Università degli Studi di Bologna
DEIS

Allocation, Scheduling and Voltage Scaling on Energy Aware MPSoCs

Luca Benini Davide Bertozzi Alessio Guerri
Michela Milano

March 6, 2007

DEIS Technical Report no. DEIS-LIA-004-07 *LIA Series no. 82*

Allocation, Scheduling and Voltage Scaling on Energy Aware MPSoCs

Luca Benini [1] Davide Bertozzi [2] Alessio Guerri [1]
Michela Milano [1]

[1] *DEIS*
Università di Bologna
V.le Risorgimento, 2
40136 Bologna, Italy
{lbenini,aguerri,mmilano}@deis.unibo.it

[2] *Dipartimento di Ingegneria*
University of Ferrara
V. Saragat 1
41100, Ferrara, Italy
dbertozzi@ing.unife.it

March 6, 2007

Abstract. In this paper we introduce a complex allocation and scheduling problem for variable voltage Multi-Processor System-on-Chip (MPSoC) platforms. We propose a methodology to formulate and solve to optimality the allocation, scheduling and discrete voltage selection problem, minimizing the system energy dissipation and the overhead for frequency switching. Our approach is based on the Logic Benders decomposition technique where the allocation is solved through an Integer Programming solver, and the scheduling through a Constraint Programming solver. The two solvers are interleaved and their interaction regulated by cutting plane generation. The objective function depends on both master and sub-problem variables. We demonstrate the efficiency of our approach on a set of realistic instances.

Contents

1 Introduction

As silicon technology keeps scaling, it is becoming technically feasible to integrate entire and complex systems on the same silicon die. This solution provides scalable computation power, and it is expected that hundreds of processor cores will be integrated on these Multi-Processor Systems-on-Chip (MPSoCs) in future technologies. MPSoCs are widely used in embedded systems (such as cellular phones, automotive control engines, etc.) where, once deployed in field, they always run the same set of applications. Since for many multimedia and signal processing applications the workload is highly predictable at design time, with minimum run-time fluctuations, an optimal allocation and scheduling for such applications can be statically derived off-line.

A critical task for recent MPSoCs is the minimization of the energy consumed since the speed of each processor can be tuned by changing its frequency. We start from a well-characterized task graph, a directed acyclic graph representing a functional abstraction of the application that will run on the MPSoCs. Each task is characterized by the number of clock cycles used for its execution. Clearly the duration of each task and the energy spent for running it depends on the clock frequency used during the task execution. In addition, tasks connected by arcs in the task graph communicate and if they are allocated to different processors, additional communicating tasks are created for reading and writing data on a shared memory.

Defining the optimal allocation, scheduling and voltage scaling for minimizing energy in MPSoCs is the aim of this paper. Energy is consumed during task execution, task communication and for switching between two voltages (setup costs).

The problem we face is very complex. It has never been solved to optimality by the system design community and it cannot be solved by any complete commercial solver that models the problem as a whole. The method we use is the Logic Based Benders Decomposition [8], an extension of the well known OR Benders Decomposition [1] approach for dealing with solvers of any kind. In this setting, we allocate tasks to processors and decide their execution frequency in the master problem, while the subproblem schedules tasks with a fixed duration and static resource assignment. The interaction between the master and the subproblem is regulated via cutting planes generation.

The approach has been followed several times for similar problems, but never applied to scheduling for minimizing costs and setup costs. In particular, there are a number of papers using Benders Decomposition in a CP setting. [12] proposes the branch and check framework using Benders Decomposition (BD). [4] embeds BD in the CP environment ECLiPSe and shows that it can be useful in practice. [5] applied Benders decomposition to minimum cost planning and scheduling problems; in this work the objective function involves only master problem variables, while the subproblem is simply a feasibility problem. [6] and [7] used Benders decomposition for Planning and Scheduling problems with several objective functions: either minimizing the cost (involving only master problem variables), or minimizing the makespan or the tardiness or the number of late tasks (involving the last three cases only subproblem variables); here the objective function involves both master problem and subproblem variables since the execution energy is minimized by the allocation problem solver while the setup cost due to frequency switches can be minimized only at scheduling time.

