Reducing the Abstraction and Optimality Gaps in the Allocation and Scheduling for Variable Voltage/Frequency MPSoC Platforms

Martino Ruggiero, Davide Bertozzi, Luca Benini, *Fellow, IEEE*,
Michela Milano, and Alexandru Andrei, *Student Member, IEEE*

***Abstract*—This paper proposes a novel approach to solve the allocation and scheduling problems for variable voltage/frequency multiprocessor systems-on-chip, which minimizes overall system energy dissipation. The optimality of derived system configurations is guaranteed, while the computation efficiency of the optimizer allows for solving problem instances that were traditionally considered beyond reach for exact solvers (optimality gap). Furthermore, this paper illustrates the development- and run-time software infrastructures that assist the user in developing applications and implementing optimizer solutions. The proposed approach guarantees a high level of power, performance, and constraint satisfaction predictability as from validation on the target platform, thus bridging the abstraction gap.**

***Index Terms*—Allocation, Benders decomposition, multiprocessor systems-on-chip (MPSoCs), scheduling, virtual platform.**

I. Introduction

A NUMBER of multiprocessor system-on-chip (MPSoC) platforms support variable frequency and voltage operation [14]–[16], and many authors have pointed out that optimal allocations, schedules, and frequency/voltage settings lead to major power savings [3], [17]. Unfortunately, this optimization problem is known to be NP-hard [17] even in much simplified variants [18], and most authors propose simplified models and heuristic approaches to solve it in reasonable time.

Model simplification is often achieved by abstracting away platform implementation details such as the penalties and the discrete range for frequency and voltage switching or the actual connectivity of communicating cores. As a result, optimization problems become more tractable, even reaching polynomial time complexity [17]. Unfortunately, this approach creates an *abstraction gap* between the optimization model and the real hardware–software (HW–SW) platform. Validation is therefore required, and the accuracy of the solutions must be carefully assessed through extensive simulation runs or executions on the target HW platform.

Heuristic approaches rely on two main strategies: *problem decomposition* and *incomplete search*. Decomposition splits the problem into a set of subproblems that are then solved in sequence. A common decomposition strategy for the mapping problem is to first perform allocation, followed by scheduling, and, finally, voltage and frequency assignment [19], [20]. Incomplete search [21] relies on flexible iterative algorithmic frameworks (e.g., genetic algorithm or tabu search) that are customized for the target problem and generally find good solutions in a reasonable computation time. The main issue with decomposition and incomplete search is that they introduce an *optimality gap* of an unknown size. In other words, they provide very limited or no information on the distance between the best computed solution and the optimal one. Even worse, when attempting to solve constrained problems, they may fail to find existing feasible solutions.

The goal of this paper is to address both abstraction and optimality gaps. Namely, we formulate an accurate model for allocation, scheduling, and frequency/voltage settings, which accounts for a number of nonidealities in real-life HW platforms. We also developed a novel mapping algorithm that deterministically finds optimal solutions. Although its worst-case run time is obviously exponential, our search strategy is computationally efficient in practice and achieves low run times (i.e., minutes) for problem instances of practical relevance (i.e., up to hundreds of tasks). This is much beyond the instance sizes that could be handled in the past by complete-search algorithms. On the other hand, we achieve consistently lower power results than previously reported heuristics. More importantly, we find feasible solutions for tightly constrained problem instances where heuristic search fails.

Our optimizer is based on an algorithmic framework called logic-based Benders decomposition (LBBD) [22], [23] which solves the allocation, scheduling, and voltage/frequency selection problems to optimality in a computation-efficient fashion through the cooperation between two solvers: an integer linear programming (IP) solver for allocation and voltage/frequency setting and a constraint programming (CP) solver for scheduling. It is important to emphasize that LBBD is *not* a heuristic decomposition strategy; the two solvers interact in an iterative fashion that is guaranteed to achieve convergence to optimality. The computational efficiency of our optimizer comes from three main factors: 1) We use solvers that are well matched to the subproblems they handle; 2) we use problem-specific

Manuscript received December 4, 2007; revised June 6, 2008 and October 12, 2008. Current version published February 19, 2009. The works of M. Ruggiero, L. Benini, and M. Milano were supported in part by the PREDATOR Project funded by the European Community's 7th Framework Programme under Contract FP7-ICT-216008 and in part by the European Network of Excellence ARTIST DESIGN. This paper was recommended by Associate Editor V. Narayanan.

M. Ruggiero, L. Benini, and M. Milano are with the Electronics, Computer Sciences, and Systems Department, University of Bologna, 40126 Bologna, Italy (e-mail: mruggiero@unibo.it; lbenini@unibo.it; mmilano@unibo.it).

D. Bertozzi is with the Engineering Department, University of Ferrara, 44100 Ferrara, Italy (e-mail: dbertozzi@ing.unife.it).

A. Andrei is with Embedded Systems Laboratory, Linköping University, 581 83 Linköping, Sweden, and also with Ericsson AB, 164 83 Stockholm, Sweden (e-mail: alean@ida.liu.se).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2009.2013536

relaxations and Benders cuts to propagate information from one solver to the other, which rapidly achieve convergence in practice; 3) by splitting the problem in two, we obtain easier to solve and balanced subproblems.

Furthermore, a second main contribution of this paper is the implementation of the static (design time) and dynamic (run time) SW infrastructure required to run applications on the target platform. This is a critical and nontrivial task, as we must guarantee that actual execution accurately matches, in time and space, the solution computed by the optimizer. We can therefore validate the optimizer against cycle-accurate performance and power analysis on a virtual platform. Experiments on a large number of problem instances demonstrate that the accuracy of our model is high and that execution traces match model predictions with an average error below 5% (worst case, 10%).

We target statically configured systems, where allocation, scheduling, and frequency settings are precomputed at design time. Such systems require design-time knowledge of application behavior under the assumption of minimum run-time fluctuations. Many signal processing, data encryption, or video graphics applications fall into this category. For them, our methodology makes the precomputation of optimal solutions still affordable in spite of the increasing number of integrated processor cores and of the growing exposition of task-level parallelism. Without lack of generality, in this paper, we do not consider conditional task graphs, which we leave for future work.

This paper is structured as follows. After reviewing previous work, the target architecture and the virtual platform environment are presented in Section III. Section IV provides background on LBBDs. Our approach to the mapping problem is presented in Sections V and VI. Computation efficiency is assessed in Section VII. The design- and run-time supports to make the optimization framework interact with the HW–SW platform are illustrated in Sections VIII and IX, respectively. Accuracy validation of the optimizer and demonstrators follow in Section X, while a comparison with a heuristic approach is reported in Section XI.

II. Related Work

In the following, we focus on offline voltage/frequency selection techniques, since our approach falls into this category.

A number of techniques have been developed for single-processor systems. Yao *et al.* proposed, in [24], the first dynamic voltage scaling (DVS) approach which can dynamically change the supply voltage over a continuous range. Ishihara and Yasuura [25] modeled the discrete voltage selection problem using an IP formulation. Xie *et al.* [26] present an algorithm for calculating the bounds on the power savings achievable through voltage selection. Jejurikar and Gupta [27] propose an algorithm that combines voltage scaling and shutdown in order to minimize dynamic and leakage energies.

Andrei *et al.* [3], [17] proposed an approach that optimally solves the voltage scaling problem for multiprocessor systems with imposed time constraints. The continuous voltage scaling is solved using convex nonlinear programming with polynomial time complexity, while the discrete problem is proved to be strongly NP-hard and is formulated as mixed IP (MILP).

The previously mentioned approaches assume that the mapping and scheduling are given. However, the achievable energy savings of DVS are greatly affected by the mapping and the scheduling of the tasks on the target processors.

Task mapping and scheduling are known NP-hard problems [28] that have been previously addressed, without and with the objective of minimizing the energy. Both heuristic [4], [29], [30] and exact solutions [31] have been proposed.

Assuming that the mapping of the tasks on the processors is given as input, Gruian and Kuchcinski [32] present a scheduling technique that maximizes the available slack, which is then used to reduce the energy via voltage scaling. Schmitz *et al.* [29] present a heuristic approach for mapping, scheduling, and voltage scaling on multiprocessor architectures.

A leakage-aware approach for combined dynamic voltage selection and adaptive body biasing has been proposed in [17] and [33]. However, the approach in [33] is restricted to the single-processor case. A multiprocessor setting is addressed in [17] through an MILP approach. Although we concentrate, in this paper, on the dynamic power and supply voltage selection, our methodology can handle (with minor changes) the combined supply and body-bias scaling problem with only marginal implications on computational complexity.

The closest approach to this paper is the one of Leung *et al.* [34]. They propose a mixed integer nonlinear programming formulation for mapping, scheduling, and voltage scaling of a given task graph to a target multiprocessor platform. They assume continuous voltages; hence, the overall result is suboptimal. More importantly, they do not take into account communication time and energy in their model. In this paper, this aspect is strategic for reducing the abstraction gap between the model and the real application behavior.

Whenever allocation and scheduling can be performed offline due to the features of the application (predictable workload), our approach is to solve these problems to optimality. In order to overcome the challenge posed by the increasing level of parallelism both in the HW and in the SW architectures, we address the need for more computation-efficient solving techniques in this paper.

III. Target Architecture

The objective of this paper is to map an application with exposed task-level parallelism onto a homogeneous multicore platform while minimizing overall system power and meeting real-time constraints.

