Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

Load-balancing iterative computations in heterogeneous clusters with shared communication links

Arnaud Legrand,
Hélène Renard,
Yves Robert,
Frédéric Vivien

April 2003

Research Report N° 2003-23

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

INRIA

Load-balancing iterative computations in heterogeneous clusters with shared communication links

Arnaud Legrand, Hélène Renard, Yves Robert, Frédéric Vivien

April 2003

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, and data distribution schemes.

Keywords: Heterogeneous clusters, load-balancing, shared communication links, complexity.

Résumé

Ce rapport est consacré à l'application d'algorithmes sur plateformes hétérogènes. Les données sont réparties sur l'ensemble des ressources, qui sont organisées en anneau virtuel. À chaque itération, les calculs indépendants sont tranmis en parallèle et les communications ont lieu entre les ressources consécutives de l'anneau. Le problème est de déterminer comment partitionner les données et comment les répartir pour que le temps total d'exécution soit minimal. Une difficulté majeure est d'inclure un anneau de ressources dans un réseau n'étant pas forcément un graphe complet, de telle sorte que certains liens de communication soient partagés par plusieurs couples de ressources. Nous avons démontré un résultat de complexité qui établit la difficulté de ce problème, et nous proposons une heuristique pratique qui prouve l'efficacité de l'application, du routage et de la distribution de données.

Mots-clés: Ressources hétérogènes, équilibrage de charge, liens de communication partagés, complexité.

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 uni-dimensional partitioning of the data, and to map it onto a uni-dimensional ring of processors: more general data partitionings, such as two-dimensional, recursive, or even arbitrary slicings into rectangles, could be considered. But uni-dimensional 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 cluster, composed of different-speed processors that communicate through links of different bandwidths. On the architecture side, the problem is twofold: (i) select the processors that will participate in the solution and decide for their ordering, that will represent the arrangement into a ring; (ii) 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 clusters 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 fraction 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.

The rest of the paper is organized as follows. The next section (Section 2) is devoted to the precise and formal specification of the previous optimization problem, which we denote as SharedRing; we discuss several variations, depending upon the assumptions made for routing communications in the network. We also work out a small-size example in Section 2.4. Next, in Section 3, we state a complexity result: we show that the decision problem associated to SharedRing is NP-complete. After the proof of this result, Section 4 deals with the design of polynomial-time heuristics to solve the SharedRing optimization problem. We report some experimental data in Section 5. We survey related work in Section 6. Finally, we state some concluding remarks in Section 7.

2 Framework

In this section, we start with the model used for the platform graph (Section 2.1), and we formally state the optimization problem to be solved (Section 2.2). We discuss several variants of the model in Section 2.3. Finally, we work out a toy example (Section 2.4) to illustrate the various constraints to satisfy during the construction of the solution ring.

2.1 Modeling the platform graph

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 L/b_e time-units to transfer a single message of size L from P_i to P_j using link e. When there are several messages sharing the link, each of them receives a fraction (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 fractions of the bandwidth that are allocated to the messages can be freely determined by the user, the only rule if that the sum of all these fractions 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 [28].

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 L from P_i to P_j, along a path composed of k edges $e_1, e_2, \dots, e_k$. Along each edge e_m, the message will be allocated a fraction 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 L/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: computations

Let W be the total size of the work to be performed at each step of the algorithm. Processor P_i will accomplish a share $\alpha_i.W$ 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 do 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 $\alpha_i W$, each active processor P_i sends a message of fixed length H (typically some boundary data) to its successor. To illustrate the relationship between W and H, we can view the original data matrix as a rectangle composed of W columns of height H, so that one single column is exchanged between any pair of consecutive processors in the ring (but clearly, the parameter H can represent any fixed volume of communication).

Let $\mathrm{succ}(i)$ and $\mathrm{pred}(i)$ denote the successor and the predecessor of P_i in the virtual ring. There is a communication path $\mathcal{S}_i$ ($\mathcal{S}$ stands for "successor") from P_i to $P_{\mathrm{succ}(i)}$ in the network: let $s_{i,m}$ be the fraction 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 $c_{i,\mathrm{succ}(i)} = \frac{1}{\min_{e_m \in \mathcal{S}_i} s_{i,m}}$: then P_i requires $H.c_{i,\mathrm{succ}(i)}$ time-units to send its message of size H to its successor $P_{\mathrm{succ}(i)}$.

