Data storage onto the scratchpad memory is directly managed by the application, and not automatically in hardware as it is the case for processor caches.

The scratchpad memory is of limited size, therefore data in excess must be stored externally in a remote on-chip memory, accessible via the bus. The bus for state-of-the-art MPSoCs is a shared communication resource, and serialization of bus access requests of the processors (the bus masters) is carried out by a centralized arbitration mechanism. The bus is re-arbitrated on a transaction basis (e.g., after single read/write transfers, or bursts of accesses of pre-defined length), based on several policies (fixed priority, round-robin, latency-driven, etc.). Modeling bus allocation at such a fine granularity would make the problem overly complex, therefore a more abstract additive bus model was devised, explained and validated in [5] where each task can simultaneously access the bus requiring a portion of the overall bandwidth. The communication resource in most cases ends up to be the most congested resource. Communication cost

is therefore critical for determining overall system performance, and will be minimized in our task allocation framework.

2.2 The target application

The target application to be executed on top of the hardware platform is the input to our algorithm. It is represented as a Conditional Task Graph (CTG). A CTG is a triple $\langle T, E, C \rangle$, where T is the set of nodes modeling generic tasks (e.g. elementary operations, subprograms, ...), E the set of edges modeling precedence constraints (e.g. due to data communication), and C is a set of conditions, each one associated to an arc, modeling what should be true in order to choose that branch during execution (e.g. the condition of a if-then-else construct). A node with more than one outgoing arc is said to be a *branch* if all arcs are conditional, a *fork* if all arcs are not conditional; mixed nodes are not allowed. A node with more than one ingoing arc is an *or-node* if all arcs are mutually exclusive, it is instead an *and-node* if all arcs are not mutually exclusive; again, mixed nodes are not allowed.

Since the truth or the falsity of conditions is not known in advance, the model is stochastic. In particular, we can associate to each branch a stochastic variable $\mathcal{B}$ with probability space $\langle C, \mathcal{A}, p \rangle$, where C is the set of possible branch exit conditions c, $\mathcal{A}$ the set of events (one for each condition) and p the branch probability distribution (in particular $p(c)$ is the probability that condition c is true).

We can associate to each node and arc an activation function, expressed as a composition of conditions by means of the logical operators $\wedge$ and $\vee$. We call it $f_i(X(\omega))$, where X is the stochastic variable associated to the composite experiment $\mathcal{B}_0 \times \mathcal{B}_1 \times ... \times \mathcal{B}_b$ ($b =$ number of branches) and $\omega \in \mathcal{D}(\mathcal{B}_0) \times \mathcal{D}(\mathcal{B}_1) \times ... \times \mathcal{D}(\mathcal{B}_b)$ (i.e. ω is a scenario).

Computation, storage and communication requirements are annotated onto the graph. In detail, the worst case execution time (WCET) is specified for each node/task and plays a critical role whenever application real time constraints (expressed here in terms of deadlines) are to be met.

Each node/task also has three kinds of associated memory requirements:

- Program Data: storage locations are required for computation data and for processor instructions.
- Internal State
- Communication queues: the task needs queues to transmit and receive messages to/from other tasks, eventually mapped on different processors.

Each of these memory requirement can be allocated either locally in the scratchpad memory or remotely in the on-chip memory.

Finally, the communication to be minimized counts two contributions: one related to single tasks, once computation data and internal state are physically allocated to the scratchpad or remote memory, and obviously depending on the size of such data; the second related to pairs of communicating tasks in the task graph, depending on the amount of data the two tasks should exchange.

3 Model definition

As already presented and motivated in [5], the problem we are facing can be split into the resource allocation master problem and the scheduling sub-problem.

3.1 Allocation problem model

With regards to the platform described in section 2.1, the allocation problem can be stated as the one of assigning processing elements to tasks and storage devices to their memory requirements.

Suppose n is the number of tasks, p the number of processors, and m the number of arcs. We introduce for each task and each PE a variable T_{ij} such that $T_{ij} = 1$ iff task i is assigned to processor j. We also define variables M_{ij} such that $M_{ij} = 1$ iff task i allocates its computation memory locally, $M_{ij} = 0$ otherwise. Similarly we introduce variables S_{ij} for task i state requirements and E_{rj} for arc r communication queue. X is the stochastic variable associated to the scenario ω.

