Static Load-Balancing Techniques for Iterative Computations on Heterogeneous Clusters

Hélène Renard, Yves Robert, and Frédéric Vivien

LIP, UMR CNRS-INRIA-UCBL 5668
ENS Lyon, France
{Helene.Renard | Yves.Robert | Frederic.Vivien}@ens-lyon.fr

Abstract. This paper is devoted to static load balancing techniques for mapping iterative algorithms onto heterogeneous clusters. The application data is partitioned over the processors. At each iteration, independent calculations are carried out in parallel, and some communications take place. 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. We establish a complexity result that assesses the difficulty of this problem, and we design practical heuristics that provide efficient distribution schemes.

1 Introduction

In this paper, we investigate static load balancing techniques for iterative algorithms that operate on a large collection of application data. The application data 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 to signal processing, and image processing algorithms. The target architecture is a fully heterogeneous cluster, composed of different-speed processors that communicate through links of different capacities. The question is to determine the best partitioning of the application data. The difficulty comes from the fact that both the computation and communication capabilities of each resource must be taken into account.

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. 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.

We assume that the target computing platform can be modeled as a complete graph:

- Each vertex in the graph models a computing resource P_i, and is weighted by the relative cycle-time of the resource. Of course the absolute value of the time-unit is application-dependent, what matters is the relative speed of one processor versus the other.
- Each edge models a communication link, and is weighted by the relative capacity of the link. Assuming a complete graph means that there is a *virtual* communication link between any processor pair P_i and P_j. Note that this link does not necessarily need to be a direct physical link. There may be a path of physical communication links from P_i to P_j: if the slowest link in the path has maximum capacity $c_{i,j}$, then the weight of the edge will be $c_{i,j}$.

We suppose that the communication capacity $c_{i,j}$ is granted between P_i and P_j (so if some communication links happen to be physically shared, we assume that a fraction of the total capacity, corresponding to the inverse of $c_{i,j}$, is available for messages from P_i to P_j). This assumption of a fixed capacity link between any processor pair makes good sense for interconnection networks based upon high-speed switches like Myrinet [2].

Given these hypotheses, the optimization problem that we want to solve is the following: how to slice the matrix data into chunks, and assign these chunks to the processors, so that the total execution time for a given sweep step, namely a computation followed by two neighbor communications, is minimized? We have to perform resource selection, because there is no reason *a priori* that all available processors will be involved in the optimal solution (for example some fast computing processor may be left idle because its communication links with the other processors are too slow). Once some resources have been selected, they must be arranged along the best possible ring, which looks like a difficult combinatorial problem. Finally, once a ring has been set up, there remains to load-balance the workloads of the participating resources.

The rest of the paper is organized as follows. In Section 2, we formally state the previous optimization problem, which we denote as SLICERING. If the network is homogeneous (all links have same capacity), then SLICERING can be solved easily, as shown in Section 3. But in the general case, SLICERING turns out to be a difficult problem: we show in Section 4 that the decision problem associated to SLICERING is NP-complete, as could be expected from its combinatorial nature. We derive in Section 5 a formulation of the SLICERING problem in terms of an integer linear program, thereby providing a (costly) way to determine the optimal solution. In Section 6, we move to the design of polynomial-time heuristics, and we report some experimental data. We survey related work in Section 7, and we provide a brief comparison of static versus dynamic strategies. Finally, we state some concluding remarks in Section 8.

2 Framework

In this section, we formally state the optimization problem to be solved. As already said, the target computing platform is modeled as a complete 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 $S.w_i$ time-units to process a task of size S. Edges are labeled with communication costs: the time needed to transfer a message of size L from P_i to P_j is $L.c_{i,j}$, where $c_{i,j}$ is the capacity of the link, i.e. the inverse of its bandwidth. 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, $\tau_i + S.w_i$ for computations and $\beta_{i,j} + L.c_{i,j}$ for communications.

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 so large: the extra communications incurred by adding more processors may slow down the whole process, despite the increased cumulated speed.