2 Problem description

The new MPSoC paradigm for hardware platform design is pushing the parallelization of applications, so that instead of running them at a high frequency on a single monolithic core, they can be partitioned into a set of parallel tasks, which are mapped and executed on top of a set of parallel processor cores operating at lower frequencies. Power minimization is a key design objective for MPSoCs to be used in portable, battery-operated devices. This goal can be pursued by means of low power design techniques at each level of the design process, from physical-level techniques (e.g., low swing signaling) up to application optimization for low power. In this paper, we focus on system-level design, where the main knobs for tuning power dissipation of an MPSoC are: allocation and scheduling of a multi-task application onto the available parallel processor cores, voltage and frequency setting of the individual processor cores. For those systems where the workload is largely predictable and not subject to run-time fluctuations (e.g., signal processing or some multimedia applications), the above design parameters can be statically set at design time. Traditional ways to tackle the mapping and configuration problem either incur overly large computation times already for medium-size task sets, or are inaccurate (e.g., use of heuristics and problem modelling with highly simplifying assumptions on system operation). Therefore, design technology for MPSoCs strongly needs accurate, scalable and composable modelling and solving frameworks.

In this paper we consider a reference template [10] for a distributed MPSoC architecture. The platform consists of computation tiles, a shared bus for inter-tile communication and a shared memory. The computation tiles are supposed to be homogeneous and consist of ARM7 processor cores (including instruction and data caches) and of tightly coupled software-controlled scratchpad memories. These latter devices can be viewed as local, low access cost memories (see Fig. 1). Messages can be exchanged by tasks through communication queues [9], which can be allocated at design time either in scratch-pad memory or in remote shared memory, depending on whether tasks are mapped onto the same processor or not.

In this architecture, each processor core can run at different clock frequencies. The frequency of each processor core is derived from a baseline system frequency by means of integer dividers. Moreover, a synchronization module must be inserted

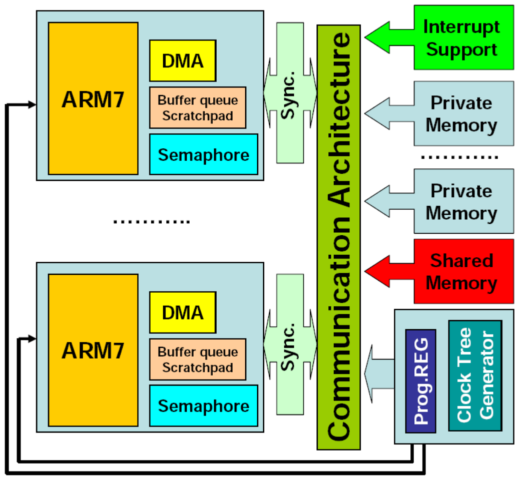

Figure 1. *Distributed MPSoC architecture.*

between the bus and the processor cores to allow frequency decoupling (usually a dual-clock FIFO). The bus operates at the maximum frequency (e.g., 200 MHz). For each processor core, a set of voltage and frequency couples is specified, since the feasible operating points for these cores are not continuous but rather discrete. For modern variable voltage/variable frequency cores, this set is specified in the data-sheet.

Finally, in real-life MPSoC platforms, switching voltage and frequency of a processor core is not immediate nor costless, therefore the switching overhead in terms of switching delay (referred to as setup times) and energy overhead (referred to as setup costs) must be carefully considered when selecting the optimal configuration of a system. In practice, interesting trade-offs have to be studied. On one hand, tasks can be spread across a large number of processor cores, so that these cores can operate at lower frequencies, but more communication arises and the energy cost of many running cores has to be compensated by a more energy-efficient execution of tasks. On the other hand, tasks have to be grouped onto the processor cores and scheduled taking care of minimizing the number of frequency switchings. It must be observed that application real-time requirements play a dominant role in determining solutions for the MPSoC mapping and configuration problem. A good methodology should be conservative with respect to task deadlines, so to minimize the probability of timing violations in the real system.

3 Dynamic Voltage Scaling Problem - DVSP: the model

We consider a directed acyclic task graph G whose nodes represent a set of T tasks, are annotated with their deadline dl_t and with the worst case number of clock cycles WCN_t. Arcs represent dependencies/communications among tasks. Each arc is annotated with the amount of data two dependent tasks should exchange, and therefore the number of clock cycles for exchanging (reading and writing) these data WCN_R and WCN_W. Tasks are running on a set of processors P. Each processor can run with M energy/speed modes and has a maximum load constraint dl_p. Each task spends energy both in computing and in communicating. In addition, when the processor switches between two modes it spends time and energy. We have energy overhead E_{ij} for switching from frequency i to frequency j, and time overhead T_{ij} for switching from frequency i to j.

The Dynamic Voltage Scaling Problem is the problem of allocating tasks to processors, define the running speed of each task and schedule each of them minimizing the total energy consumed.

