

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

Anne Benoit — Rami Melhem — Paul Renaud-Goud — Yves Robert

N° 7521

April 2011

Distributed and High Performance Computing

 **R**
*apport
de recherche*

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

Anne Benoit , Rami Melhem , Paul Renaud-Goud , Yves Robert

Theme : Distributed and High Performance Computing
Équipe-Projet GRAAL

Rapport de recherche n° 7521 — April 2011 — 40 pages

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 comprehensive simulations using the *StreamIt* workflow suite and randomly generated series-parallel graphs, and various CMP grid sizes.

Key-words: series-parallel graph; DAG; mapping; multicore; CMP; energy; power; period; throughput; DVS; DVFS; complexity; simulation; streaming applications; optimization.

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

Résumé : Dans ce rapport de recherche, nous nous intéressons au placement d'applications de type streaming représentées sous la forme d'un graphe série-parallèle sur un processeur multi-cœur, en essayant de minimiser l'énergie consommée tout en n'excédant pas une borne sur un critère de performance, la période. La partie théorique démontre la NP-complétude ou la polynomialité du problème, selon des propriétés structurelles du multi-cœur (chaîne de cœurs, uni- ou bi-directionnelle, grille de cœurs) et la largeur du graphe de l'application (bornée ou non). Le problème le moins contraint étant NP-complet, nous décrivons dans la partie expérimentale cinq heuristiques, puis les comparons entre elles, et donnons un programme linéaire en nombres entiers qui permet d'obtenir la solution optimale en temps exponentiel.

Mots-clés : graphe série-parallèle; DAG; mapping; multi-cœur; énergie; puissance; période; DVS; DVFS; complexité; simulation; streaming; optimisation.

Contents

1	Introduction	4
2	Related work	5
3	Framework	7
3.1	Applicative framework	7
3.2	Platform	10
3.3	Mapping strategies	10
3.4	Period	11
3.5	Energy model	12
4	Complexity results	12
4.1	Uni-directional uni-line CMP	13
4.2	Bi-directional uni-line CMP	14
4.3	Square CMP	16
4.4	Integer linear program	19
4.4.1	Constants	19
4.4.2	Variables	20
4.4.3	Constraints	20
4.4.4	Objective function	22
5	Heuristics	22
5.1	Random heuristic	22
5.2	Greedy heuristic	23
5.3	2D dynamic programming algorithm	24
5.4	1D heuristics	25
6	Simulation results	26
6.1	Simulation setting	26
6.1.1	Streaming applications	26
6.1.2	CMP configuration	26
6.1.3	Period bound T	27
6.2	Simulation results	27
6.2.1	<i>StreamIt</i> suite	27
6.2.2	Random SPGs	30
7	Conclusion	36

1 Introduction

The *energy consumption* of computational platforms has recently become a critical problem, both for economic and environmental reasons [34]. 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 [16, 44, 43, 52]. Most of these applications have simple and regular task graphs, such as linear chains, trees, fork-join graphs, or general SPGs (see Section 3.1 for a formal definition of SPGs). For instance, all the benchmarks of the *StreamIt* suite [45] 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 [46, 16, 20]. 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. [38] 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 simulations.

The paper is organized as follows. We first survey related work in Section 2. Then we detail the framework in Section 3, and we provide complexity results in Section 4. The heuristics are described in Section 5, and simulation results in Section 6. Finally, we conclude and discuss future research directions in Section 7.

2 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 [31, 29, 19]. The dynamic voltage and frequency scaling (DVFS) technique has been extensively studied, since it may lead to efficient energy/performance trade-offs [26, 18, 4, 12, 28, 51, 48]. Current microprocessors (for instance, from AMD [2] and Intel [35]) 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.

In this paper, we aim at minimizing the energy consumption for series-parallel graph (SPG) applications which are mapped onto a chip multiprocessor (CMP). We first discuss related work on SPG applications, then we review different energy minimization approaches. Finally, we relate work on mapping problems on CMPs.

Series-parallel workflow applications. Classical workflow applications usually consists of a directed acyclic graph: the application is made of several tasks, and there are dependencies between these tasks. However, it turns out that many of these graphs are series-parallel graphs. For instance, in [33], McClatchey et al. introduce a prototype scientific workflow management system entitled CRISTAL, and the distributed scientific workflow applications that they consider are SPGs. In [41], Qin and Fahringer discuss several scientific grid workflow applications, which are all structured as SPGs: the WIEN2k workflow performs electronic structure calculations of solids using density functional theory [7], the MeteoAG workflow is a meteorology simulation application [43], and the GRASIL workflow calculates the spectral energy distribution of galaxies [44]; this latter application has actually a fork-join graph. A last example is the fMRI workflow [52], which is a cognitive neuroscience application.

DVFS and optimization problems. When dealing with energy consumption, the most usual optimization function consists of minimizing the energy consumption, while ensuring some performance guarantees (real-time constraints, such as a bound on the total execution time, or a threshold throughput). Specifically, in [37], Okuma et al. demonstrate that voltage scaling is far more effective than the shutdown approach, which simply stops the power supply when the system is inactive. De Langen and Juurlink [30] discuss leakage-aware scheduling heuristics which investigate both DVS and processor shutdown, since static power consumption due to leakage current is expected to increase significantly. In the context of real-time embedded systems, Lee and Sakurai [31] show how to exploit slack time arising from workload variation, thanks to a software feedback control of supply voltage. Prathipati [40] discusses techniques to take advantage of run-time variations in the execution time of tasks; it determines the minimum voltage under which each task can be executed, while guaranteeing the deadlines of each task. In [50], dynamic programming algorithms are given to minimize the expected energy consumption in real time systems using frequency and voltage scaling. Yang and Lin [51] discuss algorithms with preemption, using DVS techniques; substantial energy can be saved using these algorithms, which succeed to claim the static and dynamic slack time, with little overhead. Most of these papers deal with classical scheduling of task graph applications, which are not streaming applications. The techniques are similar, but the performance guarantee is a deadline on the total execution time. Rather, we consider workflow applications, i.e., several data sets must be processed by the task graph, and hence we bound the application period.

The problem of mapping workflow applications with the structure of a linear chain onto parallel platforms has already been widely studied, in particular on homogeneous platforms (see the pioneering paper [47]) and later for heterogeneous platforms [6]. These results are extended to account for energy consumption in [5], where the target problem is mapping several linear chain applications on a fully interconnected platform, with three optimization criteria: power, period, and latency (execution time for one data set).

Mapping applications to chip multiprocessors. Many researchers have considered the mapping of tasks and threads to CMPs that are connected by a 2-dimensional network on a chip. The work in [3] introduces an approach to multi-objective exploration of mapping general task graphs to mesh-based CMPs using evolutionary algorithms. The approach is an efficient and accurate way to

obtain the Pareto mappings that optimize performance and power consumption. In [25], an architecture-aware analytic mapping algorithm is presented. It uses a metric space that exactly captures the CMP topology to efficiently solve the problem. In [10], a compiler framework is presented to map the source code of an application to a mesh-based chip multiprocessor system. Compiler techniques are also used in [9] to dynamically change the speed of communication channels in CMPs to reduce energy consumption. In [1], the mapping of applications to heterogeneous multi-processor systems is performed by invoking runtime agents that are distributed among the processors. None of the above work considers the mapping of streaming applications onto CMP with the objective of minimizing power consumption while maintaining a specified throughput (period).

Summary. In this paper, the application is a workflow whose structure is series-parallel task graph, and the goal is to map this application onto a CMP, with the objective of minimizing the energy consumption, given a threshold on the period of the workflow. We are therefore extending previous work, 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 [49] 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 5.3 follows a similar two-phase strategy but obeys the DAG-partition mapping rule (see Section 3.3).

3 Framework

In this section, we first describe the applicative framework (Section 3.1) and the target platform (Section 3.2). Then we detail the mapping strategies in Section 3.3. Finally, we formally define the bi-criteria optimization problem: we aim at minimizing the energy consumption while not exceeding a prescribed period. The formula to check that the period is not exceeded is given in Section 3.4, and the model for energy consumption is outlined in Section 3.5.

3.1 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 $S_1^{(1)}, \dots, S_{n_1}^{(1)}$, and SPG_2 with nodes $S_1^{(2)}, \dots, S_{n_2}^{(2)}$, 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 serial 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 [45] 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 4.1). Dealing with bounded-elevation graphs enables us to identify polynomial instances, thus providing optimal solutions for some problem instances.

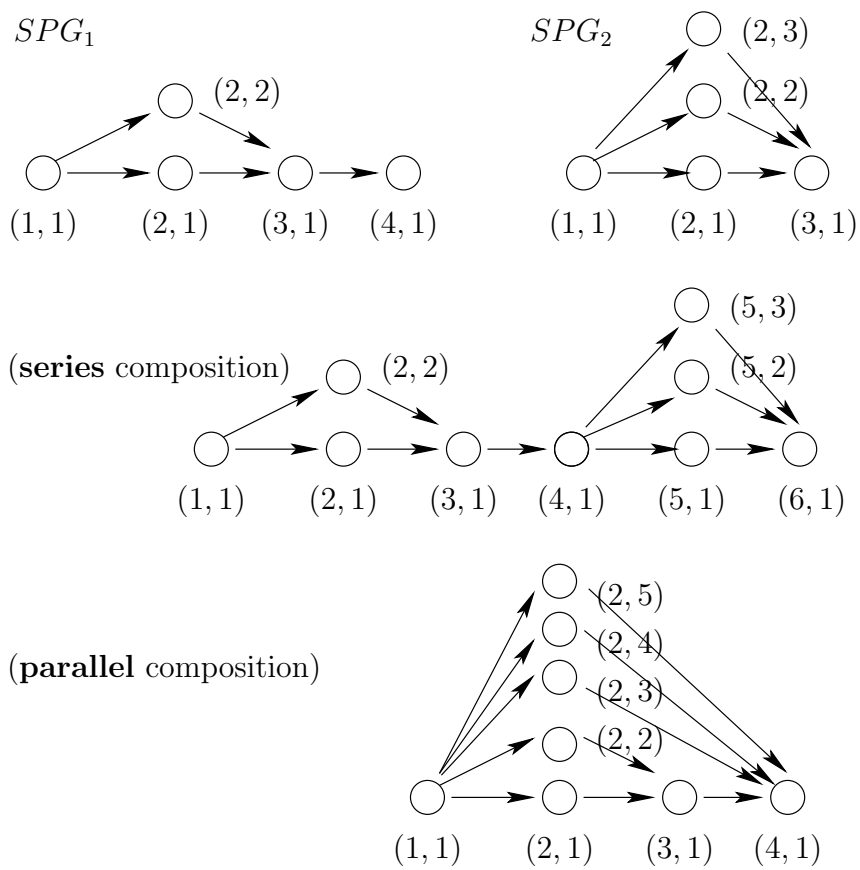


Figure 1: Examples of SPG composition.

