Energy-aware mappings of series-parallel workflows onto chip multiprocessors

Anne Benoit, Paul Renaud-Goud, Yves Robert
Ecole Normale Supérieure de Lyon, France
Email: {Anne.Benoit|Paul.Renaud-Goud|Yves.Robert}@ens-lyon.fr

Rami Melhem
University of Pittsburgh,USA
Email: melhem@cs.pitt.edu

***Abstract*—This paper studies the problem of mapping streaming applications that can be modeled by a series-parallel graph, onto a 2-dimensional tiled CMP architecture. The objective of the mapping is to minimize the energy consumption, using dynamic voltage scaling techniques, while maintaining a given level of performance, reflected by the rate of processing the data streams. This mapping problem turns out to be NP-hard, but we identify simpler instances, whose optimal solution can be computed by a dynamic programming algorithm in polynomial time. Several heuristics are proposed to tackle the general problem, building upon the theoretical results. Finally, we assess the performance of the heuristics through a set of comprehensive simulations.**

***Keywords*-series-parallel graphs; chip multiprocessors; scheduling algorithms; power consumption minimization.**

I. Introduction

The *energy consumption* of computational platforms has recently become a critical problem, both for economic and environmental reasons [1]. To help reduce energy consumption, processors can run at different speeds. Faster speeds allow for a faster execution, but they also lead to a much higher (superlinear) power consumption. Energy-aware scheduling aims at minimizing the energy consumed during the execution of the target application, both for computations and for communications. Obviously, this approach makes sense only if coupled with some performance bound to be achieved. Otherwise, the optimal solution is to run each resource at the slowest possible speed. In other words, we have a bi-criteria optimization problem, with one objective being energy minimization, and the other being performance-related.

In this paper, we aim at minimizing the energy consumption of streaming applications whose task graph is a series-parallel graph (SPG). Streaming applications, or workflows, are ubiquitous in many domains, as for instance image processing applications, astrophysics, meteorology, neuroscience, and so on [2], [3]. Most of these applications have simple and regular task graphs, such as linear chains, trees, fork-join graphs, or general SPGs (see Section III-A for a formal definition of SPGs). For instance, all the benchmarks of the *StreamIt* suite [4] are SPGs.

The performance-related objective coupled with energy minimization is the *period* of the streaming application. Typically, a series of data sets enter the input stage and progress from stage to stage, following the dependencies of the application, until the final result is computed. Each stage has its own communication and computation requirements: it reads inputs from the previous stage(s), processes the data and outputs results to the next stage(s). The pipeline operates in a dataflow mode: after a transient behavior due to the initialization delay, a new data set is completed every period. The period, which corresponds to the inverse of the throughput, is a key performance-related objective for streaming applications [2], [5], [6]. Formally, the period is the time interval between the arrival of two consecutive data sets in the application. Given a mapping of the application onto a platform, the time spent in each resource (processor or communication link) should not exceed the period.

Finally, the target platform for this study is a Chip MultiProcessor (CMP), which is composed of $p \times q$ homogeneous cores arranged along a 2D grid. During the last century, advances in integrated circuit technology have led chip designers to increase microprocessor performance by increasing the integration density thus allowing for higher clock rates and new innovations in micro-architectures. Such innovations included wider instructions, speculative execution, branch prediction and dynamic scheduling. However, in 1996, Olukotum et al. [7] argued that such a trend would not continue because of the diminishing return caused by limited instruction level parallelism and they argued that a better way for using the denser integration would be to layout multiple simpler processors on the same chip. Moreover, power consumption consideration prevented the push towards faster clocks, thus leaving the design of chip multiprocessors as the only alternative for increasing the on-chip computational capability. Specifically, increasing the number of cores rather than the processor's complexity translates into slower growth in power consumption. Currently, chip multiprocessors are commercially available and the trend is towards the continuous increase in the number of cores on single chips. The challenge is now to be able to efficiently utilize the parallelism available on chip [8].

An essential step for exploring the parallelism available in a streaming application is to provide algorithms and scheduling strategies for mapping a series-parallel graph onto a CMP, with the objective of minimizing the energy consumption while not exceeding a prescribed period. In some applications, data sets arrive at fixed time intervals, and hence the period of the application is given a priori, before any mapping is computed. In other applications, there is the freedom to choose between a set of possible periods,

which are prescribed by the user. In all cases, the main goal is to reduce the energy consumption of the mapping, while enforcing the constraint on the prescribed period.

The contribution of this paper is twofold. On the theoretical side, we assess the complexity of the above-mentioned mapping problem, using a *DAG-partition* mapping rule that partitions the application SPG into an acyclic graph of node clusters. In turn, each cluster is mapped onto a different processor of the CMP. Our cost model accounts for communication delays and cost (in terms of consumed energy). The problem turns out to be NP-hard, so we also study the complexity of simpler problem instances, either with a simpler target platform (uni-directional or bi-directional uni-line CMP), or by restricting to particular applications whose graph has a bounded degree of parallelism (*bounded-elevation* SPGs). The only problem instance that can be solved in polynomial time, thanks to a dynamic programming algorithm, is the mapping of bounded-elevation SPGs onto a uni-directional uni-line CMP. For other problem instances, we provide sophisticated NP-completeness proofs. On the practical side, building upon the theoretical results, we design some polynomial-time heuristics to solve the most general problem, and we assess their performance through simulation.

The paper is organized as follows. We first survey related work in Section II. Then we detail the framework in Section III, and we provide complexity results in Section IV. The heuristics are described in Section V, and simulation results in Section VI. Finally, we conclude and discuss future research directions in Section VII.

II. Related work

Reducing the energy consumption of computational platforms is an important research topic, and many techniques at the process, circuit design, and micro-architectural levels have been proposed. The dynamic voltage and frequency scaling (DVFS) technique has been extensively studied, since it may lead to efficient energy/performance tradeoffs [9], [10], [11], [12]. Current microprocessors (for instance, from AMD [13] and Intel [14]) allow the speed to be set dynamically. Indeed, by lowering supply voltage, hence processor clock frequency, it is possible to achieve important reductions in power consumption, without necessarily increasing the execution time.

Our objective is to minimize the energy consumption for series-parallel graph (SPG) applications which are mapped onto a chip multiprocessor (CMP). A detailed related work section in the companion research report [15] shows that (i) many of the directed acyclic graphs which represent classical workflow applications turn out to be series-parallel graphs [3]; (ii) most of the papers dealing with energy minimization approaches are also ensuring some performance guarantees (real-time constraints, such as a bound on the total execution time, or a threshold period) [16], [17]; (iii) even though some papers have considered the mapping of tasks and threads to CMPs [18], [19], none considers the mapping of streaming applications with the objective of minimizing power consumption while maintaining a specified period.

In this paper, the application is a workflow whose structure is a series-parallel task graph, and the goal is to map this application onto a CMP with minimum energy consumption and a given threshold on the period. We are extending previous work [17], which was conducted for simpler application structures (linear chains instead of series-parallel graphs), and for a realistic platform (the CMP) instead of virtual cliques. To the best of our knowledge, this paper is the first to investigate the complexity of this problem, and to propose practical solutions (polynomial time heuristics) for applications modeled by series-parallel graphs. The work in [20] shares the same objective as the work in this paper but is purely empirical. It presents a two-phase heuristic for mapping a general acyclic graph onto a CMP by first assigning the levels of the graph to the rows of the CMP and then mapping the tasks in each level to the nodes of the row assigned to that level. The heuristic described in Section V-C follows a similar two-phase strategy but obeys the DAG-partition mapping rule (see Section III-C).

