Matrix Multiplication on Heterogeneous Platforms

Olivier Beaumont, Vincent Boudet,
Fabrice Rastello, and Yves Robert, *Member*, *IEEE*

Abstract—In this paper, we address the issue of implementing matrix multiplication on heterogeneous platforms. We target two different classes of heterogeneous computing resources: heterogeneous networks of workstations and collections of heterogeneous clusters. Intuitively, the problem is to load balance the work with different speed resources while minimizing the communication volume. We formally state this problem in a geometric framework and prove its NP-completeness. Next, we introduce a (polynomial) column-based heuristic, which turns out to be very satisfactory: We derive a theoretical performance guarantee for the heuristic and we assess its practical usefulness through MPI experiments.

Index Terms—Parallel algorithms, load balancing, communication volume, matrix multiplication, numerical linear algebra libraries, heterogeneous platforms, cluster computing, metacomputing.

1 Introduction

In this paper, we deal with the implementation of a very simple but important linear algebra kernel, namely, matrix multiplication (MM for short), on heterogeneous platforms. Several parallel MM algorithms are available for parallel machines or homogeneous networks of workstations or PCs (see [1], [20], [31], among others). The popular ScaLAPACK library [7] includes a highly-tuned, very efficient routine targeted to two-dimensional processor grids. This routine uses a block-cyclic distribution of the matrices in both grid dimensions. We briefly recall parallel MM algorithms for homogeneous machines in Section 2.1.

Why extend parallel MM algorithms to heterogeneous platforms? The answer is clear: Future computing platforms are best described by the keywords *distributed* and *heterogeneous*. We target two different classes of heterogeneous computing resources:

Heterogeneous Networks of Workstations (HNOWs) are ubiquitous in university departments and companies. They represent the typical poor man's parallel computer: Running a large PVM or MPI experiment (possibly all night long) is a cheap alternative to buying supercomputer hours. When implementing MM algorithms on HNOWs, the idea is to make use of *all* available resources, namely, slower machines *in addition to* more recent ones. This is a challenging but very useful task given the importance of MM in scientific computing. Also, it is a first step towards understanding how to implement more complicated linear algebra kernels on HNOWs.

Collections of Clusters are made up of nodes or clusters, each of them being itself a HNOW of a parallel machine. These nodes may well be geographically scattered all around the world. Internode communications are typically an order of magnitude slower than intranode communications. The need to design a MM algorithm which would execute on a collection of clusters is less obvious. Are there actual applications which involve huge matrices whose product cannot be computed with a single parallel machine or workstation network? Larger and larger experiments are conducted throughout the world within the NPACI[1] initiative using tools such as Globus [18] and Legion [24]. Huge linear algebra kernels are often at the core of these experiments, so investigating "metacomputing" MM algorithms is quite natural. Anyway, we view MM algorithms as a perfect case study for the implementation of tightly coupled high-performance applications on the metacomputing grid [19]: Indeed, such applications are much more difficult to tackle than loosely-coupled cooperative applications. Because MM is a simple kernel which encompasses a lot of data movements, we view it as a perfect testbed to be studied before experimenting more challenging computational problems on the grid, especially those which exhibit a high spatial locality (e.g., such as finite difference schemes).

The major limitation to programming heterogeneous platforms arises from the additional difficulty of balancing the load when using processors running at different speeds. Data and computations are not evenly distributed to processors. Minimizing communication overhead becomes a challenging task: In fact, the MM problem with different-speed processors turns out to be surprisingly difficult. The main result of this paper is the NP-completeness of the MM problem on heterogeneous platforms. Rather than the

• *The authors are with the Laboratoire de l'Informatique du Parallelisme, UMR CNRS-ENS Lyon-INRIA 5668, École Normale Supérieure de Lyon, F-69364 Lyon Cedex 07, France. E-mail: {Yves.Robert, Olivier.Beaumont, Vincent.Boudet, Fabrice.Rastello}@ens-lyon.fr.*

Manuscript received 24 Jan. 2000; revised 1 Aug. 2000; accepted 1 May 2001. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 111321.