3.2 Platform

The target platform is a CMP (Chip MultiProcessor), 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 (in each direction). 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 [32] 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 [27]. Therefore, communication through shared memory ultimately translates to exchange of coherence traffic between the tiles [22, 14, 24]. 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 is 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 coherent¹. Hence, irrespectively of the programming model used to implement the SPG, the weight on a directed 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 2, 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 $\mathbf{S} = \{s^{(1)}, \dots, s^{(m)}\}$ 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 $s^{(k)} \in \mathbf{S}$ on a given core. Each speed induces a different dynamic power consumption, as discussed in Section 3.5 below.

3.3 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

¹It 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 [21].

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 allow 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 [17]).

Formally, the mapping is defined by an allocation function

$$alloc : \{1, \dots, n\} \rightarrow \{1, \dots, p\} \times \{1, \dots, q\},$$

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.

3.4 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 $s_{u,v} \in \mathbb{S}$. The cycle-time of $\mathcal{C}_{u,v}$ for computations is $w_{u,v}/s_{u,v}$. For communications, $b_{(u,v) \rightarrow (u',v')} = \sum_{1 \leq i,j \leq n | (u,v) \rightarrow (u',v') \in path_{i,j}} \delta_{i,j}$ is the number of bits sent from $\mathcal{C}_{u,v}$ to a neighbor core $\mathcal{C}_{u',v'}$ ². The cycle-time of the communication link $(u, v) \rightarrow (u', v')$ is $b_{(u,v) \rightarrow (u',v')}/BW$.

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 .

² $(u'=u+1$ and $v'=v)$ or $(u'=u-1$ and $v'=v)$ or $(u'=u$ and $v'=v+1)$ or $(u'=u$ and $v'=v-1)$.

3.5 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: $\mathcal{A} = \{\mathcal{C}_{u,v}, 1 \leq u \leq p, 1 \leq v \leq q \mid \exists 1 \leq i \leq n, \text{alloc}(i) = (u,v)\}$. For each core $\mathcal{C}_{u,v} \in \mathcal{A}$, let $w_{u,v}$ be its assigned work and $s_{u,v}$ its speed. The total energy consumed for computations is

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

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

$$E^{(\text{comm})} = P_{\text{leak}}^{(\text{comm})} \times T + \left(\sum_{u,v} \sum_{u',v'} b_{(u,v) \rightarrow (u',v')} \right) \times E^{(\text{bit})},$$

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:

Definition 1 (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.*

4 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 4.1), bi-directional uni-line (see Section 4.2), or bi-directional 2D mesh (see Section 4.3). 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 4.4.

4.1 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:

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

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 $n^{y_{\max}}$ 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 $O(q \times n \times n^{y_{\max}})$, which is polynomial since y_{\max} is a constant. ■

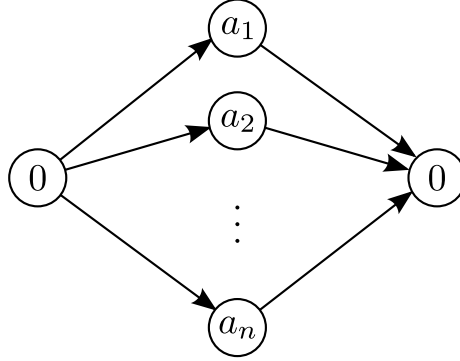


Figure 2: Unbounded-elevation SPG for the uni-directional uni-line CMP proof.

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

Proposition 1 *The extension of $\text{MINENERGY}(T)$ to unbounded-elevation SPGs on a uni-directional uni-line CMP is NP-complete.*

Proof. 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, it is easy to check in polynomial time that it is valid by computing its period.

To establish the completeness, we use a reduction from 2-PARTITION [17]. We consider an instance \mathcal{I}_1 of 2-PARTITION: given n strictly positive integers a_1, a_2, \dots, a_n , does there exist a subset I of $\{1, \dots, n\}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$? Let $S = \sum_{i=1}^n a_i$.

We build an instance \mathcal{I}_2 of our problem: the application consists of a fork-join graph of elevation n , as illustrated in Figure 2. We denote by S_0 the source node, S_{n+1} the sink node, and S_i , for $1 \leq i \leq n$, is the i^{th} node of the fork-join. For computation costs, we have $w_0 = w_{n+1} = 0$, and $w_i = a_i$, and there are no communication costs. The platform consists of two cores which can operate only at a unique speed $s = 1$. Finally, we ask whether we can achieve a period $\frac{S}{2}$.

Clearly, the size of \mathcal{I}_2 is polynomial in the size of \mathcal{I}_1 . The equivalence between both problems is straightforward: if \mathcal{I}_1 has a solution I , then we assign S_0 and $\{S_i\}_{i \in I}$ to the first core, S_{n+1} and $\{S_i\}_{i \notin I}$ to the second core. The mapping is a DAG-partition, and its period is $\frac{S}{2}$, therefore we find a solution to \mathcal{I}_2 . On the other hand, if \mathcal{I}_2 has a solution, the period on each core must be exactly $\frac{S}{2}$ because the total computation requirement is S , and therefore we have a 2-partition of the stages S_i , for $1 \leq i \leq n$. This concludes the proof. ■

4.2 Bi-directional uni-line CMP

Theorem 2 *The $\text{MINENERGY}(T)$ problem on a bi-directional uni-line CMP is NP-complete.*

Proof. As for Proposition 1, the simpler mono-criterion problem of matching a prescribed period T , without any energy consideration, is NP-complete. How-

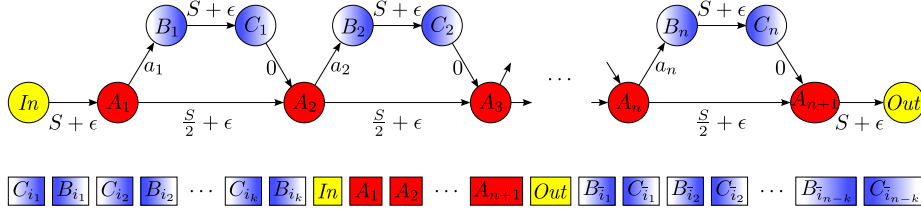


Figure 3: Bounded-elevation SPG and mapping for the bi-directional uni-line CMP proof.

ever, the reduction proof becomes quite involved, since we consider a bounded-elevation SPG.

The problem is obviously in NP: given a period and a mapping, it is easy to check in polynomial time that it is valid by computing its period. To establish the completeness, we use a reduction from 2-PARTITION [17]. We consider an instance \mathcal{I}_1 of 2-PARTITION: given n strictly positive integers a_1, a_2, \dots, a_n , does there exist a subset I of $\{1, \dots, n\}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$? Let $S = \sum_{i=1}^n a_i$.

We build an instance \mathcal{I}_2 of our problem: the bounded-elevation SPG of the application is represented in Figure 3. There are $3n + 3$ stages, computation costs of each stage are equal to 1, and communication costs are depicted in the figure. The platform is a bi-directional uni-line CMP of $1 \times q$ cores, where $q = 3n + 3$. Each core can operate only at a unique speed $s = 1$, and the bandwidth of each link is $BW = 3S/2 + \epsilon$. Finally, we ask whether we can achieve a period of 1. Clearly, the size of \mathcal{I}_2 is polynomial in the size of \mathcal{I}_1 . We now show that instance \mathcal{I}_1 has a solution if and only if instance \mathcal{I}_2 does. First note that any solution of \mathcal{I}_2 is a one-to-one mapping, because of the constraint on the period and the computation costs of stages. Indeed, if two or more stages were mapped onto the same core, the period would be at least 2.

Assume first that \mathcal{I}_1 has a solution, I . We assume that $I = \{i_1, \dots, i_k\}$ and $\bar{I} = \{1, \dots, n\} \setminus I = \{\bar{i}_1, \dots, \bar{i}_{n-k}\}$, with $\sum_{j=1}^k a_{i_j} = \sum_{j=1}^{n-k} a_{\bar{i}_j} = S/2$. For \mathcal{I}_2 , we map the application graph onto the CMP as illustrated in Figure 3: for all $j \in \{1, \dots, k\}$, C_j is mapped onto \mathcal{C}_{2j-1} and B_j onto \mathcal{C}_{2j} . Then the stage In is mapped onto \mathcal{C}_{2k+1} , for all $j \in \{1, \dots, n+1\}$, A_j is mapped onto \mathcal{C}_{2k+1+j} , and the stage Out is mapped onto \mathcal{C}_{2k+n+3} . Finally for all $j \in \{1, \dots, n-k\}$, B_j is mapped onto $\mathcal{C}_{2k+n+2+2j}$, and C_j onto $\mathcal{C}_{2k+n+2j+3}$. The mapping is one-to-one so that the period constraint is fulfilled for computations. We now show that, on each link, the sum of communications does not exceed BW , and hence the bound on the period is not violated.

