Mapping and Load-Balancing Iterative Computations

Arnaud Legrand, Hélène Renard, Yves Robert, *Senior Member*, *IEEE*, and Frédéric Vivien

Abstract—This paper is devoted to mapping iterative algorithms onto heterogeneous clusters. The application data is partitioned over the processors, which are arranged along a virtual ring. At each iteration, independent calculations are carried out in parallel, and some communications take place between consecutive processors in the ring. The question is to determine how to slice the application data into chunks, and to assign these chunks to the processors, so that the total execution time is minimized. One major difficulty is to embed a processor ring into a network that typically is not fully connected, so that some communication links have to be shared by several processor pairs. We establish a complexity result that assesses the difficulty of this problem, and we design a practical heuristic that provides efficient mapping, routing, link-sharing, and data distribution schemes.

Index Terms—Scheduling, load-balancing, iterative computations, heterogeneous clusters.

1 Introduction

In this paper, we investigate the mapping of iterative algorithms onto heterogeneous clusters. Such algorithms typically operate on a large collection of application data, which will be partitioned over the processors. At each iteration, some independent calculations will be carried out in parallel, and then some communications will take place. This scheme is very general and encompasses a broad spectrum of scientific computations, from mesh-based solvers (e.g., elliptic PDE solvers) to signal processing (e.g., recursive convolution) and image processing algorithms (e.g., mask-based algorithms such as thinning).

An abstract view of the problem is the following: The iterative algorithm repeatedly operates on a large rectangular matrix of data samples. This data matrix is split into vertical slices that are allocated to the computing resources (processors). At each step of the algorithm, the slices are updated locally, and then boundary information is exchanged between consecutive slices. This (virtual) geometrical constraint advocates that processors be organized as a virtual ring. Then, each processor will only communicate twice, once with its (virtual) predecessor in the ring and once with its successor. Note that there is no reason a priori to restrict to a unidimensional partitioning of the data and to map it onto a unidimensional ring of processors: More general data partitionings, such as two-dimensional, recursive or even arbitrary slicings into rectangles, could be considered. But, unidimensional partitionings are very natural for most applications and, as will be shown in this paper, the problem to find the optimal one is already very difficult.

The target architecture is a fully heterogeneous platform composed of different-speed processors that communicate through links of different bandwidths. On the architecture side, the problem is twofold: 1) select the processors that will participate in the solution and decide for their ordering that will represent the arrangement into a ring and 2) assign communication routes from each participating processor to its successor in the ring. One major difficulty of this ring embedding process is that some of the communication routes will (most probably) have to share some physical communication links: Indeed, the communication networks of heterogeneous platforms typically are sparse, i.e., far from being fully connected. If two or more routes share the same physical link, we have to decide which portion of the link bandwidth is to be assigned to each route.

Once the ring and the routing have been decided, there remains to determine the best partitioning of the application data. Clearly, the quality of the final solution depends on many application and architecture parameters, and we should expect the optimization problem to be very difficult to solve. To assess the impact of sharing the link bandwidths, we first deal with the simplified version of the problem where we view the target interconnection network as fully connected: Between any node pair, the routing is fixed (shortest paths in terms of bandwidth) and the bandwidth is assumed to be that of the slowest link in the routing path. This simplified model is not very realistic because no link contention is taken into account, but it will lead to a solution ring that can be compared to that obtained with link sharing, thereby providing a convenient way to evaluate the significance of the different hypotheses on the communications.

The rest of the paper is organized as follows: In Section 2, we formally state the previous optimization problems, which we denote as SLICERING for the simplified version without link sharing, and as SHAREDRING for the more general version with link sharing. Section 3 is devoted to the complexity of the SLICERING problem. After formally stating the optimization problem, we show that the associated decision problem is NP-complete. Section 4 deals with the more realistic SHAREDRING problem and contains the major contributions of this paper. We state the problem (Section 4.1), we work out a small-size example (Section 4.2), and we prove a complexity result (Section 4.3). Then, in Section 4.4, we proceed to the design of polynomial-time heuristics to solve the SHAREDRING optimization problem. We report some experimental results

• *The authors are with LIP, UMR CNRS-INRIA-UCBL 5668, École Normale Supérieure de Lyon, France.*
E-mail: {Arnaud.Legrand, Helene.Renard, Yves.Robert, Frederic.Vivien}@ens-lyon.fr.

Manuscript received 20 Aug. 2003; revised 2 Feb. 2004; accepted 4 Feb. 2004. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0141-0803.

in Section 5. We start by experimenting with the SHAREDRING heuristics using a network generator. Then, we move to a comparison of the two approaches, with and without link sharing, using two heterogeneous networks of workstations. We survey related work in Section 6, including dynamic strategies that are not considered in this paper (our scheduling approach is fully static). Finally, we state some concluding remarks in Section 7.

2 Framework

In this section, we start with the framework that is common to the two optimization problems SLICERING and SHAREDRING. Next, we state specific assumptions for each of them in Section 2.2 for SLICERING, and in Section 2.3 for SHAREDRING, and we discuss some variants (Section 2.4).

2.1 Common Assumptions

Computing costs. The target computing platform is modeled as a directed graph $G = (P, E)$. Each node P_i in the graph, $1 \leq i \leq |P| = p$, models a computing resource, and is weighted by its relative cycle-time w_i: P_i requires w_i time-steps to process a unit-size task. Of course, the absolute value of the time-unit is application-dependent; what matters is the relative speed of one processor versus the other.

Communication costs. Graph edges represent communication links and are labeled with available bandwidths. If there is an oriented link $e \in E$ from P_i to P_j, we let b_e denote the bandwidth of the link. It will take D_c/b_e time-units to transfer a single message of size D_c from P_i to P_j using link e.

Application parameters: computations. Let D_w be the total size of the work to be performed at each step of the algorithm. Processor P_i will accomplish a share $\alpha_i.D_w$ of this total work, where $\alpha_i \geq 0$ for $1 \leq i \leq p$ and $\sum_{i=1}^{p} \alpha_i = 1$. Note that we allow $\alpha_j = 0$ for some index j, meaning that processor P_j does not participate in the computation. Indeed, there is no reason a priori for all resources to be involved, especially when the total work is not very large: The extra communications incurred by adding more processors may slow down the whole process, despite the increased cumulated speed.

Application parameters: communications in the ring. We arrange the participating processors along a ring (yet to be determined). After updating its data slice of size $\alpha_i.D_w$, each active processor P_i sends a message of fixed length D_c (typically some boundary data) to its successor. To illustrate the relationship between D_w and D_c, we can view the original data matrix as a large rectangle composed of D_w columns of height D_c, so that one single column is exchanged between any pair of consecutive processors in the ring. Let $\mathrm{succ}(i)$ and $\mathrm{pred}(i)$ denote the successor and the predecessor of P_i in the virtual ring. The time needed to transfer a message of size D_c from P_i to P_j is $D_c.c_{i,j}$, where the communication delay $c_{i,j}$ will be instantiated below, according to whether links are assumed to be shared or not.

Objective function. The total cost of a single step in the iterative algorithm is the maximum, over all participating processors, of the time spent computing and communicating:

$$T_{\text{step}} = \max_{1 \leq i \leq p} \mathbb{I}\{i\}[\alpha_i.D_w.w_i + D_c.(c_{i,\text{pred}(i)} + c_{i,\text{succ}(i)})], \quad (1)$$

where $\mathbb{I}\{i\}[x] = x$ if P_i is involved in the computation, and 0 otherwise. In summary, the goal is to determine the best way to select q processors out of the p available, to assign them computational workloads, to arrange them along a ring, and to share the network bandwidth so that the total execution time per step is minimized.

2.2 Specific Assumptions for SLICERING

As already mentioned, the target computing platform is modeled as a complete graph for the simplified optimization problem SLICERING. See Section 5.2.1 for an example where such a complete graph is built out of a real platform using shortest-paths routing.

Communication costs. The time needed to transfer a message of size D_c from P_i to P_j is $D_c.c_{i,j}$, where $c_{i,j}$ is the capacity of the link e from P_i to P_j, i.e., the inverse of its bandwidth b_e.

Application parameters: communications in the ring. Processor P_i requires $D_c.c_{i,\text{succ}(i)}$ time-units to send a message of size D_c to its successor, plus $D_c.c_{i,\text{pred}(i)}$ to send a message of same size to its predecessor.

2.3 Specific Assumptions for SHAREDRING

Communications and routing are much more complicated to deal with in the general optimization problem SHAREDRING.

Communication costs. It takes D_c/b_e time-units to transfer a single message of size D_c from P_i to P_j using link e. When there are several messages sharing the link, each of them receives a portion (to be determined later) of the available bandwidth. For instance, if there are two messages sharing link e and if the first message is allocated two-thirds of the bandwidth, i.e., $2b_e/3$, then the second message cannot use more than $b_e/3$. The portions of the bandwidth that are allocated to the messages can be freely determined by the user; the only rule is that the sum of all these portions cannot exceed the total link bandwidth. In practice, such a freedom for the routing strategy will only be available with future-generation networks like IPv6, with a suitable QoS policy framework [33]. Note, however, that the eXplicit Control Protocol XCP [26] is already enabled to implement a bandwidth allocation strategy that complies with our hypotheses.