Similarly, we define the communication path $\mathcal{P}_i$ ($\mathcal{P}$ for "predecessor") from P_i to $P_{\mathrm{pred}(i)}$ in the network; $p_{i,m}$ is the fraction of the bandwidth b_{e_m} of the physical link e_m that has been allocated to the path $\mathcal{P}_i$, and $c_{i,\mathrm{pred}(i)} = \frac{1}{\min_{e_m \in \mathcal{P}_i} p_{i,m}}$. Then P_i requires $H.c_{i,\mathrm{pred}(i)}$ time-units to send its message of size H to its predecessor $P_{\mathrm{succ}(i)}$.

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:

$$T_{\mathrm{step}} = \max_{1 \le i \le p} \mathbb{I}\{i\}[\alpha_i.W.w_i + H.(c_{i,\mathrm{pred}(i)} + c_{i,\mathrm{succ}(i)})]$$

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 The SharedRing optimization problem

We state the optimization problem as follows:

Definition 1 (SharedRing(p,w_i,E,b_{e_m},W,H)). *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 W and the communication volume H at each step, minimize*

$$T_{step} = \min_{1 \le q \le p} \left\{ \min_{\substack{\sigma \in \Theta_{q,p} \\ \sum_{i=1}^{q} \alpha_{\sigma(i)} = 1}} \max_{1 \le i \le q} \left(\alpha_{\sigma(i)}.W.w_{\sigma(i)} + H.(c_{\sigma(i),\sigma(i-1 \bmod q)} + c_{\sigma(i),\sigma(i+1 \bmod q)}) \right) \right\} \quad (1)$$

In Equation 1, $\Theta_{q,p}$ 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. 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 $P_{\mathrm{succ}(\sigma(i))} = P_{\sigma(i+1 \bmod q)}$ and the path $\mathcal{P}_i$ from $P_{\sigma(i)}$ to its predecessor $P_{\mathrm{pred}(\sigma(i))} = P_{\sigma(i-1 \bmod q)}$, for $1 \le i \le q$. For each link e_m in the interconnection network, let $s_{\sigma(i),m}$ (resp. $p_{\sigma(i),m}$) be the fraction of the bandwidth b_{e_m} that is allocated to the path $\mathcal{S}_{\sigma(i)}$ (resp. $\mathcal{P}_{\sigma(i)}$). We have the equations:

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

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).

From Equation 1, we see that the optimal solution will involve all processors as soon as the ratio $\frac{W}{H}$ is large enough: in that case, the impact of the communications becomes small in front of the cost of the computations, and these computations should be distributed to all resources. But even in that case, we still have to decide how to arrange the processors along a ring, to construct the communicating paths, to assign bandwidths ratios and finally to allocate data chunks. Extracting the "best" ring seems to be a difficult combinatorial problem. Before assessing this result (see Section 3), we discuss some variants (Section 2.3) and we work out a small-size example (Section 2.4).

To conclude this section, we point out that this framework is more general than iterative algorithms: in fact, our approach applies to any problem where independent computations are distributed over heterogeneous resources. The only hypothesis is that the communication volume is the same between adjacent processors, regardless of their relative workload.

2.3 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.

Bi-directional 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 fraction of the bandwidth, so that the total available of the link bandwidth is not exceeded. In other words, we assign a bandwidth fraction 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

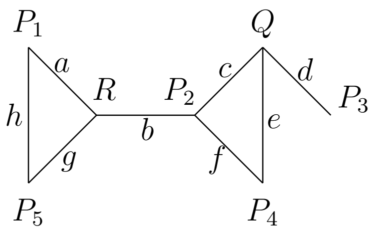

Figure 1: A small-size cluster.

$\sum f_{\text{path}} \leq b$; 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 multi-graph 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 fraction $f_{\text{path}} = b$ to each communication path requesting the link, regardless of their number: in other words, we replace the constraint $\sum f_{\text{path}} \leq b$ by $f_{\text{path}} \leq b$ for each path using the link.

2.4 Toy example

Consider the heterogeneous cluster represented in Figure 1. There are 7 processors and 8 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 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 $\mathcal{S}_1 = \{a, b\}$. But for the path from P_2 to P_1, we may use links b, g and h, so that $\mathcal{P}_2 = \{b, g, h\}$. 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:

Link a: $s_{1,a} \leq b_a$

Link b: $s_{1,b} + s_{4,b} + p_{2,b} + p_{5,b} \leq b_b$

Link c: $s_{2,c} \leq b_c$

Link d: $s_{2,d} + s_{3,d} + p_{3,d} + p_{4,d} \leq b_d$

Link e: $s_{3,e} + p_{3,e} + p_{4,e} \leq b_e$

Link f: $s_{4,f} + p_{3,f} + p_{5,f} \leq b_f$

Link g: $s_{4,g} + p_{2,g} + p_{5,g} \leq b_g$

Link h: $s_{5,h} + p_{1,h} + p_{2,h} \leq b_h$

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 2 explicits the whole system of (in)equations which is quadratic in the unknowns α_i, $s_{i,j}$ and $p_{i,j}$ [1].