The allocation model, where the objective function is the minimization of bus traffic expected value, is defined as follows:

$$\min z = E(\textit{busTraffic}(M, S, E, X(\omega)))$$

$$s.t. \quad \sum_{j=0}^{p-1} T_{ij} = 1 \qquad \forall i = 0, .., n \tag{1}$$

$$S_{ij} \leq T_{ij} \qquad \forall i = 0, .., n-1, j = 0, .., p-1 \tag{2}$$

$$M_{ij} \leq T_{ij} \qquad \forall i = 0, .., n-1, j = 0, .., p-1 \tag{3}$$

$$E_{rj} \leq T_{ij} \qquad \forall e_r = (t_i, t_k), r = 0, .., m \tag{4}$$

$$E_{rj} \leq T_{kj} \qquad \forall e_r = (t_i, t_k), r = 0, .., m \tag{5}$$

$$\sum_i [s_i S_{ij} + m_i M_{ij}] + \sum_r c_r E_{rj} \leq C_j \qquad \forall j = 0, .., p-1 \tag{6}$$

Constraints (1) force each task to be assigned to a single processor. Constraints (2) and (3) state that computation and state memory can be locally allocated on the PE j only if task i runs on it. Constraints (4) and (5) enforce that the communication queue of arc r can be locally allocated only if both the source and the destination tasks run on processor j. Finally, constraints (6) ensure that the sum of locally allocated state (s_i), computation (m_i) and communication (c_r) memory cannot exceed the scratchpad device capacity (C_j). All tasks have to be considered here, regardless they will execute or not at runtime, since a scratchpad memory is, by definition, statically allocated. In addition, some symmetries breaking constraints have been added to the model.

The bus traffic expression is composed by two contributions: one depending on single tasks and one due to the communication between pairs of tasks.

$\textit{busTraffic} = \sum_{i=0}^{n-1} \textit{taskBusTraffic}_i + \sum_{e_r=(t_i,t_k)} \textit{commBusTraffic}_r$
where
$\textit{taskBusTraffic}_i = \sum_{i=0}^{n-1} f_i(X(\omega)) \left[\overline{m}_i(1 - \sum_{j=0}^{p-1} M_{ij}) + \overline{s}_i(1 - \sum_{j=0}^{p-1} S_{ij})\right]$
$\textit{commBusTraffic}_r = f_i(X(\omega)) f_k(X(\omega)) \overline{c}_r (1 - \sum_{j=0}^{p-1} E_{rj})$

where $\overline{m}_i$ and $\overline{s}_i$, are the bus traffic contributions due to task i program data and internal state, and $\overline{c_r}$ is the traffic due to communication through arc r.

Note that, in the "task bus traffic" expression, if task i executes (thus $f_i(X(\omega)) = 1$), then $1 - \sum_{j=0}^{p-1} M_{ij}$ is 1 iff task i computation memory is allocated on the remote memory and the same holds for the state. Traffic contributions due to communications have to be considered if both the source and the destination task execute ($f_i(X(\omega)) = f_k(X(\omega)) = 1$) and the queue is remotely allocated ($1 - \sum_{j=0}^{p-1} E_{rj} = 1$).

In most cases, the minimization of a stochastic functional, such as the expected value, is a very complex operation (even more than exponential), since it often requires to repeatedly solve a deterministic subproblem [12]. The cost of such a procedure is not affordable for hardware design purposes if the deterministic subproblem is by itself NP-hard, which is our case.

One of the main contributions of this paper is the way to reduce the bus traffic expected value to a deterministic expression.

Since all task have to be assigned before running the application, the allocation is a stochastic *one phase* problem: thus, for a given task-PE assignment, the expected value depends only on the stochastic variables.

Intuitively, if we properly weight the bus traffic contributions according to task probabilities we should be able to get an analytic expression for the expected value.

Now, since both the expected value operator and the bus traffic expression are linear, the objective function can be decomposed into task related and arc related blocks:

$$E(\textit{busTraffic}) = \sum_{i=0}^{n-1} E(\textit{taskBusTraffic}_i) + \sum_{e_r=(t_i,t_k)} E(\textit{commBusTraffic}_r)$$

Since for a given allocation the objective function depends only on the stochastic variables, the contributions of decision variables are constants: we call them $KT_i = \overline{m}_i(1 - \sum_{j=0}^{p-1} M_{ij}) + \overline{s}_i(1 - \sum_{j=0}^{p-1} S_{ij})$, $KE_r = \overline{c}_r(1 - \sum_{j=0}^{p-1} E_{rj})$. Let us call $p(\omega)$ the probability of scenario ω.