Routing. We assume that we can freely decide how to route messages from one processor to another. Assume that we want to route a message of size D_c from P_i to P_j, along a path composed of k edges $e_1, e_2, \ldots, e_k$. Along each edge e_m, the message will be allocated a portion f_m of the bandwidth b_{e_m}. The overall speed of the communication along the path is bounded by the link where the smallest amount of bandwidth is available for the message: We need D_c/b time-units to route the message, where $b = \min_{1 \leq m \leq k} f_m$: This is as if we had a direct link dedicated to the message, but of reduced bandwidth b. The constraint on total link bandwidths still holds: If several messages simultaneously circulate on the network and happen to share links, the total bandwidth capacity of each link cannot be exceeded.

Application parameters: communications in the ring. There is a communication path $\mathcal{S}_i$ ($\mathcal{S}$ stands for "successor") from P_i to $P_{\text{succ}(i)}$ in the network: Let $s_{i,m}$ be the portion of the bandwidth b_{e_m} of the physical link e_m that has been allocated to the path $\mathcal{S}_i$. Of course, if a link e_r is not used in the path, then $s_{i,r} = 0$. Let $c_{i,\text{succ}(i)} = \frac{1}{\min_{e_m \in \mathcal{S}_i} s_{i,m}}$: Then, P_i requires $D_c.c_{i,\text{succ}(i)}$ time-units to send its message of size D_c to its successor $P_{\text{succ}(i)}$.

Similarly, we define the communication path $\mathcal{P}_i$ ($\mathcal{P}$ for "predecessor") from P_i to $P_{\text{pred}(i)}$ in the network; $p_{i,m}$ is the portion of the bandwidth b_{e_m} of the physical link e_m that has been allocated to the path $\mathcal{P}_i$, and $c_{i,\text{pred}(i)} = \frac{1}{\min_{e_m \in \mathcal{P}_i} p_{i,m}}$.

Then, P_i requires $D_c.c_{i,\mathrm{pred}(i)}$ time-units to send its message of size D_c to its predecessor $P_{\mathrm{pred}(i)}$.

2.4 Variants

We discuss here several variants of the previous application/architecture framework:

Start-up overheads. The motivation to use a simple linear-cost model, rather than an affine-cost model involving start-ups, both for the communications and the computations, is the following: Only large-scale applications are likely to be deployed on heterogeneous platforms. Each step of the algorithm will be both computation and communication-intensive, so that start-up overheads can indeed be neglected. Anyway, most of the results presented here extend to an affine cost modeling.

Bidirectional links. It is easy to model a bidirectional link between a given processor pair P_i and P_j: Regardless of their orientation (from P_i to P_j or the other way), all the communications using that link will be allocated a portion of the bandwidth, so that the total available of the link bandwidth is not exceeded. In other words, we assign a bandwidth portion f_{path} to each communication path requesting a bidirectional link of bandwidth b, regardless of the orientation of the path, and we state the constraint $\sum f_{\mathrm{path}} \leq b$; the sum extends to all paths using the link, regardless of their orientation. In fact, unidirectional links and bidirectional links can simultaneously exist in the network, and it is easy to model both.

Multiple links. Similarly, multiple links between a given processor pair can easily be taken into account: We would simply model G as a multigraph rather than as a simple graph.

Backbone links. Backbone links can accommodate several communications at the same bandwidth rate b: To model such a link, we assign the same portion $f_{\mathrm{path}} = b$ to each communication path requesting the link, regardless of their number: In other words, we replace the constraint $\sum f_{\mathrm{path}} \leq b$ by $f_{\mathrm{path}} \leq b$ for each path using the link.

To conclude this section, we point out that this framework is not restricted to iterative algorithms. In fact, our approach applies to problems where independent computations are distributed over heterogeneous resources arranged along a ring, and are interleaved with communications from adjacent processors. The major hypothesis is that communications only occur between adjacent processors, and that their volume is independent of their relative workload.

There are other architectural models that would be worth investigating. We use a model where each processor sequentially sends messages to its two neighbors, and we implicitly assume asynchronous receptions. In particular, we could assume that each processor is able to send both messages in parallel, which would lead to a *Max* operator instead of a sum in the communication overhead of (1).

3 The SliceRing Optimization Problem

3.1 Problem Statement

The goal of our problem is to minimize the maximum of computation cost plus communication cost between neighbors. We state the simplified optimization problem as follows.

Definition 1 (SLICERING($p, w_i, c_{i,j}, D_w, D_c$)). *Given p processors of cycle-times w_i and $p(p-1)$ communication links of capacity $c_{i,j}$, given the total workload D_w and the communication volume D_c at each step, determine*

$$T_{\mathrm{step}} = \min_{1\leq q\leq p,\ \sigma\in\Theta_{q,p},\ \sum_{i=1}^{q}\alpha_{\sigma(i)}=1} \left\{ \max_{1\leq i\leq q} \left(\alpha_{\sigma(i)}.D_w.w_{\sigma(i)} + D_c.(c_{\sigma(i),\sigma(i-1 \bmod q)} + c_{\sigma(i),\sigma(i+1 \bmod q)})\right) \right\}. \quad (2)$$

In (2), $\Theta_{q,p}$ denotes the set of one-to-one functions $\sigma : [1..q] \to [1..p]$ which index the q selected processors, for all candidate values of q between 1 and p. From this equation, it is not clear whether all processors will be involved or not. It will be the case only if the ratio $\frac{D_w}{D_c}$ is large enough. Given application parameters D_c and D_w, there is an upper bound in the number q of participating processors: Above this value, the relative cost of communications becomes too important in front of the average computation load (see Figs. 9, 10, 11, and 12 for an illustration). In any case, after deciding how many and which processors to use, we still have to decide how to arrange them along a ring. Extracting the "best" ring out of the interconnection graph seems to be a difficult combinatorial problem. Before assessing this result (see Section 3.3), we deal with the much easier situation when the network is homogeneous (see Section 3.2).

3.2 Homogeneous Networks

Solving the optimization problem, i.e., minimizing (2), is easy when all communication times are equal. This corresponds to a homogeneous network where each processor pair can communicate at the same speed, for instance, through a bus or an Ethernet backbone.

Assume that $c_{i,j} = c$ for all i and j, where c is a constant. There are only two cases to consider: 1) only the fastest processor is active and 2) all processors are involved. Indeed, as soon as a single communication occurs, we can have several ones for the same cost, and the best is to divide the computing load among all resources. In the former case 1), we derive that $T_{\mathrm{step}} = D_w.w_{\min}$, where $w_{\min}$ is the smallest cycle-time. In the latter case 2), the load is most balanced when the execution time is the same for all processors: Otherwise, removing a small portion of the load of the processor with largest execution time, and giving it to a processor finishing earlier, would decrease the maximum computation time. This leads to $\alpha_i.w_i = \mathrm{Constant}$ for all i, with $\sum_{i=1}^{p}\alpha_i = 1$. We derive that $T_{\mathrm{step}} = D_w.w_{\mathrm{cumul}} + 2D_c.c$, where $w_{\mathrm{cumul}} = \frac{1}{\sum_{i=1}^{p}\frac{1}{w_i}}$. We summarize these results.

Proposition 1. *The optimal solution to* SLICERING *(p, w_i, c, D_w, D_c) is*

$$T_{\mathrm{step}} = \min\{D_w.w_{\min}, D_w.w_{\mathrm{cumul}} + 2D_c.c\},$$

where $w_{\min} = \min_{1\leq i\leq p} w_i$ and $w_{\mathrm{cumul}} = \frac{1}{\sum_{i=1}^{p}\frac{1}{w_i}}$.

If the platform is given, there is a threshold, which is application-dependent, to decide whether only the fastest computing resource, as opposed to all the resources, should be involved. Given D_c, the fastest processor will do all the job for small values of D_w, namely, $D_w \leq D_c.\frac{2c}{w_{\min}-w_{\mathrm{cumul}}}$. Otherwise, for larger values of D_w, all processors should be involved.

3.3 Complexity

The decision problem associated to the SLICERING optimization problem is:

Definition 2 (SLICERINGDEC($p, w_i, c_{i,j}, D_w, D_c, K$)). *Given p processors of cycle-times w_i and $p(p-1)$ communication links of capacity $c_{i,j}$, given the total workload D_w and the communication volume D_c at each step, and given a time bound K, is it possible to find an integer $q \leq p$, a one-to-one mapping $\sigma : [1..q] \to [1..p]$, and nonnegative rational numbers α_i with $\sum_{i=1}^{q} \alpha_{\sigma(i)} = 1$, such that*

$$T_{\text{step}} = \max_{1 \leq i \leq q} \left\{ \alpha_{\sigma(i)}.D_w.w_{\sigma(i)} + D_c.(c_{\sigma(i),\sigma(i-1 \bmod q)} + c_{\sigma(i),\sigma(i+1 \bmod q)}) \right\} \leq K?$$

The following result states the intrinsic difficulty of the problem.

Theorem 1. *SLICERINGDEC($p, w_i, c_{i,j}, D_w, D_c, K$) is NP-complete.*

Proof. Obviously, SLICERINGDEC belongs to NP. To prove its completeness, we use a reduction from HAMPATH, the Hamiltonian Path Problem, which is NP-complete [21]. Consider an arbitrary instance $\mathcal{I}_1$ of HAMPATH: Given a graph $G_h = (V_h, E_h)$, is there a Hamiltonian cycle in G_h, i.e., a cycle that visits all the vertices of G exactly once?

