Sharing resources for performance and energy optimization of concurrent streaming applications

Anne Benoit, Paul Renaud-Goud, Yves Robert
LIP, ENS Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France
UMR 5668 - CNRS - ENS Lyon - UCB Lyon - INRIA
{Anne.Benoit|Paul.Renaud-Goud|Yves.Robert}@ens-lyon.fr

***Abstract*—We aim at finding optimal mappings for concurrent streaming applications. Each application consists of a linear chain with several stages, and processes successive data sets in pipeline mode. The objective is to minimize the energy consumption of the whole platform, while satisfying given performance-related bounds on the period and latency of each application. The problem is to decide which processors to enroll, at which speed (or mode) to use them, and which stages they should execute. We distinguish two mapping categories, interval mappings without reuse, and fully arbitrary general mappings. On the theoretical side, we establish complexity results for this tri-criteria mapping problem (energy, period, latency). Furthermore, we derive an integer linear program that provides the optimal solution in the most general case. On the experimental side, we design polynomial-time heuristics, and assess their absolute performance thanks to the linear program. One main goal is to evaluate the impact of processor sharing on the quality of the solution.**

***Index Terms*—mapping; concurrent streaming applications; heterogeneous platforms; resource sharing; energy; latency; period.**

I. Introduction

In this paper, we aim at optimizing the parallel execution of several pipelined applications on a given platform. Such streaming applications are ubiquitous in streaming environments, as for instance video and audio encoding and decoding, DSP applications, image processing, and so on ([1], [2], [3], [4], [5]). For each application, a sequence of data sets enters the input stage and progresses from stage to stage at a fixed rate until the final result is computed. Each stage has its own communication and computation requirements: it reads an input from the previous stage, processes the data and outputs a result to the next stage. A new data set enters the system each application period, and results are output at the same periodic interval.

The objective is to minimize the energy consumption of the whole platform, while satisfying given performance-related bounds on the period and latency of each application. This multi-criteria approach targets a trade-off between the users and the platform manager. The formers have specific requirements for their applications, while the latter has crucial economical and environmental constraints. Indeed, the energy saving problem is becoming increasingly important, not only because of the sole cost of energy, but also because of the cost of cooling systems and related infrastructures. To help reduce energy costs, modern computing centers provide multi-modal processors: each processor has a discrete number of predefined speeds (or modes), which correspond to different voltages that the processor can be subjected to. The power consumption is the sum of a static part (the cost for a processor to be turned on) and a dynamic part. This dynamic part is a strictly convex function of the processor speed, so that the execution of a given amount of work costs more energy if a processor runs in a higher mode [6]. On the one side, faster modes allow for fulfilling the performance criteria, and on the other side, they lead to a higher energy consumption.

The main performance-oriented criteria for pipelined applications are period and latency. The period of an application is the inverse of the throughput, i.e., it corresponds to the time interval between the arrival of two consecutive data sets. The period is fixed by the applicative setting, and we must ensure that data sets are processed fast enough so that there is no accumulation of data sets in the pipeline. The latency of an application is the time elapsed between the beginning and the end of the execution of a given data set, hence it measures the response time of the system to process the data set entirely. These two criteria alone already are antagonistic. The smallest latency is obtained when no communication occurs, i.e., when the same processor executes all the stages of an application. However, such a mapping may well exceed the bound on the period, since the same processor must process an entire application. Adding energy consumption as a third criterion renders everything even more complex. Obviously, energy is minimized by enrolling a single processor for all applications, namely the one with the smallest mode available; but such a mapping would most certainly exceed period and latency bounds.

This work is a follow-on of [7], where we have provided a comprehensive analysis of various instances of this tri-criteria optimization problem. However, the mapping rules and performance models used in this paper are different. In a nutshell, a comprehensive assessment of *one-to-one* and *interval mappings* is given in [7]. Such mappings restrict the assignment of stages to processors: each enrolled resource can execute only a single stage (one-to-one mapping) or a set of consecutive stages (interval mapping) of a given application. Therefore no inter-application reuse of resources is authorized. While prohibiting such a reuse does make sense in some situations (e.g., for security reasons), it is also very likely to waste resources and to increase energy consumption. Indeed, without reuse, more processors are enrolled, hence the static energy gets higher, and these processors cannot benefit from a good load balancing of computation costs across applications, hence a worse resource utilization. From the platform manager's point of view, resource sharing among (non critical) applications is a key ingredient to efficiently servicing several users.

In this paper, we investigate the impact of resource sharing on the quality of the solution with respect to the three optimization criteria (energy, period and latency). We thus deal with *general* mappings where application stages can arbitrarily be assigned to processors. Unfortunately, general mappings come with a price, that of intricate scheduling problems for period and latency: even when the mapping is given, scheduling the execution is a problem of combinatorial nature [8].With general mappings, a processor typically has several incoming and/or outgoing communications, and it is difficult to orchestrate these operations so as to minimize conflicting objectives such as period and latency. Therefore, we focus in this paper on the problem in which bounds on period and latency are fixed by the application designer, and we relax the definition of the latency using the approach of Hary and Ozguner [3]. Instead of computing the longest path, we approximate the latency L as $L = (2m - 1)P$, where P is the period, i.e., the rate at which data sets enter the system, and m is the number of intervals of consecutive stages mapped onto a same processor in the mapping. A processor change occurs each time when a stage and its successor are not mapped onto the same processor, i.e., $m-1$ times. The intuition is that the whole application is executed synchronously, and each data set progresses concurrently within a period. With m successive computations and $m-1$ processor changes (i.e., communications), each data set traverses the platform within $2m - 1$ periods. We adopt the model of [3] throughout the paper, and refer to Section III for further details on mapping rules and objectives. The problem can then be defined as follows: given a period P_a and a bound on the latency L_a for each application a, find a mapping which consumes the minimum amount of energy, while satisfying the performance constraints: application a is processed at a period P_a, and its latency is not greater than L_a; in other words, for application a, the number m_a does not exceed $\lfloor (L_a/P_a + 1)/2 \rfloor$.