We will arrange the participating processors along a ring (yet to be determined). After updating its data slice, each active processor P_i sends some boundary data to its neighbors: let $\mathrm{pred}(i)$ and $\mathrm{succ}(i)$ denote the predecessor and the successor of P_i in the virtual ring. Then P_i requires $H.c_{i,\mathrm{succ}(i)}$ time-units to send a message of size H to its successor, plus $H.c_{i,\mathrm{pred}(i)}$ to receive a message of same size from its predecessor. In most situations, we will have symmetric costs $(c_{i,j} = c_{j,i})$ but we do not make this assumption here. 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).

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

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

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, and to arrange them along a ring so that the total execution time per step is minimized. We formally state this optimization problem as follows:

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

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

Here $\mathcal{S}_{q,p}$ 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 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 smaller 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. Extracting the "best" ring out of the interconnection graph seems to be a difficult combinatorial problem. Before assessing this result (see Section 4), we deal with the much easier situation when the network is homogeneous (see Section 3).

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.

3 Homogeneous Networks

Solving the optimization problem, i.e. minimizing expression (1), 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.

Let us assume that $c_{i,j} = c$ for all i and j, where c is a constant. There are only two cases to consider: (i) only the fastest processor is active; (ii) 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 (i), we derive that $T_{\text{step}} = W.w_{\text{min}}$, where w_{min} is the smallest cycle-time. In the latter case (ii), 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 $\alpha_i.w_i = \text{Constant}$ for all i, with $\sum_{i=1}^{p} \alpha_i = 1$. We derive that $T_{\text{step}} = W.w_{\text{cumul}} + 2H.c$, where $w_{\text{cumul}} = \frac{1}{\sum_{i=1}^{p} \frac{1}{w_i}}$. We summarize these results:

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

$$T_{step} = \min \{W.w_{\min}, W.w_{cumul} + 2H.c\}$$

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

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 H, the fastest processor will do all the job for small values of W, namely $W \leq H.\frac{2c}{w_{\min} - w_{cumul}}$. Otherwise, for larger values of W, all processors should be involved.

4   Complexity

The decision problem associated to the SLICERING optimization problem is:

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

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

The following result states the intrinsic difficulty of the problem (see [9] for the proof):

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

5   ILP Formulation

When the network is heterogeneous, we face a complex situation: how to determine the number of processors that should take part to the computation already is a difficult question.

In this section, we express the solution to the SLICERING optimization problem, in terms of an Integer Linear Programming (ILP) problem. Of course the complexity of this approach may be exponential in the worst case, but it will provide useful hints to design low-cost heuristics. We start with the case where all processors are involved in the optimal solution. We extend the approach to the general case later on.

5.1   When All Processors Are Involved

Assume first that all processors are involved in an optimal solution. All the p processors require the same amount of time to compute and communicate: otherwise, we would slightly decrease the computing load of the last processor to
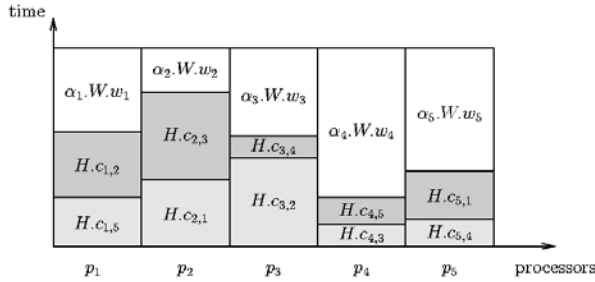
Fig. 1. Summary of computation and communication times with p processors.

complete its assignment (computations followed by communications) and assign extra work to another one. Hence (see Figure 1 for an illustration) we have

$$T_{\text{step}} = \alpha_i.W.w_i + H.(c_{i,i-1} + c_{i,i+1}) \tag{2}$$

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

$$\frac{T_{\text{step}}}{W.w_{\text{cumul}}} = 1 + \frac{H}{W} \sum_{i=1}^{p} \frac{c_{i,i-1} + c_{i,i+1}}{w_i}. \tag{3}$$

Therefore, T_{step} will be minimal when $\sum_{i=1}^{p} \frac{c_{i,i-1}+c_{i,i+1}}{w_i}$ is minimal. This will be achieved for the ring that corresponds to the shortest Hamiltonian cycle in the graph $G = (P, E)$, where each edge $e_{i,j}$ is given the weight $d_{i,j} = \frac{c_{i,j}+c_{j,i}}{w_i}$. Once we have this path, we derive T_{step} from Equation 3, and then we determine the load α_i of each processor using Equation 2.