The method we use for handling the DVSP uses the logic-based Benders decomposition technique [8]. Similarly to [2], the problem is decomposed into two parts: the first, called Master Problem, is the allocation of processors and frequencies to tasks and the second, called Subproblem, is the scheduling of tasks given the static allocation and frequency assignments provided by the master. Note that the frequency assignment could be done in the subproblem. However, the scheduling part becomes extremely slow and performances highly decrease. In addition, the relaxation of the subproblem (introduced in section 4.1) become extremely loose. Differently from [2], the objective function depends on master and subproblem variables. In fact, the master problem minimizes the communication and execution energy, while only during the scheduling phase we could minimize the switching energy overhead.

The master problem is tackled by an Integer Programming solver (through a traditional Branch and Bound) while the subproblem through a Constraint Programming solver. The two solvers interact via no-good and cutting planes generation. The solution of the master is passed to the subproblem. We have two possible cases: (1) there is no feasible schedule: we have to compute a no-good avoiding the same allocation to be found again; (2) there is a feasible and optimal schedule minimizing the second component of the objective function: here we cannot simply stop the iteration since we are not sure we have the optimal solution overall. We have to generate a cut saying that this is the optimal solution unless a better one can be computed with a different allocation.

The procedure converges when the master problem produces a solution with the same objective function of the previous one.

4 Example

As an example, let consider 5 tasks and 5 communications, with the precedence constraints as described in Figure 2. Table 1 shows the duration (in clock cycles) of execution and communication tasks (the durations of the reading and the writing phase R_i and W_i of each communication Com_i are the half of these values). We have 2 processors, running at 2 different frequencies, 200MHz and 100MHz (so, e.g. $Task_1$ will last 500ns if runs at 200MHz and 1μs if runs at 100MHz). The processors waste 10mW when running at 200MHz and 3mW when running at 100MHz. Switching

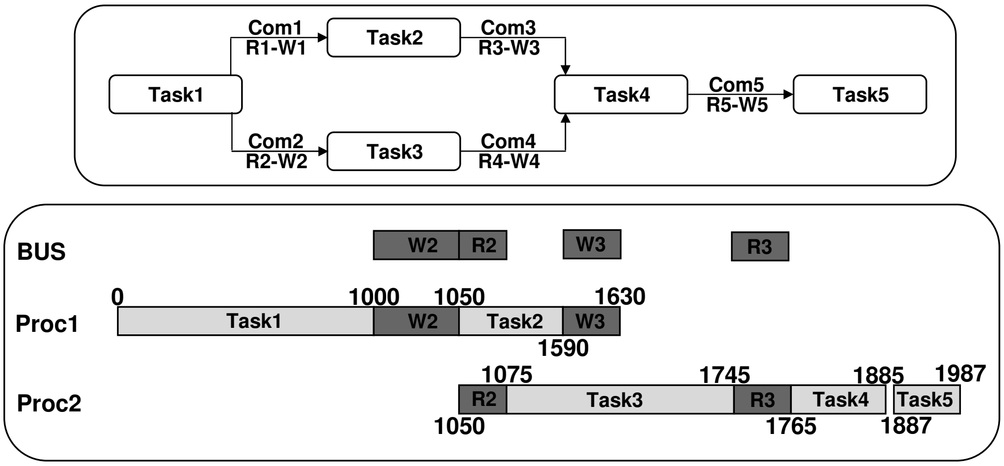

Figure 2. *Task graph and schedule for the example in Table 1*

The first allocation found tries to assign the lower frequency to the third task, being the longest one and thus the most power consuming one; this solution is however not schedulable due to the deadline constraint. The second allocation found is schedulable and is also the optimal one w.r.t. the power consumption minimization (the total power consumption is 13502mW). The first two tasks are allocated on the first processor at the higher frequency and the other three tasks on the second processor: here only $Task_5$ runs at the higher frequency. The Gantt chart in Figure 2 shows the schedule of this solution.

4.1 The Master Problem model

We model the allocation problem with binary variables X_{ptm} which take value 1 if task t is mapped on the processor p and runs in mode m, 0 otherwise. Since we also take into account communication, we assume that two communicating tasks running on the same processor do not consume any energy and do not spend any time (indeed the communication time and energy spent are included in the execution time and energy), while if they are allocated on two different processors, they both consume energy and spend time. The first task spends time and energy for writing data on a shared memory. This operation makes the duration of the task becoming longer: it increases of a quantity WCN_W/f_m where WCN_W is the number of clock cycles for writing data (it depends on the amount of data we should write), and f_m is the frequency of the clock when task t is performed. The second task should read data from the shared memory. Again its duration increases of a quantity WCN_R/f_m where WCN_R is the number of clock cycles for reading data (it depends on the amount of data we should read), and f_m is the frequency of the clock when task t is performed.