A first contribution of this paper is to provide complexity results for the tri-criteria optimization problem under the resource-sharing model. We restrict to homogeneous platforms whose processors have identical modes; otherwise

even the simplest mono-criterion problem, namely period minimization for a single application mapped onto homogeneous and uni-modal processors, paying no communication cost, is NP-complete (straightforward reduction from 2-PARTITION [9]. We show that the problem is polynomial for interval mappings on homogeneous platforms with Hary and Ozguner's model, while it was NP-complete with the longest path model [7], thereby demonstrating the impact of the model for the latency. We also show that the tri-criteria problem becomes NP-complete for general mappings on homogeneous platforms. Another contribution is to evaluate the impact of resource sharing, by comparing the quality of interval mappings and of general mappings. To this end, we design a set of polynomial-time heuristics, with and without reuse, and we experimentally compare their performance on a large set of experiments. We also evaluate the absolute performance of the heuristics on small problem instances, by comparing the solutions of the heuristics to the optimal solution obtained with an integer linear program.

The paper is organized as follows. We first review related work in Section II. The framework is described in Section III, then we provide complexity results in Section IV. In Section V, we design several polynomial-time heuristics to provide polynomial-time solutions to the tri-criteria problem; and finally we study their relative performance, and their absolute performance with respect to the integer linear program, in Section VI. We conclude in Section VII.

II. RELATED WORK

In this paper, we use the Dynamic Voltage Scaling (DVS) technique in order to adjust energy consumption. DVS has been extensively studied in several papers, for mapping onto a single-core processor, a multi-core processor, or a set of processors.

Slack reclamation techniques are used for frame-based hard real-time embedded system in [10]: a set of independent tasks, provided with their WCEC (Worst Case Execution Cycle) and sharing a common deadline, has to be mapped onto a processor. If a task needs less cycles than its WCEC, the dynamically obtained slack allows the processor to run at a lower frequency and therefore to spare energy. This work is extended in [11], where the energy model includes time and energy penalties when the processor frequency is changing. Those transition overheads are also taken into account in [12], but tasks are interdependent.

Then [13] maps applications which consists of a program modeled with a sequential part and another part which can be parallel, onto a multi-core processor. Bunde [14] focuses on the problem of offline scheduling unit time tasks with release dates, while minimizing the makespan or the total flow time on one processor. He extends this work from one processor to multi-processors.

Authors in [15] study the problem of scheduling real-time tasks on two heterogeneous processors. They provide a FPTAS to derive a solution very close to the optimal energy consumption with a reasonable complexity, while in [16], the authors design heuristics to map a set of dependent tasks with deadlines onto a set of homogeneous processors, with the possibility of changing a processor speed during the execution of a task. [17] proposes a greedy algorithm based on affinity to assign frame-based real-time tasks, and then they re-assign them in pseudo-polynomial time when any processing speed can be assigned for a processor. In [18] leakage energy is the focus for mapping applications represented as DAGs. In [19], the authors are interested about scheduling task graphs with data dependencies while minimizing the energy consumption of both the processors and the inter-processor communication devices, while assuming the communication times are negligible compared to the computation times.

All these problems are quite different from ours, since we focus on pipelined applications of infinite duration, thus considering power instead of total energy consumption. Due to the streaming nature of the applications, we do not allow for changing the processor speed during execution.

III. Framework

Applicative framework. We consider A application workflows ($A \geq 1$) to be executed concurrently; each application operates on a collection of data sets that are executed in a pipeline fashion. For $1 \leq a \leq A$, application a consists in n_a stages, and for $1 \leq k \leq n_a$, we denote by $\mathcal{S}_a^k$ the k-th stage of application a. Stage $\mathcal{S}_a^k$ receives an input data of size δ_a^{k-1}, performs w_a^k computations, and finally outputs a data of size δ_a^k. A new data set enters the system every P_a time-units; P_a is the period of application a. The total number of stages is $N = \sum_{a=1}^{A} n_a$.

Target platform. The platform is composed of p processors, which are fully interconnected; there is a bidirectional link between any processor pair $\mathcal{P}_u$ and $\mathcal{P}_v$, of bandwidth $b_{u,v}$. For simplification, we assume that $2A$ additional processors $\mathcal{P}_{\mathsf{in}_1}, \dots, \mathcal{P}_{\mathsf{in}_A}$ and $\mathcal{P}_{\mathsf{out}_1}, \dots, \mathcal{P}_{\mathsf{out}_A}$ are devoted to input/output operations of the applications (in fact these additional processors are virtual processes that may well be shared by the same physical resource). Initially, for each $a \in \{1, \dots, A\}$, the input data for each task of application a resides on $\mathcal{P}_{\mathsf{in}_a}$, while all results must be returned to and stored on $\mathcal{P}_{\mathsf{out}_a}$. These special processors are connected to the p processors of the platform.

We use a linear cost model for communications; hence it takes $X/b_{u,v}$ time-units for $\mathcal{P}_u$ to send (resp. receive) a message of size X to (resp. from) $\mathcal{P}_v$. In addition to link bandwidths, we have processor network cards that bound the total communication capacity of each computing resource (bounded multi-port model with overlap [20]). We denote by B_u^{in} (resp. B_u^{out}) the capacity of the input (resp. output) network card of processor $\mathcal{P}_u$. In other words, $\mathcal{P}_u$ cannot receive more than B_u^{in} data items per time-unit, and it cannot send more than B_u^{out} data items per time-unit, but several communications along different links can take place simultaneously (provided that the link bandwidths are not exceeded either). In addition, independent communications and computations can overlap. It has been pointed out that multi-threaded communication libraries such as MPICH2 [21] now allow for initiating multiple concurrent send and receive operations, thereby providing practical realizations of the multi-port model [22].