$$\text{minimize} \quad \max_{1\leq i\leq 5}\left(\alpha_i.W.w_i + H.(c_{i,i-1}+c_{i,i+1})\right) \quad \text{subject to}$$

$$\left\{\begin{array}{lll}
\sum_{i=1}^{5}\alpha_i = 1 & & \\
s_{1,a} \leq b_a & s_{1,b}+s_{4,b}+p_{2,b}+p_{5,b} \leq b_b & s_{2,c} \leq b_c \\
s_{2,d}+s_{3,d}+p_{3,d}+p_{4,d} \leq b_d & s_{3,e}+p_{3,e}+p_{4,e} \leq b_e & s_{4,f}+p_{3,f}+p_{5,f} \leq b_f \\
s_{4,g}+p_{2,g}+p_{5,g} \leq b_g & s_{5,h}+p_{1,h}+p_{2,h} \leq b_h & \\
s_{1,a}.c_{1,2} \geq 1 & s_{1,b}.c_{1,2} \geq 1 & p_{1,h}.c_{1,5} \geq 1 \\
s_{2,c}.c_{2,3} \geq 1 & s_{2,d}.c_{2,3} \geq 1 & p_{2,b}.c_{2,1} \geq 1 \\
p_{2,g}.c_{2,1} \geq 1 & p_{2,h}.c_{2,1} \geq 1 & s_{3,d}.c_{3,4} \geq 1 \\
s_{3,e}.c_{3,4} \geq 1 & p_{3,d}.c_{3,2} \geq 1 & p_{3,e}.c_{3,2} \geq 1 \\
p_{3,f}.c_{3,2} \geq 1 & s_{4,f}.c_{4,5} \geq 1 & s_{4,b}.c_{4,5} \geq 1 \\
s_{4,g}.c_{4,5} \geq 1 & p_{4,e}.c_{4,3} \geq 1 & p_{4,d}.c_{4,3} \geq 1 \\
s_{5,h}.c_{5,1} \geq 1 & p_{5,g}.c_{5,4} \geq 1 & p_{5,b}.c_{5,4} \geq 1 \\
p_{5,f}.c_{5,4} \geq 1 & &
\end{array}\right. \tag{2}$$

To build up Equation 2, 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 too... Not surprisingly, the decision problem associated to the SharedRing optimization problem is NP-complete, as shown in Section 3.

3 Complexity

The decision problem associated to the SharedRing optimization problem is the following:

Definition 2 (SharedRingDec(p,w_i,E,b_{e_m},W,H,K)). *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 W and the communication volume H at each step, and given a time bound K, is it possible to find a subset of $q \leq p$ processors, a one-to one mapping $\sigma : [1..q] \rightarrow [1..p]$, $2q$ communicating paths $\mathcal{S}_i$ and $\mathcal{P}_i$ such that no total link bandwidth is exceeded, and nonnegative rational numbers α_i with $\sum_{i=1}^{q}\alpha_{\sigma(i)} = 1$, such that $T_{step} \leq K$, where T_{step} is given by Equation 1?*

The following result states the intrinsic difficulty of the problem:

[1] We did not express in Equation 2 the inequations stating that all the unknowns are nonnegative.

Theorem 1. SharedRingDec*(p,w_i,E,b_{e_m},W,H,K) is NP-complete.*

Proof. Obviously, SharedRingDec belongs to NP. To prove its completeness, we use a reduction from HamCycle, the Hamiltonian Cycle Problem, which is NP-complete [19]. Consider an arbitrary instance $\mathcal{I}_1$ of HamCycle: 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 SharedRingDec: we let $p = |V_h|$ (assume $p \geq 2$ without loss of generality), and we define a complete interconnection graph $G = (P, E)$. In G all edges are unidirectional: in particular, from each edge of E_h we derive two edges in E. The bandwidths of the edges in E are given by

$$b_e = \begin{cases} 1/\varepsilon & \text{if } e \text{ is derived from } e \in E_h \\ 1/2 & \text{otherwise} \end{cases}$$