Both the read and write activities are performed at the same speed of the task and use the bus (which instead works at the maximum speed). For modelling this aspect, we introduce in the model two variables $R_{pt_1t_2m}$ and $W_{pt_1t_2m}$ taking value 1 if the task t_1 running on processor p reads (resp. writes) data at mode m from (resp. for) a task t_2 not running on p.

Any task can be mapped on only one processor and can run at only one speed. This translates in the following constraints:

$$\sum_{p=1}^{P}\sum_{m=1}^{M} X_{ptm} = 1 \quad \forall t$$

Also the communication between two tasks happens at most once:

$$\sum_{p=1}^{P}\sum_{m=1}^{M} R_{pt_1t_2m} \leq 1 \quad \forall t_1, t_2$$

$$\sum_{p=1}^{P}\sum_{m=1}^{M} W_{pt_1t_2m} \leq 1 \quad \forall t_1, t_2$$

The objective function is to minimize the energy consumption of the task execution, and of the task communication (read and write)

$$E_{comp} = \sum_{p=1}^{P}\sum_{m=1}^{M}\sum_{t=1}^{T} X_{ptm} WCN_t t_{clock_m} P_{tm}$$

$$E_{Read} = \sum_{p=1}^{P} \sum_{m=1}^{M} \sum_{t,t_1=1}^{T} R_{ptt_1m} WCN_{Rtt_1} t_{clock_m} P_{tm}$$

$$E_{Write} = \sum_{p=1}^{P} \sum_{m=1}^{M} \sum_{t,t_1=1}^{T} W_{ptt_1m} WCN_{Wtt_1} t_{clock_m} P_{tm}$$

where P_{tm} is the power consumed in a clock cycle (lasting t_{clock_m}) by the task t at mode m.

$$OF = E_{comp} + E_{Read} + E_{Write}$$

The objective function defined up to now depends only on master problem variables. However, switching from one speed to another introduces transition costs, but their value can be computed only at scheduling time. In fact, they are not constrained in the master problem original model. They are constrained by Benders Cuts instead, after the first iteration. We will present Benders Cuts in section 4.3. Therefore, in the master problem the objective function is:

$$OF_{Master} = OF + Setup$$

$$Setup = \sum_{p=1}^{P} Setup_p$$

It is worth noting that this contribution should be added to the master problem objective function, but, being the $Setup_p$ variables not constrained at the first iteration in the master problem, they are all forced to be 0. From the second iteration, instead, cuts are produced constraining variables $Setup_p$ and this contribution could be no longer 0.

This formulation will result in tasks that are potentially running initially with lower frequencies on the same processor (thus avoiding communication). A measure of control is provided by constraints on deadlines in order to prevent the blind selection of the lowest frequencies and the allocation of all tasks on the same processor. The timing is not yet known in this phase, but we can introduce some constraints that represent a relaxation of the subproblem and will reduce the solution space. For each processor, only a certain load is allowed. Therefore, on each processor the sum of the time spent for computation, plus the time spent for communication (read and write) should be less than or equal to the processor deadline dl_p:

$$T^p_{comp} = \sum_{t=1}^{T} \sum_{m=1}^{M} X_{ptm} \frac{WCN_t}{f_m}$$

$$T^p_{read} = \sum_{t=1}^{T}\sum_{m=1}^{M}\sum_{t_1=1}^{T} R_{ptt_1m}\frac{WCN_{Rtt_1}}{f_m}$$

$$T^p_{write} = \sum_{t=1}^{T}\sum_{m=1}^{M}\sum_{t_1=1}^{T} W_{ptt_1m}\frac{WCN_{Wtt_1}}{f_m}$$

$$T^p_{comp} + T^p_{read} + T^p_{write} \leq dl_p \quad \forall p \quad (1)$$

These relaxations can be tightened by considering chains of tasks in the task graphs instead of groups of tasks running on the same processor. For example consider tasks t_1, t_2, t_3, t_4 linked by precedence constraints so that $t_1 \rightarrow t_2$, $t_2 \rightarrow t_3$ and $t_3 \rightarrow t_4$. Now suppose that t_1 and t_4 are allocated on processor 1 and t_2 and t_3 on other processors. Instead of summing only the durations of t_1 and t_4 that should be less than or equal to the processor deadline, one could add also the duration of t_2 and t_3 since they should be executed before t_4. The chains in a graph can be many, we added only some of them.