In this paper, we mainly target *communication-homogeneous* platforms, with identical communication devices for each processor: all link bandwidths are identical ($b_{u,v} = b$ for $1 \leq u, v \leq p$), and all network cards are identical ($B_u^{in} = B^{in}$, $B_u^{out} = B^{out}$ for all $1 \leq u \leq p$). However, the linear program also applies to heterogeneous platforms as well.

As stated above, processors are multi-modal. Each processor $\mathcal{P}_u$ is associated with a set $\mathsf{S_u}$ of speeds, or modes: $\mathsf{S_u} = \{s_{u,1}, \dots, s_{u,m_u}\}$. To ease notations, we add a special mode 0 in which the processor is inactive: $s_{u,0} = 0$. *Speed-homogeneous* platforms have processors of identical speeds, i.e., they share a common speed set ($\mathsf{S_u} = \mathsf{S}$ for $1 \leq u \leq p$); we further assume that they are communication-homogeneous, so that they represent typical parallel machines. *Speed-heterogeneous* platforms also are communication-homogeneous, but they have different-speed processors ($\mathsf{S_u} \neq \mathsf{S_v}$). They correspond to networks of workstations with plain TCP/IP interconnects or other LANs.

Mapping strategies. The mapping is an allocation function, which associates a processor number to each stage number, as well as a speed at which each processor is running. For *general mappings with processor reuse*, there is no constraint on the allocation function. We must carefully decide how the speed of each processor is shared among all stages it is assigned to. Similarly, a communication link or processor network card may be involved in several communications, which implies to sharing bandwidths and card capacities, too. Hence the question is the following: given the mapping, and a threshold period P_a and latency L_a for each application $a \in \{1, \dots, A\}$, is it possible to determine which fraction of computing and communicating resources to assign to each operation so that all period and latency thresholds are met?

Recall that we consider the latency model described in [3], in which one period is accounted for each computation of an interval of stages and for each inter-processor communication. We observe that given the mapping, we know m_a, the number of intervals ($m_a - 1$ processor changes), for each application a. We can thus check immediately whether the bounds on the latency are respected, i.e., $(2m_a - 1)P_a \leq L_a$ for $a \in \{1, \dots, A\}$.

Now for the periods, the key idea is to distribute platform resources parsimoniously, and to allocate only the needed CPU fraction to each computation, and the needed bandwidth fraction to each communication, so that the period constraint is fulfilled. The mapping is valid if neither processor speeds, nor link bandwidths, nor network card capacities are exceeded. First, we merge consecutive stages $[\mathcal{S}_a^i, \dots, \mathcal{S}_a^j]$ of application a mapped onto a same processor as one single coalesced stage $\hat{\mathcal{S}_a^k}$, with computing cost $\hat{w_a^k} = \sum_{k'=i}^{j} w_a^{k'}$, and output communication cost $\hat{\delta_a^k} = \delta_a^j$. The transformed application now has exactly m_a stages. In the following, stage $\hat{\mathcal{S}_a^k}$ corresponds to the k-th stage of the transformed application a, for $1 \leq k \leq m_a$. As for computations, consider a processor $\mathcal{P}_u$ and an application a. We define $\mathcal{K}_a^u$ such that $k \in \mathcal{K}_a^u$ if and only if $\hat{\mathcal{S}_a^k}$ is processed by processor $\mathcal{P}_u$; $\mathcal{K}_a^u$ is the set of stages of (transformed) application a processed by $\mathcal{P}_u$. Then, for all a and u, and for each $k \in \mathcal{K}_a^u$, we allocate the speed fraction $s_{a,u}^k = \hat{w_a^k}/P_a$ for $\mathcal{P}_u$ to execute $\hat{\mathcal{S}_a^k}$. Similarly for communications, we define $\mathcal{K}_a^{u,v}$ such that $k \in \mathcal{K}_a^{u,v}$ if and only if $\hat{\mathcal{S}_a^k}$ is processed by $\mathcal{P}_u$ and $\hat{\mathcal{S}_a^{k+1}}$ is processed by $\mathcal{P}_v$, i.e., there is a communication to pay between $\mathcal{P}_u$ and $\mathcal{P}_v$. Note that $u \neq v$, otherwise stages $\hat{\mathcal{S}_a^k}$ and $\hat{\mathcal{S}_a^{k+1}}$ would have been merged as a single stage. Formally, $k \in \mathcal{K}_a^{u,v} \Leftrightarrow k \in \mathcal{K}_a^u$ and $k+1 \in \mathcal{K}_a^v$. Then we allocate the bandwidth fraction $b_{a,u,v}^k = \hat{\delta_a^k}/P_a$ to the communication.

The period of each application can be respected if and only if all the following inequalities are satisfied. There might be some spare speed and bandwidth if these are strict inequalities, and resources are fully utilized in case of equalities:

- $\forall 1 \leq u \leq p$, $\sum_{a=1}^{A} \sum_{k \in \mathcal{K}_a^u} s_{a,u}^k \leq s_u$,
  $\sum_{v=1}^{p} \sum_{a=1}^{A} \sum_{k \in \mathcal{K}_a^{u,v}} b_{a,u,v}^k \leq B_u^{out}$,
  $\sum_{v=1}^{p} \sum_{a=1}^{A} \sum_{k \in \mathcal{K}_a^{v,u}} b_{a,v,u}^k \leq B_u^{in}$;
- $\forall 1 \leq u, v \leq p$, $u \neq v$,
  $\sum_{a=1}^{A} \left(\sum_{k \in \mathcal{K}_a^{u,v}} b_{a,u,v}^k + \sum_{k \in \mathcal{K}_a^{v,u}} b_{a,v,u}^k \right) \leq b_{u,v}$.