III. Framework

A. Applicative framework

The application that is to be scheduled is a streaming application: it operates on a collection of data sets that are executed in a pipelined fashion. In this study, the application is a series-parallel graph $\mathcal{G} = (\mathcal{S}, \mathcal{E})$, or SPG. Nodes of the graph correspond to different application stages, and are denoted by S_i, with $1 \leq i \leq n$, where $n = |\mathcal{S}|$ is the size of the graph. For each precedence constraint in the application, say from stage S_i to stage S_j, we have an edge $L_{i,j} \in \mathcal{E}$. For $1 \leq i \leq n$, w_i is the computation requirement of stage S_i, and for each $L_{i,j} \in \mathcal{E}$, with $1 \leq i, j \leq n$, $\delta_{i,j}$ is the volume of communication to be sent from S_i to S_j before S_j can start its computation.

A SPG is built from a sequence of compositions (parallel or series) of smaller-size SPGs. The smallest SPG consists of two nodes connected by an edge. The first node is the source of the SPG while the second is its sink. When composing two SPGs in series, we merge the sink of the first SPG with the source of the second. For a parallel composition, the two sources are merged, as well as the two sinks (see Figure 1 for illustrative examples).

We recursively define the *label* of each node in a SPG, which corresponds to its coordinates along a 2D-grid in the recursive construction: $\ell_i = (x_i, y_i)$ is the label of stage S_i, for $1 \leq i \leq n$. First, for a two-node SPG $(S_1 \rightarrow S_2)$, the label of the source S_1 is $(1, 1)$, while the label of the sink S_2 is $(2, 1)$. The labels are then updated when composing the SPG. Consider two SPGs, SPG_1 with nodes $S_1^{(1)}, \ldots, S_{n_1}^{(1)}$,

and SPG_2 with nodes $S_1^{(2)}, \ldots, S_{n_2}^{(2)}$, and their corresponding labels $\ell_i^{(1)} = (x_i^{(1)}, y_i^{(1)})$ and $\ell_j^{(2)} = (x_j^{(2)}, y_j^{(2)})$, for $1 \leq i \leq n_1$ and $1 \leq j \leq n_2$.

- For a series composition, we merge the sink of SPG_1, $S_{n_1}^{(1)}$, with the source of SPG_2, $S_1^{(2)}$. The resulting SPG has $n = n_1 + n_2 - 1$ nodes with the following labels: for $1 \leq i \leq n_1$, $S_i = S_i^{(1)}$ and its label is $\ell_i = \ell_i^{(1)}$, and for $1 < j \leq n_2$, $S_{n_1+j-1} = S_j^{(2)}$ and the x values of the labels are incremented by $x_{n_1}^{(1)} - 1$, i.e., $\ell_{n_1+j-1} = (x_j^{(2)} + x_{n_1}^{(1)} - 1, y_j^{(2)})$.
- For a parallel composition, assume that $x_{n_1}^{(1)} \geq x_{n_2}^{(2)}$ (otherwise exchange the two SPGs, so that the first contains the longest path). We merge both sources ($S_1^{(1)}$ and $S_1^{(2)}$), and both sinks ($S_{n_1}^{(1)}$ and $S_{n_2}^{(2)}$). The resulting SPG has $n = n_1 + n_2 - 2$ nodes with the following labels: S_1 is the source and $\ell_1 = \ell_1^{(1)}$; S_n is the sink and $\ell_n = \ell_{n_1}^{(1)}$; for $1 < i < n_1$, $S_i = S_i^{(1)}$ and its label is $\ell_i = \ell_i^{(1)}$; for $1 < j < n_2$, $S_{n_1+j-2} = S_j^{(2)}$, and the y values of the labels are incremented by $y_{\max}^{(1)} = \max_{1 \leq i \leq n_1}(y_i^{(1)})$, i.e., $\ell_{n_1+j-2} = (x_j^{(2)}, y_j^{(2)} + y_{\max}^{(1)})$.

This construction is illustrated on the examples given in Figure 1. Note that these rules ensure that the source is always stage S_1, with $\ell_1 = (1, 1)$, and the sink is always stage S_n, with $\ell_n = (x_n, 1)$. Therefore, $\max_{1 \leq i \leq n} x_i = x_n$, and we denote by $y_{\max} = \max_{1 \leq i \leq n} y_i$ the maximum y value of the labels in the SPG, which we call *maximum elevation*. Intuitively, the maximum elevation denotes the maximal degree of parallelism of the SPG.

In the following, we focus the discussion on *bounded-elevation* SPGs, i.e., SPGs whose maximum elevation $y_{\max}$ is bounded by a constant. Indeed, dealing with bounded-elevation SPGs, rather than arbitrary SPGs, or even arbitrary DAGs, is a trade-off between tractability and generality. On the one hand, bounded-elevation SPGs correspond to a wide spectrum of applications, and nicely generalize linear chains and trees (a tree can easily be transformed into a SPG by adding fake nodes mirroring the tree). For instance, all the benchmarks of the *StreamIt* suite [4] are bounded-elevation SPGs: their maximum elevations range from $y_{\max} = 1$ (linear chain) to $y_{\max} = 17$. On the other hand, the problem of mapping a simple fork-join graph with n nodes (unbounded-elevation graph) onto two processors, in order to minimize the energy given a period bound, is NP-complete (reduction from 2-PARTITION, see Section IV-A). Dealing with bounded-elevation graphs enables us to identify polynomial instances, thus providing optimal solutions for some problem instances.

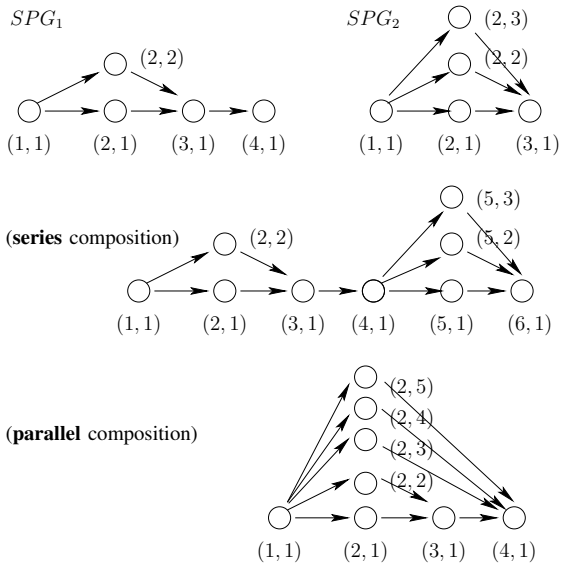

Figure 1. Examples of SPG composition.