where $0 < \varepsilon < \frac{1}{2}$ is a small constant. We let $W = H = 1$ 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 cycle. We use the edges of this path to build the ring. All processors are involved, and we let $\alpha_i = 1/p$ for $1 \leq i \leq p$. From now on, indices are taken modulo p. We re-index processor and edges so that the Hamiltonian path is $e_1, e_2, \ldots, e_p$ where e_i connects P_i to P_{i+1}; from the construction of G, there is a path in the reverse direction, i.e. edges e_{p+i} from P_{i+1} to P_i, that go in the opposite direction of the e_i. For $1 \leq i \leq p$, $\mathcal{S}_i$ reduces to e_i; we let $s_{i,i} = 1/\varepsilon$ and $s_{i,m} = 0$ for $m \neq i$. Similarly, $\mathcal{P}_i$ reduces to e_{p+i-1} (except $\mathcal{P}_1$ which reduces to e_{2p}); we let $p_{i,p+i-1} = 1/\varepsilon$ and $p_{i,m} = 0$ for $m \neq p + i - 1$ (except for $i = 1$: $p_{i,2p} = 1/\varepsilon$ and $p_{1,m} = 0$ for $m \neq 2p$). Each edge e_i is used once, and its total bandwidth $1/\varepsilon$ is not exceeded. 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 were 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 the cost of the path including that edge would be at least 2, and $T_{\text{step}} \geq H.2 = 2 > K$, 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 $\frac{1}{q}.W.p > 1$, which would imply that $T_{\text{step}} > K$, because the cost of any communicating path is at least ε. Finally, $q = p$, and the edges of the solution ring define a Hamiltonian cycle in G_h, thereby providing a solution to $\mathcal{I}_1$. □

4 Heuristics

In this section we describe a polynomial-time heuristic to solve the SharedRing optimization problem to construct a solution ring. We describe the heuristic in three steps: (i) the greedy algorithm used to construct a solution ring; (ii) the strategy used to assign bandwidth fractions during the construction; and (iii) a final refinement.

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,

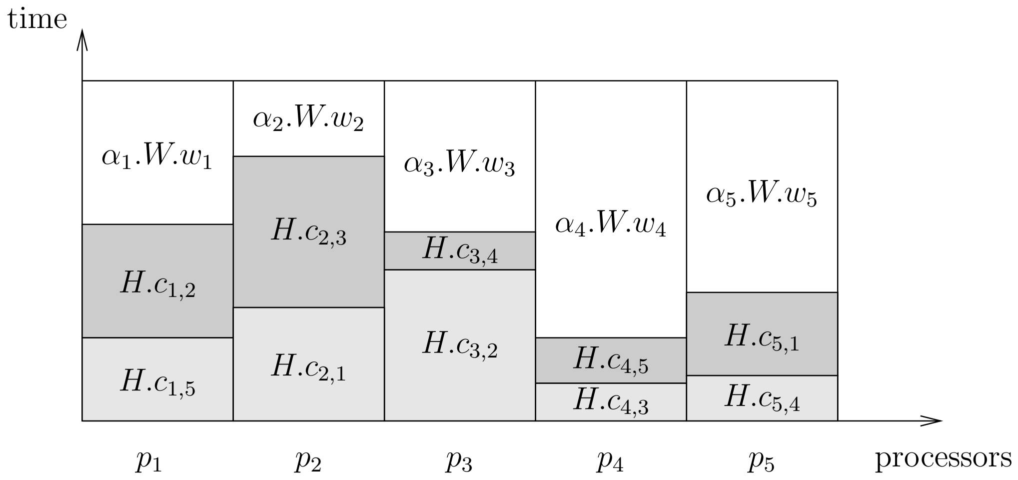

Figure 2: Summary of computation and communication times with $q = 5$ processors.

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 [2]. Hence (see Figure 2 for an illustration) we have

$$T_{\text{step}} = \alpha_i.W.w_i + H.(c_{i,i-1} + c_{i,i+1}) \tag{3}$$

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

$$T_{\text{step}} = W.w_{\text{cumul}} \left(1 + \frac{H}{W} \sum_{i=1}^{q} \frac{c_{i,i-1} + c_{i,i+1}}{w_i} \right) \tag{4}$$

We will use Equation 4 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 fractions (explained in Section 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

[2]Here we implicitly make the assumption that the total workload can be arbitrarily partitioned. Therefore we are using the "divisible load" framework. See Section 6 for pointers to papers on the divisible load theory.

- 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 $i = \sigma(s)$ 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 Figure 11 below).

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 fraction 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 re-inject in the network the bandwidths fractions 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 fractions minimizing Equation 4 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 $O(p+E)$.

4.3 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 fractions 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 fractions and communication costs for these paths, and a feasible value of T_{step}.

Because the heuristic could appear over-simplistic, we have implemented two variants aimed at refining the solution. The idea for the two variants is to keep everything but the bandwidth fractions 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 fractions 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 re-evaluate bandwidth fractions, 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 fractions 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 2.4, once we have a ring and communicating paths, the program to minimize T_{step} is quadratic in the unknowns α_i, $s_{i,j}$ and $p_{i,j}$. We use the KINSOL library [29] to solve it.