We also consider *interval mappings without reuse*, which partition the stages of each (original) application into intervals, and map each interval onto a different processor. More precisely, if we transform each application a as explained above, the allocation function of stages $\hat{\mathcal{S}_a^k}$ (for $1 \leq a \leq A$ and $1 \leq k \leq m_a$) is a one-to-one function: each coalesced stage is allocated onto a distinct processor. It becomes then much easier to check the validity of the mapping, since each processor is only handling one single stage, receiving input data from one single other processor, and sending output data to one single other processor.

Energy model. The energy consumption of the platform is defined as the sum of the energy $E(u, \ell)$ consumed by each processor $\mathcal{P}_u$ enrolled in the mapping in mode ℓ. We assume that $E(u, \ell)$ consists of a static part and of a dynamic part. The static part $E_{stat}(u)$ is the static cost for a processor to be in service, and does not depend on the speed $s_{u,\ell}$ at which the processor is running. However, the static energy is consumed only in mode $\ell \neq 0$ (otherwise, the processor is inactive, and not enrolled in the mapping). On the contrary, the dynamic part $E_{dyn}(u, \ell)$ is of the form $E_{dyn}(u, \ell) = s_{u,\ell}^{\alpha}$, where $\alpha > 1$ is an arbitrary rational number. It is sometimes assumed that $\alpha = 2$ [23], but all our results hold for any value of α. Finally, for $\ell \neq 0$, we have $E(u, \ell) = E_{stat}(u) + E_{dyn}(u, \ell)$, while $E(u, 0) = 0$. The energy $E(u, \ell)$ is an energy consumed per time unit, so we could also speak of dissipated power. Note that it is mandatory to minimize energy consumption per time unit, because the execution of streaming applications with arbitrarily many data sets may last for an unbounded amount of time.

Problem definition. We consider the problem in which the applications and their characteristics (stage weights, communication costs, periods) are provided, as well as a target execution platform and its characteristics (multi-modal processor speeds, network card capacities and link bandwidths). Then, given a bound on the latency for each application, we aim at minimizing the power consumption while matching the period and latency constraints. Therefore, we formally define the problem as follows:

TriCriteria($E[P_a, L_a]$): *given A applications, p multi-modal processors, one array of periods $[P_a]$ and one array of latencies $[L_a]$, both of length A, what is the minimum power consumption of the platform, so that for each $a \in \{1, \ldots, A\}$, application a is processed at a period P_a, and its latency does not exceed L_a?*

IV. Complexity study

We first provide results for interval mappings without reuse, which turn out to be polynomial on fully homogeneous platforms, even with several concurrent applications (and the problem was known to be NP-complete on processor-heterogeneous platforms). Then, we establish the NP-completeness of the tri-criteria problem for general mappings with reuse, even on homogeneous platforms.

Theorem 1: TriCriteria($E[P_a, L_a]$) is polynomial on speed-homogeneous platforms for interval mappings without reuse.

Due to lack of space, we only provide the proof sketch of Theorem 1. Please refer to the companion research report [24] for the detailed proof.

The optimal solution relies on an intricate nesting of two dynamic programming algorithms. The first one solves the problem with one single application: it recursively computes the optimal energy consumption that can be achieved by mapping one stage interval to exactly q processors. Then another dynamic programming algorithm finds the minimum energy consumption with several applications, recursively trying all possible distributions of processors to applications, and using the first algorithm to compute the optimal energy consumption for each application, given the number of processors allocated to this application.

Note that on speed-heterogeneous platforms, the problem of finding an interval mapping which minimizes the power consumption for a given period and latency by application is NP-complete. Indeed, the problem of finding an interval mapping which minimizes the period of one single application for speed-heterogeneous platforms without communication cost already is NP-complete [25].

Theorem 2: TriCriteria($E[P_a, L_a]$) is NP-complete on speed-homogeneous platforms for general mappings with reuse.

Proof: We consider the associated decision problem: given periods P_a, latencies L_a ($1 \leq a \leq A$) and an energy E, does there exist a general mapping such that, for all $a \in \{1, \ldots, A\}$, application a is processed at period P_a, its latency is not larger than L_a, and the total energy does not exceed E?

The problem is obviously in NP: given periods, latencies, an energy and a mapping, it is easy to check in polynomial time that the mapping is valid.

To establish the completeness, we use a reduction from 2-partition [9]. We consider an instance $\mathcal{I}_1$ of 2-partition: given n strictly positive integers $x_1, x_2, \ldots, x_n$, does it exist a subset I of $\{1, \ldots, n\}$ such that $\sum_{i \in I} x_i = \sum_{i \notin I} x_i$? Let $S = \sum_{i=1}^{n} x_i$.

We build an instance $\mathcal{I}_2$ of our problem with two identical processors, each with a single possible speed $s = S/2$, and we consider that the cost of static energy is null. We have then n single-stage applications, whose stage weights are x_a, $1 \leq a \leq n$. We ask whether it is possible to achieve an energy $E^o = 2 \times (S/2)^\alpha$, with periods of 1, and latencies not exceeding 1. Clearly, the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$.

We now prove that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ does. Assume first that $\mathcal{I}_1$ has a solution. For each $a \in I$, the stage of application a is executed by the first processor. Other stages are executed by the second processor. The mapping consumes an energy $2 \times (S/2)^\alpha$, and all applications have a period and latency equal to 1. Now assume that $\mathcal{I}_2$ has a solution. Since $\sum_{i=1}^{n} x_i = S = S/2 + S/2$, all the periods must be 1, and each processor must run an amount of work of size exactly $S/2$; in other words, $\mathcal{I}_1$ has a solution. ■

Since all problems are NP-complete on speed-heterogeneous platforms, we wrote an integer linear program, which returns the optimal solution for both interval and general mappings, and provides us with a basis for comparisons (see [24]).