The target architecture for our mapping strategy is a general template for a distributed MPSoC architecture. The platform consists of a scalable number of homogeneous processing cores, a shared communication infrastructure, and a shared memory for intertile communication. Processing cores embed instruction and data caches and are directly connected to tightly coupled SW-controlled scratch-pad memories.

The architecture is assumed to provide a harmonized HW–SW support for messaging, targeting scalability to a large number of communicating cores. Messages can be exchanged by tasks through SW communication queues, which can be physically allocated either in scratch-pad or shared memory, depending on whether tasks are mapped onto the same processor or not. This assumption avoids the generation of bus traffic and incurring congestion delays for local communications. We also target architectures where synchronization between producer–consumer pairs does not give rise to semaphore

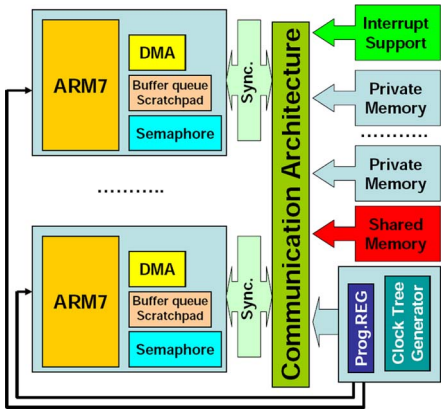

Fig. 1. Distributed MPSoC architecture.

polling traffic on the bus, since this might unpredictably and unacceptably degrade performance of ongoing message exchanges. Interrupt-based synchronization and the implementation of distributed semaphores at each computation tile are two example mechanisms matching our requirements.

As in many recent multicore architectures, we assume that the target platform supports different working frequencies and voltages for each processor core. In practice, each computation tile has its own clock tree, and synchronization mechanisms are provided for interfacing with the system bus (clock domain crossing). Moreover, we assume that the voltage/frequency settings can be adjusted at run time.

An embodiment of this template architecture is considered in this paper and is shown in Fig. 1. SystemC-based simulation models of this architecture were developed within the MPARM simulation environment [5] and feature systemwide clock-cycle accuracy. The virtual platform serves to provide input data to the optimization framework and to validate its solutions with functional simulation (accuracy of objective function values, constraint satisfaction).

ARM7 processor cores with 32-kB instruction and data caches build up the computation section of the platform, while an interconnect compliant with advance microcontroller bus architecture (AMBA) advanced high-performance bus (AHB) specification is selected. Frequency/voltage decoupling between the processor cores and the bus is implemented through dual-clock FIFOs featuring a latency of four clock cycles of the slowest clock frequency [36]. A maximum operating frequency of 200 MHz (with 1-V supply voltage) is assumed for the bus, to which per-core frequency dividers are applied. The correspondent scaling factor for the power supply was inspired by Nowka *et al.* [10].

We set up a communication and synchronization library, abstracting away low-level architectural details to programmers, such as memory maps or explicit management of HW semaphores and shared memory. More details can be found in Section IX and, more extensively, in [35]. System resources are controlled by the RTEMS real-time operating system running on each processor core.

Our virtual platform environment provides power statistics for ARM cores, caches, on-chip memories, and AMBA AHB bus by leveraging technology-homogeneous power models for a 0.13-μm technology provided by STMicroelectronics. When all tasks mapped on a processor core are suspended, the core enters power save mode, where the power consumption is assumed to be negligible.

IV. LBBD

The technique we use in this paper for finding the optimal allocation, voltage/frequency assignment, and scheduling of tasks is derived from a method, known in operations research as Benders decomposition [1] and refined by Hooker and Ottosson [22] with the name of LBBD. The classical Benders decomposition method decomposes a problem into two loosely connected subproblems. It enumerates values for the connecting variables. For each set of enumerated values, it solves the subproblem that results from fixing the connecting variables to these values. The solution of the subproblem generates a constraint, called Benders cut, which the connecting variables must satisfy in all subsequent solutions enumerated. The process continues until the master problem fails to find a solution better than the current upper bound. The classical Benders approach, however, requires that the subproblem be a continuous linear or nonlinear programming problem. This requirement poses severe applicability restrictions. For instance, scheduling is a combinatorial problem that has no practical linear or nonlinear programming model. Therefore, the Benders decomposition idea can be extended to a logic-based form (LBBD) that accommodates an arbitrary subproblem, such as a discrete scheduling problem. More formally, as introduced in [22], a problem can be written as

$$\min f(x,y) \tag{1}$$
$$\text{s.t. } p_i(y)\ i \in I_1 \text{ Master Problem Constraints} \tag{2}$$
$$g_i(x)\ i \in I_2 \text{ Subproblem Constraints} \tag{3}$$
$$q_i(y) \rightarrow h_i(x)\ i \in I_3 \text{ Conditional Constraints} \tag{4}$$
$$y \in Y \text{ Master Problem Variables} \tag{5}$$
$$x_j \in D_i \text{ Subproblem Variables.} \tag{6}$$

We have master problem constraints, subproblem constraints, and conditional constraints linking the two models. If we solve the master problem to optimality, we obtain values for variables y in I_1, namely, $\bar{y}$, and the subproblem is thus formulated as

$$\min\ f(x,\bar{y}) \tag{7}$$
$$g_i(x)\ i \in I_2 \text{ Subproblem Constraints} \tag{8}$$
$$q_i(\bar{y}) \rightarrow h_i(x)\ i \in I_3 \text{ Conditional Constraints} \tag{9}$$
$$x_j \in D_i \text{Subproblem Variables.} \tag{10}$$

The heart of Benders decomposition is somehow to derive a function that gives a valid lower bound on the optimal value of the original problem for any fixed value of y. This function yields to a valid Benders cut. The algorithm proceeds as follows. At each iteration $1,\ldots,h$, the Benders cuts so far generated are added to the master problem model that is formed by (1), (2), (5), and

$$B_{y_i}(y)\ i \in 1,\ldots,h \text{ Benders cuts.} \tag{11}$$

y_i is the solution found at iteration i of the master problem.

In practice, to avoid the generation of master problem solutions that are trivially infeasible for the subproblem, it is worth adding a relaxation of the subproblem to the master problem.

V. High-Impact Modeling Choices

Deciding to use the LBBD to solve a combinatorial optimization problem does not simply require the definition of a problem model but implies a number of design choices that strongly affect the overall performance of the algorithm. First, we should decide the master problem and the subproblem variables, along with the corresponding constraints. Second, depending on the variables and constraints that each problem is composed of, we should choose the solver that is more suited for the problem at hand. To improve efficiency, a proper relaxation of the subproblem should be inserted in the master so as to avoid the generation of trivially infeasible master problem solutions. Finally, we have to design proper Benders cuts that remove a set of infeasible and suboptimal solutions in the master problem and enable the performance of a low number of iterations.

The degrees of freedom of LBBD were fixed as follows. Master problem variables represent allocation and voltage selection choices for both execution and communication tasks. Corresponding constraints are also part of the master problem. The objective function is the energy consumed for running each task. The subproblem variables instead correspond to scheduling choices for execution and communication activities. Precedence and resource constraints are enforced in the subproblem. As an objective function, we minimize the energy dissipation associated with frequency switchings.

The choice of the best solver comes from the structure of the master and the subproblem. For some problems, it is widely recognized that either the integer programming or CP is the technique of choice. We therefore propose for the master problem an integer-programming model. It is effective for coping with optimization problems, since the integer-programming model has a global problem view due to the use of linear relaxations, and it therefore better copes with objective functions based on the sum of assignment costs. The model feeding the IP solver will be described in Section VI-A.

For the scheduling subproblem, the solver is instead based on CP. It has an effective way of coping with the so-called *feasibility reasoning*, encapsulating efficient and incremental filtering algorithms into global constraints. It better copes with temporal resource constraints and finer time granularities, and this explains why scheduling is the most successful application area of CP to date. In particular, many resource and temporal constraints have been devised so as to solve large problem instances, see [12]. A corresponding IP model for the scheduling problem would require a binary variable for each activity and each time step of the schedule horizon, leading to overly large models. Our CP scheduling subproblem model will be illustrated in Section VI-B.

An important addition to the master problem is a relaxation of the subproblem that avoids the generation of trivially infeasible master problem solutions. Real-time constraints are not taken into account in the allocation problem solver, leading this latter to pack computation and communication activities in the same processor and exceeding the real-time constraints. We should state in the master problem that the sum of the overall execution and communication times should not exceed the deadline for that processor. In addition, using a subproblem relaxation, we can compute a bound on the energy and the time for frequency switching for tasks allocated on the same processor. The relaxations used are described in Section VI-C.

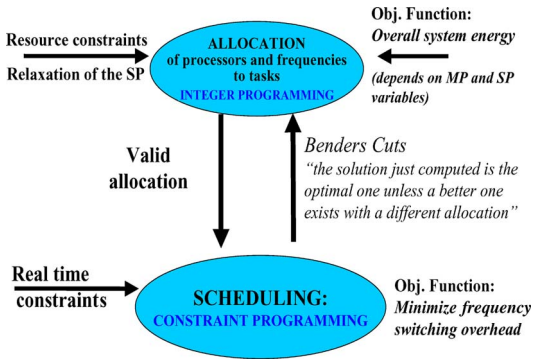

Fig. 2. Application of LBBD to the DVSP. MP and SP stand for master and subproblem, respectively.