B. Platform

The target platform is a CMP (**C**hip **M**ulti**P**rocessor), composed of $p \times q$ homogeneous cores $\mathcal{C}_{u,v}$, with $1 \leq u \leq p$, $1 \leq v \leq q$, arranged along a rectangular grid. There is a vertical (internal and bi-directional) communication link between $\mathcal{C}_{u,v}$ and $\mathcal{C}_{u+1,v}$, for $1 \leq u \leq p-1$, $1 \leq v \leq q$, and a horizontal link between $\mathcal{C}_{u,v}$ and $\mathcal{C}_{u,v+1}$, for $1 \leq u \leq p$, $1 \leq v \leq q-1$. All links have the same bandwidth BW.This means that it takes a time $\frac{\delta}{BW}$ to send δ bytes from one processor to a neighboring processor. It is possible to use only some of the communication links, and for instance to configure the $p \times q$ CMP as a $1 \times pq$ bi-directional linear array, called *bi-directional uni-line CMP*.

Although the cores of a CMP share the same memory space, it is possible to implement the message passing models on CMPs [21] by writing and reading from shared memory locations. However, for scalability purpose, CMPs with large number of cores will be organized as a mesh of tiles, each with its own cache [22]. Therefore, communication through shared memory ultimately translates to exchange of coherence traffic between the tiles [23], [24], [25]. Specifically, in the streaming model assumed in this paper, when a stage S_i, mapped to a core $\mathcal{C}_{u,v}$, writes into a shared variable, X, that shared variable is cached in the local cache of $\mathcal{C}_{u,v}$. Then, when a stage S_j with $L_{i,j} \in \mathcal{E}$, mapped to a core $\mathcal{C}_{u',v'} \neq \mathcal{C}_{u,v}$, reads X, the cache coherence protocol guarantees that X is cached in the local cache of $\mathcal{C}_{u',v'}$. Therefore, the values of the cache line containing X are sent from $\mathcal{C}_{u,v}$ to $\mathcal{C}_{u',v'}$. In other words, if two stages S_i and S_j, connected with an edge $L_{i,j}$, are mapped onto two distinct processors, a communication of size $\delta_{i,j}$ must occur (implicit messages) to keep the cached values coherent[1]. Hence, irrespectively of the programming model used to implement the SPG, the weight on a directed

[1]It is assumed that the cache coherence protocol supports cache-to-cache transfer and exploits communication locality by tracking in each core the location of frequently accessed blocks [26].

edge between two nodes in the SPG represents the volume of communication to be sent between the cores executing the corresponding application stages.

As mentioned in Section II, the voltage and frequency of each core of the CMP can be set to different values. Altogether, there is a set of possible supply voltages, together with a set of possible frequencies (or modes, or speeds), for each core. Let $\mathsf{S} = \{s^{(1)}, \ldots, s^{(m)}\}$ denote the set of all possible speeds. It takes a time $\frac{w_i}{s^{(k)}}$ to execute one data set for stage S_i at speed $s^{(k)} \in \mathsf{S}$ on a given core. Each speed induces a different dynamic power consumption, as discussed in Section III-E below.

C. Mapping strategies

We discuss several mapping rules to map the SPG application onto the CMP. As for the application graph, we use *DAG-partition* mappings, which represent a trade-off between *one-to-one* and *general* mappings. The rationale is the following. One-to-one mappings obey the simplest rule: each application stage is mapped onto a distinct core. While easier to optimize and implement, this rule may be unduly restrictive, and is likely to lead to high communication costs. Obviously, it also requires that $p \times q \geq n$, thereby limiting its applicability to large platforms or small applications. A natural extension is to search for DAG-partition mappings: we first partition the initial SPG into subsets, or clusters, such that the resulting graph is acyclic. Hence this mapping rule states that if two stages S_i and S_j are in the same subset of the partition, then any other stage S_k which has an incoming dependency from S_i and an outgoing dependency to S_j, must be in the same subset of the partition. Then we map the subsets of the partition onto the cores in a one-to-one fashion. Using this mapping rule, a core which is executing a subset I of stages $\{S_i\}_{i \in I}$ will perform at most one input and one output communication for each elevation value $\{y_i\}_{i \in I}$. This is well in accordance with our initial assumption that the SPG has bounded elevation $y_{\max}$, because it ensures that each core has at most $y_{\max}$ communications to perform at each period. In contrast, a fully general mapping, that allows for arbitrary partitions of the original application graph, would require an arbitrary number of communications, only bounded by the total number of stages n, hence an unlimited amount of buffer space. Moreover, even for bounded-elevation SPGs, the problem of finding the general mapping which minimizes the energy given a period bound is trivially NP-complete (linear chain onto two processors, reduction from 2-PARTITION [27]).

Formally, the mapping is defined by an allocation function

$$alloc : \{1, \ldots, n\} \to \{1, \ldots, p\} \times \{1, \ldots, q\} ,$$

which maps stages onto cores. In other words, if stage S_i is mapped onto core $\mathcal{C}_{u,v}$, we have $alloc(i) = (u, v)$. Once application stages are mapped onto cores, there remains to decide how to route communications between two cores which need to communicate because of the stage assignment. Therefore, for each application edge $L_{i,j} \in \mathcal{E}$, if $alloc(i) \neq alloc(j)$, we define $path_{i,j}$ as the set of communication links that are used to communicate from core $alloc(i)$ to core $alloc(j)$. Note that these paths must be defined for the mapping to be fully determined.

D. Period

As motivated above, we assume that data sets arrive at regular time intervals, which is called the *period* of the application, and denoted by T. Then, given a mapping and an execution speed for each core, we can check whether the application can be executed at the prescribed rate: we must ensure that the cycle-time of each resource (computation or communication link) does not exceed T.

Let $w_{u,v} = \sum_{1 \leq i \leq n | alloc(i) = (u,v)} w_i$ be the total amount of work assigned to core $\mathcal{C}_{u,v}$, running at speed $s_{u,v} \in \mathsf{S}$. The cycle-time of $\mathcal{C}_{u,v}$ for computations is $w_{u,v}/s_{u,v}$.

Let $b^{(ver)}_{(u,v)} = \sum_{1 \leq i,j \leq n | (u,v) \leftrightarrow (u+1,v) \in path_{i,j}} \delta_{i,j}$ be the number of bits sent between $\mathcal{C}_{u,v}$ and $\mathcal{C}_{u+1,v}$, and let $b^{(hor)}_{(u,v)} = \sum_{1 \leq i,j \leq n | (u,v) \leftrightarrow (u,v+1) \in path_{i,j}} \delta_{i,j}$ be the number of bits sent between $\mathcal{C}_{u,v}$ and $\mathcal{C}_{u,v+1}$. The cycle-time of the communication link $(u, v) \leftrightarrow (u + 1, v)$ (resp. $(u, v) \leftrightarrow (u, v + 1)$) is $b^{(ver)}_{(u,v)}/BW$ (resp. $b^{(hor)}_{(u,v)}/BW$).

We can then compute the maximum cycle-time, which is the maximum cycle-time of all resources, and check that it is not greater than T.

E. Energy model

Once a SPG application has been mapped onto the CMP, there are two sources of energy consumption: the cores consume energy for computations and the routers consume additional energy for communications.