So

$$E(\textit{taskBusTraffic}_i) = \sum_{\omega \in \Omega} p(\omega) f_i(X(\omega)) KT_i$$

and

$$E(\textit{commBusTraffic}_r) = \sum_{\omega \in \Omega} p(\omega) f_i(X(\omega)) f_k(X(\omega)) KE_r$$

Substituting

$$\sum_{\omega \in \Omega} p(\omega) f_i(X(\omega)) KT_i \leftrightarrow KT_i \sum_{\omega \in \Omega_i^1} p(\omega)$$

and

$$\sum_{\omega \in \Omega} p(\omega) f_i(X(\omega)) f_k(X(\omega)) KE_r \leftrightarrow KE_r \sum_{\omega \in \Omega_i^1 \cap \Omega_k^1} p(\omega)$$

with $\Omega_i^1 = \{\omega \mid \text{task } i \text{ executes}\}$, every stochastic dependence is removed and the expected value is reduced to a deterministic expression.

Note that $\sum_{\omega \in \Omega_i^1} p(\omega)$ is simply the existence probability of node/task i while $\sum_{\omega \in \Omega_i^1 \cap \Omega_k^1} p(\omega)$ is the coexistence probability of nodes i and k.

To apply the transformation we need both those probabilities; moreover, to achieve an effective overall complexity reduction, they have to be computed in a reasonable time. We developed a set of polynomial cost algorithms to compute those probabilities.

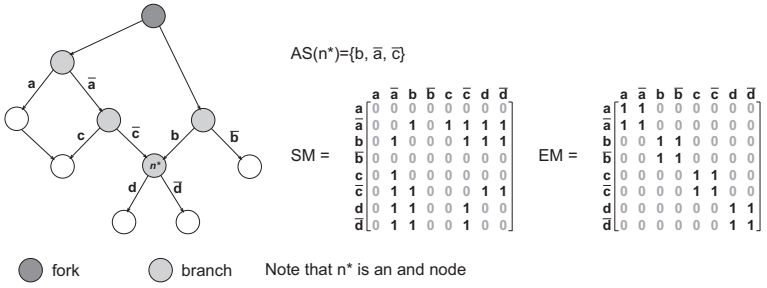

Fig. 2. An example of the three data structures

All developed algorithms are based on three types of data structure referred in Figure 2 to the CTG on the left: the first is said *activation set* of a node n ($AS(n)$), or of an arc e ($AS(e)$) and it is the set of all upstream conditions on the paths from the starting node to the specific node n or arc e; we introduced also a $c \times c$ (c is the number of conditions) binary *exclusion matrix (EM)* such that $EM_{ij} = 1$ iff c_i and c_j are mutually exclusive (i.e. they originate at the same branch), and a $c \times c$ *sequence matrix (SM)* such that $SM_{ij} = 1$ iff c_i and c_j are both needed to activate some node or arc in the CTG.

algorithm: Activation set probability (A1) – probability of a node or an arc

1. let S be the input set for this iteration; initially $S = AS(n)$
2. find a condition $c_h \in S$ such that $(EM_h \setminus c_h) \cap S \neq \emptyset$
3. if such a condition doesn't exist return $p = \prod_{c \in S} p(c)$
4. otherwise, set $B = EM_h \cap S$ (branch conditions)
5. compute set $C = S \cap \bigcap_{c_i \in B} SM_i$ (common conditions)
6. compute set $R = \bigcap_{c_i \in B} (S \setminus SM_i)$ (rest)
7. set $p = 0$
8. for each condition $c_i \in B$:
8.1. set $p = p + A1((S \cap SM_i) \setminus (C \cup R))$
9. set $p = p * A1(C) * A1(R)$
10. return p

end

All these data structures can be extracted from the graph in polynomial time. Once they are available, we can determine the probability of a node or an arc using algorithm A1, which has $O(c^3)$ complexity representing sets as bit vectors; in the algorithm the notation SM_i stands for the set of conditions "sequenced" with a given one ($SM_i = \{c_j | SM_{ij} = 1\}$); the same holds for EM_i.

Algorithm A1 works recursively partitioning the activation set of the target node or arc: first it looks for a group of mutually exclusive conditions (the B set), see `b` and `not b` in figure 3. If there is no such condition the probability of set S is the product of the ones of its elements (step 3). The algorithm then builds a "common" (C) and a "rest" (R) set: the first contains conditions c_j such that $SM_{ij} = 1 \quad \forall c_i \in B$, the second conditions c_h such that $SM_{ih} = 0 \quad \forall c_i \in B$. Finally A1 builds for each found branch condition a set containing the sequenced conditions ($S \cap SM_i$ at step 8.1), chains `b` and `not c` and `not b` and `d` in figure 3.