Benders cuts are essential for the interaction between the two solvers. Note that the overall system energy minimization function involves both master and subproblem variables. We solve the allocation problem first and the scheduling problem later. The subproblem could return infeasibility, indicating that no feasible schedule exists for a given allocation. In this case, the master problem solver will be constrained not to return the same allocation through proper Benders cuts. Alternatively, a feasible schedule is derived for the given allocation, and a new iteration of the master problem solver is triggered. This way, the computed allocation and scheduling solution at the first iteration is retained as the optimal solution unless a more energy-efficient one exists with a different allocation and frequency assignment. More details follow in Section VI-D.

The resulting cooperative solving framework for the power-aware mapping problem is shown in Fig. 2 and detailed in Section VI.

VI. DVSP

We consider a task graph G whose nodes represent a set of T tasks, which are annotated with their deadline dl_t and with the worst-case number of clock cycles WCN_t. Arcs represent dependences among tasks. Each arc is annotated with the amount of data that two dependent tasks t_i and t_j should exchange and, therefore, the number of bus clock cycles for exchanging (reading and writing) these data $WCN_{Rt_it_j}$ and $WCN_{Wt_it_j}$. 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). Execution, read, and write activities are modeled as atomic. Tasks run on a set of processors P. Without lack of generality, we assume that each task has enough local memory to meet its storage requirements, since these latter can be easily included in an extended model version.

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 a matrix $\mathbf{E}$ describing energy overhead $E_{f_if_j}$ for switching from any frequency f_i to any f_j. Similarly, a matrix $\mathbf{T}$ describing time switching overhead $T_{f_if_j}$ is defined.

The DVS problem (DVSP) is the problem of allocating tasks to processors, defining the running speed of each task, and scheduling each of them while minimizing the total energy consumed. In order to solve the DVSP to optimality without simplifying assumptions relieving computation constraints but

impairing solution accuracy, we applied the LBBD technique [22] to this new application domain.

As introduced in Section V, we decompose the problem in 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 allocations and frequency assignments provided by the master.

A. Master Problem Model

We model the allocation problem using IP. We have binary variables X_{ptm} which take the value of one if task t is mapped on the processor p and runs in (energy speed) mode m; it is zero otherwise. Since we also take into account communication, we assume that two tasks consume energy and time for communication only if they are allocated on two different processors. Variables $R_{pt_1t_2m}$ and $W_{pt_1t_2m}$ take the value of one 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. We assume that tasks running on the same processor do not consume energy and do not spend time in communication for the sake of the optimization problem, while we include the actual minor costs for local communication in execution time and energy for the sake of modeling accuracy. They are input data provided with the task graph.

Any task can be mapped to only one processor and can run at only one speed, i.e.,

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

Moreover, each task reads data (resp. writes data) atomically while executing in a given mode and on a given processor, thus constraining variables $R_{pt_1t_2m}$ and $W_{pt_1t_2m}$

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

Since each write activity corresponds to a related read activity, we have

$$\sum_{p=1}^{P}\sum_{m=1}^{M} (W_{pt_1t_2m} - R_{pt_2t_1m}) = 0 \qquad \forall t_1, t_2.$$

The objective function OF is to *minimize the energy consumption for task execution* E_{comp} and *for task communication* E_{read} and E_{write}

$$E_{\text{comp}} = \sum_{p=1}^{P}\sum_{m=1}^{M}\sum_{t=1}^{\text{T}} X_{ptm} WCN_t t_{clock_m} P_{tm}$$
$$E_{\text{read}} = \sum_{p=1}^{P}\sum_{m=1}^{M}\sum_{t,t1=1}^{\text{T}} R_{ptt_1m} WCN_{Rtt_1} t_{clock_m} P_{tm}$$
$$E_{\text{write}} = \sum_{p=1}^{P}\sum_{m=1}^{M}\sum_{t,t1=1}^{\text{T}} W_{ptt_1m} WCN_{Wtt_1} t_{clock_m} P_{tm}$$
$$OF = E_{\text{comp}} + E_{\text{read}} + E_{\text{write}}$$

where P_{tm} is the power consumed by task t when running in execution mode m and t_{clock_m} is the clock cycle at mode m.

The objective function defined until 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. Therefore, we update the objective function of the master problem with frequency transition (or setup) costs

$$OF_{\text{Master}} = OF + \sum_{p=1}^{P} Setup_p$$

where $Setup_p$ is the cost of frequency switching on processor p. Note that, in the master problem model, the $Setup_p$ variables are not constrained. This is true only in the first iteration of the LBBD algorithm, where all the $Setup_p$ variables are forced to be zero. From the second iteration on, instead, cuts are produced by the subproblem, constraining variables $Setup_p$ such that they might no longer be zero. These cuts will be described in Section VI-D. In addition, for this variable, we can compute a bound using a relaxation of the subproblem. We will explain this relaxation in Section VI-C.

The problem exhibits symmetries. For instance, given a solution with cost C, where tasks $t1$ and $t3$ are allocated on processor P1 while tasks $t2$ and $t4$ are allocated on processor P2, the solution obtained by allocating $t1$ and $t3$ on P2 and $t2$ and $t4$ on P1 does not change the solution cost. To avoid the generation and exploration of symmetric solutions, we add to the model a set of symmetry-breaking constraints. In particular, the first task is always allocated on the first processor. Each task i should be allocated on a processor j only if $j \leq i$. In addition, a task uses a new processor only if it is not mappable to an already-used one.

B. Subproblem Model

For the scheduling part, we use a CP model. Each task t has an associated variable representing its starting time $Start_i$. The duration is fixed since the frequency has been decided in the master problem, 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 to the shared memory and one for reading data from the shared memory). We model the starting time of these activities $StartWrite_{ij}$ and $StartRead_{ij}$. 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 $dRead_{ij}$ depends on the frequency and on the amount of data t_j reads, i.e., $WCN_{R_{ij}}/f_j$. Clearly, the read and write activities are linked to the corresponding task

$$Start_i + duration_i \leq StartWrite_{ij} \qquad \forall j$$
$$StartRead_{ij} + dRead_{ij} \leq Start_j \qquad \forall i.$$

The constraint is not an equality constraint since each task can produce data for and can read from many tasks. Moreover, reads and writes on the same queue are linked from

$$StartWrite_{ij} + dWrite_{ij} \leq StartRead_{ij} \qquad \forall i, j.$$

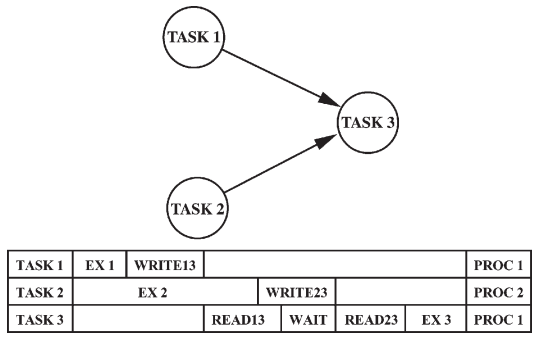

Fig. 3. Example of multiple input data reads and their scheduling in time.

The way reading and writing activities are scheduled heavily depends on the task graph structure. If we restrict our analysis to pipelined task graphs (i.e., dependences among tasks are such that they are logically ordered in a pipeline, as in [2] and [8]), then input data reading activities can be considered tightly coupled with the computation activities of each task. Therefore, tasks writing their output data to shared memory just have their execution time increased by 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 to write) between a task and its successor in the pipeline and f_m is the frequency of the clock when task t is performed. Similarly, tasks reading input data from shared memory have their duration increased by a quantity WCN_R/f_m.

On the contrary, for generic task graphs, a task might need to read multiple input queues before executing, with possible suspensions between the consecutive reading activities, as shown in Fig. 3. Our modeling framework accounts for this general case.

Therefore, we introduce constraints forcing the execution of a task to start immediately after its last reading activity is completed, and the writes of one task to be executed sequentially without intermediate suspensions beginning from the execution completion of that task. For this purpose, we need to introduce two additional activities for each task named $MacroRead_i$ and $MacroWrite_i$. These latter group all the reading and writing activities of the associated task with index i. Durations of these macroactivities can be expressed as (symbol $\rightarrow$ indicates a precedence constraint)

$$dMacroRead_i \geq \sum_{j, j \rightarrow i} dRead_{ij} \qquad \forall i$$
$$dMacroWrite_i = \sum_{j, i \rightarrow j} dWrite_{ij} \qquad \forall i.$$

This leads to new constraints linking communication and execution activities

$$Start_i + duration_i = StartMacroWrite_i \qquad \forall i$$
$$StartMacroRead_i + dMacroRead_i = Start_i \qquad \forall i.$$

In the subproblem, we model precedence constraints in the following way. If tasks 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, instead, the two tasks run on the same processor at different speeds, we should add the time $T_{f_i f_j}$ for switching between the two frequencies

$$Start_i + Duration_i + T_{f_i f_j} \leq Start_j.$$

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

$$Start_i + Duration_i + dWrite_{ij} + dRead_{ij} \leq Start_j.$$

The scheduling engine must also verify that timing and resource requirements are met. As regards timing, the task and processor deadlines are forced with proper constraints. In the simplifying assumption that task and processor deadlines are set to the same value, we just have to check that

$$Start_i + Duration_i \leq Deadline \qquad \forall i$$

otherwise the generalization is straightforward.