For the computations, we assume that each core involved in the execution consumes some static energy during the whole period T, and some dynamic energy that depends on the amount of operations, and on the speed at which these operations are executed. Let $\mathcal{A}$ be the set of active cores: $\mathcal{A} = \{\mathcal{C}_{u,v}, 1 \leq u \leq p, 1 \leq v \leq q \mid \exists 1 \leq i \leq n, alloc(i) = (u, v)\}$. For each core $\mathcal{C}_{u,v} \in \mathcal{A}$, recall that $w_{u,v}$ is its assigned work and $s_{u,v}$ its speed. The total energy consumed for computations is

$$E^{(\text{comp})} = |\mathcal{A}| \times P^{(\text{comp})}_{\text{leak}} \times T + \sum_{\mathcal{C}_{u,v} \in \mathcal{A}} \frac{w_{u,v}}{s_{u,v}} \times P^{(\text{comp})}_{s_{u,v}} ,$$

where T is the prescribed period, $P^{(\text{comp})}_{\text{leak}}$ is the leakage power dissipated together with computations, and $P^{(\text{comp})}_{s_{u,v}}$ is the dynamic power associated with speed $s_{u,v}$.

For the communications, there is also a static part due to leakage, which is paid for all cores: even if a core is not enrolled in the computation, its routers and communication links may be used to route data between remote processors. The dynamic part is directly proportional to the number of

bits that are sent across each link, $b_{(u,v)}^{(ver)}$ and $b_{(u,v)}^{(hor)}$. Hence, the total energy consumed for communications is

$$E^{(\text{comm})} = P_{\text{leak}}^{(\text{comm})} \times T + \left(\sum_{u=1}^{p-1} \sum_{v=1}^{q} b_{(u,v)}^{(ver)} + \sum_{u=1}^{p} \sum_{v=1}^{q-1} b_{(u,v)}^{(hor)} \right) \times E^{(\text{bit})},$$

where T is the period, $P_{\text{leak}}^{(\text{comm})}$ is the aggregated leakage power dissipated by all routers and links, and $E^{(\text{bit})}$ is the energy to transfer a bit across neighboring cores. Finally, the total energy consumption is $E = E^{(\text{comp})} + E^{(\text{comm})}$.

We are ready to formally define the optimization problem MINENERGY(T): given a bounded-elevation SPG and a period threshold T, find a mapping whose maximal cycle-time does not exceed T and whose energy E is minimum.

IV. Complexity results

In this section, we assess the complexity of the MINENERGY(T) problem for various instances. We classify results depending upon the target CMP, which may be uni-directional uni-line (see Section IV-A), or bi-directional (uni-line or 2D mesh, see Section IV-B). The only polynomial instance of MINENERGY(T) is for the uni-directional uni-line CMP. In this case, we exhibit a dynamic programming algorithm that finds the optimal solution. It is worth noting that this polynomial instance becomes NP-complete for SPGs of unbounded elevation. All other problem instances are NP-hard, and we formulate the problem as an integer linear program in Section IV-C.

A. Uni-directional uni-line CMP

In this section, we assume that the CMP is configured as a uni-directional linear array of q processors. First we provide a polynomial algorithm to solve the case of bounded-elevation SPGs. As a digression from the main focus of this paper (bounded-elevation SPGs), we prove that the problem becomes NP-hard for SPGs of unbounded elevation.

Theorem 1: The MINENERGY(T) problem on a uni-directional uni-line CMP has polynomial complexity.

Proof: We exhibit a dynamic programming algorithm which computes the optimal solution. Let $\mathcal{G}$ be a bounded-elevation SPG. First we define *admissible* subgraphs of $\mathcal{G}$ recursively:

- $\mathcal{G}$ is admissible;
- if a subgraph G of $\mathcal{G}$ is admissible, then any subgraph of G obtained by deleting one node which has no successor in G is admissible too.

Let H be a set of one or several nodes deleted from G with this process, and let $G' = G \setminus H$. Note that the partition $\{G', H\}$ is acyclic, and that any possible acyclic partition of G into two subgraphs can be obtained with this construction. If we iterate the construction on G', we can build any DAG-partition of $\mathcal{G}$.

How many admissible subgraphs can we have? Let $y_{\max}$ be the maximal elevation of $\mathcal{G}$. Consider any admissible subgraph G. By definition, two nodes with the same y coordinate are linked by a dependence. Therefore, for each value of y between 1 and $y_{\max}$, there can be at most one node of elevation y and without successor in G. Hence there are at most $n^{y_{\max}}$ admissible subgraphs (and this bound is asymptotically met for a fork-join shaped graph composed of $y_{\max}$ chains of length $n/y_{\max}$ assembled with a source and sink node).

For any admissible subgraph G of $\mathcal{G}$, let $\mathcal{E}(G,k)$ be the minimum energy consumption required to execute the subgraph G onto exactly the first k processors. The goal is to determine $\min_{k=1}^{q} \mathcal{E}(\mathcal{G},k)$. The dynamic programming formulation can be expressed as:

$$\mathcal{E}(G,k) = \min_{G' \subseteq G} \left(\mathcal{E}(G', k-1) \oplus \mathcal{E}^{\text{cal}}(G \setminus G') \right) ,$$

with the initialization $\mathcal{E}(G,1) = \mathcal{E}^{\text{cal}}(G)$.

The minimum is taken over all admissible subgraphs G' such that communications between G' and $G \setminus G'$ do not exceed the bandwidth: $\frac{C^{\text{out}}(G')}{BW} \leq T$, where $C^{\text{out}}(G')$ denotes the aggregated output data volume of G', i.e., the sum of the output data δ_i of all stages $S_i \in G'$ which have no successor in G'.

$\mathcal{E}^{\text{cal}}(H)$ represents the energy consumed for the computations of the nodes in H when mapped to the same processor. Given such a node set H, we select the minimum speed that allows for computing all the stages in H within the period T, and we compute the corresponding energy consumption. If no such speed exists, we let $\mathcal{E}^{\text{cal}}(H) = +\infty$. Finally, the $\oplus$ operator means that the energy consumed by the induced communications is added to the sum.

At each step, there are no more than $n^{y_{\max}}$ admissible graphs G', and therefore we have at most $n^{2y_{\max}}$ values of $\mathcal{E}^{\text{cal}}(H)$ to compute, which is done in $O(n)$. Altogether, we have designed an algorithm whose worst-case complexity is $O(q \times n \times n^{2y_{\max}})$, which is polynomial since $y_{\max}$ is a constant. ■

The previous theorem only holds for bounded-elevation SPGs. With unbounded-elevation SPGs, the problem becomes NP-hard:

Proposition 1: The extension of MINENERGY(T) to unbounded-elevation SPGs on a uni-directional uni-line CMP is NP-complete.

Proof sketch: In fact, without any energy consideration, the simpler mono-criterion problem of matching a prescribed period is NP-complete. The associated decision problem is as follows: given a period T, is there a DAG-partition mapping whose period is no more than T? The problem is obviously in NP: given a period and a mapping, we can check in polynomial time that it is valid (compute its period).

The completeness comes from a reduction from 2-PARTITION [27]. For our problem, the application consists of a fork-join graph of elevation n, with no communication costs, and the computation cost of the i-th node of the fork-join is the a_i from 2-PARTITION. The source and the sink

have no computation cost. The platform consists of two cores which can operate only at a unique speed 1, and we ask whether we can achieve a period $(\sum_{i=1}^{n} a_i)/2$. The equivalence is then immediate, see [15] for the proof. ■

B. Bi-directional CMPs

First we consider uni-line CMP, before investigating the general case of 2D meshes.