Let us first consider the link $\ell_{2k+1+j}^{(h)}$, with $j \in \{1, \dots, n\}$: the amount of communications on this link is equal to $S/2 + \epsilon$ (communication from A_j to A_{j+1}), plus at most $\sum_{i=1}^n a_i = S$ (communications from A_i to B_i), therefore a total of no more than $3S/2 + \epsilon = BW$.

Then we consider the link $\ell_{2j-1}^{(h)}$, with $j \in \{1, \dots, k\}$: the amount of communications is then $S + \epsilon$ (from B_{i_j} to C_{i_j}) plus at most $\sum_{i \in I} a_i = S/2$ (communications from A_i to B_i , for $i \in I$), which is no more than BW . Finally, on the link $\ell_{2j}^{(h)}$, with $j \in \{1, \dots, k\}$, there are at most $\sum_{i \in I} a_i = S/2$ communications

(from A_i to B_i , for $i \in I$). Similarly, we can prove that the bandwidth is not exceeded on link $\ell_{2k+n+2+j}^{(h)}$, for $j \in \{1, \dots, 2(n-k)\}$.

Only two links remain now: $\ell_{2k+1}^{(h)}$ and $\ell_{2k+n+2}^{(h)}$. The only communications not equals to 0 that go through $\ell_{2k+1}^{(h)}$ are the communications from A_i to B_i , for $i \in I$, thus the link bandwidth constraint is fulfilled. This holds true for $\ell_{2k+n+2}^{(h)}$, reasoning with \bar{I} instead of I . We conclude that \mathcal{I}_2 has a solution.

Assume now that \mathcal{I}_2 has a solution. We prove that the mapping is necessary similar to that of Figure 3, and that stages B_i and C_i must be 2-partitioned.

Let σ be the permutation of $\{1, \dots, n+1\}$ such that, for each $i \in \{1, \dots, n\}$, the core assigned to $A_{\sigma(i)}$ is to the left of the one assigned to $A_{\sigma(i+1)}$. First, let us assume that there exists $i^{(0)} \in \{1, \dots, n\}$ such that the stage In is mapped between $A_{\sigma(i^{(0)})}$ and $A_{\sigma(i^{(0)+1})}$. Since there is a path (with edges of weight $S/2 + \epsilon$) going through all the $A_{\sigma(i)}$, a communication of size $S/2 + \epsilon$ occurs on all links between the core assigned to $A_{\sigma(i^{(0)})}$ and the core assigned to $A_{\sigma(i^{(0)+1})}$. Because of the mapping of In , we deduce that there is a link on which the amount of communications is at least $3S/2 + 2\epsilon$, which leads to a contradiction. Therefore, we showed that the core that is assigned to In is either to the left of the core assigned to $A_{\sigma(1)}$ or to the right of the core assigned to $A_{\sigma(n+1)}$. The same result holds for Out (similar proof).

Moreover note that In and Out cannot be on the same side, otherwise either the link after the core assigned to $A_{\sigma(n+1)}$ or the link before the core assigned to $A_{\sigma(1)}$ would transmit at least two communications of size $S + \epsilon$. We can assume, without loss of generality that In is mapped on the left, and Out on the right.

Similarly, for all $i \in \{1, \dots, n\}$, B_i and C_i cannot be mapped onto a core between the core assigned to a $A_{\sigma(i')}$ and the one assigned to $A_{\sigma(i'+1)}$, or In and $A_{\sigma(1)}$, or $A_{\sigma(n+1)}$ and Out . The B_i are thus mapped either to the left of In , or to the right of Out , similarly to Figure 3.

Finally, let I be a subset of $\{1, \dots, n\}$ such that $i \in I$ if and only if B_i is mapped to the left of In . Then, since the bandwidth bound is not exceeded between the core assigned to In and the one assigned to $A_{\sigma(1)}$ on one hand, and between $A_{\sigma(1)}$ and Out on the other hand, we have necessarily $\sum_{i \in I} a_i + S + \epsilon \leq 3S/2 + \epsilon$ and $\sum_{i \notin I} a_i + S + \epsilon \leq 3S/2 + \epsilon$. Therefore, $\sum_{i \in I} a_i = \sum_{i \notin I} a_i = S/2$, \mathcal{I}_1 has a solution, which concludes the proof. ■

4.3 Square CMP

In this section, we focus on square CMPs. We know from Theorem 2 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 now establish this complexity:

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

Proof. As for Theorem 2, the simpler mono-criterion problem of matching a prescribed period T , without any energy consideration, is NP-complete. The problem is obviously in NP: given a period and a mapping, it is easy to check in polynomial time that it is valid by computing its period.

To establish the completeness, we use a reduction from 2-PARTITION [17]. We consider an instance \mathcal{I}_1 of 2-PARTITION: given $3n + 1$ strictly positive integers $a_1, a_2, \dots, a_{3n+1}$, does there exist a subset I of $\{1, \dots, 3n + 1\}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$? Let $S = \sum_{i=1}^{3n+1} a_i$.

We build the following instance \mathcal{I}_2 for our problem, re-using the construction proposed in Theorem 2. The CMP is composed of $p \times p$ cores with a single speed 1, linked with a bandwidth $BW = 3S/2 + \epsilon$, where $p = 6n + 4$. The series-parallel graph is described as a directed acyclic graph (DAG) in Figure 5, using some widgets introduced in Figure 4. To transform this DAG into a SPG, we use the transformation explained in Figure 6: the blue nodes in widgets G can be replaced by two nodes with computation cost $1/2$, which must be mapped onto the same core because of bandwidth constraints. All computation costs in the DAG are equal to 1, and in the following we conduct the reasoning on the DAG. The size of blue and green communications is equal to the bandwidth BW , and there is no communication between two H_i widgets, neither between E_2 and H_{6n} . The size of communications from E_1 to E_{12} on the one side, and from E_{13} to E_2 , on the other side, is equal to ϵ . The subgraph between E_{12} and E_{13} is the graph of Figure 3, replacing n by $3n + 1$. Finally, we ask whether we can achieve a period of 1. Clearly, the size of \mathcal{I}_2 is polynomial in the size of \mathcal{I}_1 .

We now show that instance \mathcal{I}_1 has a solution if and only if instance \mathcal{I}_2 does. First note that any solution of \mathcal{I}_2 is a one-to-one mapping, because of the constraint on the period and the computation costs of stages. Indeed, if two or more stages were mapped onto the same core, the period would be at least 2.

Assume first that \mathcal{I}_2 has a solution. We show that the red nodes are necessary mapped onto a linear chain of cores, and communications never escape out of this linear chain.

In each widget $G_{i,j}$, the two communications between $D_{i,2k-1}$ and $D_{i,2k}$, for $k \in \{1, \dots, j\}$, must occur on at least 4 links and no other communications not equal to 0 can use those links, because one communication fills entirely a link, and the mapping is one-to-one. In the same way, in every widget H_i , the three communications between $P_{i,2k-1}$, $P_{i,2k}$ and $P_{i,2k+1}$, for $k \in \{1, \dots, i\}$, must occur on at least 4 links and no other communication (not equal to 0) can use those links. Moreover, there are 19 more communications of size BW , which thus require at least 19 more communication links. Altogether, we need at least:

$$\begin{aligned} \sum_{i=1}^{6n} 4i + 3 \times (15n - 1) + (15n + 4) + 19 &= 2(6n)(6n + 1) + 60n + 20 \\ &= 72n^2 + 72n + 20 \\ &= 2p(p - 1) - 2(p - 2) \end{aligned}$$

communication links to map all communications except the red ones. If we use more links to map blue or green communications, there would be at most $2(p - 2) - 1$ free remaining links. Now the graph contains $2(p - 2) + 1$ red nodes, thus a red node would be isolated (i.e., it would have no available communication link), which is not possible, because each red node must communicate with at least one other red node. Thus, we have exactly $2(p - 2)$ communication links for the red communications and $2p \times (p - 1) - 2(p - 2)$ communication links for blue and green communications.

The blue nodes of degree 3 are on the border of the CMP: those nodes cannot be mapped onto a corner core, because they need at least 3 free communication

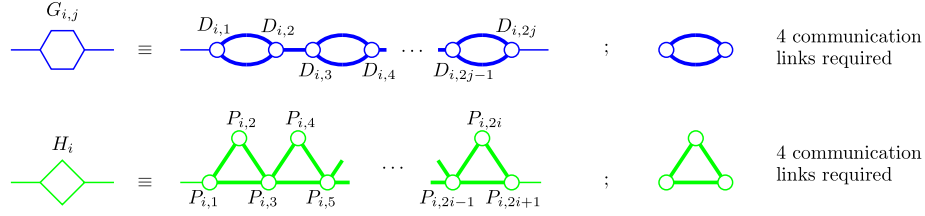


Figure 4: Widgets.