We construct the following instance $\mathcal{I}_2$ of SLICERINGDEC: We let $p = |V_h|$ (assume $p \geq 2$ without loss of generality), and we define a complete interconnection graph $G = (P, E)$, whose edge costs are given by

$$c_e = \begin{cases} \varepsilon & \text{if } e \in E_h \\ 2 & \text{otherwise,} \end{cases}$$

where $0 < \varepsilon < \frac{1}{2}$ is a small constant. We let $D_w = D_c = 1$ and $w_i = p$ for $1 \leq i \leq p$. Clearly, $\mathcal{I}_2$ can be constructed in time polynomial in the size of $\mathcal{I}_1$. Finally, we let $K = 1 + 2\varepsilon$.

Assume first that $\mathcal{I}_1$ has a solution, i.e., that G_h possesses a Hamiltonian path. We use the edges of this path to build the ring. All processors are involved, and we let $\alpha_i = 1/p$ for $1 \leq i \leq p$. The execution time and the communication time are the same for all processors, we obtain that $T_{\text{step}} = \frac{1}{p} \cdot p + 2\varepsilon = K$, hence, a solution to $\mathcal{I}_2$.

Assume now that $\mathcal{I}_2$ has a solution. If a single processor was participating in that solution, then we would have $T_{\text{step}} = 1.p \geq 2 > K$, a contradiction. Hence, there are q processors, with $q \geq 2$, participating in the solution. If the ring used a communication edge that did not belong to G_h, then its cost would be 2 and $T_{\text{step}} \geq D_c.2 = 2 > K$, again a contradiction. There remains to show that we do use all the p processors in the solution. But, otherwise, if $q < p$, one computation load would be at least equal to $\frac{1}{q}.D_w.p > 1$, which would imply that $T_{\text{step}} > K$. Finally, $q = p$, and all the edges of the solution ring enable to find a Hamiltonian path in G_h, thereby providing a solution to $\mathcal{I}_1$. □

In [31], we have expressed the solution to the SLICERING optimization problem, in terms of an Integer Linear Programming (ILP) problem. When all processors are involved in the optimal solution, we also proved that finding the optimal solution is equivalent to solving the Traveling Salesman Problem (TSP) in the weighted graph (P, E, d), where $d_{i,j} = \frac{c_{i,j} + c_{j,i}}{w_i}$.

4 The SharedRing Optimization Problem

4.1 Sharing Links

The SHAREDRING optimization problem looks very similar to the simplified version.

Definition 3 (SHAREDRING($p, w_i, E, b_{e_m}, D_w, D_c$)). *Given p processors P_i of cycle-times w_i and E communication links e_m of bandwidth b_{e_m}, given the total workload D_w and the communication volume D_c at each step, determine*

$$T_{\text{step}} = \min_{1 \leq q \leq p,\ \sigma \in \Theta_{q,p},\ \sum_{i=1}^{q} \alpha_{\sigma(i)} = 1} \left\{ \max_{1 \leq i \leq q} \left(\alpha_{\sigma(i)}.D_w.w_{\sigma(i)} + D_c.(c_{\sigma(i),\sigma(i-1 \bmod q)} + c_{\sigma(i),\sigma(i+1 \bmod q)}) \right) \right\}. \quad (3)$$

As before, in (3), $\Theta_{q,p}$ denotes the set of one-to-one functions $\sigma : [1..q] \to [1..p]$ which index the q selected processors that form the ring, for all candidate values of q between 1 and p. But, in the general version, for each candidate ring represented by such a σ function, there are constraints hidden by the introduction of the quantities $c_{\sigma(i),\sigma(i-1 \bmod q)}$ and $c_{\sigma(i),\sigma(i+1 \bmod q)}$, which we gather now. There are $2q$ communicating paths, the path $\mathcal{S}_i$ from $P_{\sigma(i)}$ to its successor $P_{\text{succ}(\sigma(i))} = P_{\sigma(i+1 \bmod q)}$ and the path $\mathcal{P}_i$ from $P_{\sigma(i)}$ to its predecessor $P_{\text{pred}(\sigma(i))} = P_{\sigma(i-1 \bmod q)}$, for $1 \leq i \leq q$. For each link e_m in the interconnection network, let $s_{\sigma(i),m}$ (respectively, $p_{\sigma(i),m}$) be the portion of the bandwidth b_{e_m} that is allocated to the path $\mathcal{S}_{\sigma(i)}$ (respectively, $\mathcal{P}_{\sigma(i)}$). We have the equations:

$$\begin{cases} s_{\sigma(i),m} \geq 0,\ p_{\sigma(i),m} \geq 0 & 1 \leq i \leq q, 1 \leq m \leq E, \\ c_{\sigma(i),\text{succ}(\sigma(i))} = \frac{1}{\min_{e_m \in \mathcal{S}_{\sigma(i)}} s_{\sigma(i),m}} & 1 \leq i \leq q, \\ c_{\sigma(i),\text{pred}(\sigma(i))} = \frac{1}{\min_{e_m \in \mathcal{P}_{\sigma(i)}} p_{\sigma(i),m}} & 1 \leq i \leq q, \\ \sum_{i=1}^{q} (s_{\sigma(i),m} + p_{\sigma(i),m}) \leq b_{e_m} & 1 \leq m \leq E. \end{cases}$$

The last equation states that the bandwidth of link e_m is not exceeded. Since each communicating path $\mathcal{S}_{\sigma(i)}$ or $\mathcal{P}_{\sigma(i)}$ will typically involve a few edges, most of the quantities $s_{\sigma(i),m}$ and $p_{\sigma(i),m}$ will be zero. In fact, we have written $e_m \in \mathcal{S}_{\sigma(i)}$ if the edge e_m is actually used in the path $\mathcal{S}_{\sigma(i)}$, i.e., if $s_{i,m}$ is not zero (and, similarly, $e_m \in \mathcal{P}_{\sigma(i)}$ if $p_{i,m}$ is not zero).

Because of the link sharing, the SHAREDRING problem looks even more combinatorial than the SLICERING problem. We have to select the participating resources, to arrange them along a ring, to construct the communicating paths, to assign bandwidths ratios and, finally, to allocate data chunks. Before stating a complexity result (see Section 4.3), we work out a small-size example (Section 4.2).

4.2 Toy Example

Consider the heterogeneous network represented in Fig. 1. There are seven processors and eight bidirectional communication links. For the sake of simplicity, we have labeled the processors P_1 to P_5 in the order that they appear in the 5-processor ring that we construct, leaving out the other two processors Q and R. Also, links are labeled with letters from

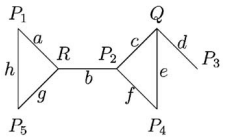


Fig. 1. A small-size cluster.

a to h instead of indices; we use b_x to denote the bandwidth of link x.

For the path from P_1 to P_2, we choose to use links a and b so that the communication path from P_1 to its successor P_2 is $\mathcal{S}_1 = \{a, b\}$. But, for the path from P_2 to P_1, we may use links b, g, and h so that the communication path from P_2 to its predecessor is $\mathcal{P}_2 = \{b, g, h\}$. Here is the complete list of the paths (note that many other choices could have been made):

- From P_1: to P_2, $\mathcal{S}_1 = \{a, b\}$ and to P_5, $\mathcal{P}_1 = \{h\}$.
- From P_2: to P_3, $\mathcal{S}_2 = \{c, d\}$ and to P_1, $\mathcal{P}_2 = \{b, g, h\}$.
- From P_3: to P_4, $\mathcal{S}_3 = \{d, e\}$ and to P_2, $\mathcal{P}_3 = \{d, e, f\}$.
- From P_4: to P_5, $\mathcal{S}_4 = \{f, b, g\}$ and to P_3, $\mathcal{P}_4 = \{e, d\}$.
- From P_5: to P_1, $\mathcal{S}_5 = \{h\}$ and to P_4, $\mathcal{P}_5 = \{g, b, f\}$.

Next, we define the path costs. For P_1, because $\mathcal{S}_1 = \{a, b\}$, we get $c_{1,2} = \frac{1}{\min(s_{1,a}, s_{1,b})}$; and because $\mathcal{P}_1 = \{h\}$, we get $c_{1,5} = \frac{1}{p_{1,h}}$. We proceed likewise for P_2 to P_5. Finally, here is the list of all the equations that must be satisfied:

$$
\begin{array}{ll}
\textbf{Link a: } s_{1,a} \le b_a & \textbf{Link b: } s_{1,b}+s_{4,b}+p_{2,b}+p_{5,b} \le b_b \\
\textbf{Link c: } s_{2,c} \le b_c & \textbf{Link d: } s_{2,d}+s_{3,d}+p_{3,d}+p_{4,d} \le b_d \\
\textbf{Link e: } s_{3,e}+p_{3,e}+p_{4,e} \le b_e & \textbf{Link f: } s_{4,f}+p_{3,f}+p_{5,f} \le b_f \\
\textbf{Link g: } s_{4,g}+p_{2,g}+p_{5,g} \le b_g & \textbf{Link h: } s_{5,h}+p_{1,h}+p_{2,h} \le b_h.
\end{array}
$$