Theorem 2: The MINENERGY(T) problem on a bi-directional uni-line CMP is NP-complete.

This theorem ensures that the problem is NP-hard for a $1 \times q$ CMP, hence for CMPs of arbitrary shapes. However, the problem complexity for a square CMP of size $p \times p$ is not a consequence of Theorem 2. We also establish this complexity:

Theorem 3: The MINENERGY(T) problem on a square CMP is NP-complete.

The proofs of both Theorems 2 and 3 are surprisingly long and difficult. Due to lack of space, we refer to the companion research report [15] for full details.

C. Integer linear program

The general problem of finding the optimal DAG-partition mapping, for a given period, has been shown to be NP-hard. However, we have been able to formulate the problem as an integer linear program (ILP), which allows us to find the optimal solution of the problem (in exponential time) for small problem instances. Actually, this ILP can also find the optimal general mapping (without the restriction of DAG-partition mappings), by removing the DAG-partition constraint from the program.

Unfortunately, because of the large number of variables needed to express communication paths in the CMP, we were unable to obtain results on a platform larger than a 2×2 CMP with ILOG CPLEX (www.ilog.com/products/cplex/). The ILP formulation can be found in the companion research report [15].

V. HEURISTICS

In this section, we describe the five heuristics that we have designed and implemented, thus providing practical solutions to the MINENERGY(T) problem.

A. Random heuristic

This first heuristic calls a procedure which works in two steps. The procedure first randomly builds a DAG-partition of the initial SPG, while ensuring that the period is matched for computations: we choose randomly a speed for the core which will handle the current subgraph G (initially, the source of the SPG), and we keep a list of stages of the SPGs that can be added to G while maintaining a DAG-partition. We pick a stage from this list randomly as long as computations do not exceed the period. When moving to the next core, we choose the first stage in the current list and iterate. In the second step, we decide randomly on which core each subgraph is mapped, and communications are done following an XY routing: a communication from $\mathcal{C}_{u,v}$ to $\mathcal{C}_{u',v'}$ follows horizontal links from $\mathcal{C}_{u,v}$ to $\mathcal{C}_{u',v}$, and then vertical links from $\mathcal{C}_{u',v}$ to $\mathcal{C}_{u',v'}$. If the period is not exceeded on any communication link, then the mapping is valid, otherwise there is no solution.

For each problem instance, **Random** calls ten times this procedure, and keeps the solution which minimizes the energy consumption, if there is at least one valid solution; otherwise it fails.

B. Greedy heuristic

Given a speed $s \in \mathsf{S}$, this heuristic greedily assigns the SPG onto the platform, on which all cores are running at speed s. The greedy assignment is done through procedure **greedy**(s). The idea is to try all possible speed values, and to keep the best solution.

The greedy procedure **greedy**(s) works as follows: we keep a list of cores which are ready to be processed, and for each core, a list of successors, together with the corresponding outgoing communications. Initially, the only core in the list is $\mathcal{C}_{1,1}$, and we assign to this core the source stage S_1. The corresponding list of successors corresponds to the successors of S_1 in the SPG, and they are sorted by non-increasing communication volume to S_1. When we process a core $\mathcal{C}_{u,v}$, we successively try to add some of the successors (from the current list) to this same core until the list is empty or the period is exceeded for computations on $\mathcal{C}_{u,v}$.

For each set of stages mapped onto $\mathcal{C}_{u,v}$ and the corresponding list of successors, we greedily share the corresponding communications between neighboring cores $\mathcal{C}_{u,v+1}$ and $\mathcal{C}_{u+1,v}$: communications are taken from the sorted list and assigned to the core which has currently the smallest amount of incoming communications. Then, we check that the partitioning is correct (no cycles in the dependence graph, i.e., we have a DAG-partition), and we check whether the bound on the period is achieved, both for computations and communications. If it is correct, we save the current solution before adding one more stage onto core $\mathcal{C}_{u,v}$ and iterating with one more stage on $\mathcal{C}_{u,v}$.

At the end of the iteration, we keep the last valid (saved) solution, i.e., the valid solution with the most number of stages onto $\mathcal{C}_{u,v}$. Cores $\mathcal{C}_{u,v+1}$ and $\mathcal{C}_{u+1,v}$ are then added to the list of ready cores, together with the list of successors (i.e., the stages that can either be assigned to this core, or forwarded to the neighboring cores).

The procedure finishes when the list of ready cores is empty, which means that all stages have been processed. Otherwise, the heuristic fails, and we move to the next speed. The energy for the mapping obtained with a given speed is computed by first *downgrading* the speed of each core, if possible: the procedure returns the mapping, and then

we compute the amount of computations on each core, and set the core to the slowest possible speed, in order to save energy. Cores which are not used are turned off. Finally, the **Greedy** heuristic selects the mapping which corresponds to the lowest energy consumption.

C. 2D dynamic programming algorithm

This heuristic, called **DPA2D**, starts by mapping the initial SPG onto an $x_{\max} \times y_{\max}$ grid, following the labels of the nodes (see Section III-A). Then, this grid is mapped onto the CMP, thanks to a double nested dynamic programming algorithm.

First, we perform a dynamic programming algorithm to cut the grid into a set of columns, which are to be mapped onto a column of cores. Let $\mathcal{E}(m, v, D)$ be the optimal energy consumption to compute the first m levels of the SPG (i.e., all stages S_i with $x_i \leq m$), using v columns of cores, regardless of the outgoing communications. D is then the corresponding distribution of outgoing communications, i.e., a list of triplets (y, b, i), where y is the row from which communication is outgoing (i.e., the communication is initiated by core $\mathcal{C}_{y,v}$), b is the amount of data, and S_i is the destination stage. We enforce these communications to go through $\mathcal{C}_{y,v+1}$, and then the communication will be redistributed to the destination core through vertical links. The solution is $\mathcal{E}(x_{\max}, q, D)$, and the recurrence is written as:

$$\mathcal{E}(m, v, D) = \min_{m' < m} \left(\begin{array}{r} \mathcal{E}(m', v-1, D') + \mathcal{E}^{\text{comm}}(D') \\ +\mathcal{E}^{\text{col}}(m'+1, m, D', D) \end{array} \right),$$

with the initialization $\mathcal{E}(m, 1, D) = \mathcal{E}^{\text{col}}(1, m, \emptyset, D)$.

D' is the distribution of outgoing communications corresponding to the m' which leads to the optimal energy consumption, i.e., obtained with $\mathcal{E}(m', v-1, D')$.

$\mathcal{E}^{\text{comm}}(D')$ is the energy consumption induced by communications from column $v-1$ to column v (on horizontal links), given the distribution D' of outgoing communications of column $v+1$. If the bandwidth is exceeded on one of these horizontal links (i.e., $\exists 1 \leq y \leq p$ such that $\sum_{(y,b,i) \in D'} b > BW$), we set $\mathcal{E}^{\text{comm}}(D') = +\infty$.