V. Heuristics

In this section, we present several heuristics for mapping streaming applications onto communication homogeneous platforms. The code of these heuristics is available at http://graal.ens-lyon.fr/~prenaud/Codes/tri-crit.tar. Except for H2, which does not use the possibility of sharing the processors (one application onto one processor), each of the heuristic variants has two versions, with or without processor reuse, which allows us to observe the impact of resource sharing.

In several heuristics, for each processor $\mathcal{P}_u$, we denote by s_u^{needed} the minimum speed at which the processor must run in order to be able to process all stages that it is currently assigned to, within the given period. When a stage of weight w of application a is assigned to processor $\mathcal{P}_u$, we add w/P_a to s_u^{needed}. When a stage is de-assigned, we perform a subtraction instead of the addition.

H1: random. We randomly cut each application into intervals (not too many, in order to match the latency constraints) and assign each interval to a random processor. The heuristic fails if a processor does not have enough speed to run its assigned stages.

H2: one-to-one. This heuristic assigns each application (as a single interval) to one single processor. This problem corresponds to the well-known assignment problem, and we implement the Hungarian algorithm to solve it.

H2-split: one-to-one with split. We perform a first assignment thanks to H2 if it succeeds, otherwise we assign all application stages to the first processor. All stages are then assigned (and we can consider, if the applications are concatenated, that each processor is assigned an "interval"), and for each processor $\mathcal{P}_u$, ℓ_u is the lowest mode such that $s_{u,\ell_u} \geq s_u^{\text{needed}}$, or m_u otherwise (note that in this case, the mapping is not valid). The main idea of this heuristic is then to try to split each "interval" at any place, and to keep the best split. While we find a better mapping, we try another split. More precisely, a split consists in (i) de-assigning one part of the concerned "interval"; (ii) assigning it to another processor $P_{u'}$; (iii) updating the two concerned modes ℓ_u and $\ell_{u'}$ as mentioned previously, thanks to the new values s_u^{needed} and $s_{u'}^{\text{needed}}$. Then we have to define a way to sort the different resulting mappings in order to choose the best one. We sort the mappings by: (1) increasing $\sum_{u=1}^{p} \max(s_u^{\text{needed}} - s_{u,\ell_u}, 0)$ (the mapping is valid if and only if this value is equal to 0); (2) increasing energy of the platform $E = \sum_{u \in \{1,\ldots,p\}} E(u, \ell_u)$; (3) decreasing $\max \left\{ \frac{E(u,\ell_u) - E(u,\ell_u - 1)}{s_u^{\text{needed}} - s_{u,\ell_u - 1}} | u \in \{1, \ldots, p\}, \ell_u \neq 0 \right\}$ (for large values, we expect the next split to be better).

H3: increasing speeds. We start with all processors in their smallest mode. Then we map applications onto the current platform, and check whether the mapping is valid or not. If the algorithm returns true, then we are done. Otherwise, we repeatedly change the distribution of the modes and call the mapping procedure until we find a valid mapping. There are different ways to change the distribution of the modes, thus leading to different variants of the heuristic (see below for variants *speed*, *energy* and *upDown*). The mapping procedure is quite different from that of previous heuristics. Indeed, we never assign a stage to a processor if it has not enough speed to run it while not exceeding the bound on the period. In other words, H3 never allows $s_{u,\ell_u} < s_u^{\text{needed}}$. In the previous heuristics, we first decided for the mapping, and then we chose the modes. In H3, we first choose the mode of each processor, and then we try to find an assignment which is valid with these modes and may either success or fail.

H3-sort: application sorting. This heuristic proposes a modification in the H3 mapping procedure, in which we assign the stages interval by interval instead of application by application. It can also work with any of the three variants below.

H3-speed/energy/upDown. We outline the three variants:
speed: the processors are sorted by increasing speed of the current mode (and if there is a tie, by increasing speed gain between the current mode and the next higher one). We check whether we find a mapping; if yes, we stop, and if not, we upgrade the first processor (in the previous order) and repeat.
energy: the processors are sorted by increasing energy spent (which is different from an ordering based on modes because of static energy).
upDown: we use the same ordering of processors as in the "energy" variant, but we improve the upgrade.

Summary of heuristics. Each heuristic is denoted by its heuristic number, followed by variants. For instance, H3-sort-speed is the H3-sort heuristic with the speed variant. Also, we add "-n" at the end of the heuristic name for the "without reuse" version of the heuristic, and "-r" for the "with reuse" version. Thus, H2-split-n is the H2-split heuristic with no reuse. Finally we consider another heuristic, called the "best" heuristic, which simply takes the minimum energy returned by