Now that we have all these constraints, we can (try to) compute the α_i, $s_{i,j}$, and $p_{i,j}$ that minimize the objective function T_{step}. Equation 4 explicits the whole system of (in) equations which is quadratic in the unknowns α_i, $s_{i,j}$, $c_{i,j}$, and $p_{i,j}$.[1]

$$
\begin{array}{l}
\text{minimize} \quad \max_{1 \le i \le 5} (\alpha_i.D_w.w_i + D_c.(c_{i,i-1} + c_{i,i+1})) \quad \text{subject to} \\
\left\{ \begin{array}{lll}
\sum_{i=1}^{5} \alpha_i = 1 & & \\
s_{1,a} \le b_a & s_{1,b}+s_{4,b}+p_{2,b}+p_{5,b} \le b_b & s_{2,c} \le b_c \\
s_{2,d}+s_{3,d}+p_{3,d}+p_{4,d} \le b_d & s_{3,e}+p_{3,e}+p_{4,e} \le b_e & s_{4,f}+p_{3,f}+p_{5,f} \le b_f \\
s_{4,g}+p_{2,g}+p_{5,g} \le b_g & s_{5,h}+p_{1,h}+p_{2,h} \le b_h & \\
s_{1,a}.c_{1,2} \ge 1 & s_{1,b}.c_{1,2} \ge 1 & p_{1,h}.c_{1,5} \ge 1 \\
s_{2,c}.c_{2,3} \ge 1 & s_{2,d}.c_{2,3} \ge 1 & p_{2,b}.c_{2,1} \ge 1 \\
p_{2,g}.c_{2,1} \ge 1 & p_{2,h}.c_{2,1} \ge 1 & s_{3,d}.c_{3,4} \ge 1 \\
s_{3,e}.c_{3,4} \ge 1 & p_{3,d}.c_{3,2} \ge 1 & p_{3,e}.c_{3,2} \ge 1 \\
p_{3,f}.c_{3,2} \ge 1 & s_{4,f}.c_{4,5} \ge 1 & s_{4,b}.c_{4,5} \ge 1 \\
s_{4,g}.c_{4,5} \ge 1 & p_{4,e}.c_{4,3} \ge 1 & p_{4,d}.c_{4,3} \ge 1 \\
s_{5,h}.c_{5,1} \ge 1 & p_{5,g}.c_{5,4} \ge 1 & p_{5,b}.c_{5,4} \ge 1 \\
p_{5,f}.c_{5,4} \ge 1. &
\end{array} \right.
\end{array}
\tag{4}
$$

To build up (4), we have used arbitrary communication paths, and there are many others to try. Worse, there are many other rings to build, even with the same processors that could be arranged differently, or with other processors. And, the number of processors q must be varied also. Not surprisingly, the decision problem associated to the SHARED-DRING optimization problem is NP-complete, as shown in Section 4.3.

1. We did not express in (4) the inequations stating that all the unknowns are nonnegative.

4.3 Complexity

The decision problem associated to the SHAREDRING optimization problem turns out to be NP-complete. The proof of this result is quite similar to that of Theorem 1 (see [27]), and we state it for the sake of reference.

Theorem 2. *SHAREDRINGDEC($p, w_i, E, b_{e_m}, D_w, D_c, K$) is NP-complete.*

4.4 Heuristics

In this section, we describe a polynomial-time heuristic to solve the SHAREDRING optimization problem. We describe the heuristic in three steps: 1) the greedy algorithm used to construct a solution ring, 2) the strategy used to assign bandwidth portions during the construction, and 3) a final refinement.

4.4.1 Ring Construction

We consider a solution ring involving q processors, numbered from P_1 to P_q. Ideally, all these processors should require the same amount of time to compute and communicate: Otherwise, we would slightly decrease the computing load of the last processor to complete its assignment (computations followed by communications) and assign extra work to another one. Hence (see Fig. 2 for an illustration), we have

$$T_{\text{step}} = \alpha_i.D_w.w_i + D_c.(c_{i,i-1} + c_{i,i+1})$$

for all i (indices in the communication costs are taken modulo q). Since $\sum_{i=1}^{q} \alpha_i = 1$, we derive that $\sum_{i=1}^{q} \frac{T_{\text{step}} - D_c.(c_{i,i-1}+c_{i,i+1})}{D_w.w_i} = 1$. Defining $w_{\text{cumul}} = \frac{1}{\sum_{i=1}^{q} \frac{1}{w_i}}$, we rewrite this as:

$$T_{\text{step}} = D_w.w_{\text{cumul}} \left(1 + \frac{D_c}{D_w} \sum_{i=1}^{q} \frac{c_{i,i-1} + c_{i,i+1}}{w_i} \right). \tag{5}$$

We will use (5) as a basis for a greedy algorithm to grow a solution ring iteratively. The greedy heuristic starts by selecting the best pair of processors. Then, it iteratively includes a new node in the current solution ring. Assume that we have already selected a ring of r processors. For each remaining processor P_i, we search where to insert it in the current ring: For each pair of successive processors (P_j, P_k) in the ring, we compute the cost of inserting P_i between P_j and P_k in the ring. We retain the processor and the pair that minimize the insertion cost, and we store the new value of T_{step}.

How do we compute the cost of inserting P_i between P_j and P_k? We have to resort to another heuristic to construct communicating paths and allocate bandwidth portions (explained in Section 4.4.2), in order to compute the new costs $c_{k,j}$ (path from P_k to its successor P_j), $c_{j,k}$ (the other way round), $c_{k,i}$ (path from P_k to its predecessor P_i), and $c_{k,i}$ (the other way round). Once we have these costs, we can compute the new value of T_{step} as follows:

- We update w_{cumul} by adding the new processor P_k into the formula, which will decrease its value.
- In the summation $\sum_{s=1}^{r} \frac{c_{\sigma(s),\sigma(s-1)}+c_{\sigma(s),\sigma(s+1)}}{w_{\sigma(s)}}$, we suppress the two terms corresponding to the two paths between P_i to P_j (by hypothesis we had $i = \sigma(s)$

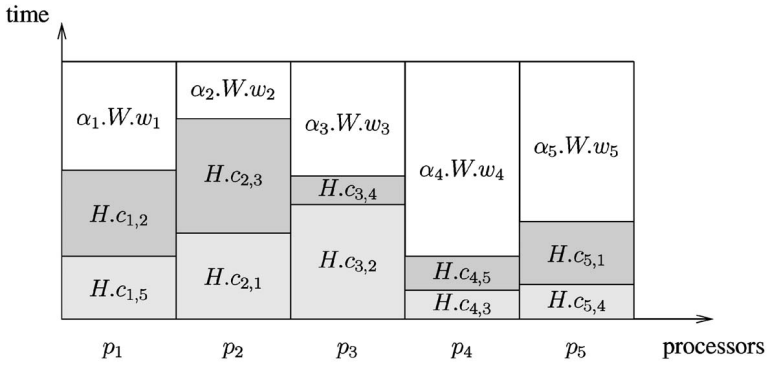


Fig. 2. Summary of computation and communication times with $q = 5$ processors.

and $j = \sigma(s+1)$ for some s), and we insert the new terms $\frac{c_{k,j}+c_{k,i}}{w_k}$, $\frac{c_{j,k}}{w_j}$, and $\frac{c_{i,k}}{w_i}$.

This step of the heuristic has a complexity proportional to $(p-r).r$ times the cost to compute four communicating paths. Finally, we grow the ring until we have p processors. We return the minimal value obtained for T_{step}. The total complexity is $\sum_{r=1}^{p}(p-r)rC = O(p^3)C$, where C is the cost of computing four paths in the network. Note that it is important to try all values of r because T_{step} may not vary monotonically with r (for instance, see Fig. 11).

4.4.2 Bandwidth Allocation

In this section, we assume that we already have a r-processor ring, a pair (P_i, P_j) of successive processors in the ring, and a new processor P_k to be inserted between P_i and P_j. Together with the ring, we have constructed $2r$ communicating paths, and a certain portion of the initial bandwidth has been allocated to these paths. To build the new four paths involving P_k, we reason on the graph $G = (V, E, b)$, where each edge is labeled with the remaining available bandwidth: Now, $b(e_m)$ is not the initial bandwidth of edge e_m, but what has been left by the $2r$ paths.

The first thing to do is to reinject in the network the bandwidth portions used by the two communication paths between P_i and P_j (because these paths will be replaced by the new four paths). We use a simple shortest path algorithm to determine the four paths, from P_k to P_i and P_j and vice-versa. There is a subtlety here because these four paths may share some links. The strategy that we use is the following:

- We independently compute four paths of maximal bandwidth, using a standard shortest path algorithm [13] in G.
- If some paths happen to share some links, we do not change the paths; instead, we use a brute force (analytical) method to compute the bandwidth portions minimizing (5) to be allocated to each path, and we update the four path costs accordingly.

Now that we have the paths and their costs, we compute the new value of T_{step} as explained above. Note that from T_{step}, we can derive the values of the computing workloads α_i, but we do not need them until the end. The cost C of computing four paths in the network is $O(p+E)$.

4.4.3 Refinements

A concise way to describe the heuristic is the following: We greedily grow a ring by peeling off the bandwidths to insert new processors. To diminish the cost of the heuristic, we never recalculate the bandwidth portions that have been assigned to previous communicating paths. When we are done with the heuristic, we have a q-processor ring, q workloads, $2q$ communicating paths, bandwidth portions, and communication costs for these paths, and a feasible value of T_{step}.