To summarize, we have the following result:

Proposition 2. *When all processors are involved, finding the optimal solution is equivalent to solving the Traveling Salesman Problem (TSP) in the weighted graph* (P, E, d), $d_{i,j} = \frac{c_{i,j}+c_{j,i}}{w_i}$.

Of course we are not expecting any polynomial-time solution from this result, because the decision problem associated to the TSP is NP-complete [6] (even worse, because the distance d does not satisfy the triangle inequality, there is no polynomial-time approximation [1]), but this equivalence gives us two lines of action:

- For platforms of reasonable size, the optimal solution can be computed using an integer linear program that returns the optimal solution to the TSP
- For very large platforms, we can use well-established heuristics which approximate the solution to the TSP in polynomial time, such as the Lin-Kernighan heuristic [8,7].

The following result (see [9] for a proof) solves the problem whenever all processors are involved:

Proposition 3. *When all processors are involved, finding the optimal solution is equivalent to solving the following integer linear program:*

> **TSP integer linear programming formulation**
> MINIMIZE $\sum_{i=1}^{p} \sum_{j=1}^{p} d_{i,j}.x_{i,j}$,
> SUBJECT TO
> $$\begin{cases} (1) \ \sum_{j=1}^{p} x_{i,j} = 1 & 1 \leq i \leq p \\ (2) \ \sum_{i=1}^{p} x_{i,j} = 1 & 1 \leq j \leq p \\ (3) \ x_{i,j} \in \{0,1\} & 1 \leq i,j \leq p \\ (4) \ u_i - u_j + p.x_{i,j} \leq p - 1 & 2 \leq i,j \leq p, i \neq j \\ (5) \ u_i \ integer, u_i \geq 0 & 2 \leq i \leq p \end{cases}$$

5.2 General Case

How to extend the ILP formulation to the general case? For each possible value of q, $1 \leq q \leq p$, we will set up an ILP problem giving the optimal solution with exactly q participating resources. Taking the smallest solution among the p values returned by these ILP problems will lead to the optimal solution.

For a fixed value of q, $1 \leq q \leq p$, we use a technique similar to that of Section 5.1, but we need additional variables. Here is the ILP (see [9] for the proof that the solution is a ring of q participating resources):

q-ring integer linear programming formulation
MINIMIZE T,
SUBJECT TO

$$\begin{cases} (1) \ \alpha_i.w_i + \dfrac{H}{W} \sum_{j=1}^{p} (x_{i,j}c_{i,j} + x_{j,i}c_{j,i}) \leq T + \left(1 - \sum_{i=1}^{p} x_{j,i}\right).K & 1 \leq i \leq p \\ (2) \ \sum_{i=1}^{p} x_{i,j} = \sum_{i=1}^{p} x_{j,i} & 1 \leq j \leq p \\ (3) \ \sum_{i=1}^{p} x_{i,j} \leq 1 & 1 \leq j \leq p \\ (4) \ \sum_{i=1}^{p} \sum_{j=1}^{p} x_{i,j} = q & \\ (5) \ x_{i,j} \in \{0,1\} & 1 \leq i,j \leq p \\ (6) \ \sum_{i=1}^{p} y_i = 1 & \\ (7) \ -p.y_i - p.y_j + u_i - u_j + q.x_{i,j} \leq q - 1 & 1 \leq i,j \leq p, i \neq j \\ (8) \ y_i \in \{0,1\} & 1 \leq i \leq p \\ (9) \ u_i \ integer, u_i \geq 0 & 1 \leq i \leq p \end{cases}$$

where $K = \max_{1 \leq j \leq p} w_{i,j} + 2\frac{H}{W}.\max_{1 \leq j \leq p} c_{i,j}$. In any optimal solution of this linear program, $T = \frac{T_{\text{step}}}{W}$ and, because of the term $(1 - \sum_{i=1}^{p} x_{j,i}).K$, the constraints (1), for $1 \leq i \leq p$, induce constraints on T only for resources participating in the solution ring.