A1 is then recursively called on all these sets. The probabilities of sets corresponding to mutually exclusive conditions are summed (step 8.1), the ones of C and R are multiplied (step 9).

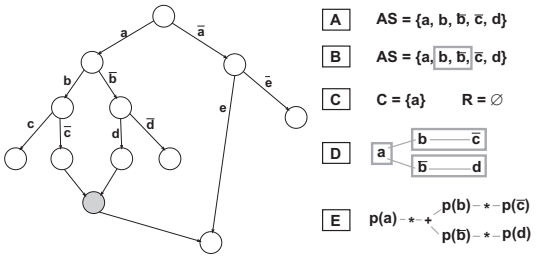

Fig. 3. Activation Set Probability (A1)

---

algorithm: Coexistence set determination (A2)

---

1. if $AS_i = \emptyset$ then $CS = AS_i$; the same if $AS_j = \emptyset$
2. otherwise, if there are still not processed conditions in AS_i, let c_h be the first of them:
  2.1. compute set $S = AS_i \cap SM_h$ (sequenced conditions)
  2.2. compute the exclusion set $EX(S)$
  2.3. compute set $C = AS_j \cap \bigcup_{c_k \in AS_j \cap EX(S)} SM_k$ (conditions candidates for clear operation)
  2.4. compute set $R = AS_j \cap \bigcup_{c_k \in AS_j \setminus C} SM_k$ (conditions to spare)
  2.5. set $D = C \setminus R$ (conditions to clear)
  2.6. if AS_j is not a subset of D:
    2.6.1. set $CS(AS_i, AS_j) = CS(AS_i, AS_j) \cup S \cup (AS_j \setminus D)$

end

Given a set of nodes or arcs, we can determine a kind of common activation set (*coexistence set (CS)*) using algorithm A2, whose inputs are two AS (AS_i, AS_j) and whose complexity is again $O(c^3)$:

The notation $EX(S)$ stands for the exclusion set, i.e. the set of conditions surely excluded by those in S; it can be computed in $O(c^2)$.

Suppose we have the activation sets of two nodes n_i and n_j: then A2 works trying to find all paths from n_i to a source (backward paths) and from the source to n_j (forward paths). The algorithm starts building a group of backward paths; it does it by choosing

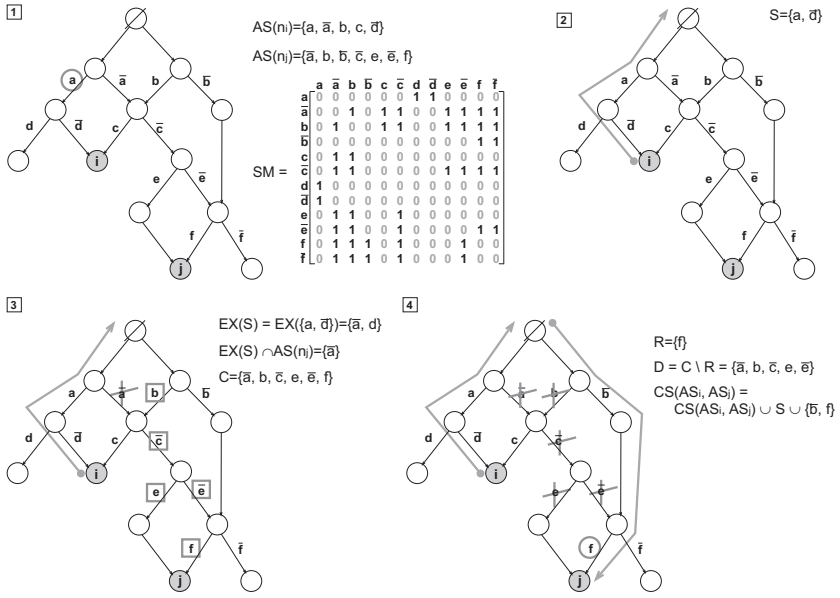

Fig. 4. Coexistence set computation

a condition (for instance condition `a` in 1 figure 4) and finding all other conditions sequenced with it (set S in 2 figure 4).

Then the algorithm finds the exclusion set ($EX(S)$) of set S and intersects it with $AS(n_j)$. In 3 figure 4 the only condition in the intersection is `not a` (crossed arc): conditions in the intersection and those sequenced with them are called "candidates conditions" (set C in 3 figure 4). These conditions will be removed from $AS(n_j)$, unless they are sequenced with one or more non-candidate conditions, i.e., they belong to the set R (for instance condition `f` is in sequence with `not b` and is not removed from $AS(n_j)$ in 4, figure 4). The conditions not removed from $AS(n_j)$ identify a set of forward paths we are interested in. The algorithm goes on until all conditions in $AS(n_i)$ are processed. If there is no path from n_i to n_j (i.e. the coexistence set is empty) the two nodes are mutually exclusive.