Because the heuristic could appear oversimplistic, we have implemented two variants aimed at refining the solution. The idea for the two variants is to keep everything but the bandwidth portions and the workloads, and to recompute these each time we have inserted a new processor in the ring. In other words, once we have selected the processor and the pair minimizing the insertion cost in the current ring, we perform the insertion and we recompute all the bandwidth portions and the workloads. We keep the ring (both the processors and their ordering) and the communication paths as such. Since we know all the $2q$ paths, we can reevaluate bandwidth portions, hence, communication costs, using a global approach:

Method 1: Max-min fairness. This is the traditional bandwidth-sharing algorithm [5], which is designed to maximize the minimum bandwidth allocated to a path. Once we have computed the bandwidths portions with the algorithm, we have the communication costs, and we compute the α_i so as to equate all execution times (computations followed by communications), thereby minimizing T_{step}.

Method 2: quadratic resolution using the KINSOL software. As mentioned in Section 4.2, once we have a ring and communicating paths, the program to minimize T_{step} is quadratic in the unknowns α_i, $s_{i,j}$, $c_{i,j}$, and $p_{i,j}$. We use the KINSOL library [34] to solve it.

5 Experimental Results

5.1 SharedRing with a Network Generator

To evaluate the heuristics designed for SHAREDRING, we experimented with two platforms generated with the Tiers network generator [10], [16]. This generator produces graphs having three levels of hierarchy referred as LAN, MAN, and WAN levels.

5.1.1 Platform Description

The platforms are generated by selecting a fraction of the LAN nodes, referred to as the boxed nodes in Figs. 3 and 4. These boxed nodes are the selected machine nodes that are used in the computing process. They represent about

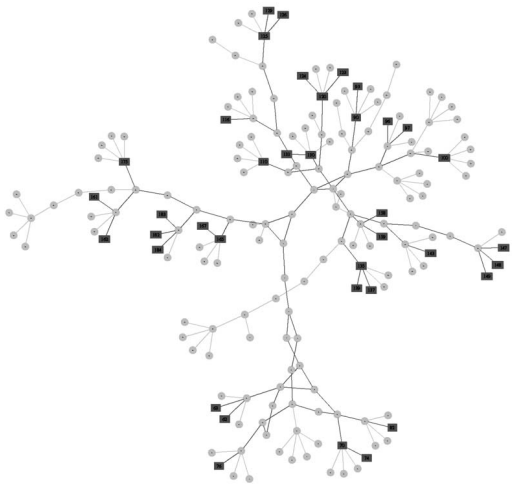


Fig. 3. First platform. Boxed nodes are selected machine nodes. There are 38 selected machine nodes, connected through 49 routers and 92 communication links.

30 percent of the initial LAN nodes. All other nodes are handled as simple routers. The processing powers of the selected machine nodes are randomly chosen in a list of values corresponding to the processing powers (expressed in MFlops and evaluated thanks to a benchmark taken from LINPACK [8]) of a wide variety of machines (Pentium Pro 200MHz, Pentium 2 350MHz, Celeron 400MHz, Athlon 1.4GHz, Pentium 4 1.7GHz, etc.). The capacities of the edges are assigned using the classification of the Tiers generator (local LAN link, LAN/MAN link, MAN/WAN link, etc.). For each link type, we use values measured using `pathchar` [17] between some machines in ENS Lyon and some other machines scattered in France (Strasbourg, Lille, Grenoble, and Orsay), in the US (Knoxville, San Diego, and Argonne), and in Japan (Nagoya and Tokyo). The second platform is (roughly) twice larger than the first platform.

5.1.2 Results

In Figs. 5 and 6, we plot the number of processors used in the solution ring. The ratio D_c/D_w ranges between values which have been roughly estimated as follows: Suppose we have a square matrix of size n to which we want to apply some filters. D_c is then roughly equal to $n \times 64/1,024^2$ Mbits (when using double) and D_w is roughly equal to $n^2/1,024^2$ Mflops (if there is one Floating Point operation to perform per cell). D_c/D_w is then equal to $64/n$ and for reasonnable values of n, D_c/D_w ranges between 0.0064 (for $n = 10,000$) and 0.64 (for $n = 100$).

As expected, the size of the ring increases as the ratio D_c/D_w decreases: additional computational power pay off the communication overhead. In Figs. 7, 8, 9, 10, 11, and 12, we represent the normalized execution time as a function of the size of the solution ring. For both platforms, we use various communication-to-computation ratios. For each ratio, there is an optimal size, which is reached with more and more processors as the ratio decreases. When D_c/D_w is small enough, the *bath tub* shape of the curves is readily explained: The execution time first decreases as the number of processors increases because the load is distributed over more resources; then, the relative cost of communications becomes more important, and it no longer pays off to use more resources. Note that, when the ratio D_c/D_w is too large, we can see that, as expected, it is not worth it to distribute the application among different processors.

Finally, we assess the usefulness of the two variants (max-min fairness and quadratic programming) introduced to refine the heuristic. Surprisingly enough, the impact of both variants is not significant at all (see Figs. 13 and 14). In fact, they turn out to be even less efficient in most cases than the plain version of the heuristic, which turns out to be both low-cost and very efficient.

Fig. 4. Second platform. Boxed nodes are selected machine nodes. There are 90 selected machine nodes, connected through 43 routers and 136 communication links.

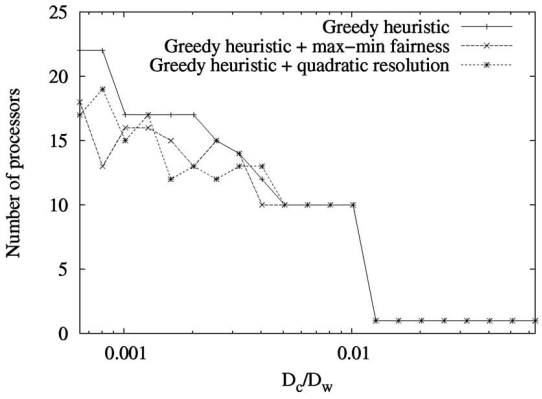


Fig. 5. First platform. Size of the optimal ring as a function of the ratio D_c/D_w.

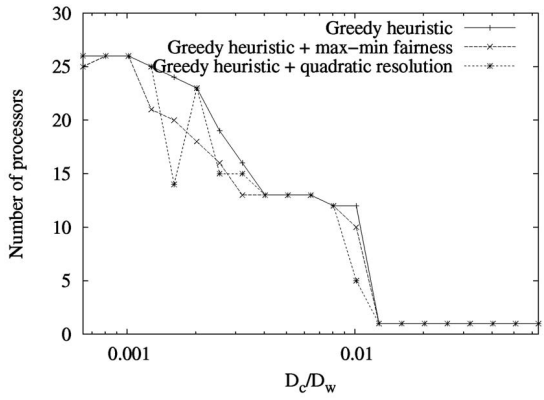


Fig. 6. Second platform. Size of the optimal ring as a function of the ratio D_c/D_w.

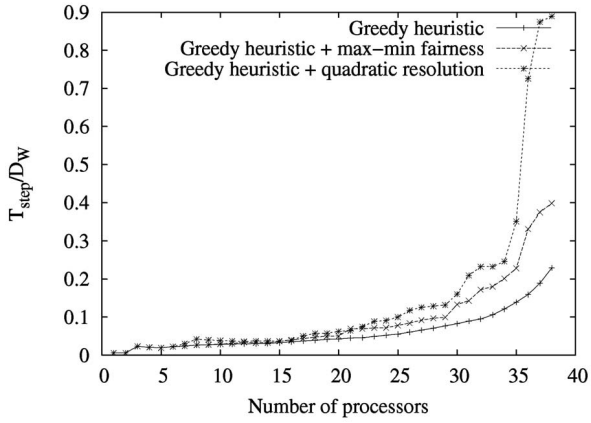


Fig. 7. First platform. Value of T_{step} normalized by D_w as a function of the size of the solution ring, with a high communication-to-computation ratio: $D_c/D_w = 0.064$. In both cases, the optimal ring size is equal to 1: There is not enough work to distribute, hence it is not worth it to distribute the application among different processors.

5.2 Assessing the Impact of Link Sharing

5.2.1 Platform Description

To evaluate the impact of link sharing, we experimented with two platforms, one heterogeneous network of workstations located in ENS Lyon and the other in the University of Strasbourg. Fig. 15 shows the Lyon platform, which is composed of 14 computing resources and three routers. An abstraction of the Lyon platform is represented in Fig. 16. In this figure, circled nodes 0 to 13 are the processors, and diamond nodes 14 to 16 are the routers. Edges are labeled with link bandwidths. Processor cycle-times are gathered in Table 1.

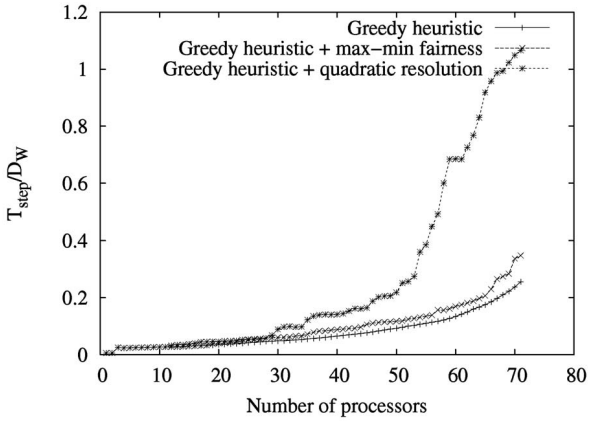


Fig. 8. Second platform. Value of T_{step} normalized by D_w as a function of the size of the solution ring, with a high communication-to-computation ratio: $D_c/D_w = 0.064$. In both cases, the optimal ring size is equal to 1: There is not enough work to distribute, hence it is not worth it to distribute the application among different processors.