Resources are modeled as follows. We have a unary resource constraint for each processor, modeled through a cumulative constraint having, as parameters, a list of all tasks sharing the same resource p, $TaskList_p$, their durations $DurationList_p$, their resource consumption (which is a list of 1 s), and the capacity of the processor, which is one

$$cumulative(TaskList_p, DurationList_p, [1], 1) \qquad \forall p.$$

We model the bus as an additive resource, and we force the system to work under those operating conditions where the model holds. When the bus bandwidth used by concurrent tasks does not exceed an upper threshold, the bus behavior can be abstracted by means of a very simple additive model; the bus delivers an overall bandwidth that is approximatively equal to the sum of the bandwidth requirements of the concurrent tasks. If the upper utilization threshold is exceeded, then the execution times of the tasks are stretched by an unpredictable amount due to bus access contention. Exceeding the threshold does not even pay off in terms of increased bus offered bandwidth, since a saturation effect was observed in [2]. This modeling approach has a number of advantages. First, it avoids the fine granularity modeling of communication in terms of each individual bus transaction. From a scheduling viewpoint, this allows one to model concurrent activities consuming a fraction of the total bus bandwidth for a given time frame. As a result, a larger granularity time unit can be used to solve the scheduling problem, thus reducing the number of variables. Second, the optimizer can schedule communication in such a way that the additive threshold is met. This way, run-time execution times of the tasks will not significantly deviate from those precharacterized at design time in a contention-free regime. We refer to [2] for a detailed description and validation of the bus additive model.

The objective function we want to minimize in the scheduling problem is the setup energy, i.e., the energy spent for frequency switchings

$$\min \sum_{p=1}^{P} Setup_p.$$

For this purpose, we use a matrix of precomputed transition costs $\mathbf{E}$, which reports the energy overhead for switching from frequency f_i (row i) to f_j (column j) and whose diagonal is obviously null. If we indicate with S_p the set of task pairs which

are scheduled consecutively on processor p, then the setup costs can be derived as

$$Setup_p = \sum_{(i,j)\in S_p} E_{f_i f_j} \qquad \forall p.$$

A bound on $Setup_p$ is computed in Section VI-C.

C. Relaxation of the Subproblem

The master problem formulation described in Section VI-A will result in allocations where tasks will potentially run at their lowest frequencies and on the same processor, since task and processor deadlines are not yet accounted for in the master problem. Feeding these allocation solutions to the subproblem solver will most probably result in infeasible schedules, thus leading to a lot of computation-inefficient iterations between the master problem and the subproblem. To avoid this, we introduce relaxations of the subproblem in the master problem model. In other words, we impose that, on each processor, the sum of the time spent for the computation plus the time spent for communication (read and write) should be less than or equal to the deadline of the processor, in order to prevent trivially infeasible solutions

$$\begin{aligned}
T^p_{comp_p} &= \sum_{t=1}^{T}\sum_{m=1}^{M} X_{ptm}\frac{WCN_t}{f_m} \\
T^p_{\text{read}} &= \sum_{t=1}^{T}\sum_{m=1}^{M}\sum_{t=1}^{T} R_{ptt_1m}\frac{WCN_{Rtt_1}}{f_m} \\
T^p_{\text{write}} &= \sum_{t=1}^{T}\sum_{m=1}^{M}\sum_{t=1}^{T} W_{ptt_1m}\frac{WCN_{Wtt_1}}{f_m} \\
T^p_{\text{comp}} + T^p_{\text{read}} + T^p_{\text{write}} &\le dl_p \qquad \forall p.
\end{aligned}$$

In the same way, task deadlines can be captured, which are the same formulas but the final sums are computed for each task.

Note that, to further improve these constraints, we can add a contribution concerning the setup time T_{switch}, i.e., the time spent to switch between two frequencies in the same processor

$$T^p_{\text{comp}} + T^p_{\text{read}} + T^p_{\text{write}} + T^p_{\text{switch}} \le dl_p \qquad \forall p.$$

On T_{switch}, we can only compute a lower bound since the real switching time can be computed once the schedule is known. The idea is the following. If we consider all the task frequencies allocated on a single processor, we know that T_{switch} is at least the sum of all switches minus the greatest switch time. For instance, if frequencies f_1, f_2, and f_3 are allocated on processor $PE0$, we have to sum the minimum time for switching to frequency f_1, f_2, and f_3 minus the maximum of the three. To this purpose, we have defined variables Z_{pf} taking the value of one if the frequency f is allocated at least once on the processor p, and it is zero otherwise. In addition, we can extract from the matrix $\mathbf{T}$ of switching time overheads a vector $\bar{T}$ corresponding to the time for the frequency switches, which will be possibly performed on the processor. The ith element in the vector $\bar{T}$ is the minimum time for switching to frequency i. The lower bound on T_{switch} can be imposed as follows:

$$T^p_{\text{switch}} \ge \sum_{f=1}^{M}\left(Z_{pf}\bar{T}_f - \max_f\{\bar{T}_f | Z_{pf}=1\}\right) \qquad \forall p.$$

Another aspect of the relaxation, which helps in avoiding the computation of suboptimal solutions, concerns the computation of a bound on the switching costs on each processor $Setup_p$. It is computed in the same way described for T_{switch}. This time, however, we have to extract, from the matrix of switching cost overheads $\mathbf{E}$, a vector $\bar{E}$ corresponding to the cost of frequency switches, which will be possibly performed on the processor. The ith element in the vector $\bar{E}$ is the minimum cost for switching to frequency i

$$Setup_p \ge \sum_{f=1}^{M}\left(Z_{pf}\bar{E}_f - \max_f\{\bar{E}_f | Z_{pf}=1\}\right) \qquad \forall p.$$

D. Benders Cuts

At each iteration h, we have an overall problem solution computed as the sum of the optimal solution of the master problem $OF^{(h)}$ and the corresponding optimal solution of the subproblem $Setup^{(h)}$. This solution might not be optimal overall; thus, we have to go on with the iterative process with a new upper bound on the overall problem objective function

$$UB^{(h)} = OF^{(h)} + Setup^{(h)}.$$

The objective function of the master problem at iteration $h+1$ will therefore be constrained as follows:

$$OF^{(h+1)} \le UB^{(h)} - 1.$$

In addition, we generate Benders cuts. The cuts are of two types.

1) If there is no feasible schedule given an allocation, we have to compute a no-good on variables X_{ptm}, avoiding the same allocation to be found again.
2) If a feasible and optimal schedule exists, we cannot simply stop the iteration since the master 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 for a different allocation. These cuts produce a lower bound on the setup costs of the processors.

The procedure converges when the master problem fails to find a solution whose cost is better then the current upper bound.

The first type of cuts are *no-goods*. We call J_p the set of (Task, Frequency) pairs allocated to processor p. We impose

$$\sum_{(t,m)\in J_p} X_{ptm} \le |J_p| - 1 \qquad \forall p.$$

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.

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

$$\begin{aligned}
&Setup_p \ge 0 \\
&Setup_p \ge LB_{Setup_p} \\
&LB_{Setup_p} = Setup^*_p - Setup^*_p \sum_{(t,m)\in J_p}(1 - X_{ptm}).
\end{aligned}$$

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

We have devised even tighter cuts, removing more solutions. However, they complicate the model too much, and our experimental results show that these cuts, even if tighter, do not lead to any advantage in terms of computational time.

VII. Computational Efficiency

We tested the computational efficiency of our hybrid approach on a 2-GHz Pentium 4 machine with 512-Mb RAM and leveraged state-of-the-art professional solving tools, namely, ILOG CPLEX 8.1, ILOG Solver 5.3, and ILOG Scheduler 5.3. We considered two kinds of DVSP instances: 1) instances with a pipelined task graph and 2) instances with a generic task graph, generated using the instance generator presented in [37] producing realistic task graphs.

If n is the number of tasks in the pipeline, after n repetitions, the pipeline is at full rate. In a pipeline with n tasks, we have n execution activities and $2 \times (n-1)$ communication activities (a read and a write for each edge in the graph); we therefore allocate $n + 2 \times (n-1)$ and schedule $n \times (n + 2 \times (n-1))$ activities.

We generated and solved 280 instances with an increasing number of tasks and processing elements. Overall, we had up to 180 activities to schedule in a system with a dozen of pipelined tasks. For all the instances, the optimal solution could be found within 4 min and the algorithm proved to scale quite smoothly for an increasing number of tasks and processing elements. The optimal solution could be found after one iteration in 50% of the cases, and the number of iterations was, at most, five in almost 90% of the cases. This result is due to the tight relaxations added to the master problem model.

We extended our analysis to instances where the task graph is generic; hence, an activity can possibly read data from more than one preceding activity and possibly write data that will be read by more than one subsequent activity. The number of reading and writing activities can become considerably higher, with the higher number of edges in the task graph. We consider here processing elements that can run at six different frequencies. This problem is much harder than the pipelined one, considering that the task graph can have a number of parallel task execution chains and, thus, the macroactivities described in Section VI-B must be considered, complicating and introducing symmetries in the model. Differently from the pipelined instances, we can schedule a single repetition of each task.