1. National Partnership for Advanced Computational Infrastructure, see http://www.npaci.edu.

proof, the result itself is interesting because it reveals the intrinsic difficulty of designing heterogeneous algorithms.

The rest of the paper is organized as follows: In Section 2, we summarize existing MM algorithms for homogeneous platforms and we discuss how to extend these to cope with heterogeneity. In Section 3, we formally state the MM optimization problem for heterogeneous platforms, which we formulate as a geometric optimization problem, and we establish its NP-completeness (the long and technical proof of this important result is given in the Appendix). In Section 4, we briefly survey related NP-complete optimization problems. Section 5 is devoted to the design of efficient (polynomial) heuristics, whose practical usefulness is demonstrated through MPI experiments on a HNOW and on a 2-cluster configuration (Section 6). We give some final remarks and conclusions in Section 7.

2 MM Algorithms

In this section, we briefly describe how to implement a parallel (or distributed) MM algorithm on a heterogeneous platform. We adopt an abstract view by assuming that we have a collection of p heterogeneous computing resources $P_1, P_2, \ldots, P_p$. If each computing resource P_i reduces to a single processor, we are dealing with a heterogeneous network of workstations or PCs (HNOW). When each computing resource P_i is itself a heterogeneous cluster or a parallel machine, we are targeting a metacomputing environment made up from a collection of clusters. The high-level algorithmic description is the same for all target machines. However, our model will have to cope with different hypotheses on communication issues. We come back to the impact of communication modeling in Section 3.2. Before dealing with heterogeneous resources, we briefly summarize existing algorithms for homogeneous machines.

2.1 Homogeneous Grids

We start by briefly recalling the MM algorithm implemented in the ScaLAPACK library [7] on 2D homogeneous grids. For the sake of simplicity, we restrict ourselves to the multiplication $C = AB$ of two square $n \times n$ matrices A and B. In that case, ScaLAPACK uses the outer product algorithm described in [1], [20], [31]. Consider a 2D processor grid of size $p = p_1 \times p_2$ and assume for a while that $n = p_1 = p_2$. In that case, the three matrices share the same layout over the 2D grid: Processor $P_{i,j}$ stores $a_{i,j}$, $b_{i,j}$, and $c_{i,j}$. Then, at each step k,

- Each processor $P_{i,k}$ (for all $i \in \{1, .., p_1\}$) horizontally broadcasts $a_{i,k}$ to processors $P_{i,*}$ and
- Each processor $P_{k,j}$ (for all $j \in \{1, .., p_2\}$) vertically broadcasts $b_{k,j}$ to processors $P_{*,j}$,

so that each processor $P_{i,j}$ can independently update $c_{i,j} = c_{i,j} + a_{i,k}b_{k,j}$.

This current version of the ScaLAPACK library uses a blocked version of this algorithm to squeeze the most out state-of-the-art processors with pipelined arithmetic units and multilevel memory hierarchy [17], [12]. Each matrix coefficient in the description above is replaced by a $r \times r$ square block, where optimal values of r depend on the memory hierarchy and on the communication-to-computation ratio of the target computer. Finally, a level of virtualization is added: Usually, the number of blocks $\lceil \frac{n}{r} \rceil \times \lceil \frac{n}{r} \rceil$ is much greater than the number of processors $p_1 p_2$. Thus, blocks are scattered in a cyclic fashion along both grid dimensions, so that each processor is responsible for updating several blocks at each step of the algorithm.

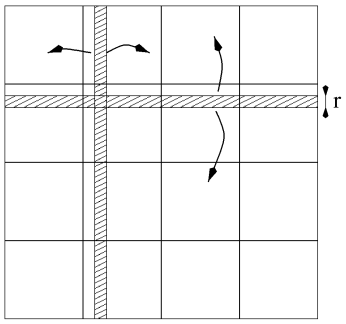

Fig. 1. One step of the MM algorithm on a 4×4 homogeneous 2D-grid.