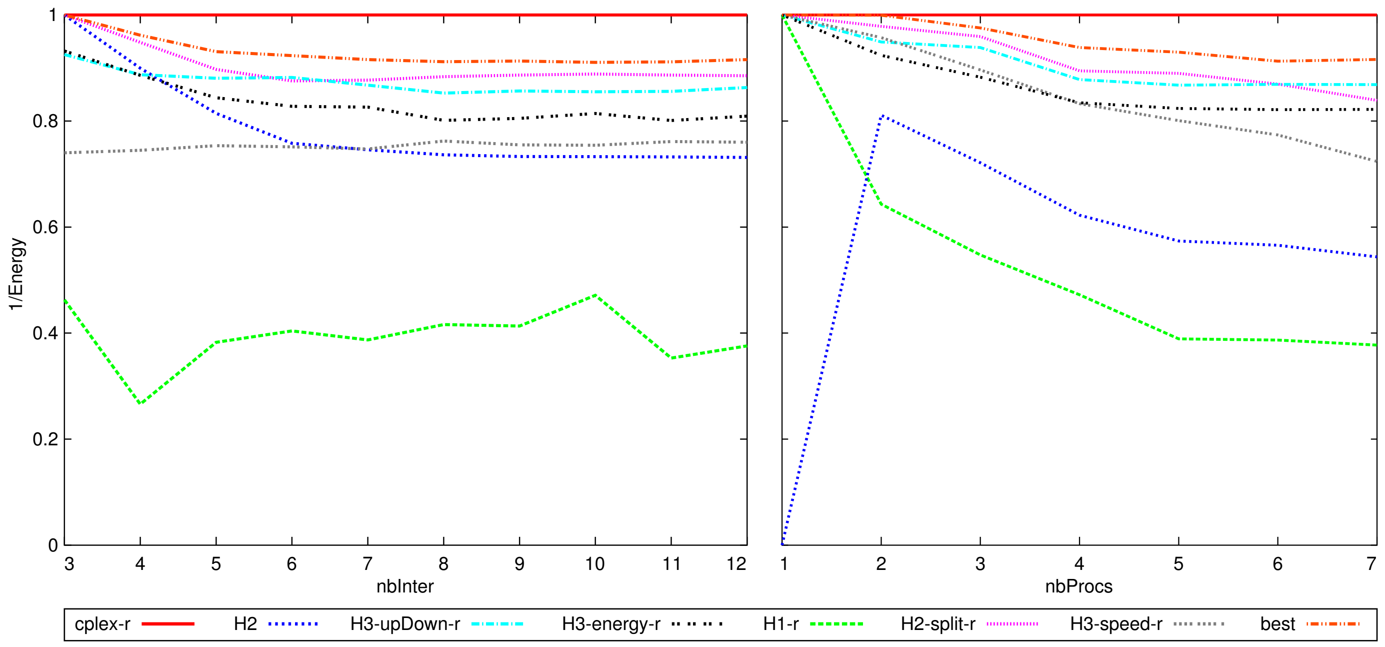


Fig. 1. Comparison with the optimal solution.

all the heuristics. Of course this value is achieved by different heuristics over all experiments, but it helps quantify what can be achieved in polynomial time vs. the linear program.

VI. Experiments

We have performed a comprehensive set of experiments in order to (i) assess the absolute performance of the heuristics, by comparing the heuristics with the linear program that finds the optimal general mapping (denoted as cplex-r); (ii) analyze the impact of reusing resources (interval vs. general mappings), using the linear program in its "without reuse" version (cplex-n); and (iii) study the scalability of the heuristics, on larger problem instances and without running the linear program (5000 problem instances, between 8 and 13 applications, and between 30 and 40 processors with 10 modes in the last experiment).

For the first two sets of experiments, since the integer linear program may run in exponential time, and effectively turns out very time consuming, we restrict the experiments to a small set of small problem instances (30 problem instances, with 3 applications and around 7 processors which have between 2 and 8 modes). In these experiments, we generate the set of random platforms and applications, and we vary one parameter of this platform: latency, number of processors, maximum energy static or average gap between two consecutive modes. For each platform, and each value of the parameter that we vary, we run all heuristics, and compute the solution of the linear program using the CPLEX software [26]. The graphs can be viewed as the average inverse of the consumed energy, normalized by the optimal solution. Platform sizes are chosen so that the optimal solution can be found in reasonable time (each graph has been obtained within a week, and the execution time of each heuristic was under 1 second per trial). Also, we do not represent the "sort" variant of H3, because it leads to negligible variations compared to H3.

(i) Latency and number of processors. In the first experiment (Fig. 1, left), we vary the latency of the applications: at the beginning, the latency constraint imposes that each application is mapped as a single interval, while it can go up to four in the end, hence twelve intervals over all applications. In the second experiment (Fig. 1, right), we increase the number of processors for a given application. All the heuristics are run in their "with reuse" variants. These experiments give a first idea of the ordering of the heuristics: H3-upDown-r, the best of the H3 variants, and H2-split-r return the best results, depending upon the platform. The "speed" variant is weak when there are fewer intervals by application or many processors, because it does not evaluate the static energy. H1 is worse than the others, therefore demonstrating that a random approach does not provide satisfying results. Finally, the "best" heuristic is quite good, never below 0.9 of the optimal.

(ii) Impact of reuse. In this second set of experiments, we compare the heuristics to the optimal solution without reuse, in order to assess the impact of reuse on the mapping. We vary the average gap between two modes in the third experiment (Fig. 2, left), and the maximum static energy in the fourth one (Fig. 2, right). When the static energy is becoming high, or when the modes are not close together, a mapping with one processor per application is the best solution without resource sharing; thus H2 and H2-split-n tend naturally to the optimal solution. Then H2-split-r and H3-upDown-r show the benefit of processor reuse: on the one hand, it allows these heuristics to use fewer processors than applications, thereby sparing static energy cost; and on the other hand, they can fill up the high modes with stages of different applications.