Table I summarizes the results. Each line represents an instance that has been solved to optimality. The first three columns contain the number of allocated and scheduled activities (execution + communication data writes and reads) and the number of processing elements considered in the instances. The last two columns represent, respectively, the search time and the number of iterations between the master problem and the subproblem. Each value is the mean over ten instances with the same number of tasks and processing elements. We can see that, typically, the behaviors are similar to those found when solving the pipelined instances; however, we can note some instances where the number of iterations or the search time is notably higher. For example, in the last but two lines, the number of iterations is very high; this is due to the particular structure of the task graph. In fact, it can happen that a high degree of parallelism between the tasks, i.e., a high number of tasks that can execute only after a single task, leads to a number of allocations that are not schedulable. The master problem solver thus looses time proposing to the scheduler a high number of infeasible allocations. On the contrary, in the last line, the number of iterations is low but the search time is extremely high; this is due to the tasks' characteristics that make the scheduling problem very hard to solve.

TABLE I
Search Time and Number of Iterations for Instances With Generic Task Graphs

Activities				
Alloc	Sched	Procs	Time(s)	Iters
8+12	8+12	3	1,48	2
8+12	8+12	3	4,26	6
8+16	8+16	2	1,57	1
8+16	8+16	3	0,81	1
8+16	8+16	4	0,86	1
9+8	9+8	2	2,73	3
9+10	9+10	4	2,60	4
9+12	9+12	4	1,40	3
9+12	9+12	4	2,14	5
9+12	9+12	2	1,11	1
9+16	9+16	3	35,95	43
9+16	9+16	4	29,59	26
9+16	9+16	4	4,84	6
9+20	9+20	3	2,51	1
9+20	9+20	6	158,43	39
9+22	9+22	3	6,62	2
9+24	9+24	2	2,51	1
10+12	10+12	4	0,37	1
10+12	10+12	4	11,50	27
10+16	10+16	3	12,81	3
10+16	10+16	4	13,92	14
10+18	10+18	2	5,90	1
10+18	10+18	3	2,12	1
10+24	10+24	4	4,18	5
12+20	12+20	5	551,92	213
14+22	14+22	2	14,11	1
14+62	14+62	6	3624,81	2

Finally, we intended to compare the computation efficiency of our hybrid approach with that of traditional approaches not leveraging problem decomposition (i.e., the whole mapping problem modeled through IP or CP). However, such a comparison was already reported in [9] for a simpler problem (power consumption was not accounted for, and only pipelined task graphs were supported), where a computation efficiency gap of orders of magnitude was already shown. Considering an upper bound of 15 min for the search time, CP and IP proved capable of finding the optimal solution only for extremely small instances, with a low number of tasks and processing elements, and of finding a solution (not the optimal one) only in 50% of the hard instances, while the hybrid approach solved 100% of the instances to optimality. Given the neat outperformance of LBBD over CP and IP for a much simpler problem than the one addressed in this paper, we do not provide a further comparison between the two approaches for the more complex problem addressed here.

VIII. Design-Time Support

An SW development and optimization flow based on the aforementioned hybrid solver addresses the optimality gap usually incurred by fast exploration frameworks. On the other hand, this flow requires a correspondent design- and run-time support in the target platform, matching the way that the application and the architecture are abstracted in the optimization framework and allowing the precise implementation of computed mapping

solutions. In practice, such a support is needed to close the abstraction gap (i.e., the deviation between the mapping problem model and the real behavior of the target platform), which is the other main objective of this paper.

A. Application and Task Computational Model

Our methodology requires modeling the multitask application to be mapped and executed on top of the target HW platform as a task graph with precedence constraints. The nodes of the graph represent concurrent tasks, while the arcs indicate mutual dependences (communication and/or synchronization).

Task execution is structured in three phases. All input communication queues are read (INPUT); computation activity is performed (EXEC), and finally, all output queues are written (OUTPUT). Each phase consists of an atomic activity. Each task also has two kinds of associated memory requirements.

1) Program data: Storage locations are required for computation data and for processor instructions.
2) Communication queues: The task needs queues to transmit and receive messages to/from other tasks, eventually mapped on different processors.

Program data are allocated on the private memory of each processor, while communication queues reside in scratch-pad memory (in case the communicating tasks run on the same processor) or in shared memory (for remote communications).

B. Customizable Application Template

We set up a generic customizable application template, allowing SW developers to easily and quickly build their parallel applications starting from a high-level task and data flow graph specification. Programmers can, at first, think about their applications in terms of task dependences and quickly draw the task graphs, and then use our tools and libraries to translate the abstract representation into C code. This way, they can devote most of their effort to the functionality of tasks rather than the implementation of their communication, sychronization, scheduling, and energy mode switching mechanisms. Following our scalable and parameterizable template, we also ensure that the final implementation of the target application will be compliant with the modeling assumptions of the optimizer and that the value of the objective function and constraint satisfaction of computed mapping solutions will be achieved in practice.

Fig. 4 shows a pictorial illustration of how our template looks like. Programmers can specify the structure of the target application by simply declaring a series of macros and data structures. In the example, we have shown a task graph with 12 tasks and with precedence constraints defined in the matrix *queue_consumer[][]*. If task i has a precedence constraint w.r.t. task j, the element $queue\_consumer[i][j]$ will be set to one. Developers can also specify information about the configuration of the target HW platform and the desired allocation and schedule, as derived from the optimization tool. *N_CPU macro* specifies the number of available processing cores. The two *task_on_core[]* and *schedule_on_core[][]* data structures specify where tasks should be allocated and which schedule to apply for each core, while with the *task_freq[]* vector, developers can associate an operating voltage/frequency pair to each task.

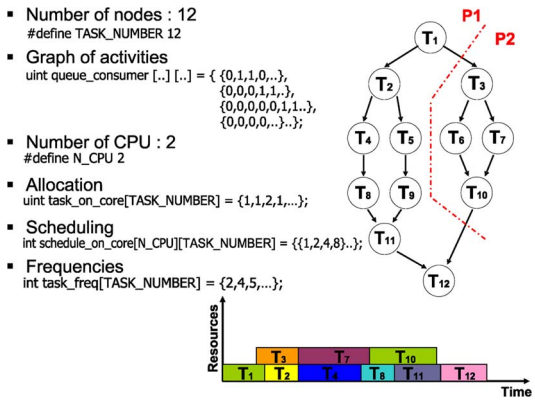

Fig. 4. Example of how to use the customizable application template.

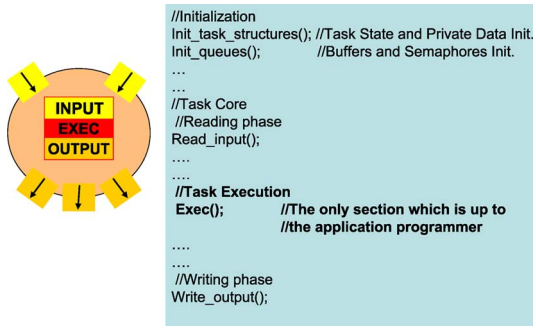

Fig. 5. Task computational model and generated C code.

For every task indicated within the application template, C code is automatically generated. Fig. 5 shows the C code for a task reflecting the considered computation model. At task creation, the task state and private data structures are instantiated and initialized, as well as all buffers and semaphores needed for communication and synchronization. The INPUT phase of the computational model corresponds to the *Read_input()* function, while the OUTPUT phase corresponds to the *Write_output()* one. These two functions are blocking, and they handle the whole communication and synchronization procedures automatically. The only section which is the burden of the programmer is the *Exec()* function; this is the customizable computational core of the task.

IX. Run-Time Support

We implemented a set of application programming interfaces (APIs), with which users can easily reproduce optimizer solutions on their target platform with great accuracy.

A. OS-Independent Allocation and Scheduling Support

Once the target application has been implemented using our generic customizable template, tasks, program data, and communication queues are allocated to the proper HW resources (processor or memory cores), as indicated by the computed allocation solution. This is done through the *init_task* of our template, which allocates and launches all the activities at boot time.

In order to reproduce the exact scheduling behavior of the optimizer, we implemented a scheduling middleware in the target platform. Using this facility, programmers only have to specify the desired scheduling for every processor core, which is handled accordingly by our middleware in a transparent way.

After the boot of the application, our framework sets to active only the first task in the scheduling list, while the other ones are set to the sleep state. In this way, we avoid any undesired task preemption by the operating system scheduler, which would induce a different behavior with respect to the optimal one provided by the optimizer.

After the active task has finished its execution, it is put to sleep, thus releasing the CPU, while the subsequent task in the scheduling list is woken up by switching its state to active. If the subsequent task is allocated to a different CPU, this remote wake-up mechanism is handled via interrupts. Every time a new task is scheduled, our middleware sets its right operating frequency as specified in the application template.

B. Communication and Synchronization Support

SW support for efficient messaging is also provided by our set of high-level APIs. The communication and synchronization library abstracts low-level architectural details to the programmer, such as memory maps or explicit management of HW semaphores or interrupt signaling.

The infrastructure for the communication between a producer–consumer pair is composed by a data queue and by two semaphores. In order to send a message, a producer core writes to the message queue in shared memory.