To prepare for the description of the heterogeneous version, we introduce another "logical" description of the algorithm:

- We take a macroscopic view and concentrate on allocating (and operating on) matrix blocks to processors: Each element in A, B, and C is a square $r \times r$ block and the unit of computation is the updating of one block, i.e., a matrix multiplication of size r.
- At each step, a column of blocks (the pivot column) is communicated (broadcast) horizontally and a row of blocks (the pivot row) is communicated (broadcast) vertically.
- The C matrix is partitioned into $p_1 \times p_2$ rectangles. There is a one-to-one mapping between these rectangles and the processors. Each processor is responsible for updating its rectangle: More precisely, it updates each block in its rectangle with one block from the pivot row and one block form the pivot column, as illustrated in Fig. 1. For square $p \times p$ homogeneous 2D-grids and when the number of blocks in each dimension n is a multiple of p (the actual matrix size is thus $n.r \times n.r$), it turns out that all rectangles are identical squares of $\frac{n}{p} \times \frac{n}{p}$ blocks.

In Fig. 1, we see that the total amount of communications performed by the MM algorithm is proportional to the sum of the half-perimeters of the rectangles allocated to the processors: More precisely, at each step each processor responsible for a rectangle of $h \times v$ blocks must receive (vertically) h blocks of matrix B and (horizontally) v blocks of matrix A. This explains why allocated rectangles are identical squares for square $p \times p$ homogeneous 2D-grids when p divides n: In that case, all rectangles of fixed area $\frac{n}{p} \times \frac{n}{p}$ are squares. Because the (half)-perimeter of a rectangle of fixed area is minimized when it is a square, this choice does minimize the communication volume.

There are other homogeneous MM algorithms: For instance, Cannon's algorithm [31] (whose main drawback

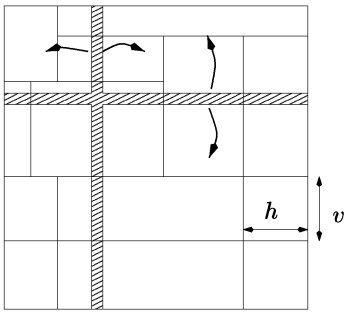

Fig. 2. The MM algorithm on a heterogeneous platform.

is to require an initial permutation of matrices A and B) replaces all the horizontal and vertical broadcasts by nearest-neighbor shifts. The total communication volume at each step is the same, but the communications are different. Still, all processors independently update their rectangle of C blocks at each step.

2.2 Heterogeneous Platforms

How to modify the previous MM algorithms for a heterogeneous platform? The idea is to keep the same framework: at each step, one pivot column and one pivot row are communicated to all processors and independent updates take place. However, with different speed processors, we cannot distribute same size rectangles from the C matrix to the processors. Intuitively, we want to balance the computing load so that each processor receives an amount of work in accordance to its computing power. Because all C blocks require the same amount of arithmetic operations, each processor executes an amount of work which is proportional to the number of blocks that are allocated to it, hence, proportional to the area of its rectangle. To parallelize the matrix product $C = AB$, we have to tile the C matrix into p nonoverlapping rectangles, each rectangle being assigned to one processor. Fig. 2 shows an example with 13 different-speed computing resources.

The question is: How to compute the *area* and *shape* of these p rectangles so as to minimize the total execution time? As usual, with parallel algorithms, there are two nonindependent and maybe conflicting goals: 1) load-balancing computations and 2) minimizing communication overhead. Goal 1) is related to the area of the rectangles that are allocated to the processors while goal 2) is related to their shapes. We discuss areas and shapes in the next section, in order to formally state (and try to solve) this difficult optimization problem.

3 The Heterogeneous MM Optimization Problem

Consider a matrix product $C = AB$, where A, B, and C are square matrices of $n \times n$ square blocks of size r. Assume that we have p computing resources $P_1, P_2, \ldots, P_p$ of (relative) cycle-times $t_1, t_2, \ldots, t_p$: If all processors have same speed, then $t_i = 1$ for $1 \leq i \leq p$. If, say, P_2 is twice faster than P_1, then $t_1 = 2t_2$. We start with load-balancing issues before dealing with communication overhead.