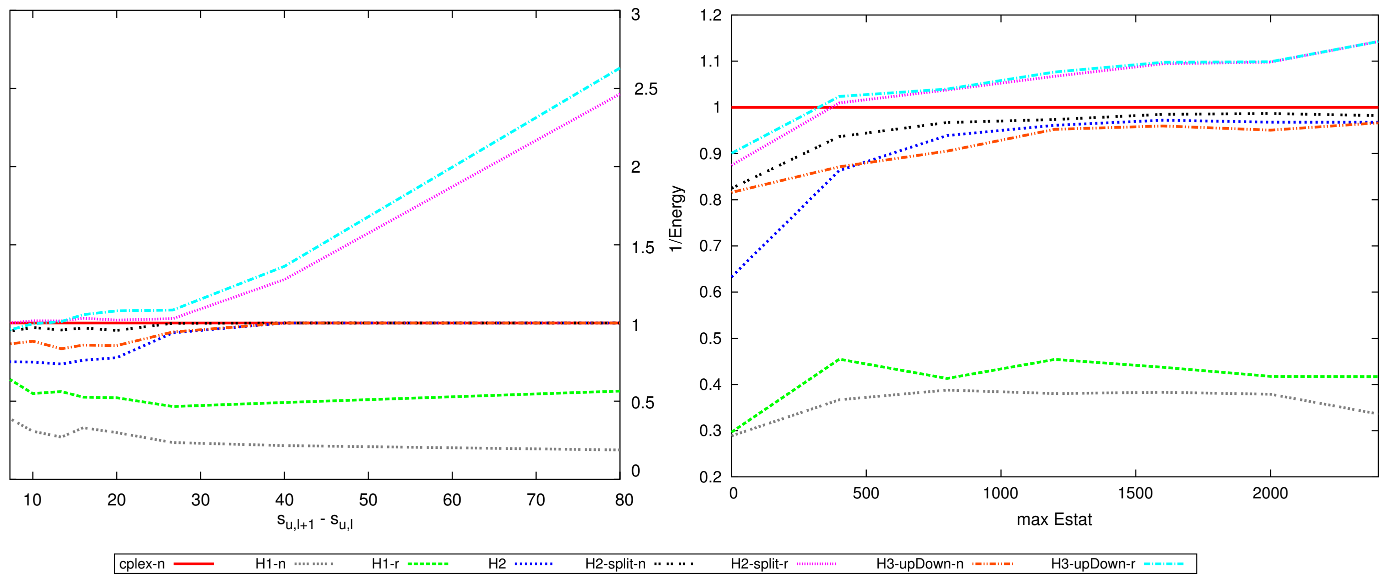


Fig. 2. Impact of reuse.

(iii) Scalability. In this last set of experiments, we study the heuristics when the instances are larger. For such real-life instances of the problem, the integer linear program cannot be used any more, due to its high complexity. In the fifth experiment (Fig. 3, left), we increase the number of processors with the number of applications, so that there are four times more processors than applications. This time, we represent the energy on the y-axis instead of its inverse, since we cannot normalize the plots with the optimal solution anymore. Therefore, the lower the plot the better the heuristic. Finally, in the sixth experiment (Fig. 3, right), we study all heuristics for some large problem instances. The main characteristics of the heuristics are shown in the table of Fig. 3. We report the number of failures in the first column, and how many times the concerned heuristic has been the best one in the second column. For the last three columns, we normalize the power consumption found by each heuristic by the power consumption found by the best one and analyze the table of normalized power. The average is computed over the platforms for which the heuristic found a solution. These experiments show the supremacy of H2-split-r, which is better in average, and gets better when problem instances get larger. Note however that for 20 applications, all heuristics execute in less than 1 second, except H2-split-r which takes 3 minutes.

VII. Conclusion

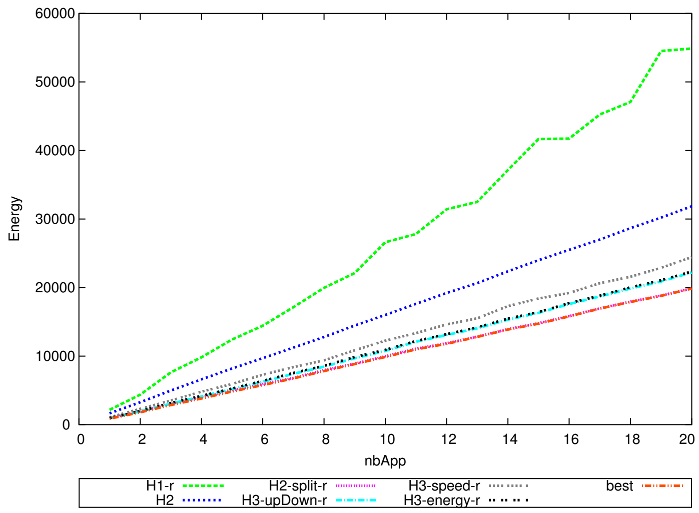
We have studied the following scheduling problem: given several pipelined applications with fixed periods and latency thresholds, and given a platform with multi-modal processors, determine the mapping (including processor modes) which minimizes the total consumed energy. We first established the complexity of this problem for different mapping strategies (interval mappings without reuse and general mappings with reuse), and different platform types (speed-homogeneous and speed-heterogeneous platforms). Thanks to a combination of two dynamic programming algorithms, we showed that finding an optimal interval mapping without reuse on speed-homogeneous platforms can be done in polynomial time. On the contrary, finding an optimal general mapping on any platform type, or finding any optimal mapping on speed-heterogeneous platforms, are NP-complete problems.

In order to tackle the difficult problem instances on speed-heterogeneous platforms, we wrote an integer linear program to compute the optimal solution (either interval-based or general) in possibly exponential time. Then we designed several heuristics, which we compared to each other, and to the optimal solution found by the linear program on small instances. At least on these small instances, the best heuristic always achieves at least 90% of the optimal solution. The comparison of heuristics with and without processor sharing does confirm that sharing is more useful when (i) the modes are not close to each other, and (ii) the static energy is high.

As for future directions, we would like to search for approximation algorithms, or to derive inapproximability results. Indeed, even though the performance of the heuristics was experimentally shown pretty good, we have no theoretical guarantee. With the tri-criteria approach of this paper, with thresholds on performance-related criteria, we could formulate the problem as follows: given three parameters α_P, α_L and α_E, does there exist a polynomial algorithm $\mathcal{A}$ such that the energy found by $\mathcal{A}$ on the problem TriCriteria($E[\alpha_P P_a, \alpha_L L_a]$) is less than α_E times the optimal energy consumption of the problem TriCriteria($E[P_a, L_a]$)? Finding approximation algorithms for some values of α_P, α_L and α_E is a challenging problem.

Acknowledgment

A. Benoit and Y. Robert are with the Institut Universitaire de France. This work was supported in part by the ANR *StochaGrid* project.