Proposition 4. *The SLICERING optimization problem can be solved by computing the solution of p integer linear programs, where p is the total number of resources.*

6   Heuristics and Experiments

After the previous theoretically-oriented results, we adopt a more practical approach in this section. We aim at deriving polynomial-time heuristics for solving the SLICERING optimization problem.

Having expressed the problem in terms of a collection of integer linear programs enables us to try to compute the optimal solution with softwares like PIP [4,3] (at least for reasonable sizes of the target computing platforms). We compare this optimal solution with that returned by two polynomial-time heuristics, one that approximates the TSP problem (but only returns a solution where all processors are involved), and a greedy heuristic that iteratively grows the solution ring.

6.1   TSP-Based Heuristic

The situation where all processors are involved in the optimal solution is very important in practice. Indeed, only very large applications are likely to be deployed on distributed heterogeneous platforms. And when W is large enough, we know from Equation 1 that all processors will be involved.

From Section 5.1 we know that the optimal solution, when all processors are involved, corresponds to the shortest Hamiltonian cycle in the graph (P, E, d), with $d_{i,j} = \frac{c_{i,j}+c_{j,i}}{w_i}$. We use the well-known Lin-Kernighan heuristic [8,7] to approximate this shortest path. By construction, the TSP-based heuristic always returns a solution where all processors are involved. Of course, if the optimal solution requires fewer processors, the TSP-based heuristic will fail to find it.

6.2   Greedy Heuristic

The greedy 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 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 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.

6.3   Platform Description

We experimented with two platforms, one located in ENS Lyon and the other in the University of Strasbourg, which are represented in Figure 2. The Lyon

platform is composed of 14 processors, whose cycle-times are described in Table 1. Table 2 shows the capacity of the links, i.e. the inverse of the bandwidth, between each processor pair (P_i, P_j). The Strasbourg platform is composed of 13 processors, whose cycle-times are described in Table 3, while Table 4 shows the capacity of the links.
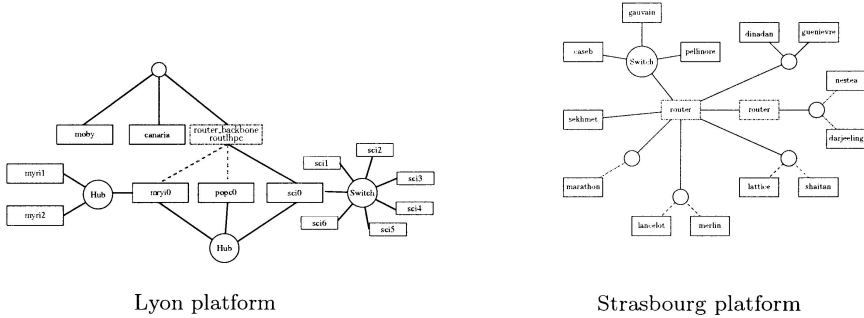

| Lyon platform | Strasbourg platform |

Fig. 2. Topology of the Lyon and Strasbourg platforms.

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	P_9	P_{10}	P_{11}	P_{12}	P_{13}
0.0291	0.00874	0.0206	0.0451	0.0206	0.0291	0.0206	0.0206	0.0206	0.0206	0.0206	0.0206	0.0206	0.0206

6.4   Results

For both topologies, we compared the greedy heuristic against the optimal solution obtained with ILP software, when available. Tables 5 and 6 show the difference between the greedy heuristic and the optimal solution (computed with PIP) on the Lyon and Strasbourg platforms. The numbers in the tables represent the minimal cost of a path of length q on the platform, i.e. the value of the objective function of the ILP program of Section 5.2 (multiplied by a scaling factor 6000, because PIP needs a matrix of integers).