$\mathcal{E}^{\text{col}}(m_1, m_2, D', D)$ is the optimal energy consumption of the column of the CMP which is processing stages S_i with $m_1 \leq x_i \leq m_2$: it accounts both for computations, and for vertical communications in the column, given the distribution of outgoing communications of the previous column, D'. The distribution of outgoing communications of this column is then D. Note that in the recurrence, D is an output of $\mathcal{E}^{\text{col}}(m'+1, m, D', D)$, while D' is an output of $\mathcal{E}(m', v-1, D')$. The values of $\mathcal{E}^{\text{col}}$ (and therefore, distribution D) are computed thanks to another dynamic programming algorithm: we compute $\mathcal{E}^{\text{col}}_{(m_1,m_2,D',D)}(g, u)$, which corresponds to the mapping of stages S_i, with $m_1 \leq x_i \leq m_2$ and $y_i \leq g$, onto the u first cores of a column of the CMP. As before, D' is an input, it corresponds to the distribution of outgoing communications arriving into the current column, while D is the distribution of outgoing communications of the current column for the solution which minimizes the energy consumption. Then we have $\mathcal{E}^{\text{col}}(m_1, m_2, D', D) = \mathcal{E}^{\text{col}}_{(m_1,m_2,D',D)}(y_{\max}, p)$.

For the distribution within a column, the recurrence writes:

$$\mathcal{E}^{\text{col}}_{(m_1,m_2,D',D)}(g, u) = \min_{g' \leq g} \left(\begin{array}{l} \mathcal{E}^{\text{col}}_{(m_1,m_2,D',D)}(g', u-1) \\ +\mathcal{E}^{\text{cal}}_{(m_1,m_2,D)}(g'+1, g) \\ +\mathcal{E}^{\text{ver}}_{(m_1,m_2,D')}(g'+1, g, u-1) \end{array} \right),$$

with the initialization $\mathcal{E}^{\text{col}}_{(m_1,m_2,D',D)}(0, u) = 0$, and no outgoing communications from row 1 to row u, except the communications from D' that must be forwarded to the next column.

$\mathcal{E}^{\text{ver}}_{(m_1,m_2,D')}(g'+1, g, u-1)$ is the energy consumption of the vertical communications between cores $u-1$ and u in the column. These communications can either come from two dependent stages of the column, or be forwarded from the previous column (D'). If the bandwidth of the link is exceeded, we set the value to $+\infty$.

Finally, $\mathcal{E}^{\text{cal}}_{(m_1,m_2,D)}(g'+1, g)$ is the optimal energy consumption of a core which is computing all stages S_i such that $m_1 \leq x_i \leq m_2$, and $g'+1 \leq y_i \leq g$. If the period cannot be respected, or if the corresponding partition does not respect the DAG-partition constraint, the value is set to $+\infty$. Moreover, this function is adding to distribution D the communications from a stage S_i to another stage S_j, with $x_j > m_2$. These communications will occur on row u.

Note that in the recursive computation of $\mathcal{E}^{\text{col}}$, we can have $g' = g$, which means that no stage is assigned to core $\mathcal{C}_{u,v}$. This may happen if there are not enough stages in the column, or if this would save communications.

D. 1D heuristics

The two last heuristics configure the CMP as a unidirectional uni-line CMP with $r = p \times q$ cores, by embedding it into the bi-directional platform as a *snake*:

$$\begin{array}{lll} \mathcal{C}_{1,1} & \rightarrow \mathcal{C}_{1,2} \rightarrow \cdots \rightarrow & \mathcal{C}_{1,q} \\ & & \downarrow \\ \mathcal{C}_{2,1} & \leftarrow \cdots \leftarrow \mathcal{C}_{2,q-1} \leftarrow & \mathcal{C}_{2,q} \\ \downarrow & & \\ \mathcal{C}_{3,1} & \rightarrow \mathcal{C}_{3,2} \rightarrow \ldots & \end{array}$$

The **DPA1D** heuristic builds upon the theoretical results of Section IV, and computes the optimal solution of the dynamic programming algorithm of Theorem 1 with $r = p \times q$ cores. The mapping is then done along the snake; no other communication link is used. Note that if the SPG is a linear chain, even if there are communication costs, then this heuristic is optimal, since any other solution could not exploit the communication links discarded with the snake structure. However, **DPA1D** may make wrong decisions when communications are intensive, since it is restricted to

a subset of communication links. Moreover, its complexity of $O(p \times q \times n \times n^{y_{\max}})$ makes it intractable for SPGs with large $y_{\max}$.

Finally, the **DPA2D1D** heuristic computes the solution with the **DPA2D** heuristic (Section V-C) on a $1 \times r$ CMP, and then do the mapping along the snake, similarly to **DPA1D**. The goal of this heuristic is to obtain efficient solutions when communications are not too intensive, and when the optimal **DPA1D** cannot find a solution in reasonable time.

VI. Simulation results

This section reports simulation results assessing the performance of the various heuristics. As for the applications, we randomly generate series-parallel graphs of various elevation and different computation-to-communication ratios. We also experimented with a real set of applications extracted from the *StreamIt* suite [4], but due to lack of space we mainly present results for the randomly generated graphs, because they allow us to better explore the parameter space. As for the target platform, we use CMP grids, whose hardware characteristics are representative of state-of-the-art devices. The source code for all simulations is publicly available at graal.ens-lyon.fr/~prenaud/sp-cmp/.

Streaming applications. We randomly build SPG applications (by applying recursively series and parallel compositions of SPG applications), and we extract their size n and their elevation $y_{\max}$, together with their *computation-to-communication ratio (CCR)*, defined as the sum $\sum_{i=1}^{n} w_i$ of all computations over the sum $\sum_{L_{i,j} \in \mathcal{E}} \delta_{i,j}$ of all communications. We generate 50 SPGs of elevation y, for $1 \leq y \leq 20$, and their size is such that $40 \leq n \leq 60$. The CCR ratio is set either to 10 (compute-intensive applications), 1 (balanced applications) or 0.1 (communication-intensive applications).

CMP configuration. For processor speeds and power consumption, we use the model of the Intel Xscale, following [28], [29]. There are five speeds for each core: $s_{u,v} = (0.15, 0.4, 0.6, 0.8, 1)\ GHz$, with power consumption $P_{s_{u,v}}^{(\text{comp})} = (80, 170, 400, 900, 1600)\ mW$. We assume that the power consumption of the processor when it is idle is $P_{\text{leak}}^{(\text{comp})} = 80mW$. We use 16-byte wide communication links [30], whose bandwidths are $BW = 16 \times 1.2\ Gbytes$, which is reasonable according to [30]. Note that from the communication prospective, decreasing CCR has the same effect on the results as decreasing the width of the communication link below 16 bytes. The link energy is assumed to be between 1 and 10 picojoule per bit [31]; we fix $E^{(\text{bit})} = 6pJ$. Finally, we use $P_{\text{leak}}^{(\text{comm})} = 0$ without loss of generality (because for all heuristics the same quantity $P_{\text{leak}}^{(\text{comm})} \times T$ will be added to the total energy).

Period bound T. We need to find a meaningful value of T for each workflow. Indeed, if T is too large, all heuristics will map all stages onto a single processor running at the slowest speed, while if T is too small, all heuristics will fail. We choose T as follows: for each workflow, we start with $T = 1s$. With such a period, we observe that at least one heuristic succeeds. Then we iteratively divide the period by a factor of 10 and run all heuristics under this new value until all heuristics fail. We retain the period as the penultimate value, which is the last one before total failure. Note that this value depends on the workflow. It is chosen to give some tightness to the mapping problem: at least one heuristic succeeds to find a mapping that matches the bound T, but none does for $T/10$.