The probability of a coexistence set can be computed once again by means of A1: thus, with A1 and A2 we are able to compute the existence probability of a single node or arc and the coexistence probability of a group of nodes or arcs. Since the algorithms complexities are polynomial, the reduction of the bus traffic to a deterministic expression can be done in polynomial time.

3.2 Scheduling Model

The scheduling subproblem was solved by means of Constraint Programming. Since the objective function depends only on the allocation of tasks and memory requirements,

scheduling is just a feasibility problem. Therefore we decided to provide a unique schedule, dynamically considering for each task the longest ingoing path.

Tasks using the same resources can overlap if they are on alternative paths (under two mutually exclusive conditions): if we model the resources in order to take into account this behavior, we get an exact schedule for the worst task track, provided there are no ambiguous nodes.We say two nodes are ambiguous if they are mutually exclusive w.r.t some paths and not mutually exclusive w.r.t. other paths.

Tasks have a five phases behavior: they read all communication queues (INPUT), eventually read their state (RS), execute (EXEC), write their states (WS) and finally write all the communications queues (OUTPUT). Each task is modeled as a group of not breakable activities; the adopted schema and precedence relations vary with the type of the corresponding node (or/and, branch/fork): figure 5 summarizes all used rules. In the picture the black arrows represent immediate precedence relations ($end(A) = start(B)$), while the gray hyphened arrows are simple precedence relations ($end(A) \leq start(B)$).

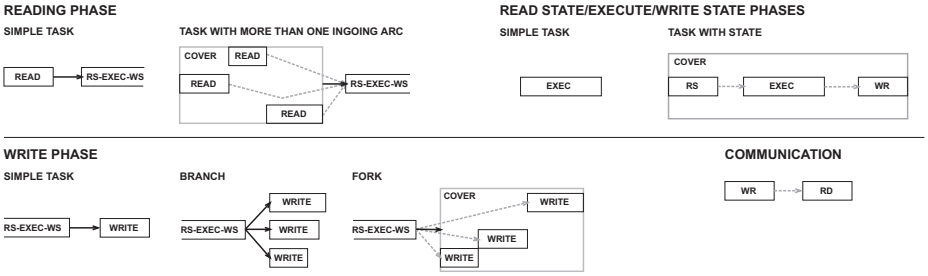

Fig. 5. Task decomposition schema

Each activity duration is an input parameter and can vary depending on the allocation of internal state and program data.

The processing elements are unary resources: we modeled them defining a simple disjunctive constraint proposed in [10].

The bus, as in [5], was modeled as a cumulative resource, according with the so called “additive model”, which allows an error less than 10% until bandwidth usage is under 60% of the real capacity.

Computing the bus usage in presence of alternative activities is not trivial, since the bus usage varies in a not linear way and every activity can have its own bus view (see fig 6).

We modeled the bus implementing a timetable like constraint for cumulative resources in the not preemptive case. The constraint keeps a list of all known entry and exit points of activities: given an activity A, if $lst(A) \leq eet(A)$ then the entry point of A is $lst(A)$ and $eet(A)$ is its exit point (where lst stands for latest start time and so on).

algorithm: Cumulative resource constraint with alternative activities (A3)

1. $time = est(a), finish = eet(a)$
2. $latestGoodTime = time$
3. $good = true$
4. While $\neg\,[(good = false \wedge time > lst(a)) \vee (good = true \wedge time >= finish)]$:
  4.1. if $busreq(a) + usedBandwith > busBandwidth$:
    4.1.1. $time =$ next exit point
    4.1.2. $good = false$
  4.2. else:
    4.2.1. $time =$ next entry point
    4.2.2. if $good = false$:
      4.2.2.1. $lastGoodTime = time$
      4.2.2.2. $finish = max(finish, time + mindur(a))$
      4.2.2.3. $good = true$
5. if $good = true$: $est(a) = lastGoodTime$
6. else: $fail$

end

Let A be the target activity: A3 scans the interval $[est(A), finish)$ checking the bus usage at all entry points (as long as $good = true$). If it finds an entry point with not enough bandwidth left it starts to scan all exit points ($good = false$) in order to determine a new possible starting time for activity A. If such an instant is found its value is stored ($lastGoodTime$) and the finish line is updated (step 4.2.2.2), then A3 restarts to scan other entry points, and so on. When the finish line is reached the algorithm updates $est(A)$ or fails. A3 has $O(a(c + b))$ complexity, where a is the number of activities, b the one of branches, c the number of conditions. The algorithm can be easily extended to update also $let(A)$: we tried to do it, but the added filtering is not enough to justify the increased propagation time.