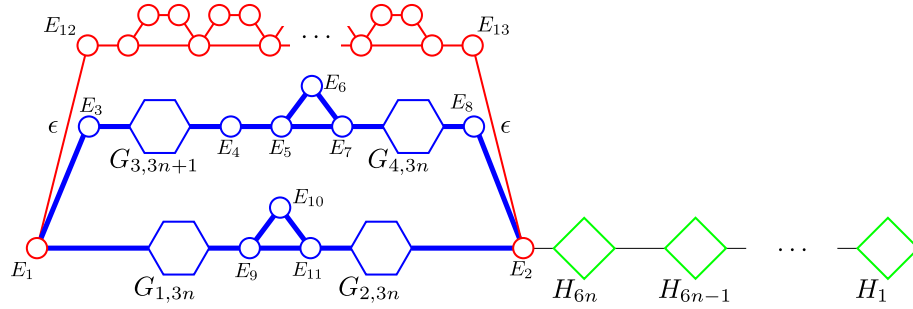


Figure 5: DAG.

links, and they cannot be mapped onto a middle core either. In this case, three of four communication links would be indeed used, and the remaining empty communication link could not be used by another communication: an incoming communication could not exit through another link. The nodes $P_{3,1}, \dots, P_{3,p-2}$ of the widget $G_{3,3n+1}$ must be mapped in order on a border, otherwise we lose at least one communication link. Without loss of generality, we can assume that they are mapped respectively onto cores $\mathcal{C}_{2,1}, \dots, \mathcal{C}_{p-1,1}$. The communication between $P_{3,1}$ and $P_{3,2}$ must occur on links $\ell_{2,1}^{(h)}$, $\ell_{2,2}^{(v)}$ and $\ell_{3,1}^{(h)}$ in order not to lose any communication link. In the same way, the communication between $P_{3,p-3}$ and $P_{3,p-2}$ takes the links $\ell_{p-2,1}^{(h)}$, $\ell_{p-2,2}^{(v)}$ and $\ell_{p-1,1}^{(h)}$. As a result, again from the fact that we cannot lose any communication link, E_4 is mapped onto $\mathcal{C}_{p,1}$, E_3 onto $\mathcal{C}_{1,1}$ then E_1 onto $\mathcal{C}_{1,2}$. In the same manner again, nodes $E_5, E_7, P_{4,1}, \dots, P_{4,p-2}$ are mapped respectively onto cores $\mathcal{C}_{p,2}, \dots, \mathcal{C}_{p,p-1}$, E_8 onto $\mathcal{C}_{p,p}$ and E_2 onto $\mathcal{C}_{p-1,p}$.

If the graph composed of the cores on which a red node is mapped, linked with the communication links where a red communication occurs, is not a chain, E_1 and E_2 cannot be connected, because there are only $2(p-2)$ remaining communication links and the Manhattan distance between E_1 and E_2 is $2(p-2)$. Since the 2 additional nodes E_1 and E_2 , and the two communications of weight ϵ do not change anything, we are in the case of proof of Theorem 2. Therefore \mathcal{I}_1 has a solution.

We assume now that \mathcal{I}_1 has a solution. We give the mapping for $n = 1$ in Figure 7, which can convince us that such a mapping, where red nodes and communications are mapped onto a linear chain, can be found for any $n > 1$. Then we use again the proof of Theorem 2 to conclude that \mathcal{I}_2 has a solution, and hence conclude the proof. ■

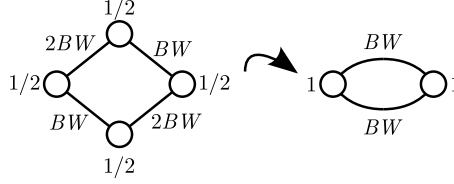
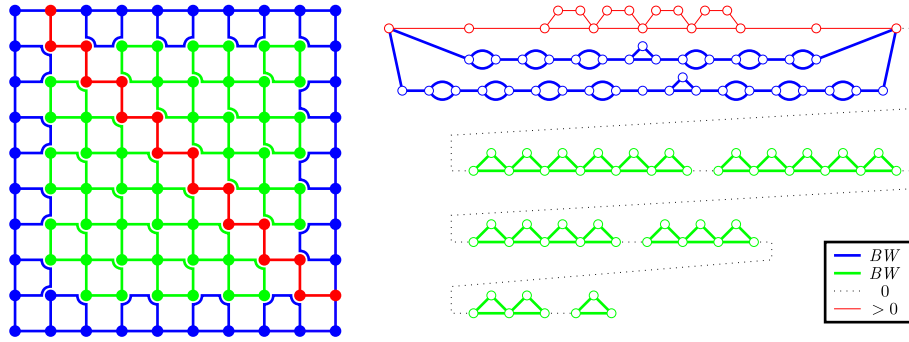


Figure 6: SPG to DAG.

Figure 7: Example 10×10 .

4.4 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 formulate in this section 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 [15].

4.4.1 Constants

We first define the set of constant values that define our problem. The application is composed of n stages S_1, \dots, S_n , and a set of edges \mathcal{E} :

- for $1 \leq i \leq n$, $w(i)$ is the amount of computations of node S_i , i.e., it corresponds to the w_i parameter;
- for $1 \leq i, j \leq n$, $\ell(i, j) = 1$ if there is a link between S_i and S_j (i.e., if $L_{i,j} \in \mathcal{E}$), and then $\delta(i, j)$ is the amount of communications between the two stages (it corresponds to the $\delta_{i,j}$ parameter); otherwise $\ell(i, j) = \delta(i, j) = 0$;
- we define ℓ^* as the transitive closure of ℓ , i.e., for $1 \leq i, j \leq n$, $\ell^*(i, j) = 1$ if there is a dependence path from S_i to S_j , otherwise $\ell^*(i, j) = 0$.

For the platform, we consider a $p \times q$ CMP, and we need to compute beforehand the energy consumed by a core when running at any speed.

- for $1 \leq k \leq m$, $s(k)$ is the k -th possible speed of a core;
- for $1 \leq k \leq m$, $E_{\text{stat}} = P_{\text{leak}}^{(\text{comp})} \times T$ (static energy consumption for one core);
- for $1 \leq k \leq m$, $E_{\text{dyn}}(k) = P_{s(k)}^{(\text{comp})}/s(k)$ (it must be multiplied by the amount of computation on the core to return the dynamic energy consumption, see Section 3.5).

Finally, BW is the link bandwidth, and T is the bound on the period.

4.4.2 Variables

Now that we have defined the constants that define our problem, we define unknown variables to be computed:

- for $1 \leq i \leq n$, $1 \leq k \leq m$, $1 \leq u \leq p$ and $1 \leq v \leq q$, $x_{i,k,u,v}$ is a boolean variable equal to 1 if stage S_i is mapped onto core $\mathcal{C}_{u,v}$, operated at speed $s(k)$, and 0 otherwise; there are $n \times m \times p \times q$ such variables;
- for $1 \leq k \leq m$, $m_{k,u,v}$ is a boolean variable equal to 1 if core $\mathcal{C}_{u,v}$ is operated at speed $s(k)$, and 0 otherwise; there are $m \times p \times q$ such variables;
- for $1 \leq i, j \leq n$, $1 \leq u \leq p$ and $1 \leq v \leq q$, $c_{i,j,u,v}^N$ (resp. $c_{i,j,u,v}^S$, $c_{i,j,u,v}^W$ and $c_{i,j,u,v}^E$) is a boolean variable equal to 1 if there is a communication for link $L_{i,j}$ between core $\mathcal{C}_{u,v}$ and its *north* (resp. *south*, *west*, *east*) neighbor $\mathcal{C}_{u-1,v}$ (resp. $\mathcal{C}_{u+1,v}$, $\mathcal{C}_{u,v-1}$, $\mathcal{C}_{u,v+1}$) and 0 otherwise; for $u = 1$ (resp. $u = p$, $v = 1$, $v = q$), we enforce that the variable is set to 0 (no possible communication because of the borders of the CMP); there are $4 \times n^2 \times p \times q$ such variables.

For convenience, we note $c_{i,j,u,v}^+ = c_{i,j,u,v}^N + c_{i,j,u,v}^S + c_{i,j,u,v}^W + c_{i,j,u,v}^E$.

4.4.3 Constraints

Finally, we must write all constraints involving our constants and variables. In the following, unless stated otherwise, i, j, i' span $\{1, \dots, n\}$ (stage indices); u, u' span $\{1, \dots, p\}$ and v, v' span $\{1, \dots, q\}$ (processor indices), and finally k, k' span $\{1, \dots, m\}$ (speed, or mode indices). First we need constraints to guarantee that the allocation of stages to cores is a valid allocation, and that the speed of each core is correctly set.

- $\forall i, k, \sum_{u,v} x_{i,k,u,v} = 1$: each stage is allocated to exactly one core;
- $\forall k, u, v, m_{k,u,v} \geq \sum_i x_{i,k,u,v}$: if stage S_i is mapped onto $\mathcal{C}_{u,v}$ operated at speed $s(k)$, then $\mathcal{C}_{u,v}$ must be operated at speed $s(k)$;
- $\forall u, v, \sum_k m_{k,u,v} \leq 1$: each core is operated at no more than one speed (either the core is on and the sum equals 1, or it is off and the sum equals 0).

Then, we need to ensure that communications are correctly scheduled, by enforcing constraints on the $c_{i,j,u,v}$ variables.