Finally, task deadlines can be captured:

$$\sum_{p=1}^{P}\sum_{m=1}^{M}\left[X_{ptm}\frac{WCN_t}{f_m} + \sum_{t1=1}^{T}\left(R_{ptt_1m}\frac{WCN_{Rtt_1}}{f_m} + W_{ptt_1m}\frac{WCN_{Wtt_1}}{f_m}\right)\right] \leq dl_t \quad \forall t$$

There are several improvements we have introduced in the master problem model. In particular we have removed many symmetries leading the solver to explore the same configurations several times.

4.2 The Sub-Problem model

Once allocation and voltage selection have been solved optimally, for the scheduling part each task t has an associated variable representing its starting time $Start_i$. The duration is fixed since the frequency is decided, i.e., $duration_i = WCN_i/f_i$. In addition, if two communicating tasks t_i and t_j are allocated on two different processors, we should introduce two additional activities (one for writing data on the shared memory and one for reading data from the shared memory). We model the starting time of these activities $StartWrite_{ij}$ and $StartRead_{ji}$. These activities are carried on at the same frequency of the corresponding task. If t_i writes and t_j reads data, the writing activity is performed at the same frequency of t_i and its duration $dWrite_{ij}$ depends on the frequency and on the amount of data t_i writes, i.e., WCN_{Wij}/f_i. Analogously, the reading activity is performed at the same frequency of t_j and its duration $dRead_{ji}$ depends on the frequency and on the amount of data t_j reads, i.e., $WCN_{R_{ji}}/f_j$. Clearly the read and write activities are linked together and to the corresponding task:

$$StartWrite_{ij} + dWrite_{ij} \leq StartRead_{ji} \quad \forall i, j \text{ s.t. i communicates with j}$$

$$Start_i + duration_i \leq StartWrite_{ij} \quad \forall i, j \text{ s.t. i communicates with j}$$

$$StartRead_{ji} + dRead_{ji} \leq Start_j \quad \forall i, j \text{ s.t. i communicates with j}$$

In the subproblem, we model precedence constraints in the following way: if task t_i should precede task t_j and they run on the same processor at the same frequency the precedence constraint is simply:

$$Start_i + duration_i \leq Start_j$$

If two tasks run on different processors and should communicate we should add the time for communicating.

$$Start_i + duration_i + dWrite_{ij} + dRead_{ji} \leq Start_j$$

Deadline constraints are captured stating that each task must end its execution before its deadline and, on each processor, all the tasks (and in particular the last one) running on it must end before the processor deadline.

$$Start_i + duration_i \leq dl_{t_i} \quad \forall \; tasks \; t_i$$

$$Start_i + duration_i \leq dl_p \quad \forall i \in p, \; \forall p$$

Resources are modelled as follows. We have a unary resource constraint for each processor, modelled through a cumulative constraint having as parameters a list of all the variables representing the starting time of the activities (tasks, readings, writings) sharing the same resource p, their durations, their resource consumption (which is a list of 1) and the capacity of the processor which is 1.

$$cumulative(StartList_p, DurationList_p, [1], 1) \quad \forall p$$

We model the bus through an additive model we have already validated in [11]. We have an activity on the bus each time a task writes or reads data to or from the shared memory. The bus is modelled as an additive resource and several activities can share the bus, each one consuming a fraction of it until the total bandwidth is reached. The cumulative constraint used to model the bus is:

$$cumulative(StartReadWriteList, DurationList, Fraction, TotBWidth)$$

where $StartReadWriteList$ and $DurationList$ are lists of the starting times and durations of all read and write activities needing the bus, $Fraction$ is the amount of bandwidth granted to any activity when accessing the bus[1] and $TotBWidth$ is total bandwidth available of the bus.

To model the setup time and cost for frequency switching we take advantage of the classes defined by ILOG Scheduler to manage transitions between activities. It is possible to associate a label to each activity and to define a transition matrix that specifies, for each couple of labels l_1 and l_2, a setup time and a setup cost that must be paid to schedule, on the same resource, an activity having the label l_1 just before an activity having the label l_2. When, during the search for a solution, two activities with labels l_1 and l_2 are scheduled one just after the other on the same resource, the solver will satisfy the additional constraint:

$$Start_{l_1} + duration_{l_1} + TransTime_{l_1 l_2} \leq Start_{l_2}$$

where $TransTime_{l_1 l_2}$ is the setup time specified in the transition matrix. Likewise, the solver introduces $TransCost_{ij}$ in the objective function. If S_p is the set of all the tasks scheduled on processor p, the objective function we want to minimize is:

$$OF = \sum_{p=1}^{P} \sum_{(i,j) \in S_p | next(i)=j} TransCost_{ij}$$

4.3 Generation of Logic-based Benders Cuts

Once the subproblem has been solved, we generate Benders Cuts. The cuts are of two types:

- if there is no feasible schedule given an allocation, the cuts are the same we computed for the single voltage problem and depend on variables X_{ptm}.
- if the schedule exists, we cannot simply stop the iteration since the objective function depends also on subproblem variables. Therefore, we have to produce cuts saying that the one just computed is the optimal solution unless a better one exists with a different allocation. These cuts produce a lower bound on the setup of single processors.

The first type of cuts are no-good: we call J_p the set of couples (Task, Frequency) allocated to processor p. We impose

$$\sum_{(t,m) \in J_p} X_{ptm} \leq |J_p| - 1 \quad \forall p$$

Let us concentrate on the second type of cuts. The cuts we produce in this case are bounds on the variable *Setup* previously defined in the Master Problem.

[1]This value was experimentally tuned to 1/4 of the total bus bandwidth.

Suppose the schedule we find for a given allocation has an optimal setup cost $Setup^*$. It is formed by independent setups, one for each processor $Setup^* = \sum_{p=1}^{P} Setup_p^*$.

We have a bound on the setup LB_{Setup_p} on each processor and therefore a bound on the overall setup $LB_{Setup} = \sum_{p=1}^{P} LB_{Setup_p}$.

$$Setup_p \geq 0$$
$$Setup_p \geq LB_{Setup_p}$$

$$LB_{Setup_p} = Setup_p^* - Setup_p^* \sum_{(t,m) \in J_p} (1 - X_{ptm})$$

These cuts remove only one allocation. Indeed, we have also produced cuts that remove some symmetric solutions.

We have devised tighter cuts removing more solutions. Intuitively, each time we consider a solution of the problem overall, we generate an optimal setup cost $Setup^*$ for the given allocation. In the current solution, we know the number of frequency switches producing $Setup^*$. We can consider each processor independently since the frequency switches on one processor are independent from the other. We can impose cuts that say that $Setup^*$ is bound for all solutions with the same set of frequency switches of the last one found or a superset of it. To do that we have to introduce in the model variables $Next_{t_1 t_2 f_1 f_2 p}$, which complicate the model too much. In fact, our experimental results show that these cuts, even if tighter, do not lead to any advantage in terms of computational time.

4.4 Relaxation of the subproblem

The iterative procedure presented so far can be improved by adding a bound on the setup cost and setup time in the master problem based only on information derived from the allocation.

Suppose we have five tasks running on the same processors using three different frequencies. So for instance, tasks t_1, t_3 and t_5 run at frequency f_1, t_2 runs at frequency f_2 and t_4 runs at frequency f_3. Since we have to compute a bound, we suppose that all tasks running at the same speed go one after the other. We can have six possible orders of these frequencies leading to different couples of frequency switches. A bound on the sum of the energy spent during the frequency switches is the minimal sum between two switches, i.e., the sum of all possible switches minus the maximum switch. This bound is extremely easy to compute and does not enlarge the allocation problem model.

Let us introduce in the model variables Z_{pf} taking value 1 if the frequency f is allocated at least once on the processor p, 0 otherwise. Let us call E_f the minimum energy for switching to frequency f, i.e. $E_f = min_{i, i \neq f}\{E_{if}\}$.

$$Setup_p \geq \sum_{f=1}^{M}(Z_{pf}E_f - max_f\{E_f|Z_{pf} = 1\})$$

This bound helps in reducing the number of iterations between the master and the subproblem.

Similarly, we can compute the bound on the setup time given an allocation. Let us consider $T_f = min_{i,i \neq f}\{T_{if}\}$. Therefore, we can compute the following bound.

$$SetupTime_p \geq \sum_{f=1}^{M}(Z_{pf}T_f - max_f\{T_f|Z_{pf} = 1\})$$

This bound can be used to tighten the constraint (1) in section 4.1 in the following way.

$$T^p_{comp} + T^p_{read} + T^p_{write} + SetupTime_p \leq dl_p \quad \forall p$$

so that solutions provided by the master problem are more likely to be feasible for the subproblem.

A tighter bound on the setup time and cost could be achieved by introducing in the allocation problem model variables $Next$, but as explained in section 4.3 they complicate too much the model and are not worth using.