3.1 Load Balancing

To perfectly load-balance the computation, each processor should receive an amount of work in accordance to its computing power. If, say, P_2 is twice faster than P_1 ($t_1 = 2t_2$), then P_2 should be assigned twice as many elements as P_1. In other words, the *area* of its rectangle should be the double of that of P_1. Let s_i be the area of the rectangle R_i allocated to processor P_i. Obviously, the first equation is $\sum_{i=1}^{p} s_i = n^2$, in order to obtain a true partition of the C matrix. Next, since P_i processes its rectangle within $s_i t_i$ time-steps, we have

$$s_1 t_1 = s_2 t_2 = \ldots = s_p t_p.$$

The last constraint is to write s_i as $s_i = h_i v_i$, where h_i and v_i are the number of rows and columns of R_i. These equations do not always have integer solutions, which means that a perfect load balancing of the computations is not always possible.

However, we are not really interested in an exact solution. A more concrete and interesting question is the following: Given the p computing resources, how to compute the respective area of the rectangles R_i so that the workload is asymptotically optimally balanced: the larger the matrix size (expressed in blocks), the more accurate the tiling into rectangles. This question translates into the following system: Given $t_1, \ldots, t_p$, search for real unknowns s_i, h_i, and v_i, $1 \leq i \leq p$, such that:

$$\begin{cases} 1) & s_1 t_1 = s_2 t_2 = \ldots = s_p t_p \\ 2) & \sum_{i=1}^{p} s_i = 1 \\ 3) & \text{The } p \text{ rectangles of size } h_i \times v_i \text{ (where } h_i v_i = s_i) \\ & \qquad\qquad\qquad\qquad \text{tile the unit square.} \end{cases}$$

Condition 1 ensures that the area of the rectangle R_i allocated to processor P_i is inversely proportional to its cycle time. Condition 2 is for normalization: The sum of the areas of the p rectangles is that of the unit square, a necessary condition for Condition 3 to hold. Note that, as expected, Conditions 1 and 2 allow to compute the s_i: We obtain

$$s_i = \frac{\frac{1}{t_i}}{\sum_{i=1}^{p} \frac{1}{t_i}}.$$

We see that s_i is computed from the harmonic mean of the t_i and it is not an integer ($0 < s_i < 1$ as soon as $p \geq 2$).

There are always solutions to the normalized problem. For instance, we fulfill Condition 3 by choosing to tile the unit square into p horizontal slices of height $v_i = s_i$ (and width $h_i = 1$) or into p vertical slices of width $h_i = s_i$ (and height $v_i = 1$). This degree of freedom comes from the fact that load balancing imposes constraints on the area of the rectangles R_i, but not on their shapes. Shapes come into the story when discussing communication issues, as explained below.

Finally, note that it is straightforward to retrieve a solution of the original problem (tiling a matrix of $n \times n$ blocks) from the solution of the normalized problem: We simply multiply all the h_i and the v_i by n, getting $h_i(n) = nh_i$ and $v_i(n) = nv_i$. Then, we round up values to integers, $h'_i(n)$ and $v'_i(n)$, while preserving the

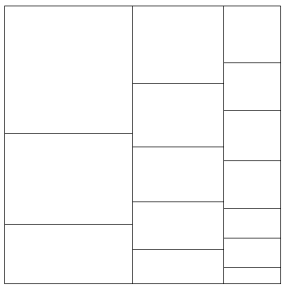

Fig. 3. Tiling the unit square into columns of rectangles.

constraint $\sum_{i=1}^{p} h_i'(n) = \sum_{i=1}^{p} v_i'(n) = n$ (there are many possible variations). Therefore, we derive a *generic* solution to the original problem, which is valid for all values of the parameter n.

3.2 Communication Overhead

At each step of the MM algorithm, communications take place between processors: The total volume of data exchanged is proportional to the sum $\hat{C} = \sum_{i=1}^{p}(h_i + v_i)$ of the half-perimeters of the p rectangles R_i. In fact, this is not exactly true: Because the pivot row and column are not sent to the processors that own them, we should subtract 2 from $\hat{C}$, 1 for the horizontal communications and 1 for the vertical ones. Since minimizing $\hat{C}$ or $\hat{C} - 2$ is equivalent, so we keep the value of $\hat{C}$ as stated.