- $\forall i, j, u, v, c_{i,j,1,v}^N = 0, c_{i,j,p,v}^S = 0, c_{i,j,u,1}^W = 0,$ and $c_{i,j,u,q}^E = 0$: no communication is allowed outside the borders of the CMP;
- $\forall i, j, u, v, c_{i,j,u,v}^+ \leq \ell(i, j)$: there is no communication from S_i to S_j if there is no dependence constraint between these two stages;
- $\forall i, j, k, u, v, x_{i,k,u,v} + x_{j,k,u,v} + c_{i,j,u,v}^+ \leq 2$: this condition enforces that if S_i and S_j are mapped onto the same core, $\mathcal{C}_{u,v}$, then there is no communication for link $L_{i,j}$ initiated from $\mathcal{C}_{u,v}$;
- $\forall i, j, k, c_{i,j,u,v}^+ \geq x_{i,k,u,v} + \sum_{k',(u,v) \neq (u',v')} x_{j,k',u',v'} + \ell(i, j) - 2$: this initiates the communication for $L_{i,j}$ if S_i and S_j are mapped onto two distinct cores; the communication must occur into one of the directions (N,S,W or E);
- $\forall i, j, u < p, v, c_{i,j,u,v}^S \leq c_{i,j,u+1,v}^+ + \sum_k x_{j,k,u+1,v} \leq 2 - c_{i,j,u,v}^S$: if there was a communication initiated from $\mathcal{C}_{u,v}$ to the south for $L_{i,j}$ ($c_{i,j,u,v}^S = 1$), then either we reach the destination core ($\sum_k x_{j,k,u+1,v} = 1$), or the communication must be forwarded on one of the links from $\mathcal{C}_{u+1,v}$ ($c_{i,j,u+1,v}^+ = 1$); otherwise there is no constraint; these constraints express both the forwarding of communications and the stopping condition;
- there are similar constraints for other directions:
 - $\forall i, j, u > 1, v, c_{i,j,u,v}^N \leq c_{i,j,u-1,v}^+ + \sum_k x_{j,k,u-1,v} \leq 2 - c_{i,j,u,v}^N$;
 - $\forall i, j, u, v < q, c_{i,j,u,v}^E \leq c_{i,j,u,v+1}^+ + \sum_k x_{j,k,u,v+1} \leq 2 - c_{i,j,u,v}^E$;
 - $\forall i, j, u, v > 1, c_{i,j,u,v}^W \leq c_{i,j,u,v-1}^+ + \sum_k x_{j,k,u,v-1} \leq 2 - c_{i,j,u,v}^W$.

A set of constraints express the fact that no cycle can occur in the communications:

- $\forall i, j, p > u > 1, q > v > 1, c_{i,j,u+1,v}^N + c_{i,j,u-1,v}^S + c_{i,j,u,v-1}^E + c_{i,j,u,v+1}^W \leq \sum_k x_{i,k,u,v}$;
- $\forall i, j, q > v > 1, c_{i,j,2,v}^N + c_{i,j,1,v-1}^E + c_{i,j,1,v+1}^W \leq \sum_k x_{i,k,1,v}$;
- $\forall i, j, q > v > 1, c_{i,j,p-1,v}^S + c_{i,j,p,v-1}^E + c_{i,j,p,v+1}^W \leq \sum_k x_{i,k,p,v}$;
- $\forall i, j, p > u > 1, c_{i,j,u+1,1}^N + c_{i,j,u-1,1}^S + c_{i,j,u,2}^W \leq \sum_k x_{i,k,u,1}$;
- $\forall i, j, p > u > 1, c_{i,j,u+1,q}^N + c_{i,j,u-1,q}^S + c_{i,j,u,q-1}^E \leq \sum_k x_{i,k,u,q}$;
- $\forall i, j, c_{i,j,2,1}^N + c_{i,j,1,2}^W \leq \sum_k x_{i,k,1,1}$;
- $\forall i, j, c_{i,j,p-1,1}^S + c_{i,j,p,2}^W \leq \sum_k x_{i,k,p,1}$;
- $\forall i, j, c_{i,j,2,q}^N + c_{i,j,1,q-1}^E \leq \sum_k x_{i,k,1,q}$;
- $\forall i, j, c_{i,j,p-1,q}^S + c_{i,j,p,q-1}^E \leq \sum_k x_{i,k,p,q}$.

Another constraint expresses the fact that the mapping is a DAG-partition:

- $\forall i, i', j, k, u, v, x_{i',k,u,v} \geq \ell_{i,i'}^* \times \ell_{i',j}^* \times (x_{i,k,u,v} + x_{j,k,u,v} - 1)$: if two stages S_i and S_j are mapped onto the same core $\mathcal{C}_{u,v}$, then any stage $S_{i'}$ which has an incoming dependency from S_i and an outgoing dependency from S_j must be mapped onto the same core, otherwise there would be a cycle in the partition.

Finally, we express the fact that the constraint on the period is fulfilled:

- $\forall u, v, k, \sum_i x_{i,k,u,v} \times w(i) \leq T \times m_{k,u,v} \times s(k)$: constraint on computations;
- $\forall u, v \sum_{i,j} c_{i,j,u,v}^N \times \delta(i, j) \leq T \times BW$: constraint on north communications;
- $\forall u, v \sum_{i,j} c_{i,j,u,v}^S \times \delta(i, j) \leq T \times BW$: constraint on south communications;
- $\forall u, v \sum_{i,j} c_{i,j,u,v}^W \times \delta(i, j) \leq T \times BW$: constraint on west communications;
- $\forall u, v \sum_{i,j} c_{i,j,u,v}^E \times \delta(i, j) \leq T \times BW$: constraint on east communications.

4.4.4 Objective function

We aim at minimizing the energy consumption, which writes:

$$\min \left(\begin{array}{l} \sum_{u,v} \left(\sum_k m_{k,u,v} \times Estat \right. \\ \left. + \sum_{i,k} x_{i,k,u,v} \times w(i) \times Edyn(k) \right) \\ \left. + \sum_{u,v,i,j} c_{i,j,u,v}^+ \times \delta(i, j) \times E^{(bit)} \right) . \end{array} \right)$$

The objective function is linear, as well as all the constraints. Since the variables are boolean, this is an integer linear program.

5 Heuristics

In this section, we describe the five heuristics that we have designed and implemented, thus providing practical solutions to the $\text{MINENERGY}(T)$ problem. The first heuristic, **Random** (Section 5.1), performs a random mapping, and it is used for comparison purposes. Then we propose a greedy heuristic, **Greedy**, in Section 5.2, a heuristic based on a two-dimensional dynamic programming algorithm, **DPA2D**, in Section 5.3. Finally, we design two one-dimensional heuristics in Section 5.4: **DPA1D** builds upon the theoretical results of Section 4.1 and computes the optimal one-dimensional solution, while **DPA2D1D** computes the solution with the **DPA2D** heuristic, used in a one-dimensional setting.

5.1 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 a *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.

5.2 Greedy heuristic

Given a speed $s \in \mathcal{S}$, 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.

5.3 2D dynamic programming algorithm

This heuristic, called **DPA2D**, starts by mapping the initial SPG onto a $x_{\max} \times y_{\max}$ grid, following the labels of the nodes (see Section 3.1). 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:

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

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}^{col} (and therefore, distribution D) are computed thanks to another dynamic programming algorithm: we compute $\mathcal{E}_{(m_1, m_2, D', D)}^{\text{col}}(g, u)$, 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 $\mathcal{E}^{\text{col}}(m_1, m_2, D', D) = \mathcal{E}_{(m_1, m_2, D', D)}^{\text{col}}(y_{\max}, p)$.

For the distribution within a column, the recurrence writes:

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

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

$\mathcal{E}_{(m_1, m_2, D')}^{\text{ver}}(g' + 1, g, u - 1)$ 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, $\mathcal{E}_{(m_1, m_2, D)}^{\text{cal}}(g' + 1, g)$ 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}^{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.

5.4 1D heuristics

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

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

The **DPA1D** heuristic builds upon the theoretical results of Section 4, 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. It is also optimal for any SPG without communication. However, **DPA1D** may take 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 5.3) 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.

6 Simulation results

This section reports simulation results assessing the performance of the various heuristics. As for the applications, we use both real-life applications taken from the *StreamIt* suite [45], and randomly generated applications, which allows us to cover a broader spectrum. As for the target platform, we use 4×4 and 6×6 CMP grids, whose hardware characteristics are representative of state-of-the-art devices. The source code for all simulations is publicly available at [42].

6.1 Simulation setting

6.1.1 Streaming applications

StreamIt suite. There are 12 workflows in the *StreamIt* suite [45]. Their main characteristics are summarized in Table 1, where we give the size n , the maximum label values y_{\max} and x_{\max} , and 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 observe in Table 1 that all workflows have a large CCR, hence are compute-intensive rather than data-intensive. In the simulations, we first use the workflows as such, with the original CCR values, and then we scale communication weights (the $\delta_{i,j}$) to change each CCR successively to 10, 1, and 0.1, so as to assess the impact of the communications on the performance of the heuristics.

Index	Name	n	y_{\max}	x_{\max}	CCR
1	Beamformer	57	12	12	537
2	ChannelVocoder	55	17	8	453
3	Filterbank	85	16	14	535
4	FMRadio	43	12	12	330
5	Vocoder	114	17	32	38
6	BitonicSort	40	4	23	6
7	DCT	8	1	8	68
8	DES	53	3	45	7
9	FFT	17	1	17	17
10	MPEG2-noparser	23	5	18	9
11	Serpent	120	2	111	9
12	TDE	29	1	29	12

Table 1: Characteristics of the *StreamIt* workflows.

Randomly generated. We randomly build SPG applications (by applying recursively series and parallel compositions of SPG applications), and we extract their size n , their elevation y_{\max} , together with their computation-to-communication ratio (CCR).

6.1.2 CMP configuration

For processor speeds and power consumption, we use the model of the Intel Xscale [23], following [13, 11, 36]. There are five speeds for each core:

$$s_{u,v} = (0.15, 0.4, 0.6, 0.8, 1) \text{ GHz},$$

with power consumption $P_{s_{u,v}}^{(\text{comp})} = (80, 170, 400, 900, 1600) \text{ mW}$. We assume that the power consumption of the processor when it is idle is $P_{\text{leak}}^{(\text{comp})} = 80 \text{ mW}$. We use 16-byte wide communication links [39], whose bandwidths are $BW = 16 \times 1.2 \text{ Gbytes}$, which is reasonable according to [39]. 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 [9]; we fix $E^{(\text{bit})} = 6 \text{ pJ}$. 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).