PIP is able to compute the optimal solution only when all processors are involved (note that we used a machine with two gigabytes of RAM!). When all processors are involved, we also tried the LKH heuristic: for both platforms, it returns the optimal result. The conclusions that can be drawn from these experiments are the following:

– the greedy heuristic is fast (and seems efficient)
– the LKH heuristic is very reliable, but its application is limited to the case where all resources are involved
– ILP softwares rapidly fail to compute the optimal solution.

Table 2. Capacity of the links (time in seconds to transfer a 1 Mb message) for the Lyon platform.

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}
P_0		0.198	1.702	1.702	0.262	0.198	1.702	1.702	0.262	0.262	0.262	0.262	0.262	0.262
P_1	0.198		1.702	1.702	0.262	0.198	1.702	1.702	0.262	0.262	0.262	0.262	0.262	0.262
P_2	1.702	1.702		0.248	0.248	1.702	0.248	0.248	0.248	0.248	0.248	0.248	0.248	0.248
P_3	1.702	1.702	0.248		0.248	1.702	0.248	0.248	0.248	0.248	0.248	0.248	0.248	0.248
P_4	0.262	0.262	0.248	0.248		0.262	0.248	0.248	0.248	0.248	0.248	0.248	0.248	0.248
P_5	0.198	0.198	1.702	1.702	0.262		1.702	1.702	0.262	0.262	0.262	0.262	0.262	0.262
P_6	1.702	1.702	0.248	0.248	0.248	1.702		0.248	0.248	0.248	0.248	0.248	0.248	0.248
P_7	1.702	1.702	0.248	0.248	0.248	1.702	0.248		0.248	0.248	0.248	0.248	0.248	0.248
P_8	0.262	0.262	0.248	0.248	0.248	0.262	0.248	0.248		0.248	0.248	0.248	0.248	0.248
P_9	0.262	0.262	0.248	0.248	0.248	0.262	0.248	0.248	0.248		0.248	0.248	0.248	0.248
P_{10}	0.262	0.262	0.248	0.248	0.248	0.262	0.248	0.248	0.248	0.248		0.248	0.248	0.248
P_{11}	0.262	0.262	0.248	0.248	0.248	0.262	0.248	0.248	0.248	0.248	0.248		0.248	0.248
P_{12}	0.262	0.262	0.248	0.248	0.248	0.262	0.248	0.248	0.248	0.248	0.248	0.248		0.248
P_{13}	0.262	0.262	0.248	0.248	0.248	0.262	0.248	0.248	0.248	0.248	0.248	0.248	0.248	

Table 3. 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	P_{10}	P_{11}	P_{12}
0.00874	0.00874	0.0102	0.00728	0.00728	0.0262	0.00583	0.016	0.00728	0.00874	0.0131	0.00583	0.0131

We considered the number p_{opt} of processors in the best solution found by the greedy heuristic as a function of the ratio W/H. As expected, when this ratio grows (meaning more computations per communication), more and more processors are used in the optimal solution, and the value of p_{opt} increases. Because the interconnection network of the Lyon platform involves links of similar capacities, the value of p_{opt} jumps from 1 (sequential solution) to 14 (all processors participate). The big jump from 1 to 14 is easily explained: once there is enough work to make a communication affordable, rather use many communications for the same price, thereby better sharing the load.

Table 4. Capacity of the links (time in seconds to transfer a one-megabit message) for the Strasbourg platform.