When the message is ready, the consumer can transfer it from shared memory space to its local scratch pad. Data can be transferred either by the processor itself or by a direct memory access controller, when available. As far as synchronization is concerned, when a producer intends to generate a message, it locally checks an integer semaphore which contains the number of free messages in the queue. If enough space is available, it decrements the semaphore and stores the message in the queue. Availability of the message is signaled to the consumer by remotely incrementing its local semaphore. This single write operation goes through the bus. Semaphores are therefore distributed among the processing elements, resulting in two advantages; the read/write traffic to the semaphores is distributed, and the producer (consumer) can locally poll whether space (a message) is available, thereby reducing bus traffic. Furthermore, our semaphores may interrupt the local processor when released, providing an alternative mechanism to polling. In fact, if the semaphore is not available, the polling task registers itself on a list of tasks waiting for that semaphore and suspends itself. Other tasks on the processor can then execute. As soon as the semaphore is released, it generates an interrupt, and the corresponding interrupt routine reactivates all tasks on the waiting list.

If one task has more than one input or output queue, our optimizer can specify the optimal reading/writing sequence from/to them. We tuned our run-time support to enable this option. This is a very important feature, since an optimal queue-usage ordering can increase the parallelism and thus boost performance. Fig. 6 better clarifies this issue. It shows a case study in which six tasks are allocated to two different processing cores.

Task T1 has to communicate with both T2 and T3, which are allocated to the same core, and with T4 allocated to a different core. At startup, let us assume that task T1 will be scheduled on CPU1 and task T4 on CPU2. While T1 immediately starts its execution, T4 has to wait for data from T1, thus keeping CPU2 stalled. The idle wait of T4 depends on the queue-fill ordering enforced by T1; it will be shorter if T1 gives maximum priority to queue C3. Both our optimization framework and our application execution support can handle this additional degree of freedom for performance optimization.

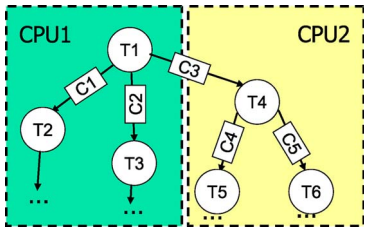

Fig. 6. Optimal queue-usage ordering example.

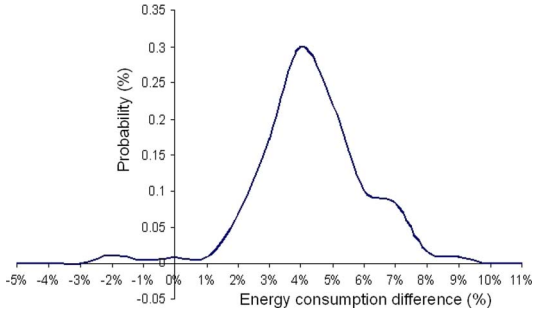

Fig. 7. Distribution of energy consumption differences.

X. Experimental Results

For each task in the input graph, the optimizer needs the execution time, the time required for writing and for reading input data from local memory, and the overhead for writing and reading input data if queues are allocated onto the shared memory in the absence of bus contention. The contribution of cache misses to execution times needs to be considered as well, and contention-free bus accesses can be assumed for this purpose.

For each task graph, this information can be collected with only two simulation runs on a virtual platform. As mentioned in Section III, we used the MPARM platform for complete MPSoC functional simulation with clock-cycle accuracy [5] in SystemC. This modeling and simulation environment was used both to provide input data for the optimization framework and to validate its solutions.

Two types of *validation experiments* were performed, namely: 1) comparison of simulation-based energy and throughput with optimizer-derived values and 2) proof of viability of the proposed approach for real-life demonstrators [global system for mobile communication (GSM) and JPEG].

A. Validation of Optimizer Solutions

We have deployed the virtual platform to implement the allocations, schedules, and frequency assignments generated by the optimizer. A tunable multitask application has been used for this experiment, allowing one to change system and application parameters (local memory size, execution times, data size, real-time requirements, etc.) and to generate the 200 problem instances used for validation. The results of the validation phase are shown in Fig. 7, which shows the distribution of energy deviations. The average difference between the measured and the predicted energy values is 4.80%, with 1.71 standard deviation and 95% statistical significance computed with the Student's t

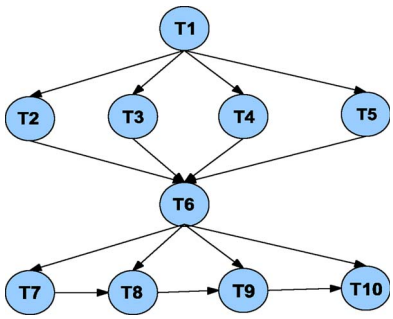

Fig. 8. GSM implementation with generic task graph structure.

distribution. The distribution of throughput deviations was found to be similar and, hence, is not reported here for lack of space. The average difference between the measured and the predicted throughputs was 4.51%, with 1.94 standard deviation and more than 90% statistical significance computed with the Student's t distribution. This confirms the high level of accuracy achieved by the developed optimization framework.

B. Demonstrators

1) GSM Demonstrator: The methodology has been applied to a GSM codec parallelized in two ways. The first variant features ten generic tasks, while the second one consists of six tasks ordered in a logic pipeline. Each task has been precharacterized by the virtual platform to provide task model parameters to the optimizer. After the optimization stage, the solution has been validated on the virtual platform.

Fig. 8 shows the task graph of the first GSM implementation variant. The time taken by the optimizer to come to a solution was 0.2 s, and Table II shows the results of the optimization run. The validation process on the virtual platform proved accuracy by 3.99% on throughput and by 2.91% on energy dissipation.

The results for the pipelined version of the GSM codec has shown an accuracy on the processor energy dissipation, as predicted by the optimizer, by 2%. We used the pipelined version of the GSM demonstrator to explore how the optimizer minimizes energy dissipation of the processor cores with varying real-time requirements, and the results are shown in Fig. 9. The behavior of the optimizer is not specific for the GSM case study but can be extended to all applications featuring timing constraints.

When the deadline is loose, all tasks are allocated to a single processor at the minimum frequency (66 MHz, corresponding to a divisor of three). As the deadline gets tighter, the optimizer prefers to employ a second processor and to progressively balance the load, instead of increasing task frequencies. This procedure is repeated every time a new processor has to be allocated to meet the timing constraints. Only under very tight deadlines, the optimizer leverages increased task frequencies to speed up the system. To the limit, the system works with one task on each processor, although not all tasks run at the maximum frequency. In fact, the GSM pipeline turns out to be unbalanced; therefore, it would be energy inefficient to run the shorter tasks at maximum speed and would not even provide performance benefits. As a result, the optimizer determines the most energy-efficient configuration that provides the best performance. The problem becomes infeasible if more stringent deadlines than 710 ns are required.

2) JPEG Demonstrator: Our methodology was then applied to a JPEG decoder, which was partitioned in four pipelined tasks: Huffman dc decoding, Huffman ac decoding, inverse quantization, and inverse discrete cosine transform. Each stage processes an 8×8 block, amounting to an exchange of 1024 b among pipeline stages. The accuracy of the energy estimation given by the optimizer was found to be within 3.1% with respect to functional simulation. In contrast to pipelined GSM, user requirements on a JPEG decoding usually consist of the minimization of the execution time and not of a deadline to be met. However, a performance–energy conflict arises, and two approaches for the allocation and scheduling of a JPEG decoder task graph are feasible. On one hand, the designer could be primarily interested in reducing execution time at the cost of increased energy. On the other hand, the primary objective function could be the minimization of energy dissipation, whatever the decoding performance. This tradeoff has been investigated with the optimizer, and the Pareto-optimal frontier in the performance–energy space is shown in Fig. 10. The constraint on the execution time on the x-axis has been translated into a constraint on the block decoding time. The curve is not linear since there is a discrete number of voltage–frequency pairs.

As we can observe for a large range of deadlines, the optimizer is good at improving system performance without significantly changing processor energy dissipation. This is done by using one or two processors, changing the allocations and using high-frequency dividers. Beyond 200 ns, the optimizer is forced to use low-frequency dividers, thus causing the energy to skyrocket. Interestingly, the increase of the task frequency is preferred to an increase of the number of processors, since the communication energy would involve even higher total energy consumption. This behavior is different from the one seen for the GSM, since this time, the computation–communication ratio is lower than for GSM due a larger size of exchanged messages.

XI. Optimal Versus Heuristic Approaches

In this section, we illustrate a comparison of mapping and frequency/voltage assignment solutions generated by our complete solver with those provided by a heuristic approach leveraging genetic algorithms.

The heuristic was derived from [4], which extends previous work in [29]. Its optimization flow is split in three parts:

1) genetic task allocation optimization;
2) genetic schedule optimization;
3) optimal frequency selection.

Originally, the approach presented in [4] associates a communication task that has to be scheduled for each message exchanged over the bus. In order to have a fair comparison with our approach, we have implemented the additive bus model used in this paper.