6.1.3 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 $T = 1 \text{ s}$. 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 upon the workflow, and that 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 $T/10$.

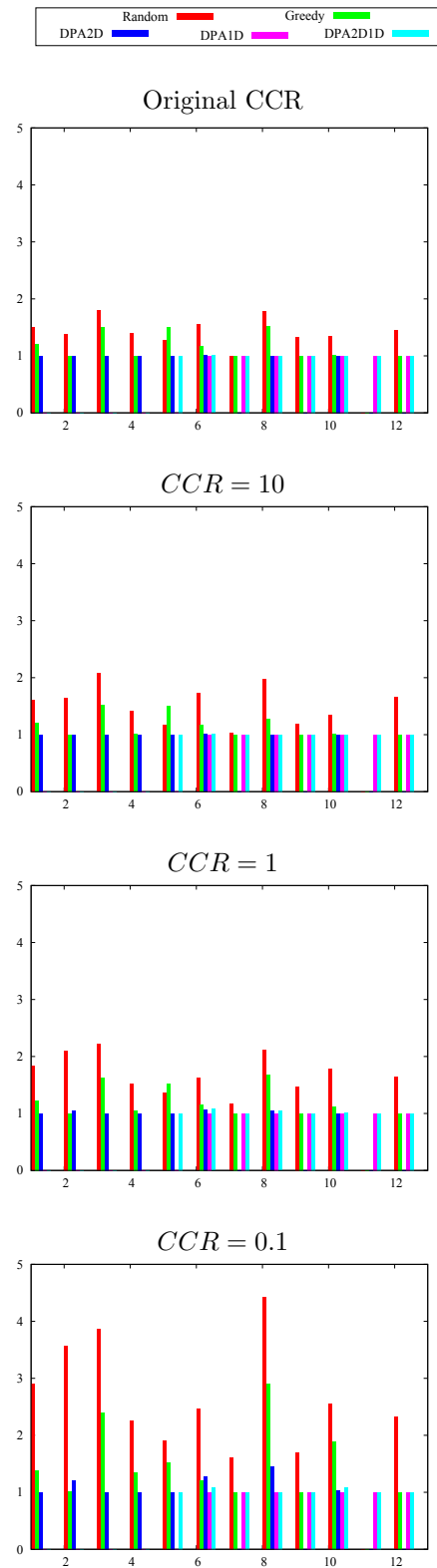
6.2 Simulation results

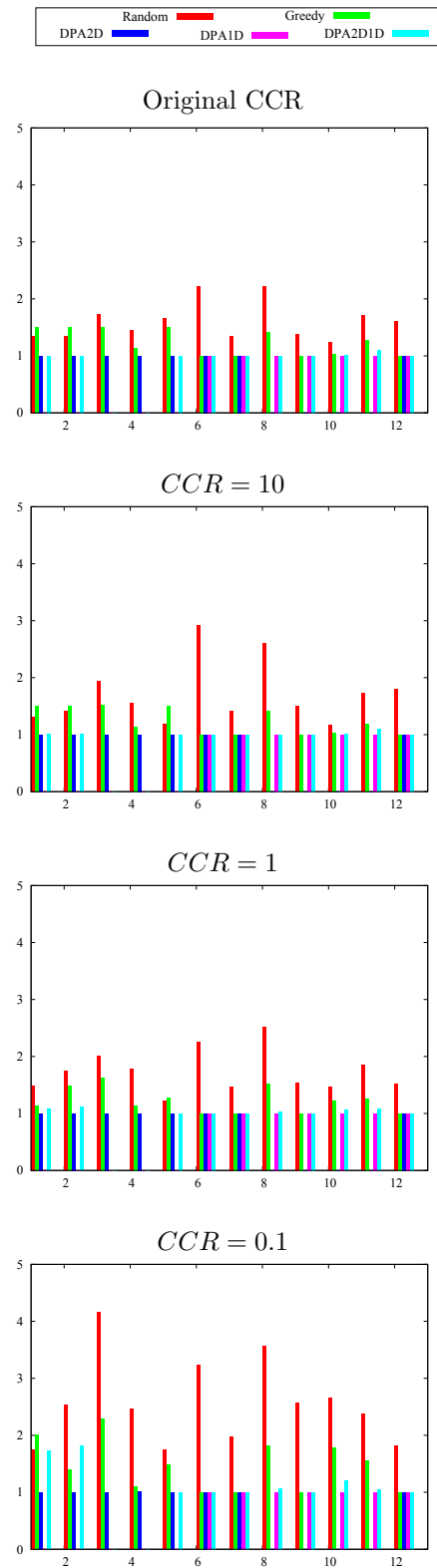
6.2.1 *StreamIt* suite

In Figures 8 and 9, we plot the energy computed by the four heuristics for each application, given a CMP size (4×4 or 6×6) and a CCR ratio (set to the original value, 10, 1 and 0.1). On the horizontal x axis, each group corresponds to an application, and x is the number of the application in Table 1. On the vertical axis, we plot the energy found by each heuristic, normalized by the minimum value obtained over all heuristics (so that the best heuristic returns 1, and the other ones return higher values). The **DPA1D** heuristic fails to return a solution for the first four applications, because there are too many possible splits to explore, and it is not plotted for those applications. More generally, each time a heuristic fails on a given application, it does not appear on the corresponding graph.

4×4 CMP grid. Results for a 4×4 CMP grid are given in Figure 8. When computations are predominant, i.e., when the CCR is set to its original value, or uniformly equal to 10, we observe that **Greedy**, **DPA2D**, **DPA1D** and **DPA2D1D** return similar results, and that **Random** always is within a factor of two. We also observe that **DPA2D** often fails on graphs with small elevation (linear graphs), because it wastes a lot of cores. For instance, if the application is exactly a pipeline (workflows numbered 7, 9 and 12), **DPA2D** can only enroll 4 cores over the 16 that are available. This fact holds true irrespective of the CCR.

When communications are more important, i.e., when the CCR is uniformly set to 1 or 0.1, **Random** gets much worse than the other heuristics: if it does

Figure 8: Normalized energy on the set of applications for a 4×4 CMP grid.

Figure 9: Normalized energy on the set of applications for a 6×6 CMP grid.

Platform size	Random	Greedy	DPA2D	DPA1D	DPA2D1D
4 × 4	5	4	16	20	16
6 × 6	0	0	17	20	8

Table 2: Number of failures for each heuristic (out of 48 instances per CMP grid size).

not fail, its energy is between 2 and 4 times worse than the best one. In a general manner, we see that **DPA2D** is the best heuristic when the application graph has a high elevation.

We point out that **DPA1D** and **DPA2D1D** are the only successful heuristics for the workflow 11, whatever the CCR ratio is. This workflow fits very well with the main design idea of **DPA1D** and **DPA2D1D**: it is a pipeline-like graph (its elevation is only 2) with numerous stages. The other heuristics fail to find a good load-balance of computations and communications for this application.

The difference between **DPA1D** and **DPA2D1D** is tiny: when **DPA1D** finds a solution, **DPA2D1D** finds a close one, and there is only one graph (numbered 5) on which **DPA2D1D** succeeds, whereas **DPA1D** fails, because of the high memory complexity. Note that, in some cases, the solution of **DPA1D** is better than that of **DPA2D1D**, confirming that **DPA2D1D** does not return the optimal 1D mapping.

Altogether, **Greedy** seems to be a general-purpose heuristic that succeeds on most graphs, and it is always superior to **Random**. On the contrary, **DPA1D**, **DPA2D1D** and **DPA2D** are “specialized” heuristics, the first two heuristics are very efficient for long and almost linear graphs but not good for fat graphs of large elevation, and the last one behaving just as the opposite.

6×6 CMP grid. Results for a 6×6 CMP grid are given in Figure 9. Because the target grid is larger, it is easier to find a mapping that matches the period bound, especially for applications with a small number of stages. This is quantified in Table 2, where we report the number of failures for each heuristic.

We observe that the difference between solutions of **DPA2D1D** and solutions of **DPA1D** almost disappears. Otherwise, the conclusion remains more or less the same as on the 4×4 CMP grid, with **Greedy** always successful but also always inferior to one of the three specialized heuristics, **DPA1D**, **DPA2D1D** and **DPA2D**, depending upon the graph shape.

6.2.2 Random SPGs

For the randomly generated SPGs, we plot four sets of three graphs; in each set, the three graphs are obtained for a given CCR (10, 1 or 0.1), whereas each set corresponds to a value of the couple (n, p) , where the number of nodes n can be 50 or 150 and the number of cores p in a row of the square CMP can be 4 or 6.

On the horizontal axis, we represent the elevation of the SPG. For each value of the elevation, we average the results obtained on 100 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).