Sequential Communications. Minimizing $\hat{C}$ seems to be a very natural goal because it represents the total volume of communications. For instance, it is natural to assume that communications will be mostly sequential on a HNOW where processors are linked by a simple Ethernet network; also, there will be little or none computation/communication overlap on such a platform. In that context, minimizing the total communication volume is the main objective: It is proportional to the communication time needed at each step of the MM algorithm with the underlying hypothesis that the network is homogeneous. In this paper, we do not investigate further the situation where different speed links are available between the processors (see Section 3.5 for a pointer).

Parallel Communications. Conversely, some communications can occur in parallel or some efficient broadcast mechanisms can be used if the computing resources are linked through a dedicated high-speed network and if parallel communication links are provided. In that context, we may want to use a columnwise allocation as depicted in Fig. 3: Vertical communications are performed in parallel in all columns and broadcasts or at least scatters can be performed horizontally.

Collections of Clusters. Finally, in a metacomputing context, intercluster communications are typically one order of magnitude slower than intracluster communications, so we may want to adopt a two-level scheme: We assign rectangles to clusters as described in Fig. 4 while inside each cluster some master-slave mechanism could be provided.

Optimization Criteria. It seems that minimizing the total communication volume is the most important optimization problem because of its wide potential applicability. Also, forgetting about MM algorithms for a while, consider the implementation of any application (such as a finite-difference scheme), where heterogeneous processors communicate boundary elements at each step (the communication scheme need not be nearest-neighbor, it can be anything): Minimizing the total communication volume while load-balancing the work amounts to solving exactly the same optimization problem.

The rest of the paper is devoted to solving the MM optimization problem using the total communication volume $\hat{C}$ as the objective function to be minimized. We formally state this optimization problem in Section 3.3. For the sake of completeness, we discuss some extensions of the problem in Section 3.5.

3.3 The MM Optimization Problem

We are ready to state the MM optimization problem for heterogeneous platforms. We have p computing resources P_i, $1 \leq i \leq p$. Each P_i is assigned a rectangle R_i of prescribed area s_i, where $\sum_{i=1}^{p} s_i = 1$. The shape of each

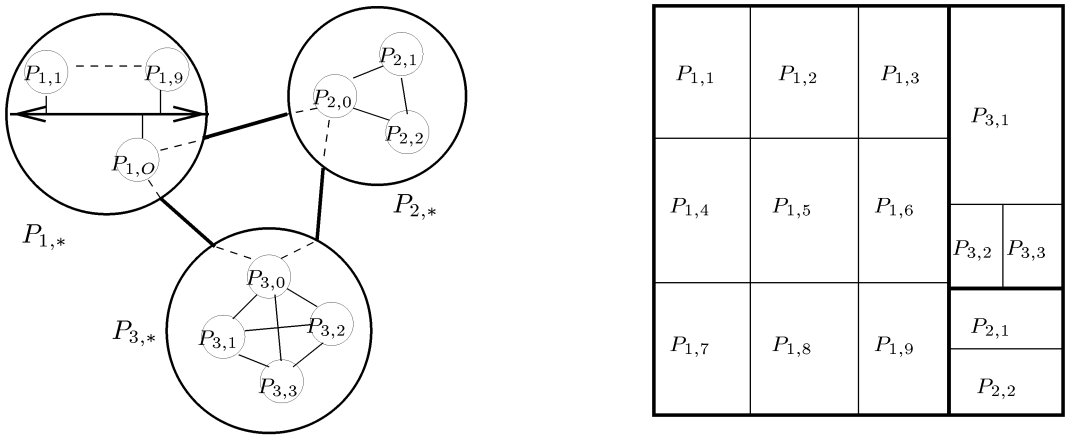

Fig. 4. Two level allocation scheme for a collection of clusters. In this example, one processor within each cluster, namely, $P_{i,0}$, is dedicated to intercluster communications.