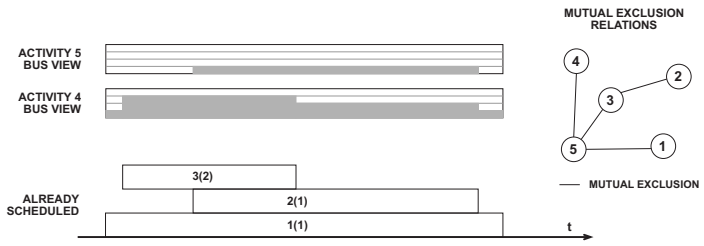

Fig. 6. Activity bus view; bus requirements of scheduled activities are between round brackets. For instance activity 4 is not exclusive with 1, 2 and 3, but 2 and 3 are mutually exclusive. Therefore, the bandwidth used by scheduled activities is the one of activity 1, then of 1 and 3 and when 3 finishes, the one of activities 1 and 2.

The main difficulty of having conditional activities, is that each activity has its own bus view depending on which activities are not exclusive with it. For instance in figure 6 activities 4 and 5 have a different view of the bus. A3 is able to compute the bandwidth usage seen from each activity in linear time by taking advantage of a particular data structure we introduced, named Branch Fork Graph (BFG). For lack of space, we give

only an intuition here. The BFG is like a skeleton of the relations between branch and fork nodes in the original graph: once this structure has been extracted, each activity can be mapped to one or more BFG nodes according to its activation set. For each activity we keep also the entry and exit point (continuously updated during search). The bandwidth usage at a given time can be computed by parsing the graph, which is a linear time operation: the activation set of the target activity is used to select which nodes of the BFG must be taken into account for the computation. The BFG makes it possible to compute bus usage in a very efficient way, by making direct use of the graph structure: if we only take into account the exclusion relations it would be an NP-hard problem.

To have a polynomial time algorithm however the graph should satisfy a particular condition (called "Control Flow Uniqueness") which states that each "and" node must have a main ingoing arc, whose activation implies the activation of the other ingoing arcs. This is not a very restrictive condition since it is satisfied by every graph resulting from the natural parsing of programs written in a language such C++ or Java.

Each activity in the presented schema needs a processing element and requires an amount of bus bandwidth if its data are remotely allocated. That amount is an input parameter and for communication and state activities is strongly greater than for execution ones.

3.3 Benders cuts and subproblem relaxation

Each time the master problem solution is not feasible for the scheduling subproblem a cut is generated which forbids that solution. Moreover, all solutions obtained by permutation of PEs are forbidden, too.

Unfortunately, this kind of cut, although sufficient, is weak; this is why we decided to introduce another cut type, generated as follows: (1) solve to feasibility a single machine scheduling model with only one PE and tasks running on it; (2) if there is no solution the tasks considered cannot be allocated to any other PE.

The cut is very effective, but we need to solve an NP-hard problem to generate it; however, in practice, the problem can be quickly solved.

With the objective to limit iteration number (which strongly influences the solution method efficiency) we also inserted in the master problem a relaxation of the subproblem. This forbids the allocator to store in a single processor a set of non mutually exclusive tasks whose duration exceeds the time limit, and to assign memory devices in such a way that the total length of a track is greater than the deadline.

4 Experimental results

We implemented all exposed algorithms in C++, using the state of the art solvers ILOG Cplex 9.0 (for ILP) and ILOG Solver 6.0 (for CP).

We tested the method on two set of instances: the first ones are characterized by means of a synthetic benchmark; peculiar input data of this problem (such as the branch probabilities) were estimated via a profiling step. Instances of this first group are only slightly structured, i.e. they have very short tracks and quite often contain singleton

nodes: therefore we decided to generate a second group of instances, completely structured (one head, one tail, long tracks).

We tested all instances on a Pentium IV pc with 512MB RAM. The time limit for the solution process was 30 minutes.

The results of the tests on the first group are summarized in the table 1, which reports results for instances subgroups; in particular it shows for each subgroup: the number of activities (acts), the number of processing elements (PEs), the number of instances in the group (inst.), the instances which were proven to be infeasible (inf.), the mean overall time (in seconds), the mean time to analyze the graph (init), to solve the master and the subproblem, to generate the no-good cuts and the mean number of iterations (it). The solution times are of the same order of the deterministic case (scheduling of Task Graphs), which is a very good result, since we are working on conditional task graphs and thus dealing with a stochastic problem.