Results. In Figure 2, we plot the energy computed by the five heuristics for each application, given a CMP of size 4×4. On the horizontal axis, we represent the elevation of the SPG. For each value of the elevation, we average the results obtained on the randomly generated applications. On the vertical axis, we plot the inverse of the energy found by each heuristic, normalized to the minimum value obtained over all heuristics (so that the best heuristic returns 1, and the other ones return smaller values).

When computations are predominant, i.e., when the CCR is uniformly equal to 10, we observe that the two 1D heuristics always return good results. For small elevations, **DPA1D** is the best, but it often fails as soon as the elevation is greater than 4, thus leading to poor results. **DPA2D1D** returns very good results whatever the elevation of the graph. The 2D heuristic **DPA2D** is the best for elevations greater than 6, but it often fails on graphs with small elevation, because it wastes a lot of cores. For instance, if the application is exactly a pipeline (elevation 1), **DPA2D** can only enroll 4 cores over the 16 that are available. This fact holds true irrespective of the CCR. **Greedy** and **Random** are not as good, but **Greedy** always outperforms **Random**.

When communications and computations are more balanced (CCR of 1), similar results can be observed, but **DPA2D1D** is a bit further from the best solution, since it cannot utilize all the communication links. Finally, for communication-intensive applications (CCR of 0.1), **Random** gets much worse than the other heuristics: its energy can be up to 10 times worse than the best one. Also, the 1D heuristics do not perform well, except for small elevation graphs, because of their restriction in the communication pattern. In a general manner, we see that **DPA2D** is the best heuristic when the application graph has a high elevation.

Random and **Greedy** are running in average in 1 ms, **DPA2D** and **DPA2D1D** in 50 ms and **DPA1D** in 10 s.

Finally, in Table I, we report the number of failures for each heuristic. With a large CCR (10 or 1), **DPA2D1D** almost always succeeds to find a solution, which are in turn pretty good (see Figure 2). **Greedy** is always reasonably robust, whatever the CCR, and is followed closely by **Random**. **DPA2D** fails a bit more frequently because it does not often succeed with graphs of small elevation, as explained earlier. Finally, **DPA1D** succeeds only for graphs of small elevation, which leads to a very high failure rate.

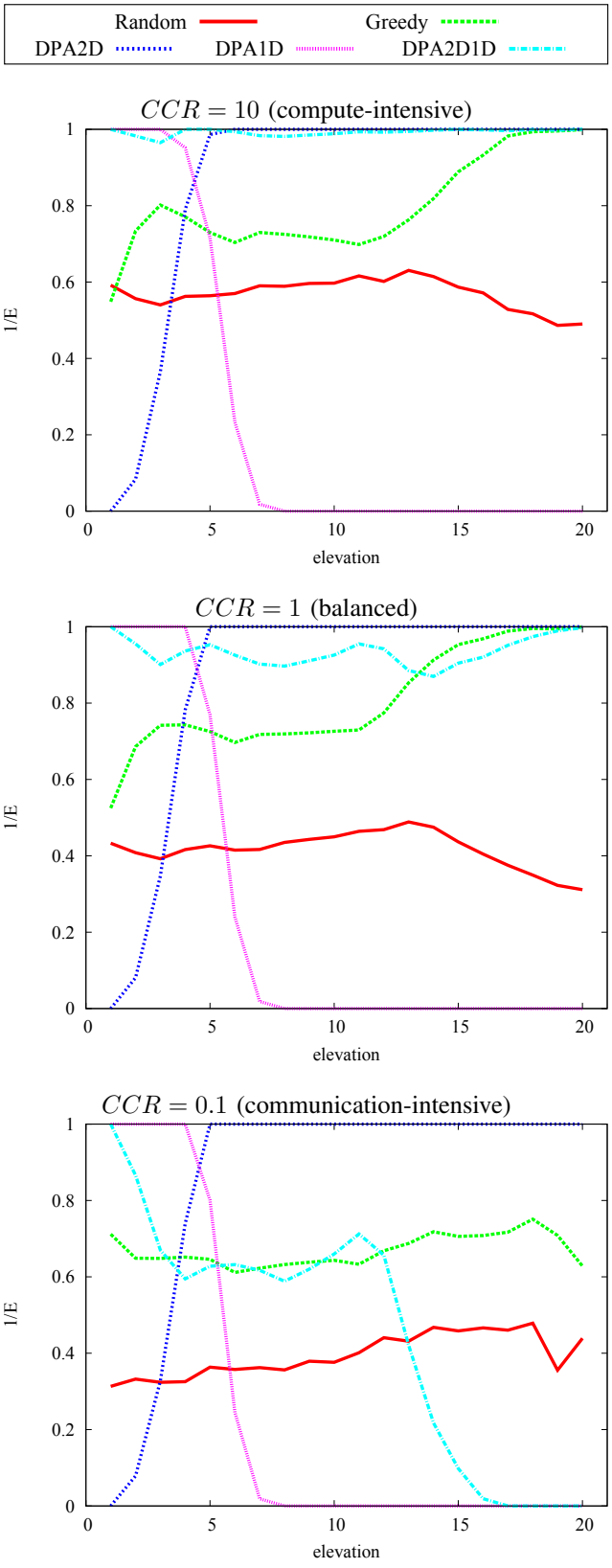

Figure 2. Normalized energy for a 4×4 CMP grid.

Table I
NUMBER OF FAILURES (OUT OF 1000 INSTANCES PER CCR VALUE).

CCR	**Random**	**Greedy**	**DPA2D**	**DPA1D**	**DPA2D1D**
10	29	28	85	758	1
1	29	28	78	760	3
0.1	300	287	348	670	458

We have performed further simulations on larger applications (with sizes up to 200), different CMP grid sizes (6×6), and also on a real set of applications extracted from the *StreamIt* suite [4]. The detailed results can be found in the companion research report [15], but overall the conclusions remain the same. **Greedy** seems to be a general-purpose heuristic that succeeds on most graphs and is always superior to **Random**. On the contrary, the three remaining heuristics are "specialized": for SPGs of small elevation, **DPA1D** is the best heuristic (it is even optimal with no communications). When the elevation gets higher, one should resort to **DPA2D1D**, which is able to find an efficient 1D solution while **DPA1D** fails, at least when communications are not too intensive. When communications increase, **DPA2D** becomes the most efficient heuristic, since it judiciously handles communications in the 2D grid.

VII. CONCLUSION

This paper contributes to the efficient utilization of multicores by considering an important class of streaming applications that can be modeled by a series-parallel graph, and studying the problem of mapping these applications to 2-dimensional tiled CMP architectures. The objective of the mapping is to minimize the energy consumption while maintaining a given level of performance, reflected by the rate of processing the data streams. Both processing/communication capabilities and power consumption are considered during the mapping, but it is assumed that only the processing power can be managed through dynamic voltage scaling. We will consider systems in which the communication power can also be managed in future work.