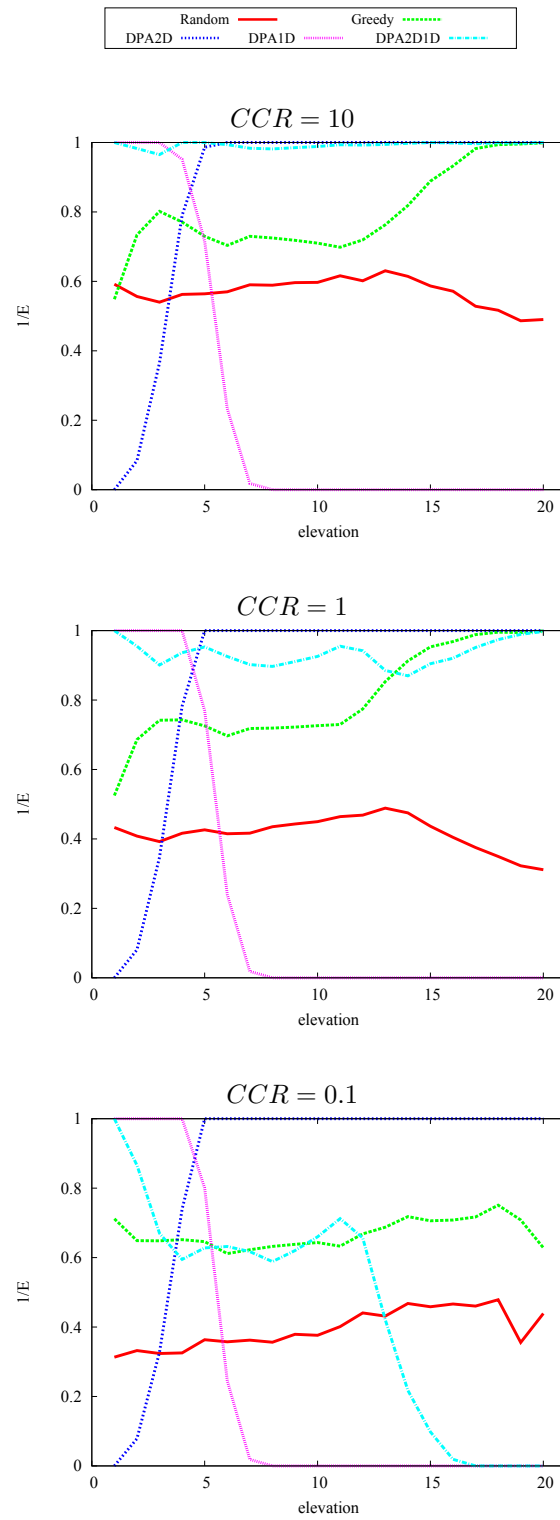


Figure 10: Normalized energy inverse on a random set of applications of 50 nodes for a 4×4 CMP grid.

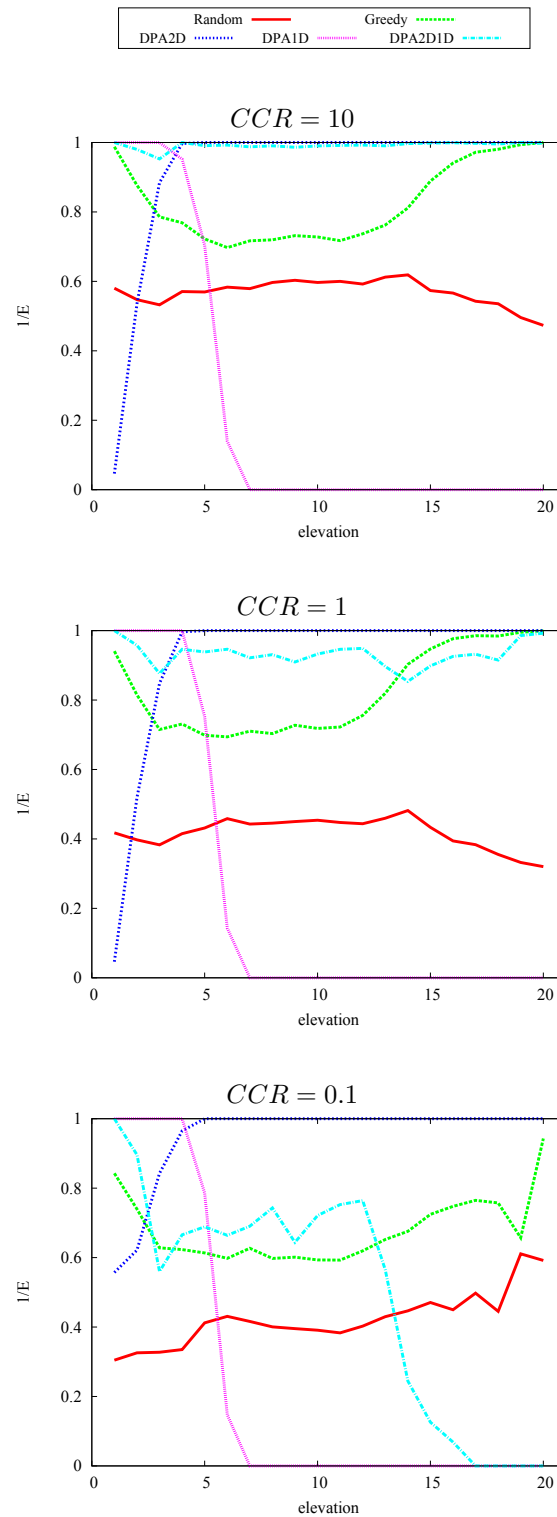


Figure 11: Normalized energy inverse on a random set of applications of 50 nodes for a 6×6 CMP grid.

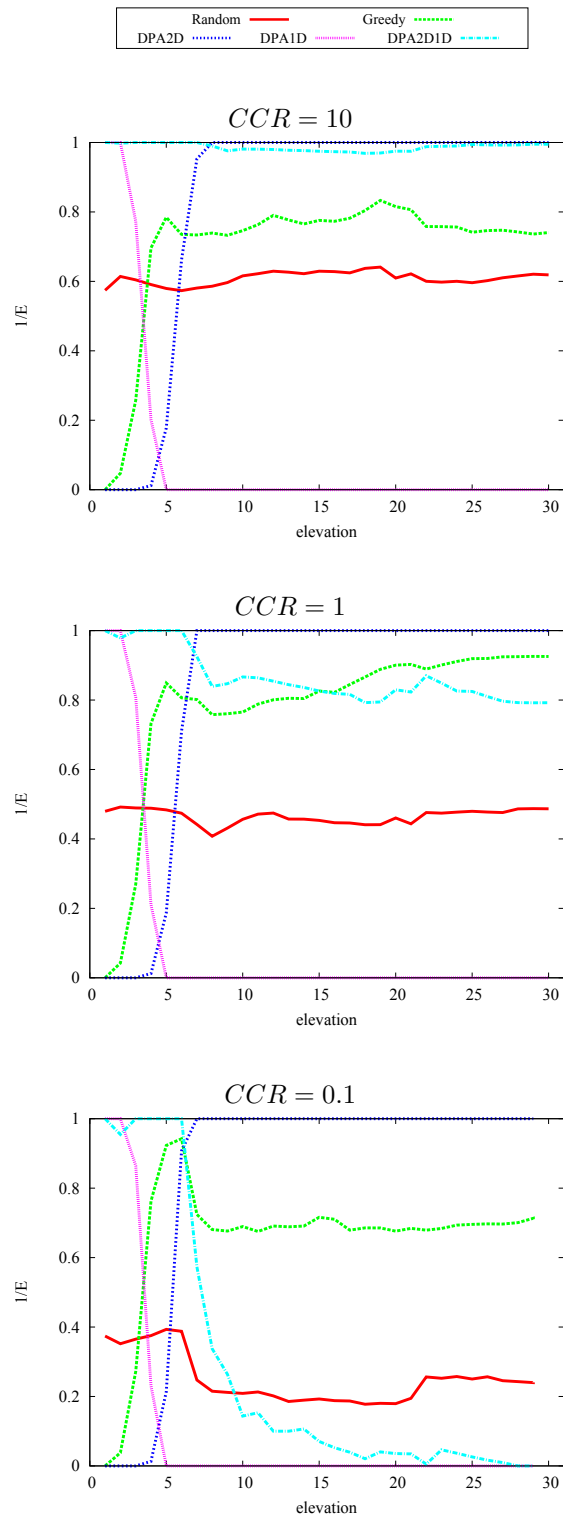


Figure 12: Normalized energy inverse on a random set of applications of 150 nodes for a 4×4 CMP grid.

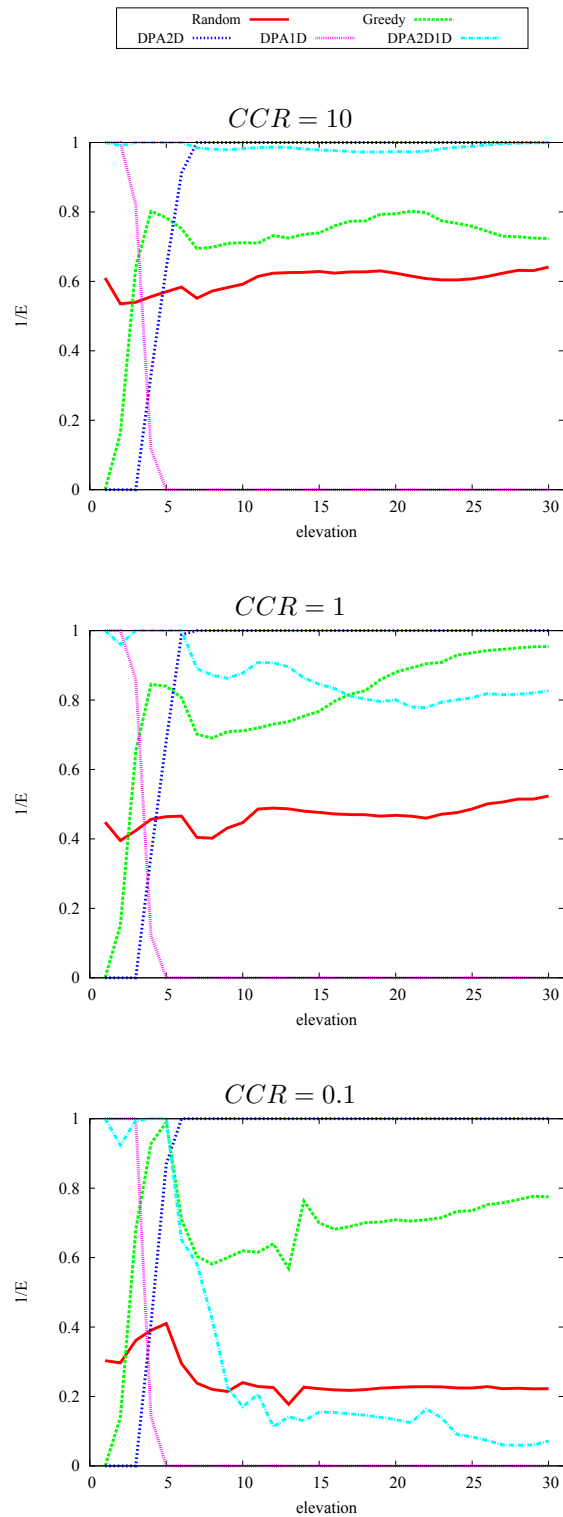


Figure 13: Normalized energy inverse on a random set of applications of 150 nodes for a 6×6 CMP grid.