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}
P_0		0.048	0.019	0.017	0.019	0.147	0.151	0.154	0.147	0.048	0.017	0.016	0.151
P_1	0.048		0.048	0.048	0.048	0.147	0.151	0.154	0.147	0.017	0.048	0.048	0.151
P_2	0.019	0.048		0.019	0.019	0.147	0.151	0.154	0.147	0.048	0.019	0.019	0.151
P_3	0.017	0.048	0.019		0.019	0.147	0.151	0.154	0.147	0.048	0.017	0.018	0.151
P_4	0.019	0.048	0.019	0.019		0.147	0.151	0.154	0.147	0.048	0.019	0.019	0.151
P_5	0.147	0.147	0.147	0.147	0.147		0.151	0.154	0.147	0.147	0.147	0.147	0.151
P_6	0.151	0.151	0.151	0.151	0.151	0.151		0.154	0.151	0.151	0.151	0.151	0.151
P_7	0.154	0.154	0.154	0.154	0.154	0.154	0.154		0.154	0.154	0.154	0.154	0.154
P_8	0.147	0.147	0.147	0.147	0.147	0.147	0.151	0.154		0.147	0.147	0.147	0.151
P_9	0.048	0.017	0.048	0.048	0.048	0.147	0.151	0.154	0.147		0.048	0.048	0.151
P_{10}	0.017	0.048	0.019	0.017	0.019	0.147	0.151	0.154	0.147	0.048		0.018	0.151
P_{11}	0.016	0.048	0.019	0.018	0.019	0.147	0.151	0.154	0.147	0.048	0.018		0.151
P_{12}	0.151	0.151	0.151	0.151	0.151	0.151	0.151	0.154	0.151	0.151	0.151	0.151	

Table 5. Comparison between the greedy heuristic and PIP for the Lyon platform.

| Processors Heuristics	3	4	5	6	7	8	9	10	11	12	13	14
Greedy	1202	556	1152	906	2240	3238	4236	5234	6232	7230	8228	10077
PIP	out of memory	out of memory	out of memory	out of memory	out of memory	out of memory	out of memory	out of memory	out of memory	out of memory	out of memory	9059

Table 6. Comparison between the greedy heuristic and PIP for the Strasbourg platform.

Processors Heuristics	3	4	5	6	7	8	9	10	11	12	13
Greedy	1520	2112	3144	3736	4958	5668	7353	8505	10195	12490	15759
PIP	out of memory	out of memory	out of memory	out of memory	out of memory	out of memory	out of memory	out of memory	out of memory	out of memory	14757

The interconnection network of the Strasbourg platform is more heterogeneous, and there the value of p_{opt} jumps from 1 (sequential solution) to 10, 11 and 13 (all processors participate).
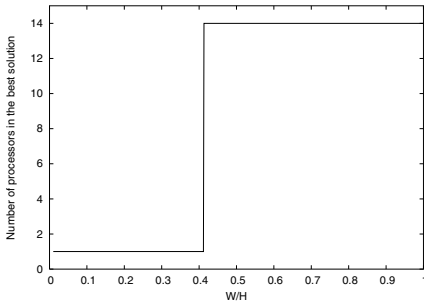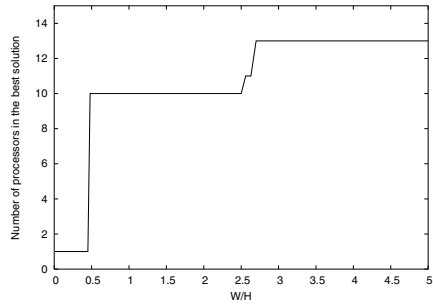
Lyon platform   Strasbourg platform

Fig. 3. Optimal number of processors for the Lyon and Strasbourg platforms according to the greedy heuristic.

7   Related Work

Due to the lack of space, see [9].

8   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 dependencies, 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 [5], may well enforce the use of dynamic schemes.

One major result of this paper is the NP-completeness of the SLICERING 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 efficient heuristics, that provide 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. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.
2. D. E. Culler and J. P. Singh. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, San Francisco, CA, 1999.
3. P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationnelle*, 22:243–268, Sept. 1988. Software available at
   `http://www.prism.uvsq.fr/~cedb/bastools/piplib.html`.
4. P. Feautrier and N. Tawbi. Résolution de systèmes d'inéquations linéaires; mode d'emploi du logiciel PIP. Technical Report 90-2, Institut Blaise Pascal, Laboratoire MASI (Paris), Jan. 1990.
5. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999.
6. M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1991.
7. K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000. Software available at `http://www.dat.ruc.dk/~keld/research/LKH/`.
8. S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.
9. H. Renard, Y. Robert, and F. Vivien. Static load-balancing techniques for iterative computations on heterogeneous clusters. Research Report RR-2003-12, LIP, ENS Lyon, France, Feb. 2003.