For a limited number of instances the overall solving time was exceptionally high: the last column in the table shows the number of instances for which this happened, mainly due to the master problem (A), the scheduling problem (S) or the number of iterations (I). The solution time of this instances was not counted in the mean; in general it was greater than than thirty minutes.

acts	PEs	inst.	inf.	time	init	master	sub	nogood	it	A/S/I
10-12	2	6	0	0.0337	0.0208	0.0075	0.0027	0.0027	1.1667	0/0/0
13-15	2	8	1	0.5251	0.1600	0.0076	0.0040	0.0020	1.1250	0/0/0
16-18	2-3	12	0	0.1091	0.0922	0.0089	0.0067	0.0013	1.0833	0/0/0
19-21	2-3	14	1	0.1216	0.0791	0.0279	0.0079	0.0046	1.2143	0/0/0
22-24	2-3	23	4	0.2336	0.1520	0.0259	0.0061	0.0081	1.1739	0/0/0
25-27	2-3	16	3	1.7849	0.0319	1.7285	0.0108	0.0088	1.3125	0/0/0
28-30	2-3	13	2	0.3331	0.0284	0.0770	0.1900	0.0338	1.6667	0/1/0
31-33	3-4	4	2	0.3008	0.2303	0.0510	0.0040	0.0000	1.0000	0/0/0
34-36	3-4	13	4	0.6840	0.0204	0.4245	0.0132	0.0108	1.2308	0/0/0
37-39	3-4	7	0	1.5670	0.0399	1.2010	0.1384	0.1877	4.4286	0/0/0
40-42	3-4	6	3	2.9162	0.0182	0.5857	2.2267	0.0390	1.6667	0/0/0
43-45	3-4	6	1	5.3670	0.2757	4.8200	0.0630	0.2005	4.1667	0/0/0
46-48	4-5	11	0	3.2719	0.0508	0.6913	2.4616	0.0683	2.0000	1/2/0
49-51	4-5	11	1	1.9950	0.1840	1.7900	0.0071	0.0087	1.1111	1/1/0
52-54	5-6	6	0	8.0000	1.3398	1.5743	4.8788	0.2073	2.7500	1/1/0
55-67	6	8	0	2.2810	0.8333	1.4377	0.0100	0.0000	1.0000	1/4/0

Table 1. Results of the tests on the first group of instances (slightly structured)

Although this extremely high solution time occurs with increasing frequency as the number of activities grows, it seems it is not completely determinated by that factor: sometimes even a very small change of the deadline or of some branch probability makes the computation time explode.

We guess that at least in several cases this happens since the scheduler efficiency greatly relies on a very effective heuristic: for some input graph topologies and parameter configurations the heuristic does not make the right choices and thus the solution

acts	PEs	inst.	inf.	time	init	master	sub	nogood	it	A/S/I
20-29	2	7	2	0.5227	0.0200	0.0134	0.0090	0.0021	8.8571	0/0/0
30-39	2-3	6	0	1.7625	0.0283	1.2655	0.2057	0.2630	5.8333	0/0/0
40-49	3	3	0	0.4380	0.0313	0.3493	0.0573	0.0000	1.0000	0/0/0
50-59	3-4	7	0	1.1403	0.0310	0.6070	0.2708	0.2315	3.6667	0/0/1
60-69	4-5	4	0	10.1598	0.0385	6.8718	1.2798	1.9698	18.0000	0/0/0
70-79	4-5	4	0	88.9650	0.0428	88.6645	0.2578	0.0000	1.0000	0/0/0
80-90	4-6	7	0	202.4655	0.0755	184.0177	6.5008	11.8715	28.6667	0/0/1

Table 2. Result of the tests on the second group of instances (completely structured)

time dramatically grows. In most cases the very high solution time is mainly due to the scheduling problem. Perhaps this could be avoided by randomizing the solution method and by using restart strategies [19].

mean time to gen. a cut			
basic case:			0.0074
with relaxation based cuts (RBC):			0.0499
number of iterations			
deadline	**basic case**	**with RBC**	**result**
8557573	2	3	opt. found
625918	1	1	opt. found
590846	1	1	opt. found
473108	19	6	opt. found
464512	190	14	opt. found
454268	195	24	opt. found
444444	78	15	opt. found
433330	9	4	opt. found
430835	5	3	opt. found
430490	5	3	opt. found
427251	3	2	inf.