In the genetic task allocation approach, solution candidates are encoded into allocation strings. Each gene in these strings describes a candidate allocation of a task to a processor. One key feature of this algorithm is the invocation of a genetic scheduling procedure for each allocation candidate. The genetic scheduling algorithm finds, for a given allocation, the schedule that meets all the task deadlines and, furthermore, has the

TABLE II
MAPPING SOLUTION FOR THE GSM ENCODER

Deadline (ns)	# of Proc.	Allocation of Tasks to Core	Task Freq. Divider	Energy Consump. (nJ)
5000	2	1,1,1,1,2,1,1,2,2,2	1,2,2,2,4,1,1,1,1,1	4784

Deadline (ns)	Number of Processors	# Task Allocated on Core	Task Frequency Divider	Energy Consumption (nJ)
6000	1	1, 1, 1, 1, 1, 1	3, 3, 3, 3, 3, 3	5840
5500	2	2, 1, 1, 1, 1, 1	3, 3, 3, 3, 3, 3	5910
5000	2	1, 1, 1, 1, 1, 2	3, 3, 3, 3, 3, 3	5938
4500	2	1, 1, 1, 1, 2, 2	3, 3, 3, 3, 3, 3	5938
4000	2	1, 1, 1, 2, 2, 2	3, 3, 3, 3, 3, 3	5938
3500	2	1, 1, 1, 2, 2, 2	3, 3, 3, 3, 3, 3	5938
3000	3	1, 2, 2, 3, 3, 3	3, 3, 3, 3, 3, 3	6008
2500	3	1, 1, 2, 2, 3, 3	3, 3, 3, 3, 3, 3	6039
2000	4	1, 2, 3, 3, 4, 4	3, 3, 3, 3, 3, 3	6109
1500	6	1, 2, 3, 4, 5, 6	3, 3, 3, 3, 3, 3	6304
1000	6	1, 2, 3, 4, 5, 6	3, 2, 2, 2, 3, 2	6807
900	6	1, 2, 3, 4, 5, 6	3, 1, 2, 2, 2, 2	9834
750	6	1, 2, 3, 4, 5, 6	2, 1, 2, 2, 2, 2	9934
730	6	1, 2, 3, 4, 5, 6	2, 1, 1, 2, 2, 2	12102
710	6	1, 2, 3, 4, 5, 6	2, 1, 1, 1, 2, 2	14193

Fig. 9. Behavior of the optimizer with varying real-time requirements. Allocation is given as an array indicating the processor identification on which each task is mapped. Similarly, the frequency of each task is expressed in terms of the integer divider of the baseline frequency. Only three dividers are used for this example.

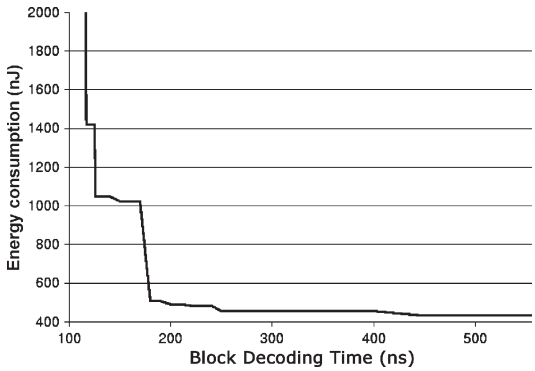

Fig. 10. Pareto-optimal frontier in the performance–energy design space.

minimum energy. We deployed one of the most widely used heuristic approaches to scheduling, namely, list scheduling.

The algorithm proceeds by passing the schedule to a voltage selection algorithm that identifies the task voltages minimizing the energy dissipation. After performing the voltage selection, the fitness F_S of each schedule candidate (capturing total power) is calculated.

During the genetic scheduling step, a fast voltage scaling heuristic is used. However, once the genetic algorithms are completed, we use the optimal frequency scaling algorithm presented in [3] (extending [17]), restricted to select one single frequency for each task. Consequently, if the genetic heuristic method finds the best allocation and schedule, after the last frequency selection step, we will obtain a globally optimal result, which is identical to the one produced by the approach proposed in this paper.

The complete and the heuristic algorithms have been compared on 70 different problem instances divided into seven groups based on their structure (i.e., number of tasks and branching factor) and a number of tasks from 10 to 20. The structure of some task graphs and their annotated values reflect those of parallelized real-life applications in the signal-processing domain (such as matrix multiply, fast Fourier transform, data encryption standard encryption, and finite-impulse response filtering), which were then taken as the starting point for parametric explorations.

The deviation of execution times and energies obtained with the heuristic methods with respect to the optimal solver is shown in Table III. We found that the heuristic approach never provides the optimal solution for the problem instances under test. Nevertheless, in 70% of the cases, the energy consumption difference is within 5%; however, in 10% of the cases, it is extremely high (up to 44%).

By setting loose search stopping criteria for the heuristic method (thus giving it more time to optimize the solution), we allowed this latter to take a search time comparable with that of our technique. In spite of this, our complete method is able to find optimal solutions that the heuristic algorithm is not able to find.

In the second experiment we performed, we solved a common problem instance while varying the real-time constraint, i.e., we varied the task deadline value from a very loose one (allowing all tasks to run at the lowest speed on the same processor) to the tightest one.

Results are shown in Fig. 11. We report the energy consumption (y-axis) of the solutions found by the complete and the heuristic approach as a function of the deadline (x-axis). We can see that the heuristic solution is never the optimum, even when the real-time constraint is weak. In fact, the relative difference for the loosest deadline value is equal to 0.55%. Such difference then grows as the real-time constraint becomes tighter, and for deadline values lower than 3 ms (on the left of the black vertical line in Fig. 11), the heuristic approach is not even able to find a solution, while the complete solver finds that the lowest possible deadline value is around 2.1 ms. We find this capability of our solver to extend the range of problem feasibility very interesting from an application viewpoint.

XII. CONCLUSION

In this paper, we addressed both the optimality and the abstraction gaps which impair the results of traditional SW optimization flows for on-chip multiprocessor systems. On one hand, we present a cooperative framework to solve the allocation, scheduling, and voltage/frequency selection problems to optimality for energy-efficient MPSoCs. This iterative framework is based on LBBD and provides optimal solutions at an affordable search time, orders of magnitude shorter than traditional IP or CP solvers. On the other hand, we set up a design-time and a run-time support for the target MPSoC platform allowing to specify applications while matching optimizer modeling assumptions and to accurately implement its mapping solutions.

APPENDIX
PROOF OF CONVERGENCE OF THE ALGORITHM

We prove here that the LBBD algorithm we propose converges to the optimal solution.

First, we show that the algorithm converges without using the bound on the switching cost of each processor $Setup_p$ introduced in Sections VI-C and D.

TABLE III
COMPARISON BETWEEN COMPLETE AND HEURISTIC ALGORITHMS

	Execution time			Energy consumption		
Group #	Optimal (ns)	Heuristic (ns)	Diff. (%)	Optimal (nJ)	Heuristic (nJ)	Diff. (%)
1	13666160	16301020	19.28	23292418	23733705	1.89
2	7856620	8648980	10.08	12961388	13158730	1.52
3	11561360	13651545	18.07	24899620	35795990	43.76
4	5612801	5584947	-0.49	12995149	13129346	1.03
5	6677140	6547810	-1.93	10934681	11731834	7.29
6	8433960	10889200	29.11	12461688	12539983	0.62
7	5093060	5100401	0.01	21638232	29224058	0.013

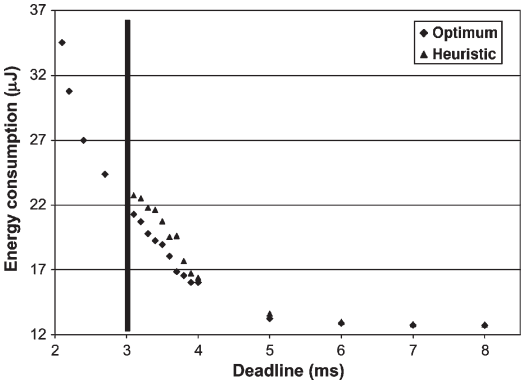

Fig. 11. Comparison between optimal and heuristic solutions.

The objective function of the master problem counts two terms

$$OF_{\text{Master}} = OF + \sum_{p=1}^{P} Setup_p.$$

Without any bound on $Setup$ variables, every $Setup_p$ is set to zero since the IP solver wants to minimize the overall objective function. Therefore, the master problem obtains a lower bound LB on the overall problem objective function.

This solution is passed to the subproblem solver that computes the optimal setups for the given allocation. If the subproblem is infeasible, a no-good is added to the master so as to avoid the generation of the solution just found. If, instead, the subproblem is solved, we obtain a feasible solution for the overall problem, which is a valid upper bound UB. The optimal solution is somewhere between LB and UB.

The convergence is ensured by the constraint that imposes that the solution just found at iteration h in the master problem is discarded (i.e., made infeasible) at iteration $h+1$ as an effect of the first type of Benders cut. Let J_p be the set of couples (Task, Frequency) allocated to processor p. We impose that the current solution at iteration h is no longer feasible in iteration $h+1$

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

Given the new constraint, the OF of the master problem cannot decrease since the solution space is narrower and the optimal solution at previous iteration has been discarded.

Since the number of solutions of the problem is finite, the algorithm converges. The optimality is proved when the master problem cannot find a solution whose allocation cost is lower than the current upper bound.

The use of the relaxation introduced in Section VI-C and the second type of Benders cuts simply help in speeding up the process. In fact, both the relaxation and the second type of Benders cut estimate an optimistic switching cost for a given allocation. If a new allocation plus this optimistic cost are worse than the current upper bound, we have proved optimality.