5 Experimental Results

We have generated 500 realistic instances, with the number of tasks varying from 4 to 10 and the number of processors from 2 to 10. We assume that each processor can run at three different frequencies. We consider, as in [2], applications with a pipeline workload. Therefore we refer to the number of tasks to be allocated and we schedule a larger number of tasks corresponding to many iterations of the pipeline. We also have generated 27 realistic instances with the number of tasks varying from 8 to 14 and the number of processors from 2 to 6, with generic task graphs. The generic task graph complicates the problem since it increases the parallelism degree. We assume that each processor can run at six different frequencies. All the considered instances are solvable and we found the proved optimal solution for each of them. Experiments were performed on a 2.4GHz Pentium 4 with 512 Mb RAM. We used ILOG CPLEX 8.1, ILOG Solver 5.3 and ILOG Scheduler 5.3 as solving tools.

5.1 Comparison with pure approaches

In [2], we compared a solving tool based on Benders Decomposition for a similar problem with pure CP or IP based solving tools. Results shown that the pure approaches were not comparable with the hybrid one, being the search times for finding a solution to a relaxed (thus easier) problem order of magnitude higher. The problem we are facing in this paper is much more complex then the one presented in [2], since we consider also frequency switching. We developed a CP and an IP-based approach to solve allocation, scheduling and voltage selection, but not even a single (feasible) solution was found within 15 minutes, while the hybrid approach, within 4 minutes, finds the optimal solution and proves optimality for all the pipelined instances considered.

5.2 Experimental results

In this section we show the results obtained solving the problem instances using the model described in section 3. We consider first the instances with task graphs representing a pipeline workflow. Note that here, since we are considering applications with pipeline workload, if n is the number of tasks to be allocated, the number of scheduled tasks is n^2. Results are summarized in Table 2. The first three columns contain the number of allocated and scheduled tasks and the number of processors considered in the instances (we remind that each processor can run at three different frequencies). The last two columns represent respectively the search time and the number of iterations. Each value is the mean over all the instances with the same number of tasks and processors. We can see that for all the instances the optimal solution can be found within four minutes. The number of iterations is typically low. Table 4 shows the percentage of occurrence of a given number of iterations. We can see that the optimal solution can be found at the first step in one half of the cases and the number of iterations is at most 5 in almost the 90% of cases. This result is due to the tight relaxations added to the master problem model. We tried to remove these relaxations and we found that the search time and the number of iterations rise, in the average case, up to 1 order of magnitude and, in the worst cases, the solution cannot be found within two hours.

We extended our analysis to instances where the task graph is a generic one, so an activity can possibly read data from more than one preceding activity and possibly write data that will be read by more than one following activity, so the number of reading and writing activities can be considerably higher, being higher the number of edges in the task graph. We remind that each processor can run at six different frequencies, so the number of alternative resources a task can use is six times the number of processors. Differently from the pipelined instances, here we schedule a single repetition of each task. Table 3 summarizes the results. Each instance presented has been solved optimally. Columns have the same meaning as those already described in Table 2. We can see that typically the behaviors are similar to those found when solving the pipelined instances, but sometimes the number of iterations, and thus the search time is notably higher. This is due to the particular

Tasks				
Alloc	**Sched**	**Procs**	**Time(s)**	**Iters**
4	16	2	1,73	1,98
4	16	3	1,43	2,91
4	16	4	2,24	3,47
5	25	2	2,91	2,36
5	25	3	4,19	4,12
5	25	4	5,65	4,80
5	25	5	6,69	3,41
6	36	2	3,84	2,90
6	36	3	10,76	2,17
6	36	4	15,25	4,66
6	36	5	23,17	4,50
6	36	6	26,14	3,66
7	49	2	4,67	1,75
7	49	3	5,90	1,90
7	49	7	34,53	6,34
8	64	2	4,09	3,28
8	64	3	10,99	1,83
8	64	4	12,34	4,45
8	64	5	22,65	10,53
8	64	7	51,07	6,98
9	81	2	1,79	1,12
9	81	5	60,07	7,15
9	81	6	70,40	9,20
10	100	2	5,52	1,83
10	100	3	3,07	1,96
10	100	6	120,02	6,23
10	100	10	209,35	10,65

Table 2. *Search time and number of iterations for instances with pipelined task graphs*