From a theoretical angle, we showed that most of the bi-criteria mapping problems were NP-complete, with the notable exception of uni-directional uni-line CMPs, for which an elaborated dynamic programming algorithm returns the optimal solution. The latter result holds true only for bounded-elevation SPGs, and the problem becomes NP-complete otherwise, which provides yet another evidence of the interest to restrict to particular graph structures rather than to deal with arbitrary DAGs. We strongly believe that bounded-elevation SPGs represent a very interesting trade-off, as they combine a large practical significance while being amenable to rigorous analysis. From a practical angle, the simulations conducted confirmed the efficiency of the main design principles underlying the various heuristics. While **Greedy** is the most robust approach, it is always superseded by one of the three specialized algorithms, either **DPA1D** or **DPA2D1D** for long pipeline-like graphs, or **DPA2D** for fat graphs of large elevation.

Finally, our future research will investigate general mappings, and assess the difference with DAG-partition mappings, both from a theoretical and a practical perspective. We also hope to succeed in simplifying the integer linear program for some problem instances, thereby providing an absolute measure of the quality of the various heuristics.

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

REFERENCES

[1] M. P. Mills, "The internet begins with coal," *Environment and Climate News*, 1999.

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

[3] J. Qin and T. Fahringer, "Advanced data flow support for scientific grid workflow applications," in *Proc. of SC'07, the Conf. on Supercomputing*, Nov. 2007, pp. 1–12.

[4] "StreamIt Project," http://groups.csail.mit.edu/cag/streamit/apps/stream-graphs.

[5] J. Subhlok and G. Vondran, "Optimal mapping of sequences of data parallel tasks," in *Proc. of PPoPP'95, the 5th Symp. on Principles and Practice of Parallel Programming*, 1995.

[6] Y. Gu and Q. Wu, "Maximizing workflow throughput for streaming applications in distributed environments," in *Proc. of ICCCN'10, the Int. Conf. on Computer Communication Networks*. IEEE CS, 2010.

[7] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The case for a single-chip multiprocessor," *SIGPLAN Not.*, vol. 31, pp. 2–11, Sep. 1996.

[8] G. Blake, R. Dreslinski, and T. Mudge, "A survey of multicore processors," *Signal Processing Magazine*, vol. 26, no. 6, pp. 26–37, Nov. 2009.

[9] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proc. of DAC'04, the 41st annual Design Automation Conference*, 2004, pp. 275–280.

[10] N. Bansal, T. Kimbrel, and K. Pruhs, "Speed scaling to manage energy and temperature," *Journal of the ACM*, vol. 54, no. 1, pp. 1–39, 2007.

[11] J.-J. Chen and C.-F. Kuo, "Energy-Efficient Scheduling for Real-Time Systems on Dynamic Voltage Scaling (DVS) Platforms," in *Proc. of the Int. Workshop on Real-Time Computing Systems and Applications*, 2007, pp. 28–38.

[12] L. Wang, G. von Laszewski, J. Dayal, and F. Wang, "Towards Energy Aware Scheduling for Precedence Constrained Parallel Tasks in a Cluster with DVFS," in *Proc. of CCGrid'2010, the 10th Int. Conf. on Cluster, Cloud and Grid Computing*, May 2010, pp. 368–377.

[13] AMD, "ACP - The Truth About Power Consumption Starts Here," http://www.amd.com/us/Documents/43761C_ACP_WP_EE.pdf, 2010.

[14] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar, "Power and Thermal Management in the Intel CoreTM Duo Processor," *Intel Technology Journal*, vol. 10, no. 2, pp. 109–122, May 2006.

[15] A. Benoit, R. Melhem, P. Renaud-Goud, and Y. Robert, "Energy-aware mappings of series-parallel workflows onto chip multiprocessors," INRIA, France, Research Report RR-7521, Jan. 2011, available at http://graal.ens-lyon.fr/~abenoit/.

[16] L. Yang and L. Man, "On-Line and Off-Line DVS for Fixed Priority with Preemption Threshold Scheduling," in *Proc. of ICESS'09, the Int. Conf. on Embedded Software and Systems*, May 2009, pp. 273–280.

[17] A. Benoit, P. Renaud-Goud, and Y. Robert, "Performance and energy optimization of concurrent pipelined applications," in *Proc. of IPDPS, the Int. Parallel and Distributed Processing Symp*. IEEE CS Press, May 2010.

[18] G. Ascia, V. Catania, and M. Palesi, "Multi-objective mapping for mesh-based NoC architectures," in *Proc. of CODES+ISSS'04, the Int. Conf. on Hardware/ Software Codesign and System Synthesis*, Sep. 2004, pp. 182–187.

[19] M. Al Faruque, R. Krist, and J. Henkel, "ADAM: Run-time agent-based distributed application mapping for on-chip communication," in *Proc. of DAC'08, the 45th Design Automation Conf.*, Jun. 2008, pp. 760–765.

[20] R. Xu, R. Melhem, and D. Mossé, "Energy-aware scheduling for streaming applications on chip multiprocessors," in *Proc. of RTSS'07, the 28th IEEE Int. Real-Time Systems Symp.*, 2007, pp. 25–38.

[21] P. Mahr, C. Lorchner, H. Ishebabi, and C. Bobda, "SoC-MPI: A Flexible Message Passing Library for Multiprocessor Systems-on-Chips," in *Proc. of ReConFig'08, the Int. Conf. on Reconfigurable Computing and FPGAs*, Dec. 2008, pp. 187–192.

[22] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 211–222, Oct. 2002.

[23] M. Hammoud, S. Cho, and R. Melhem, "A dynamic pressure-aware associative placement strategy for large scale chip multiprocessors," *Computer Architecture Letters*, vol. 9, no. 1, pp. 29–32, Jan. 2010.

[24] Z. Chishti, M. Powell, and T. Vijaykumar, "Optimizing replication, communication, and capacity allocation in CMPs," in *Proc. of ISCA'05, the 32nd Int. Symp. on Computer Architecture*, Jun. 2005, pp. 357–368.

[25] J. Jaehyuk Huh, C. Changkyu Kim, H. Shafi, L. Lixin Zhang, D. Burger, and S. Keckler, "A NUCA Substrate for Flexible CMP Cache Sharing," *IEEE Trans. on Parallel and Distributed Systems*, vol. 18, no. 8, pp. 1028–1040, 2007.

[26] M. Hammoud, S. Cho, and R. Melhem, "ACM: An Efficient Approach for Managing Shared Caches in Chip Multi- processors," in *Proc. of HiPEAC'09, the 4th Int. Conf. on High Performance Embedded Architectures and Compilers*, 2009, pp. 355–372.

[27] 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.

[28] J.-J. Chen, "Expected energy consumption minimization in DVS systems with discrete frequencies," in *Proc. of SAC'08, Symp. on Applied Computing*, 2008, pp. 1720–1725.

[29] L. Niu, "Energy Efficient Scheduling for Real-Time Embedded Systems with QoS Guarantee," in *Proc. of RTCSA, the 16th Int. Conf. on Embedded and Real-Time Computing Systems and App.*, Aug. 2010, pp. 163 –172.

[30] J. D. Owens, W. J. Dally, R. Ho, D. N. J. Jayasimha, S. W. Keckler, and L.-S. Peh, "Research Challenges for On-Chip Interconnection Networks," *IEEE Micro*, vol. 27, pp. 96–108, 2007.

[31] G. Chen, F. Li, M. Kandemir, and M. J. Irwin, "Reducing NoC energy consumption through compiler-directed channel voltage scaling," *SIGPLAN Not.*, vol. 41, pp. 193–203, 2006.