REFERENCES

[1] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numerische Mathematik*, vol. 4, no. 1, pp. 238–252, Dec. 1962.
[2] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano, "Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip," in *Proc. DATE*, 2006, pp. 3–8.
[3] A. Andrei, P. Eles, Z. Peng, M. Schmitz, and B. M. Al-Hashimi, "Energy optimization of multiprocessor systems on chip by voltage selection," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 3, pp. 262–275, Mar. 2007.
[4] M. T. Schmitz, P. Eles, and B. M. Al-Hashimi, *System-Level Design Techniques for Energy-Efficient Embedded Systems*. Norwell, MA: Kluwer, 2004.
[5] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon, "Analyzing on-chip communication in a MPSoC environment," in *Proc. DATE*, 2004, pp. 752–757.
[6] I. E. Grossmann and V. Jain, "Algorithms for hybrid MILP/CP models for a class of optimization problems," *INFORMS J. Comput.*, vol. 13, no. 4, pp. 258–276, 2001.
[7] J. N. Hooker, "A hybrid method for planning and scheduling," in *Proc. 10th Int. Conf. Principles Practice Constraint Program.—CP*, Sep. 2004, pp. 305–316.
[8] M. Ruggiero, G. Pari, A. Guerri, L. Benini, M. Milano, D. Bertozzi, and A. Andrei, "A cooperative, accurate solving framework for optimal allocation, scheduling and frequency selection," in *Proc. Int. Symp. SOC*, 2006, pp. 183–186.
[9] M. Ruggiero, A. Guerri, D. Bertozzi, M. Milano, and L. Benini, "A fast and accurate technique for mapping parallel applications on stream-oriented MPSoC platforms with communication awareness," *Int. J. Parallel Program.*, vol. 36, no. 1, pp. 3–36, Feb. 2008.
[10] K. J. Nowka, G. D. Carpenter, E. W. MacDonald, H. C. Ngo, B. C. Brock, K. I. Ishii, T. Y. Nguyen, and J. L. Burns, "A 32-bit PowerPC system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1441–1447, Nov. 2002.
[11] J. C. Régin, "A filtering algorithm for constraints of difference in CSPs," in *Proc. 12th Nat. Conf. Artif. Intell.*, 1994, vol. 1, pp. 362–367.
[12] P. Baptiste, C. Le Pape, and W. Nuijten, *Constraint-Based Scheduling*, ser. International Series in Operations Research and Management Science, vol. 39. New York: Springer-Verlag, 2001.
[13] E. P. K. Tsang, *Foundation of Constraint Satisfaction*. New York: Academic, 1993.
[14] K. Hirata and J. Goodacre, "ARM MPCore: The streamlined and scalable ARM11 processor core," in *Proc. ASP-DAC*, 2007, pp. 747–748.
[15] K. Nose, A. Shibayama, H. Kodama, M. Mizuno, M. Edahiro, and N. Nishi, "Deterministic inter-core synchronization with potentially all-in-phase clocking for low-power multi-core SoCs," in *Proc. Int. Solid-State Circuits Conf., Dig. Tech. Papers*, 2005, pp. 296–299.
[16] *Freescale Technologies for Energy Efficiency: 2007 Overview*, 2007. White Paper.
[17] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. Al-Hashimi, "Overhead-conscious voltage selection for dynamic and leakage power reduction of time-constraint systems," in *Proc. DATE*, 2004, pp. 518–523.
[18] Y. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, Dec. 1999.

[19] P. Eles, Z. Peng, K. Kuchcinski, A. Doboli, and P. Pop, "Scheduling of conditional process graphs for the synthesis of embedded systems," in *Proc. DATE*, 1998, pp. 132–139.
[20] N. Ventroux, F. Blanc, and D. Lavenier, "A low complex scheduling algorithm for multi-processor system-on-chip," in *Proc. Parallel Distrib. Comput. Netw.*, 2005, pp. 540–545.
[21] J. Axelsson, "Architecture synthesis and partitioning of real-time synthesis: A comparison of 3 heuristic search strategies," in *Proc. CODES/CASHE*, 1997, pp. 161–166.
[22] J. N. Hooker and G. Ottosson, "Logic-based Benders decomposition," *Math. Program.*, vol. 96, no. 1, pp. 33–60, Apr. 2003.
[23] M. Milano, Ed., *Constraint and Integer Programming: Toward a Unified Methodology (Operations Research/Computer Sciences Interfaces)*. Norwell, MA: Kluwer, 2003.
[24] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. IEEE Symp. Found. Comput. Sci.*, 1995, pp. 374–382.
[25] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proc. ISLPED*, 1998, pp. 197–202.
[26] F. Xie, M. Martonosi, and S. Malik, "Bounds on power savings using runtime dynamic voltage scaling: An exact algorithm and a linear-time heuristic approximation," in *Proc. ISLPED*, 2005, pp. 287–292.
[27] R. Jejurikar and R. Gupta, "Dynamic slack reclamation with procrastination scheduling in real-time embedded systems," in *Proc. Des. Autom. Conf.*, 2005, pp. 111–116.
[28] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.
[29] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Iterative schedule optimization for voltage scalable distributed embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 3, no. 1, pp. 182–217, Feb. 2004.
[30] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints," in *Proc. DATE*, 2004, pp. 234–239.
[31] L. Benini, D. Bertozzi, A. Guerri, and M. Milano, "Allocation and scheduling for MPSoCs via decomposition and no-good generation," in *Proc. IJCAI*, 2005, pp. 1517–1518.
[32] F. Gruian and K. Kuchcinski, "LEneS: Task scheduling for low-energy systems using variable supply voltage processors," in *Proc. ASP-DAC*, 2001, pp. 449–455.
[33] S. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *Proc. ICCAD*, 2002, pp. 721–725.
[34] L. F. Leung, C. Y. Tsui, and W. H. Ki, "Minimizing energy consumption of multiple-processors-core systems with simultaneous task allocation, scheduling and voltage assignment," in *Proc. ASP-DAC*, 2004, pp. 647–652.
[35] F. Poletti and A. Poggiali, "Flexible hardware/software support for message passing on a distributed shared memory architecture," in *Proc. DATE Conf.*, 2005, pp. 736–741.
[36] M. Ruggiero, A. Acquaviva, D. Bertozzi, and L. Benini, "Application-specific power-aware workload allocation for voltage scalable MPSoC platforms," in *Proc. ICCD*, 2005, pp. 87–93.
[37] A. Guerri, M. Lombardi, and M. Milano, "Challenging scheduling problems in the field of system design," in *Proc. ICAPS*, 2007. [Online]. Available: http://abotea.rsise.anu.edu.au/satellite-events-icaps07/workshop4/paper08.pdf

Martino Ruggiero received the M.S. degree in electrical engineering and the Ph.D. degree from University of Bologna, Bologna, Italy, in 2004 and 2007, respectively.

He is currently with the Electronics, Computer Sciences, and Systems Department, University of Bologna, where he holds a postdoctoral position. His research interests include embedded system architecture and software (SW) design, with particular emphasis on low-power SW and architecture design for ultraportable devices; distributed and parallel computing; development of a simulation environment at different levels of abstraction for multiprocessor systems-on-chip; application partitioning for parallel architectures; and complete algorithmic techniques for mapping and scheduling.

Davide Bertozzi received the Ph.D. degree in electrical engineering from the University of Bologna, Bologna, Italy, in 2003.

He was a Visiting Researcher with Stanford University, Stanford, CA; NEC Research America, Princeton, NJ; Philips Research Labs, Eindhoven, The Netherlands; and Samsung Electronics, Korea. He is currently an Assistant Professor with the Engineering Department, University of Ferrara, Ferrara, Italy. His main research interest includes multiprocessor system-on-chip design, with an orthogonal focus ranging from resource management to physical design issues. He is currently very active in the domain of network-on-chip design.

Luca Benini (S'94–M'97–SM'04–F'06) received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1997.

He is currently a Full Professor with the Electronics, Computer Sciences, and Systems Department, University of Bologna, Bologna, Italy, and also a Consulting Research Professor with the Belgian Interuniversity MicroElectronics Centre, Leuven, Belgium. He also holds a Visiting Faculty position with the Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland. His research interest includes design of system-on-chip platforms for embedded applications. He is also active in the area of energy-efficient smart sensors and sensor networks, including biosensors and related data mining challenges. He has published more than 400 papers in peer-reviewed international journals and conferences, four books, and several book chapters.

Dr. Benini is a member of the steering board of the ARTEMISIA European Association on Advanced Research and Technology for Embedded Intelligence and Systems.

Michela Milano received the Ph.D. degree in electronics, computer science and telecommunication from the Electronics, Computer Sciences, and Systems Department, University of Bologna, Bologna, Italy, in 1998.

She is currently an Associate Professor of computer science with the Electronics, Computer Sciences, and Systems Department, University of Bologna. Her research interests include combinatorial optimization and the integration between constraint and integer programming. In this field, she is a member of the program committee of major conferences, Area Editor of *Constraint Programming Letters*, Associate Editor of the Institute for Operations Research and the Management Sciences *Journal of Computing*, and a member of the Editorial Board of *Constraints*. She is the Editor of *Constraint and Integer Programming: Toward a Unified Methodology* (Kluwer, 2003).

Alexandru Andrei (S'03) received the M.S. degree from Politehnica University Timisoara, Timisoara, Romania, in 2001 and the Ph.D. degree in computer engineering from Linköping University, Linköping, Sweden, in 2007.

His research interests include low-power design, real-time systems, and hardware–software codesign. He is currently with Ericsson AB, Stockholm, Sweden, and also with the Embedded Systems Laboratory, Linköping University.