5 Experimental results

5.1 Platform description

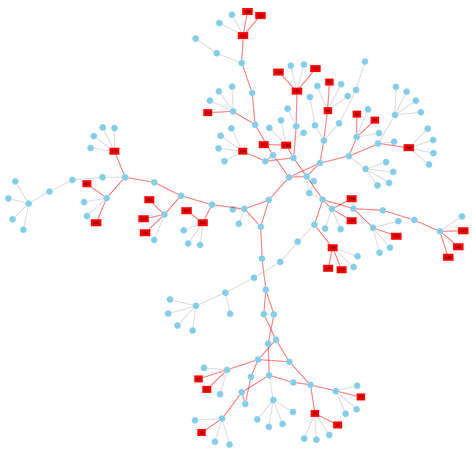

Figure 3: First platform. Boxed nodes are selected machine nodes. There are 37 selected machine nodes, connected through 47 routers and 91 communication links.

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. The platforms are generated by selecting a fraction of the LAN nodes, referred to as the boxed nodes in Figures 3 and 4. These boxed nodes are the selected machine nodes that are used in the computing process. They represent about 30% 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,

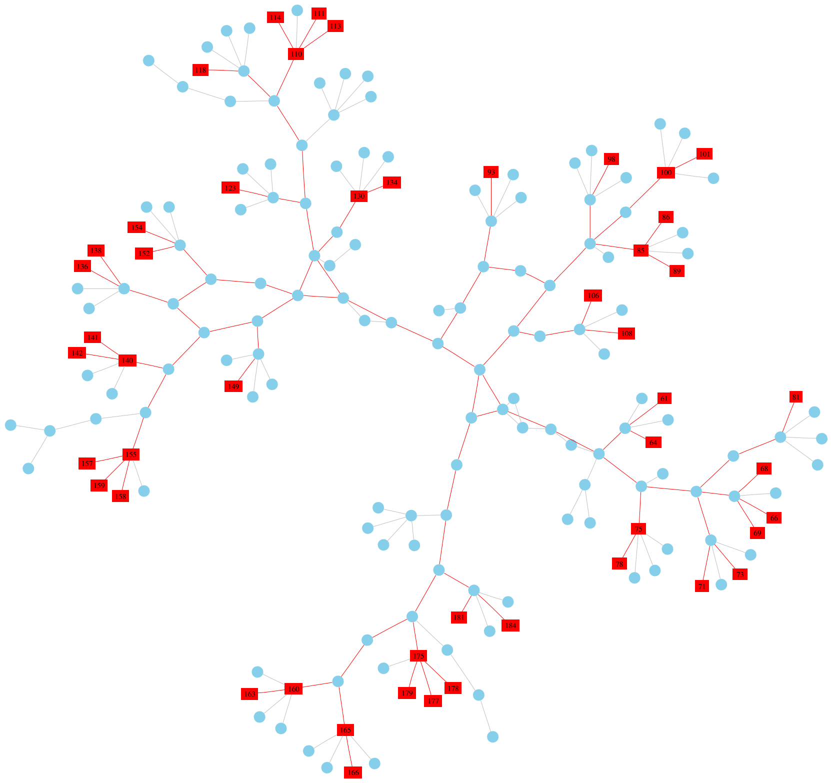

Figure 4: Second platform. Boxed nodes are selected machine nodes. There are 70 selected machine nodes, connected through 103 routers and 182 communication links.

Athlon 1.4GHz, Pentium 4 1.7GHz, ...). The capacities of the edges are assigned using the classification of the Tiers generator (local LAN link, LAN/MAN link, MAN/WAN link,...). 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, Orsay), in the USA (Knoxville, San Diego, Argonne) and in Japan (Nagoya, Tokyo). The second platform is (roughly) twice larger than the first platform.

5.2 Results

In Figures 5 and 6, we plot the number of processors used in the solution ring. As expected, the size of the ring decreases as the ratio H/W increases: additional computational power does not pay off the communication overhead.

In Figures 7 to 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 fewer processors as the ratio increases.

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: the best gain is 3%. This is good news for the plain version of the heuristic, which turns out to be both low-cost and efficient.