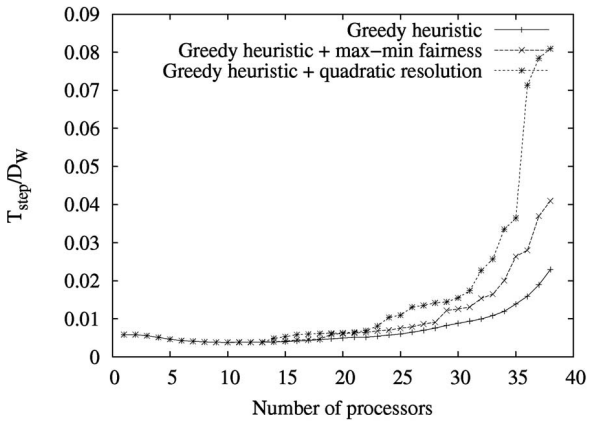


Fig. 9. First platform. Value of T_{step} normalized by D_w as a function of the size of the solution ring, with a balanced communication-to-computation ratio: $D_c/D_w = 0.0064$. The most efficient ring found has a size equal to 10.

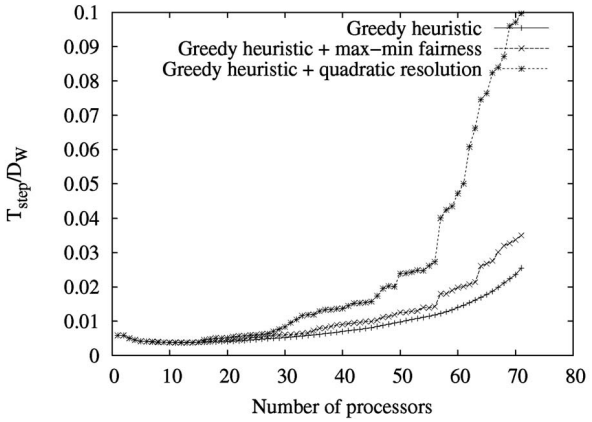


Fig. 10. Second platform. Value of T_{step} normalized by D_w as a function of the size of the solution ring, with a balanced communication-to-computation ratio: $D_c/D_w = 0.0064$. The most efficient ring found has a size equal to 13.

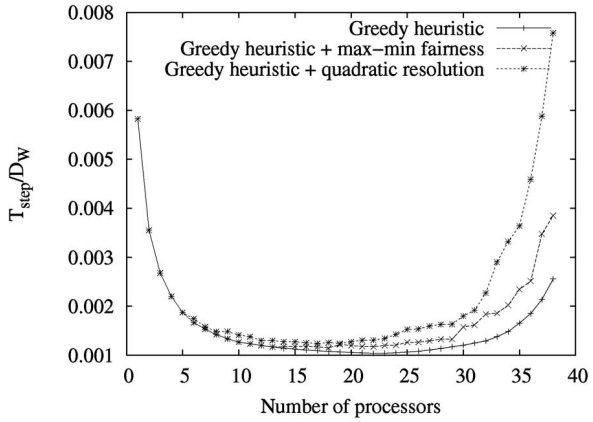


Fig. 11. First platform. Value of T_{step} normalized by D_w as a function of the size of the solution ring, with a low communication-to-computation ratio: $D_c/D_w = 0.00064$. The most efficient ring found has a size equal to 22.

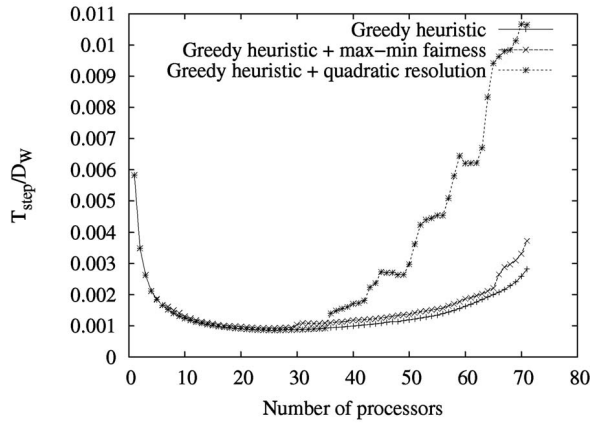


Fig. 12. Second platform. Value of T_{step} normalized by D_w as a function of the size of the solution ring, with a low communication-to-computation ratio: $D_c/D_w = 0.00064$. The most efficient ring found has a size equal to 26.

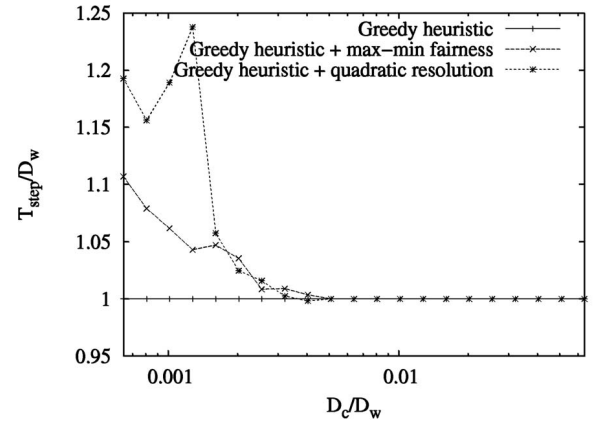


Fig. 13. First platform. Impact of the refinements on the quality of the solution.

Similarly, Fig. 17 represents the Strasbourg platform, which is composed of 13 computing resources and six routers. An abstraction of the Strasbourg platform is represented in Fig. 18. In this figure, circled nodes 0 to 12 are the processors, and diamond nodes 13 to 18 are the routers. Edges are labeled with link bandwidths. Processor cycle-times are gathered in Table 2.

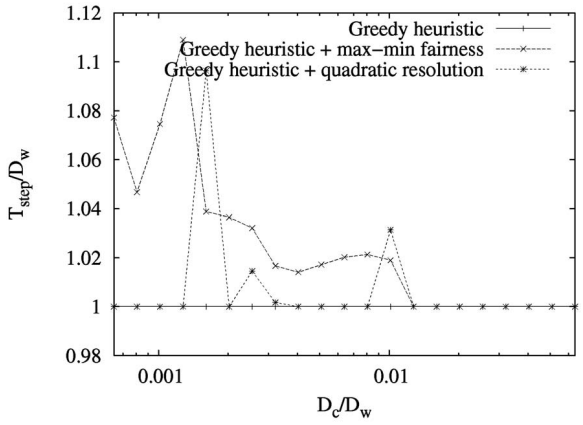


Fig. 14. Second platform. Impact of the refinements on the quality of the solution.

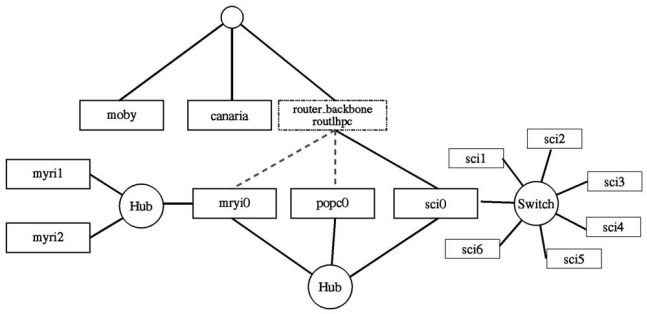


Fig. 15. Topology of the Lyon platform.

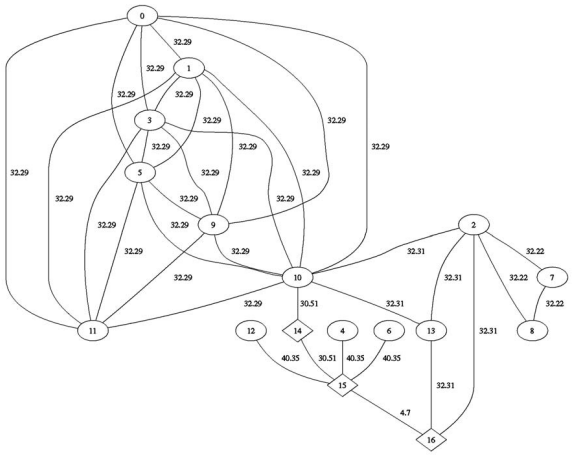


Fig. 16. Abstraction of the Lyon platform.

5.2.2 Results

To evaluate the impact of link sharing, we proceeded as follows with the Lyon and Strasbourg platforms.