Tasks				
Alloc	**Sched**	**Procs**	**Time(s)**	**Iters**
8	8	2	1,57	1
8	8	3	1,48	2
8	8	3	0,81	1
8	8	3	4,26	6
8	8	4	0,86	1
9	9	2	2,51	1
9	9	2	1,11	1
9	9	2	2,73	3
9	9	3	35,95	43
9	9	3	2,51	1
9	9	3	6,62	2
9	9	4	1,40	3
9	9	4	2,14	5
9	9	4	2,60	4
9	9	4	29,59	26
9	9	4	4,84	6
9	9	6	158,43	39
10	10	2	5,90	1
10	10	3	2,12	1
10	10	3	12,81	3
10	10	4	0,37	1
10	10	4	13,92	14
10	10	4	4,18	5
10	10	4	11,50	27
12	12	5	551,92	213
14	14	2	14,11	1
14	14	6	3624,81	2

Table 3. *Search time and number of iterations for instances with generic task graphs*

structure of the task graph; in fact it can happens that a high degree of parallelism between the tasks, that is a high number of tasks that can execute only after a single task, leads to allocations that are not schedulable. The master problem solver thus looses time proposing to the scheduler a high number of unfeasible allocation. Introducing in the master problem model some relaxations coming from an analysis of the task graph structure, and in particular from the precedence constraints, can

Iter	1	2	3	4	5	6	7	8	9	10	11+
%	50,20	18,51	7,11	4,52	4,81	2,88	2,46	2,05	1,64	1,64	4,11

Table 4. *Number of iterations distribution ratio*

lead to better results.

6 Conclusion and future research

An exact algorithm for allocation, scheduling and voltage selection has been proposed exploiting the method of Logic-based Benders Decomposition. Experimental results show that the approach using CP and IP for the problem as a whole cannot solve any of the instances considered, while our approach solves them all to optimality. A number of improvements can be conceived the most important concerning the use of a column generation approach for the master problem would most probably lead to a significant speed up. As a second improvement cutting planes that can be derived from [3] and integrated in the master problem model. In addition, we are investigating tighter cutting planes based on information derived from the precedence graph.

Acknowledgement This work has been partially supported by MIUR under the COFIN2005 project *Mapping di applicazioni multi-task basate su Programmazione a vincoli e intera.*

References

[1] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[2] D. Bertozzi, L. Benini, A. Guerri, and M. Milano. Allocation and scheduling for mpsocs via decomposition and no-good generation. In *Procs. of the 11th Intern. Conference on Principles and Practice of Constraint Programming - CP 2005*, pages 107–121, Sites, Spain, Sept. 2005. Springer.

[3] M. Fischetti E. Balas and W. Pulleyblank. The precedence constrained asymmetric travelling salesman problem. *Mathematical Programming*, 68:241–265, 1995.

[4] A. Eremin and M. Wallace. Hybrid benders decomposition algorithms in constraint logic programming. In *Procs. of the 7th Intern. Conference on Principles and Practice of Constraint Programming - CP 2001*, pages 1–15, Paphos, Cyprus, Nov. 2001. Springer.

[5] I. E. Grossmann and V. Jain. Algorithms for hybrid milp/cp models for a class of optimization problems. *INFORMS Journal on Computing*, 13:258–276, 2001.

[6] J. N. Hooker. A hybrid method for planning and scheduling. In *Procs. of the 10th Intern. Conference on Principles and Practice of Constraint Programming - CP 2004*, pages 305–316, Toronto, Canada, Sept. 2004. Springer.

[7] J. N. Hooker. Planning and scheduling to minimize tardiness. In *Procs. of the 11th Intern. Conference on Principles and Practice of Constraint Programming - CP 2005*, pages 314–327, Sites, Spain, Sept. 2005. Springer.

[8] J. N. Hooker and G. Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

[9] P. Poletti, A. Poggiali, and P. Marchal. Flexible hardware/software support for message passing on a distributed shared memory architecture. In *2005 Design, Automation and Test in Europe Conference and Exposition DATE2005*, pages 736–741, 2005.

[10] M. Ruggiero, A. Acquaviva, D. Bertozzi, and L. Benini. Application-specific power-aware workload allocation for voltage scalable mpsoc platforms. In *2005 International Conference on Computer Design*, pages 87–93, 2005.

[11] M. Ruggiero, A. Guerri, D. Bertozzi, L. Benini, and M. Milano. Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip. In *2006 Design, Automation and Test in Europe Conference and Exposition DATE2006*, 2006.

[12] E. S. Thorsteinsson. A hybrid framework integrating mixed integer programming and constraint programming. In *Procs. of the 7th International Conference on Principles and Practice of Constraint Programming - CP 2001*, pages 16–30, Paphos, Cyprus, Nov. 2001.