With 50 nodes and a 4×4 CMP grid. Results are given in Figure 10. 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.

Number of failures. In Table 3, we report the number of failures for each heuristic, again with 50 nodes and a 4×4 CMP grid. With a large CCR (10 or 1), **DPA2D1D** almost always succeeds to find a solution, which are in turn pretty good (see Figure 10). **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.

Other results. We have performed further simulations on larger applications and/or different CMP grid sizes, see Figures 11, 12, and 13. Overall, the conclusions remain the same, and they confirm the results derived from the real-life *StreamIt* applications.

CCR	Random	Greedy	DPA2D	DPA1D	DPA2D1D
10	58	56	156	1516	2
1	58	56	156	1520	4
0.1	300	287	348	1340	916

Table 3: Number of failures (out of 2000 instances per CCR value).

7 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 and 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 and frequency 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 uniline 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 with the *StreamIt* suite [45] and the randomly generated SPGs confirmed the efficiency of the main design principles underlying the various heuristics. While **Greedy** is the most robust approach, it is always superseded by at least one of the three specialized algorithms, **DPA1D** for long pipeline-like graphs, **DPA2D** for fat graphs of large elevation or **DPA2D1D** for any graph containing low communication weights and for graphs of low 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.

References

- [1] 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.*, pages 760–765, June 2008.
- [2] AMD. ACP - The Truth About Power Consumption Starts Here. http://www.amd.com/us/Documents/43761C_ACP_WP_EE.pdf, 2010.
- [3] 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*, pages 182–187, Sept. 2004.
- [4] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1):1–39, 2007.
- [5] 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.
- [6] A. Benoit and Y. Robert. Mapping pipeline skeletons onto heterogeneous platforms. *Journal of Parallel and Distributed Computing (JPDC)*, 68(6):790–808, 2008.
- [7] P. Blaha, K. Schwarz, G. Madsen, D. Kvasnicka, and J. Luitz. WIEN2k: An Augmented Plane Wave Plus Local Orbitals Program for Calculating Crystal Properties - User's guide, 2001. Vienna Univ. of Technology, Austria.
- [8] G. Blake, R. Dreslinski, and T. Mudge. A survey of multicore processors. *Signal Processing Magazine*, 26(6):26–37, Nov. 2009.
- [9] G. Chen, F. Li, M. Kandemir, and M. J. Irwin. Reducing NoC energy consumption through compiler-directed channel voltage scaling. *SIGPLAN Not.*, 41:193–203, 2006.
- [10] G. Chen, F. Li, S. Son, and M. Kandemir. Application mapping for chip multiprocessors. In *Proc. of DAC'08, the 45th Design Automation Conf.*, pages 620–625, June 2008.
- [11] J.-J. Chen. Expected energy consumption minimization in DVS systems with discrete frequencies. In *Proc. of SAC'08, Symp. on Applied Computing*, pages 1720–1725, 2008.
- [12] 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*, pages 28–38, 2007.
- [13] J.-J. Chen and T.-W. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *Proc. of ICCAD'07, the Int. Conf. on Computer-aided design*, pages 289–294, 2007.

- [14] 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*, pages 357–368, June 2005.
- [15] Cplex. ILOG CPLEX: High-performance software for mathematical programming and optimization. <http://www.ilog.com/products/cplex/>.
- [16] DataCutter. DataCutter Project: Middleware for Filtering Large Archival Scientific Datasets in a Grid Environment. <http://www.cs.umd.edu/projects/hpsl/ResearchAreas/DataCutter.htm>.
- [17] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.
- [18] R. Ge, X. Feng, and K. W. Cameron. Performance- constrained distributed DVS scheduling for scientific applications on power-aware clusters. In *Proc. of the Conf. on SuperComputing (SC)*, page 34. IEEE CS, 2005.
- [19] P. Grosse, Y. Durand, and P. Feautrier. Methods for power optimization in SOC-based data flow systems. *ACM Trans. Des. Autom. Electron. Syst.*, 14:38:1–20, June 2009.
- [20] 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.
- [21] 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*, pages 355–372, 2009.
- [22] M. Hammoud, S. Cho, and R. Melhem. A dynamic pressure-aware associative placement strategy for large scale chip multiprocessors. *Computer Architecture Letters*, 9(1):29–32, Jan. 2010.
- [23] Intel XScale technology. <http://www.intel.com/design/intelxscale>.
- [24] 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*, 18(8):1028–1040, 2007.
- [25] W. Jang and D. Pan. A3MAP: Architecture-Aware Analytic Mapping for Networks-on-Chip. In *Proc. of DAC'10, the 15th Asia and South Pacific Design Automation Conference*, pages 523–528, Jan. 2010.
- [26] 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*, pages 275–280, 2004.
- [27] 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.*, 36:211–222, Oct. 2002.

- [28] K. H. Kim, R. Buyya, and J. Kim. Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters. In *Proc. of CCGRID 2007, the 7th IEEE Int. Symp. on Cluster Computing and the Grid*, pages 541–548, May 2007.
- [29] K. Lahiri, A. Raghunathan, S. Dey, and D. Panigrahi. Battery-driven system design: a new frontier in low power design. In *Proc. of DAC'02, the 7th Design Automation Conference and the 15th Int. Conf. on VLSI Design*, pages 261–267, 2002.
- [30] P. Langen and B. Juurlink. Leakage-aware multiprocessor scheduling. *J. Signal Process. Syst.*, 57(1):73–88, 2009.
- [31] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. In *Proc. of DAC'00, the 37th Design Automation Conference*, pages 806–809, 2000.
- [32] 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*, pages 187–192, Dec. 2008.
- [33] R. McClatchey, F. Estrella, J.-M. Le Goff, Z. Kovacs, and N. Baker. Object databases in a distributed scientific workflow application. In *Proc. of BI-WIT'97, the 3rd Basque Int. Workshop on Information Technology*, pages 11–21, July 1997.
- [34] M. P. Mills. The internet begins with coal. *Environment and Climate News*, 1999.
- [35] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar. Power and Thermal Management in the Intel Core™ Duo Processor. *Intel Technology Journal*, 10(2):109–122, May 2006.
- [36] 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.*, pages 163–172, Aug. 2010.
- [37] T. Okuma, H. Yasuura, and T. Ishihara. Software energy reduction techniques for variable-voltage processors. *Design Test of Computers, IEEE*, 18(2):31–41, Mar. 2001.
- [38] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The case for a single-chip multiprocessor. *SIGPLAN Not.*, 31:2–11, Sept. 1996.
- [39] 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*, 27:96–108, 2007.
- [40] R. B. Prathipati. Energy efficient scheduling techniques for real-time embedded systems. Master's thesis, Texas A&M University, May 2004.
- [41] J. Qin and T. Fahringer. Advanced data flow support for scientific grid workflow applications. In *Proc. of SC'07, the Conf. on Supercomputing*, pages 1–12, Nov. 2007.

- [42] P. Renaud-Goud. Source Code for the Experiments. <http://graal.ens-lyon.fr/~prenaud/sp-cmp/>.
- [43] F. Schueller, J. Qin, F. Nadeem, R. Prodan, T. Fahringer, and G. Mayr. Performance, Scalability and Quality of the Meteorological Grid Workflow MeteoAG. In *Proc. of 2nd Austrian Grid Symp.*, Univ. Innsbruck, Sept. 2006.
- [44] L. Silva, G. Granato, A. Bressan, C. Lacey, C. Baugh, S. Cole, and C. Frenk. Modelling dust in galactic sed: Application to semi-analytical galaxy formation models. *Astrophysics and Space Science*, 276:1073–1078, 2001.
- [45] StreamIt Project. <http://groups.csail.mit.edu/cag/streamit/apps/stream-graphs>.
- [46] 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.
- [47] J. Subhlok and G. Vondran. Optimal latency-throughput tradeoffs for data parallel pipelines. In *Proc. of SPAA, the Symp. on Parallelism in Algorithms and Archi.*, 1996.
- [48] 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*, pages 368–377, May 2010.
- [49] 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.*, pages 25–38, 2007.
- [50] R. Xu, D. Mossé, and R. Melhem. Minimizing expected energy consumption in real-time systems through dynamic voltage scaling. *ACM Trans. Comput. Syst.*, 25(4):9, 2007.
- [51] L. Yang and L. Man. On-Line and Off-Line DVS for Fixed Priority with Preemption Threshold Scheduling. In *Proc. of ICES'09, the Int. Conf. on Embedded Software and Systems*, pages 273–280, May 2009.
- [52] Y. Zhao, M. Wilde, I. Foster, J. Voekler, T. Jordan, E. Quigg, and J. Dobson. Grid middleware services for virtual data discovery, composition, and integration. In *Proc. of MGC'04, the 2nd workshop on Middleware for Grid Computing*, pages 57–62. ACM, 2004.



Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
<http://www.inria.fr>
ISSN 0249-6399