In the first heuristic, we return the solution ring computed by a greedy heuristic for the SLICERING problem. This heuristic takes a fully connected network as input and iteratively builds a solution ring, quite similarly to the greedy heuristic described in Section 4.4. The heuristic starts by selecting the fastest processor. Then, it iteratively includes a new node in the current solution ring. Assume that we have already selected a ring of r processors. For each remaining processor P_i, we search where to insert it in the current ring: For each pair of successive processors (P_j, P_k) in the ring, we

TABLE 1
Processor Cycle-Times (in Seconds per Megaflop) for the Lyon Platform

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
0.0206	0.0206	0.0206	0.0206	0.0291	0.0206	0.0087	0.0206	0.0206
P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	
0.0206	0.0206	0.0206	0.0291	0.0451	0	0	0	

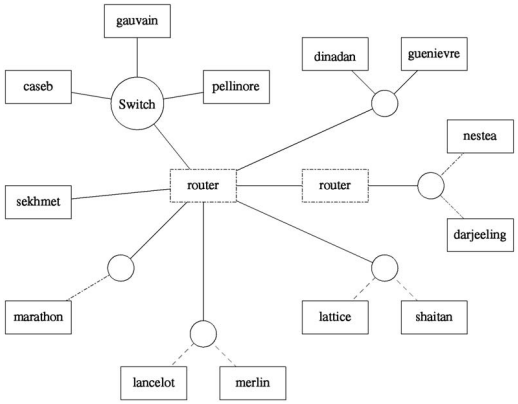


Fig. 17. Topology of the Strasbourg platform.

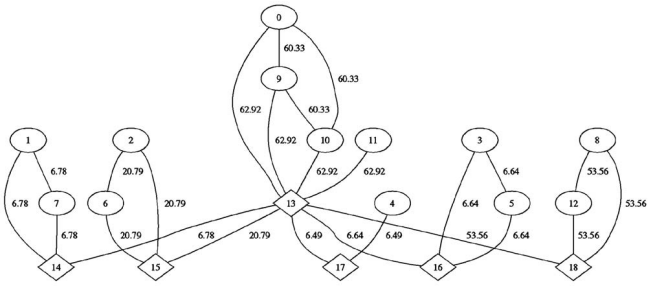


Fig. 18. Abstraction of the Strasbourg platform.

compute the cost of inserting P_i between P_j and P_k in the ring. We retain the processor and the pair that minimize the insertion cost, and we store the value of T_{step}. This step of the heuristic is much simpler than for the SHAREDRING problem, and it has a complexity proportional to $(p-r).r$. Finally, we grow the ring until we have p processors and we return the minimal value obtained for T_{step}. The total complexity is $\sum_{r=1}^{p}(p-r)r = O(p^3)$. Note that it is important to try all values of r because T_{step} may not vary monotically with r.

How do we provide a (fake) completely connected network as input of this heuristic? We have to build up the communication matrix, that gives the capacity of each (virtual) link between any processor pair. We use a shortest-paths algorithm (in terms of bandwidth) to construct this matrix, thereby simulating a complete interconnection graph. For example, with the Lyon platform, to go from processor 12 to processor 5, we use the following links: $13, 5, 4, 23$. Link 13 connects processor 12 and router 15, link 5 connects routers 15 and 14, link 4 connects router 14 and processor 10, and link 23 connects processors 10 and 5. The maximum bandwidth available is 30.51 Mb/s (these bandwidths were obtained through an ssh connection (constraint of encoding, etc.),

TABLE 2
Processor Cycle-Times (in Seconds per Megaflop) for the Strasbourg Platform

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
0.0087	0.0072	0.0087	0.0131	0.016	0.0058	0.0087	0.0262	0.0102	0.0131
P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}	
0.0072	0.0058	0.0072	0	0	0	0	0	0	

TABLE 3
T_{step}/D_w for Each Heuristic on the Lyon Platform (Number between Parentheses Denotes the Size of the Corresponding Ring)

Ratio D_c/D_w	H1 : slice-ring	H2 : shared-ring	Improvement
0.64	0.008738 (1)	0.008738 (1)	0%
0.064	0.018837 (13)	0.006639 (14)	64.75%
0.064	0.003819 (13)	0.001975 (14)	48.28%

TABLE 4
T_{step}/D_w for Each Heuristic on the Strasbourg Platform (Number between Parentheses Denotes the Size of the Corresponding Ring)

Ratio D_c/D_w	H1 : slice-ring	H2 : shared-ring	Improvement
0.64	0.005825 (1)	0.005825 (1)	0 %
0.064	0.027919 (8)	0.004865 (6)	82.57%
0.0064	0.007218 (13)	0.001608 (8)	77.72%

which explains the relatively low value), and we store this value in the matrix.

Clearly, the value of T_{step} achieved by the heuristic may well not be feasible because the actual network is not fully connected. Therefore, we keep the ring and the communicating paths between adjacent processors in the ring, and we compute feasible values using the max-min fairness heuristic.

The second heuristic is the greedy heuristic designed in Section 4.4 for the SHAREDRING problem, using the max-min fairness heuristic. The major difference between the two heuristics is that the latter takes link contention into account when building up the solution ring.

To compare the value of $\frac{T_{step}}{D_w}$ returned by both algorithms, we use various communication-to-computation ratios. Tables 3 and 4 show these values for each platform. The conclusions that can be drawn from these experiments is that an accurate modeling of the communications has a dramatic impact on the performance of the load-balancing strategies.

Note that we cannot compare our heuristics to an exhaustive search of the optimal ring. Indeed, such research would be prohibitive as it requires not only to enumerate all processor permutations, but also for each permutation to test all possible routings (as using alternate routings may decrease the amount of link sharing and thus increase the available bandwidth).

6 Related Work

Load balancing strategies have been widely studied, both for homogeneous platforms (see the collection of papers [32]) and for heterogeneous clusters (see chapter 25 in [9]). Distributing the computations (together with the associated data) can be performed either dynamically or statically, or a mixture of both.

The vast majority of the literature deals with dynamic strategies that call for periodic remapping phases to remedy observed load-imbalance. Even though we target static schemes, we briefly discuss a few important references in the field of dynamic approaches. Simple paradigms are based upon the idea "*use the past to predict the future*," i.e., use the currently observed speed of computation of each machine to decide for the next distribution of work [11], [12], [4]. Several authors [18], [30], [29], [35], [22] propose a mapping policy which dynamically minimizes system degradation (including the cost of remapping) for each computation step. Other papers [36], [15] advocate local schemes where data is exchanged only between neighbor processors. Generally speaking, there is a challenge in determining a trade off between the data distribution parameters and the process spawning and possible migration policies. Redundant computations might also be necessary to use a heterogeneous platform at its best capabilities [28].

In the context of a library oriented approach, dynamic strategies are difficult to introduce because they imply a complicated memory management. Static strategies are less general, but prove useful if enough knowledge can be injected in the scheduling and mapping decision process. In other words, if the characteristics of the target platform (processor speeds and link capacities) and of the target application (computation and communication costs associated to each data chunk) are known rather accurately, then excellent performance can be achieved through static strategies. However, sophisticated data distribution schemes (like the ones presented in this paper) are mandatory to achieve such a good performance.

A survey of static load balancing techniques for mesh computations has been written by Hu and Blake [22]. On the same subject, see also the paper by Ichikawa and Yamashita [23]. Several authors have dealt with the static implementation of linear algebra kernels on heterogeneous platforms. Matrix multiplication has been studied by [25], [2]. LU and QR decomposition have been discussed by Barbosa et al. [1]. Static partitioning schemes to map a two-dimensional data matrix onto heterogeneous resources have been investigated by Crandall and Quinn [14], Kaddoura et al. [24], and Beaumont et al. [3]. The main conclusions of these papers are drawn for three kinds of problems:

- Distributing independent chunks of work to unidimensional (linear) arrays of heterogeneous processors is easy (see the algorithm in [2]).
- Distributing independent chunks of work to two-dimensional processor grids is difficult. We have to search for the best distribution of work for each processor arrangement along the two-dimensional grid, and there is an exponential number of such arrangements as the grid size increases (see [1], [2]).
- Relaxing the geometrical constraints induced by two-dimensional grids leads to irregular partitionings [14], [24], [3] that allow for a good load-balancing, but are much more difficult to implement. This approach has been extended to three-dimensional problems [19].

In this perspective, this paper shows that the first problem, i.e., distributing independent chunks of work to unidimensional processor arrays, is no longer easy when communications are taken into account in addition to computations.

Related work also includes the vast amount of literature dealing with divisible loads (see [6], [7]): Just as in this paper, a big chunk of work can be arbitrarily divided into several pieces, and these pieces are assigned to processors so that the total execution time, i.e., the sum of the communication and the computation, is minimized. However, in the divisible load theory, the target architecture is fixed, typically a master-slave fork graph, or a tree, and the communication links are dedicated.

7 Conclusion

In this paper, the major emphasis was toward a realistic modeling of concurrent communications in cluster networks. One major result is the NP-completeness of SLICERING and SHAREDRING problems. Rather than the proof, the result itself is interesting because it provides yet another evidence of the intrinsic difficulty of designing algorithms for heterogeneous platforms. But, this negative result should not be overemphasized. Indeed, another important contribution of this paper is the design of an efficient heuristic, that provides a pragmatic guidance to the designer of iterative scientific computations. Implementing such computations on commodity clusters made up of several heterogeneous resources is a promising alternative to using costly supercomputers.

One major limitation of our work is that it assumes some static knowledge of the target heterogeneous platform to take mapping and load-balancing decisions. In fact, load balancing techniques can be introduced dynamically or statically, or a mixture of both. On one hand, we may think that dynamic strategies are likely to perform better because the machine loads will be self-regulated, hence, self-balanced, if processors pick up new tasks just as they terminate their current computation. However, data dependences, in addition to communication costs and control overhead, may well lead to slow the whole process down to the pace of the slowest processors. On the other hand, static strategies will suppress (or at least minimize) data redistributions and control overhead during execution. Furthermore, in the context of a scientific library, static allocations seem to be necessary for a simple and efficient memory allocation.