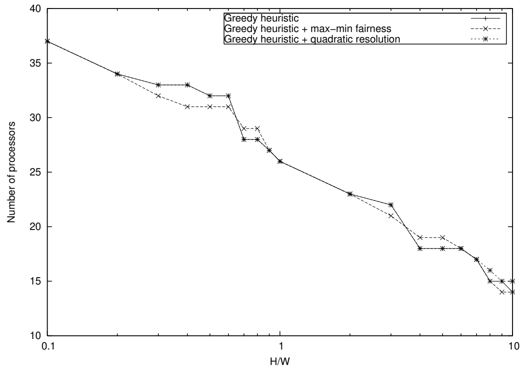

Figure 5: First platform. Size of the optimal ring as a function of the ratio H/W.

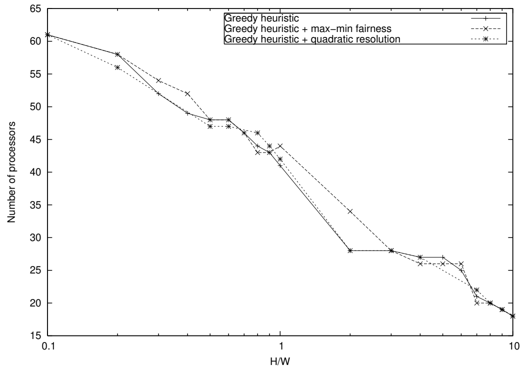

Figure 6: Second platform. Size of the optimal ring as a function of the ratio H/W.

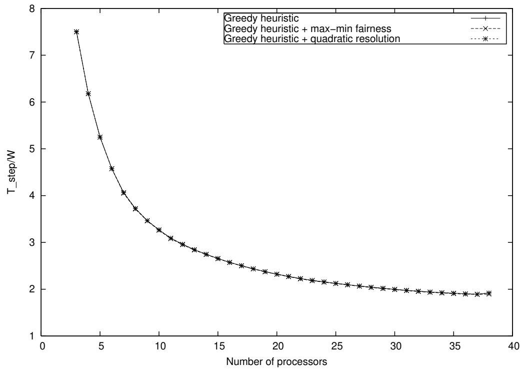

Figure 7: First platform. Value of T_{step} normalized by W as a function of the size of the solution ring, with a low communication-to-computation ratio: $H/W = 0.1$.

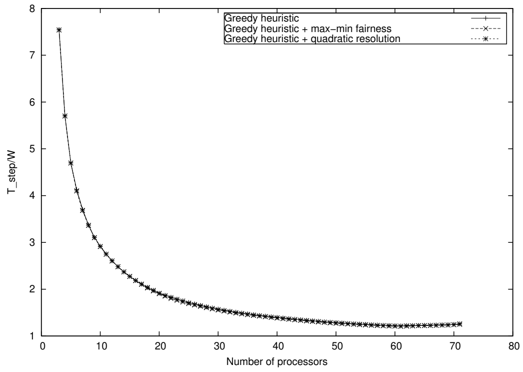

Figure 8: Second platform. Value of T_{step} normalized by W as a function of the size of the solution ring, with a low communication-to-computation ratio: $H/W = 0.1$.

6 Related work

Load balancing strategies have been widely studied, both for homogeneous platforms (see the collection of papers [27]) 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 calls for periodic re-mapping 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 [25, 24, 30, 20] propose a mapping policy which dynamically minimizes system degradation (including the cost of remapping) for each computation step. Other papers [31, 15] advocate local schemes where data is exchanged only between neighbor processors. Generally speaking, there is a challenge in determining a trade-off between

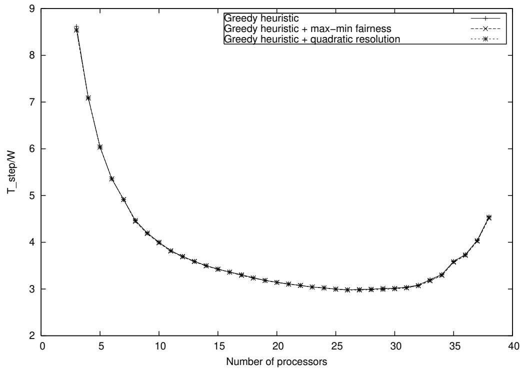

Figure 9: First platform. Value of T_{step} normalized by W as a function of the size of the solution ring, with a balanced communication-to-computation ratio: $H/W = 1$.

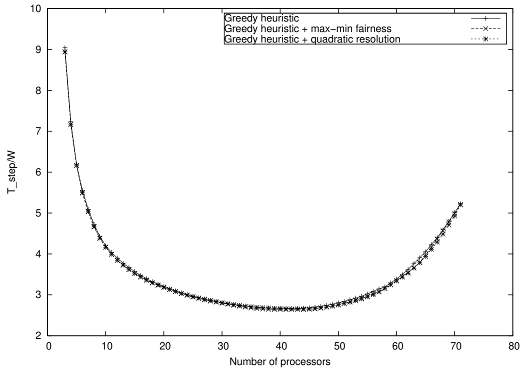

Figure 10: Second platform. Value of T_{step} normalized by W as a function of the size of the solution ring, with a balanced communication-to-computation ratio: $H/W = 1$.