Table 3. Number of iterations without and with scheduling relaxation based cuts

The results of the second group of instances (completely structured) are reported in table 2. In this case the higher number of arcs (and thus of precedence constraints) reduces the time windows and makes the scheduling problem much more stable: no instance solution time exploded due to the scheduling problem. On the other hand the increased number of arcs makes the allocation more complex and the scheduling problem approximation less strict, thus increasing the number of iterations and their duration. In two cases this leaded to a break of the time limit.

We also ran a set of tests to verify the efficiency of the scheduling approximation based cuts: in each test we repeatedly solved an instance with decreasing deadline values, until the problem became infeasible; table 3 reports results for a sample 34 activities instance. The iteration number greatly reduces, while the mean time to generate a cut grows by a factor of ten, but remains quite short. Therefore the relaxation cuts

are very effective and enough efficient, although they need the solution of an NP-hard problem; we thus decided to use them in all other tests.

5 Conclusion and future works

We have proposed a stochastic method for planning and scheduling in the stochastic case. The method proposed has two main contributions: the first is a polynomial transformation of a stochastic problem into a deterministic one based on the conditional task graph analysis. Second, the implementation of two constraints for unary and cumulative resources in presence of conditional activities. We believe the results obtained are extremely encouraging. In fact, computation times are comparable with the deterministic version of the same instances. We still have much work to do: first we have to solve the extremely hard instances possibly through randomization; second we have to take into account other aspects where stochasticity could come into play, like task duration which could not be known in advance. Third, we have to validate these results on a real simulation platform to have some feedback on the model.

References

1. Wu, D., Al-Hashimi, B., Eles, P.: Scheduling and mapping of conditional task graph for the synthesis of low power embedded systems. In: Computers and Digital Techniques, IEE Proceedings. Volume 150 (5). (2003) 262–273
2. Eles, P.P.P., Peng, Z.: Bus access optimization for distributed embedded systems based on schedulability analysis. In: International Conference on Design and Automation in Europe, DATE2000, IEEE Computer Sociery (2000)
3. Shin, D., Kim, J.: Power-aware scheduling of conditional task graphs in real-time multiprocessor systems. In: International Symposium on Low Power Electronics and Design (ISLPED), ACM (2003)
4. Wolf, W.: The future of multiprocessor systems-on-chips. In: In Procs. of the 41st Design and Automation Conference - DAC 2004, San Diego, CA, USA, ACM (2004) 681–685
5. Benini, L., Bertozzi, D., Guerri, A., Milano, M.: Allocation and scheduling for mpsocs via decomposition and no-good generation. In: Proc. of the Int.l Conference in Principles and Practice of Constraint Programming. (2005)
6. Hooker, J.N., Ottosson, G.: Logic-based benders decomposition. Mathematical Programming **96** (2003) 33–60
7. Baptiste, P., Pape, C.L., Nuijten, W.: Constraint-Based Scheduling. Kluwer Academic Publisher (2003)
8. Vilim, P., Bartak, R., Cepek, O.: Extension of o(n.log n) filtering algorithms for the unary resource constraint to optional activities. Constraints **10** (2005) 403–425
9. Laborie, P.: Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results. Journal of Artificial Intelligence **143** (2003) 151–188
10. Kuchcinski, K.: Constraints-driven scheduling and resource assignment. ACM Transactions on Design Automation of Electronic Systems **8** (2003)
11. Ahmed, S., Shapiro, A.: The sample average approximation method for stochastic programs with integer recourse. In: Optimization on line. (2002)
12. Laporte, G., Louveaux, F.V.: The integer l-shaped method for stochastic integer programs with complete recourse. Operations Research Letters **13** (1993)
13. Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik **4** (1962) 238–252
14. Norkin, V.I., Pflug, G., Ruszczynski, A.: A branch and bound method for stochastic global optimization. MAthematical Programming **83** (1998)
15. Walsh, T.: Stochastic constraint programming. In: Proc. of the European Conference on Artificial Intelligence, ECAI. (2002)
16. Tarim, A., Manandhar, S., Walsh, T.: Stochastic constraint programming: A scenario-based approach. Constraints **11** (2006) 53–80
17. Culler, D.A., Singh, J.P.: Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann (1999)
18. Compton, K., Hauck, S.: Reconfigurable computing: A survey of systems and software. ACM Computing Surveys **34** (1999) 171–210
19. Gomes, C.P., Selman, B., McAloon, K., Tretkoff, C.: Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. In: AIPS. (1998) 208–213