	Fail/Best	average	min	max
H1-r	114/0	2.625483	1.541424	FAIL
H1-n	286/0	2.570121	1.511875	FAIL
H2	0/0	1.558267	1.257960	1.954439
H2-split-r	0/3710	1.008385	1	1.226330
H2-split-n	0/514	1.022594	1	1.226330
H3-upDown-r	0/164	1.100380	1	1.338159
H3-upDown-n	0/98	1.113033	1	1.504697
H3-speed-r	0/4	1.228998	1	1.974289
H3-speed-n	0/3	1.244661	1	2.180104
H3-energy-r	0/58	1.114920	1	1.374722
H3-energy-n	0/37	1.126718	1	1.504697
H3-sort-upDown-r	0/712	1.056324	1	1.251331
H3-sort-upDown-n	0/62	1.118340	1	1.453929
H3-sort-speed-r	0/37	1.170503	1	1.902925
H3-sort-speed-n	0/5	1.210323	1	2.017221
H3-sort-energy-r	0/239	1.071706	1	1.271649
H3-sort-energy-n	0/25	1.128880	1	1.470972

Fig. 3. Scalability.

References

[1] DataCutter, "DataCutter Project: Middleware for Filtering Large Archival Scientific Datasets in a Grid Environment," http://www.cs.umd.edu/projects/hpsl/ResearchAreas/DataCutter.htm.

[2] K. Taura and A. A. Chien, "A heuristic algorithm for mapping communicating tasks on heterogeneous resources," in *Heterogeneous Computing Workshop*. IEEE Computer Society Press, 2000, pp. 102–115.

[3] S. L. Hary and F. Özgüner, "Precedence-constrained task allocation onto point-to-point networks for pipelined execution," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 10, no. 8, pp. 838–851, 1999.

[4] Q. Wu, J. Gao, M. Zhu, N. Rao, J. Huang, and S. Iyengar, "On optimal resource utilization for distributed remote visualization," *IEEE Transactions on Computers (TC)*, vol. 57, no. 1, pp. 55–68, 2008.

[5] Q. Wu and Y. Gu, "Supporting distributed application workflows in heterogeneous computing environments," in *International Conference on Parallel and Distributed Systems (ICPADS)*. Washington, DC, USA: IEEE Computer Society Press, 2008, pp. 3–10.

[6] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi, "Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2006, p. 340.

[7] A. Benoit, P. Renaud-Goud, and Y. Robert, "Performance and energy optimization of concurrent pipelined applications," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE CS Press, May 2010.

[8] K. Agrawal, A. Benoit, L. Magnan, and Y. Robert, "Scheduling algorithms for workflow optimization," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE CS Press, May 2010, also available as research report RR-LIP-2009-22 at http://graal.ens-lyon.fr/~abenoit/.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[10] R. Xu, D. Moss, and R. Melhem, "Minimizing expected energy in real-time embedded systems," in *Proceedings of the ACM Int. Conf. on Embedded Software (EMSOFT)*, 2005, pp. 251–254.

[11] R. Xu, D. Mossé, and R. Melhem, "Minimizing expected energy consumption in real-time systems through dynamic voltage scaling," *ACM Trans. Comput. Syst.*, vol. 25, no. 4, p. 9, 2007.

[12] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. M. Al-Hashimi, "Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems," in *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*. Washington, DC, USA: IEEE Computer Society Press, 2004, p. 10518.

[13] S. Cho and R. G. Melhem, "On the interplay of parallelization, program performance, and energy consumption," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 21, pp. 342–353, 2010.

[14] D. P. Bunde, "Power-aware scheduling for makespan and flow," in *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. New York, NY, USA: ACM Press, 2006, pp. 190–196.

[15] J.-J. Chen and L. Thiele, "Energy-efficient task partition for periodic real-time tasks on platforms with dual processing elements," in *International Conference on Parallel and Distributed Systems (ICPADS)*. Washington, DC, USA: IEEE Computer Society Press, 2008, pp. 161–168.

[16] F. Gruian and K. Kuchcinski, "Lenes: task scheduling for low-energy systems using variable supply voltage processors," in *Proceedings of the Asia South Pacific Design Automation Conference (ASPDAC)*. New York, NY, USA: ACM, 2001, pp. 449–455.

[17] T.-Y. Huang, Y.-C. Tsai, and E. T.-H. Chu, "A near-optimal solution for the heterogeneous multi-processor single-level voltage setup problem," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2007, p. 57.

[18] P. Langen and B. Juurlink, "Leakage-aware multiprocessor scheduling," *J. Signal Process. Syst.*, vol. 57, no. 1, pp. 73–88, 2009.

[19] G. Varatkar and R. Marculescu, "Communication-aware task scheduling and voltage selection for total systems energy minimization," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Washington, DC, USA: IEEE Computer Society Press, 2003, p. 510.

[20] B. Hong and V. K. Prasanna, "Bandwidth-aware resource allocation for heterogeneous computing systems to maximize throughput," in *International Conference on Parallel Processing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2003, p. 539.

[21] N. T. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A grid-enabled implementation of the message passing interface," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 63, no. 5, pp. 551–563, 2003.

[22] K. Agrawal, A. Benoit, and Y. Robert, "Mapping linear workflows with computation/communication overlap," in *International Conference on Parallel and Distributed Systems (ICPADS)*. Washington, DC, USA: IEEE Computer Society Press, 2008, pp. 195–202.

[23] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *International Symposium on Low Power Electronics and Design (ISLPED)*. ACM Press, 1998, pp. 197–202.

[24] A. Benoit, P. Renaud-Goud, and Y. Robert, "Sharing resources for performance and energy optimization of concurrent streaming applications," LIP, ENS Lyon, France, Research Report 2010-05, February 2010, available at http://graal.ens-lyon.fr/~abenoit/.

[25] A. Benoit and Y. Robert, "Mapping pipeline skeletons onto heterogeneous platforms," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 68, no. 6, pp. 790–808, 2008.

[26] Cplex, "ILOG CPLEX: High-performance software for mathematical programming and optimization," http://www.ilog.com/products/cplex/.