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.

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 [20]. On the same subject, see also the paper by Ichikawa and Yamashita [21]. Several authors have dealt with the static implementation of linear algebra kernels on heterogeneous platforms. Matrix multiplication has been studied by [23, 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, Ranka and Wang [22], and Beaumont et al. [3]. The main conclusions of these papers are drawn for three kinds of problems:

- Distributing independent chunks of work to uni-dimensional (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, 22, 3] that allow for a good load-balancing but are much more difficult to implement

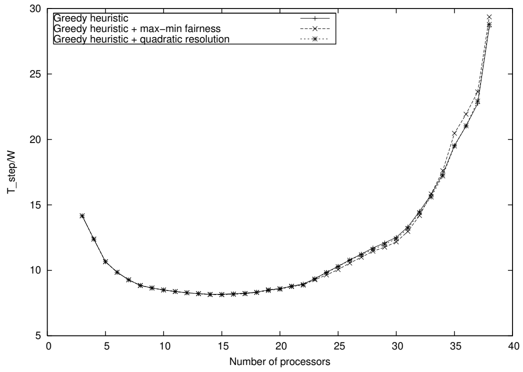

Figure 11: First platform. Value of T_{step} normalized by W as a function of the size of the solution ring, with a high communication-to-computation ratio: $H/W = 10$.

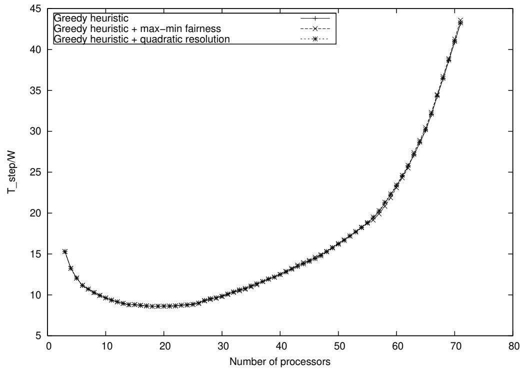

Figure 12: Second platform. Value of T_{step} normalized by W as a function of the size of the solution ring, with a high communication-to-computation ratio: $H/W = 10$.

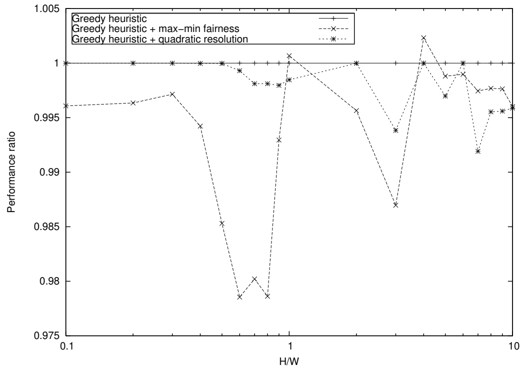

Figure 13: First platform. Impact of the refinements on the quality of the solution.

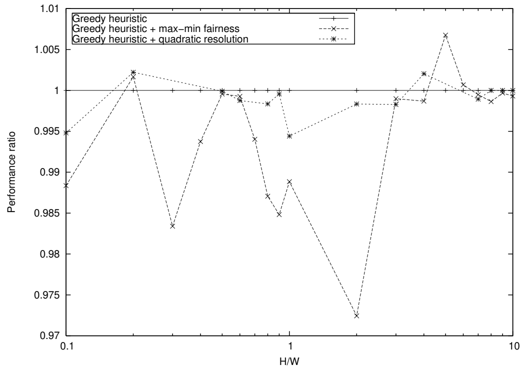

Figure 14: Second platform. Impact of the refinements on the quality of the solution.

In this perspective, this paper shows that the first problem, i.e. distributing independent chunks of work to uni-dimensional 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.

Finally, note that a simplified version of the SharedRing optimization problem is considered in [26]: in this report, we assume that the target communication network is a clique (i.e. fully connected), hence there is no need to share links when constructing the solution ring.

7 Conclusion

The major limitation to programming heterogeneous platforms arises from the additional difficulty of balancing the load. Data and computations are not evenly distributed to processors. Minimizing communication overhead becomes a challenging task.

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 [18], may well enforce the use of dynamic schemes.