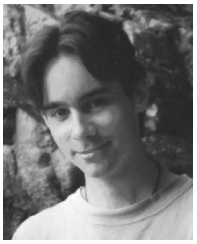
We agree, however, that targeting larger platforms such as distributed collections of heterogeneous clusters, e.g., available from the metacomputing grid [20], may well enforce the use of dynamic schemes. If the target computing platform is unstable, with rapidly changing CPU loads and communication hot-spots, a dynamic approach is likely to be the only viable alternative. Still, even in such a context, there is hope that injecting some static knowledge in the dynamic scheduler will prove useful for a wide spectrum of applications.

ACKNOWLEDGMENTS

The authors thank the reviewers for their helpful comments and suggestions, which greatly improved the final version of the paper. A much shorter version of this work appeared in the Proceedings of the Euro-PVM/MPI 2003 Conference [31].

REFERENCES

[1] J. Barbosa, J. Tavares, and A.J. Padilha, "Linear Algebra Algorithms in a Heterogeneous Cluster of Personal Computers," *Proc. Ninth Heterogeneous Computing Workshop,* pp. 147-159, 2000.
[2] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert, "A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers)," *IEEE Trans. Computers,* vol. 50, no. 10, pp. 1052-1070, 2001.
[3] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix Multiplication on Heterogeneous Platforms," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 10, pp. 1033-1051, Oct. 2001.
[4] F. Berman, "High-Performance Schedulers," *The Grid: Blueprint for a New Computing Infrastructure,* I. Foster and C. Kesselman, eds., pp. 279-309, Morgan-Kaufmann, 1999.
[5] D. Bertsekas and R. Gallager, *Data Networks.* Prentice Hall, 1987.
[6] V. Bharadwaj, D. Ghose, V. Mani, and T.G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems,* IEEE CS Press, 1996.
[7] V. Bharadwaj, D. Ghose, and T.G. Robertazzi, "A New Paradigm for Load Scheduling in Distributed Systems," *Cluster Computing,* vol. 6, no. 1, pp. 7-18, Jan. 2003.
[8] R.P. Brent, "The LINPACK Benchmark on the AP1000: Preliminary Report," *Proc. CAP Workshop,* 1991.
[9] R. Buyya, *High Performance Cluster Computing. Volume 1: Architecture and Systems.* Upper Saddle River, N.J.: Prentice Hall PTR, 1999.
[10] K.L. Calvert, M.B. Doar, and E.W. Zegura, "Modeling Internet Topology," *IEEE Comm. Magazine,* vol. 35, no. 6, pp. 160-163, June 1997.
[11] M. Cierniak, M.J. Zaki, and W. Li, "Compile-Time Scheduling Algorithms for Heterogeneous Network of Workstations," *The Computer J.,* vol. 40, no. 6, pp. 356-372, 1997.
[12] M. Cierniak, M.J. Zaki, and W. Li, "Customized Dynamic Load Balancing for a Network of Workstations," *J. Parallel and Distributed Computing,* vol. 43, pp. 156-162, 1997.
[13] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms.* MIT Press, 1990.
[14] P.E. Crandall and M.J. Quinn, "Block Data Decomposition for Data-Parallel Programming on a Heterogeneous Workstation Network," *Proc. Second Int'l Symp. High Performance Distributed Computing,* pp. 42-49, 1993.
[15] E. Deelman and B.K. Szymanski, "Dynamic Load Balancing in Parallel Discrete Event Simulation for Spatially Explicit Problems," *Proc. PADS'98 12th Workshop Parallel and Distributed Simulation,* pp. 46-53, 1998.
[16] M. Doar, "A Better Model for Generating Test Networks," *Proc. Globecom '96,* Nov. 1996.
[17] A.B. Downey, "Using Pathchar to Estimate Internet Link Characteristics," *Measurement and Modeling of Computer Systems,* pp. 222-223, 1999.
[18] J.E. Flaherty, R.M. Loy, C. Özturan, M.S. Shephard, B.K. Szymanski, J.D. Teresco, and L.H. Ziantz, "Parallel Structures and Dynamic Load Balancing for Adaptive Finite Element Computation," *Applied Numerical Math.,* vol. 26, nos. 1-2, pp. 241-263, 1997.
[19] J.E. Flaherty, R.M. Loy, M.S. Shephard, B.K. Szymanski, J.D. Teresco, and L.H. Ziantz, "Adaptive Local Refinement with Octree Load Balancing for the Parallel Solution of Three-Dimensional Conservation Laws," *J. Parallel and Distributed Computing,* vol. 47, no. 2, pp. 139-152, 1997.
[20] *The Grid: Blueprint for a New Computing Infrastructure.* I. Foster and C. Kesselman, eds., Morgan-Kaufmann, 1999.
[21] M.R. Garey and D.S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1991.
[22] Y.F. Hu and R.J. Blake, "Load Balancing for Unstructured Mesh Applications," *Parallel and Distributed Computing Practices,* vol. 2, no. 3, 1999.
[23] S. Ichikawa and S. Yamashita, "Static Load Balancing of Parallel PDE Solver for Distributed Computing Environment," *Proc. 13th Int'l Conf. Parallel and Distributed Computing Systems,* pp. 399-405, 2000.
[24] M. Kaddoura, S. Ranka, and A. Wang, "Array Decomposition for Nonuniform Computational Environments," *J. Parallel and Distributed Computing,* vol. 36, pp. 91-105, 1996.
[25] A. Kalinov and A. Lastovetsky, "Heterogeneous Distribution of Computations while Solving Linear Algebra Problems on Networks of Heterogeneous Computers," *Proc. Conf. High-Performance Computing and Networking (HPCN Europe),* pp. 191-200, 1999.
[26] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," *Proc. ACM 2002 Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM),* pp. 89-102, 2002.
[27] A. Legrand, H. Renard, Y. Robert, and F. Vivien, "Load-Balancing Iterative Computations in Heterogeneous Clusters with Shared Communication Links," Technical Report RR-2003-23, LIP, ENS Lyon, France, also available as INRIA Research Report 4800, Apr. 2003.
[28] M. Nibhanupudi and B. Szymanski, "BSP-Based Adaptive Parallel Processing," *High Performance Cluster Computing. Volume 1: Architecture and Systems,* R. Buyya, ed., pp. 702-721, Prentice-Hall, 1999.
[29] D.M. Nicol and P.F. Reynolds Jr., "Optimal Dynamic Remapping of Data Parallel Computations," *IEEE Trans. Computers,* vol. 39, no. 2, pp. 206-219, 1990.
[30] D.M. Nicol and J.H. Saltz, "Dynamic Remapping of Parallel Computations with Varying Resource Demands," *IEEE Trans. Computers,* vol. 37, no. 9, pp. 1073-1087, 1988.
[31] H. Renard, Y. Robert, and F. Vivien, "Static Load-Balancing Techniques for Iterative Computations on Heterogeneous Clusters," *Proc. Euro-Par'03: Parallel Processing,* pp. 148-159, 2003.
[32] B.A. Shirazi, A.R. Hurson, and K.M. Kavi, *Scheduling and Load Balancing in Parallel and Distributed Systems.* IEEE Computer Science Press, 1995.
[33] A.S. Tanenbaum, *Computer Networks.* Prentice Hall, 2003.
[34] A.G. Taylor and A.C. Hindmarsh, "User Documentation for KINSOL, a Nonlinear Solver for Sequential and Parallel Computers," Technical Report UCRL-ID-131185, Lawrence Livermore Nat'l Laboratory, July 1998.
[35] J. Watts and S. Taylor, "A Practical Approach to Dynamic Load Balancing," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 93, pp. 235-248, 1998.
[36] M.-Y. Wu, "On Runtime Parallel Scheduling for Processor Load Balancing," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 2, pp. 173-186, 1997.



Arnaud Legrand received the PhD degree from École normale supérieure de Lyon in 2003. He is currently a postdoctoral researcher in the LIP laboratory at ENS Lyon. He is mainly interested in parallel algorithm design for heterogeneous platforms and in scheduling techniques.

Hélène Renard is currently a PhD student in the Computer Science Laboratory LIP at ENS Lyon. She is mainly interested in parallel algorithm design for heterogeneous platforms and in load-balancing techniques.

Yves Robert received the PhD degree from the Institut National Polytechnique de Grenoble in January 1986. He is currently a full professor in the Computer Science Laboratory LIP at ENS Lyon. He is the author of four books, 85 papers published in international journals, and 110 papers published in international conferences. His main research interests are scheduling techniques and parallel algorithms for clusters and grids. He is a senior member of IEEE and the IEEE Computer Society, and serves as an associate editor of *IEEE Transactions on Parallel and Distributed Systems*.

Frédéric Vivien received the PhD degree from École normale supérieure de Lyon in 1997. From 1998 to 2002, he was an associate professor at Louis Pasteur University of Strasbourg. He spent the year of 2000 working in the Computer Architecture Group of the MIT Laboratory for Computer Science. He is currently a full reseacher at INRIA. His main research interests are scheduling techniques, parallel algorithms for clusters and grids, and automatic compilation/parallelization techniques.