In this paper, the major emphasis was towards a realistic modeling of concurrent communications in cluster networks. One major result is the NP-completeness of the SharedRing problem. Rather than the proof, the result itself is interesting, because it provides yet another evidence of the intrinsic difficulty of designing heterogeneous algorithms. But this negative result should not be over-emphasized. 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.

References

[1] J. Barbosa, J. Tavares, and A. J. Padilha. Linear algebra algorithms in a heterogeneous cluster of personal computers. In *9th Heterogeneous Computing Workshop (HCW'2000)*, pages 147–159. IEEE Computer Society Press, 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*, 50(10):1052–1070, 2001.

[3] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. Matrix multiplication on heterogeneous platforms. *IEEE Trans. Parallel Distributed Systems*, 12(10):1033–1051, 2001.

[4] F. Berman. High-performance schedulers. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 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 Computer Society Press, 1996.

[7] V. Bharadwaj, D. Ghose, and T. G. Robertazzi. A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):7–18, January 2003.

[8] R. P. Brent. The LINPACK Benchmark on the AP1000: Preliminary Report. In *CAP Workshop 91*. Australian National University, 1991. Website http://www.netlib.org/linpack/.

[9] R. Buyya. *High Performance Cluster Computing. Volume 1: Architecture and Systems*. Prentice Hall PTR, Upper Saddle River, NJ, 1999.

[10] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997. Available at http://citeseer.nj.nec.com/calvert97modeling.html.

[11] M. Cierniak, M.J. Zaki, and W. Li. Compile-time scheduling algorithms for heterogeneous network of workstations. *The Computer Journal*, 40(6):356–372, 1997.

[12] M. Cierniak, M.J. Zaki, and W. Li. Customized dynamic load balancing for a network of workstations. *Journal of Parallel and Distributed Computing*, 43:156–162, 1997.

[13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[14] P. E. Crandall and M. J. Quinn. Block data decomposition for data-parallel programming on a heterogeneous workstation network. In *2nd International Symposium on High Performance Distributed Computing*, pages 42–49. IEEE Computer Society Press, 1993.

[15] E. Deelman and B.K. Szymanski. Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. In *PADS'98, 12th Workshop on Parallel and Distributed Simulation*, pages 46–53. IEEE Computer Society Press, 1998.

[16] M. Doar. A better model for generating test networks. In *Proceedings of Globecom '96*, November 1996. Available at http://citeseer.nj.nec.com/doar96better.html.

[17] Allen B. Downey. Using pathchar to estimate internet link characteristics. In *Measurement and Modeling of Computer Systems*, pages 222–223, 1999. Available at http://citeseer.nj.nec.com/downey99using.html.

[18] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1991.

[20] Y.F. Hu and R.J. Blake. Load balancing for unstructured mesh applications. *Parallel and Distributed Computing Practices*, 2(3), 1999.

[21] S. Ichikawa and S. Yamashita. Static load balancing of parallel PDE solver for distributed computing environment. In *PDCS'2000, 13th Int'l Conf. Parallel and Distributed Computing Systems*, pages 399–405. ISCA Press, 2000.

[22] M. Kaddoura, S. Ranka, and A. Wang. Array decomposition for nonuniform computational environments. *Journal of Parallel and Distributed Computing*, 36:91–105, 1996.

[23] A. Kalinov and A. Lastovetsky. Heterogeneous distribution of computations while solving linear algebra problems on networks of heterogeneous computers. In P. Sloot, M. Bubak, A. Hoekstra, and B. Hertzberger, editors, *HPCN Europe 1999*, LNCS 1593, pages 191–200. Springer Verlag, 1999.

[24] D.M. Nicol and Jr P.F. Reynolds. Optimal dynamic remapping of data parallel computations. *IEEE Trans. Computers*, 39(2):206–219, 1990.

[25] D.M. Nicol and J.H. Saltz. Dynamic remapping of parallel computations with varying resource demands. *IEEE Trans. Computers*, 37(9):1073–1087, 1988.

[26] H. Renard, Y. Robert, and F. Vivien. Static load-balancing techniques for iterative computations on heterogeneous clusters. Technical Report RR-2003-12, LIP, ENS Lyon, France, February 2003.

[27] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Science Press, 1995.

[28] A.S. Tanenbaum. *Computer Networks*. Prentice Hall, 2003.

[29] 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 National Laboratory, July 1998.

[30] J. Watts and S. Taylor. A practical approach to dynamic load balancing. *IEEE Trans. Parallel and Distributed Systems*, 9(93):235–248, 1998.

[31] M-Y. Wu. On runtime parallel scheduling for processor load balancing. *IEEE Trans. Parallel and Distributed Systems*, 8(2):173–186, 1997.