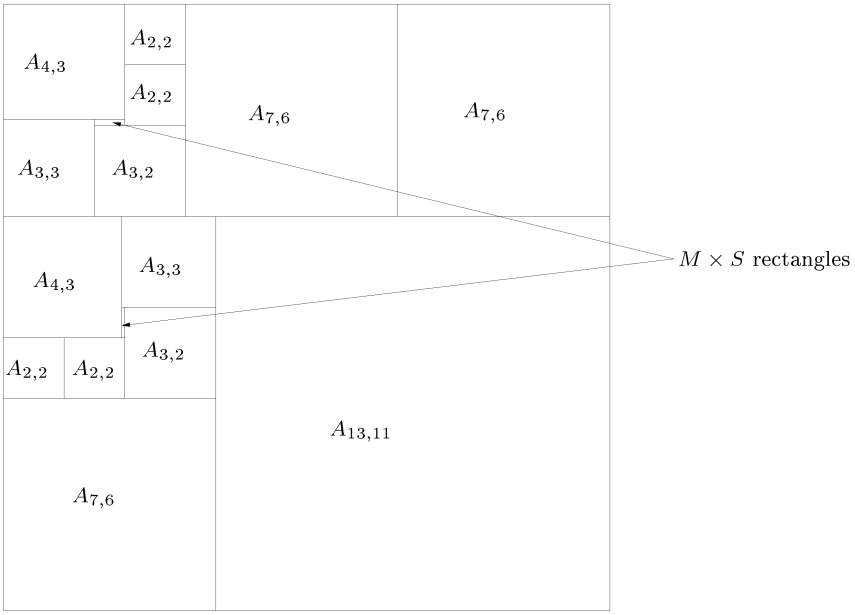

Fig. 5. General position of the squares.

R_i is the degree of freedom: We want to tile the unit square so as to minimize the total communication volume $\hat{C}$. The abstract optimization problem is the following:

Definition 1. *MM-OPT(s): Given p real positive numbers $s_1, \ldots, s_p$ s.t. $\sum_{i=1}^{p} s_i = 1$, find a partition of the unit square into p rectangles R_i of area s_i and of size $h_i \times v_i$ so that $\hat{C} = \sum_{i=1}^{p}(h_i + v_i)$ is minimized.*

Given the solution (or an approximation of the solution) of MM-OPT(s), we round up the values to the nearest integers so as to derive a concrete solution for matrices of given size n. As stated above, the integer solution will be asymptotically optimal. There is an obvious lower bound for MM-OPT(s):

Lemma 1. *For all solutions of MM-OPT(s), $\hat{C} \geq 2\sum_{i=1}^{p}\sqrt{s_i}$.*

Proof. The half-perimeter of each rectangle R_i will be always larger that $2\sqrt{s_i}$, the value when it is a square. Of course, tiling the unit square into p squares of area s_i is not always possible (think of the problem of tiling the unit square into two squares of same area 0.5), so this lower bound is not always tight. □

As already mentioned, it is easy to solve MM-OPT(s) when using a square two-dimensional grid of homogeneous processors ($s_{ij} = 1/p^2$ for $1 \leq i, j \leq p$). However, with heterogeneous processors, the MM-OPT(s) optimization problem turns out to be difficult, as shown in the next section.

3.4 NP-Completeness

The decision problem associated to the optimization problem MM-OPT is the following:

Definition 2. *MM-DEC(s,K): Given p real positive numbers $s_1, \ldots, s_p$ s.t. $\sum_{i=1}^{p} s_i = 1$ and a positive real bound K, is there a partition of the unit square into p rectangles R_i of area s_i and of size $h_i \times v_i$ so that $\sum_{i=1}^{p}(h_i + v_i) \leq K$?*

Our main result states the intrinsic difficulty of the MM optimization problem:

Theorem 1. *MM-DEC(s,K) is NP-complete.*

Because the proof is both lengthy and technical, we provide it in the Appendix. More important than the proof, the theorem itself clearly demonstrates the intrinsic difficulty of static load-balancing on heterogeneous platforms while minimizing communication cost.

The main ideas of the proof are the following:

- First, we polynomially reduce the decision problem MM-DEC(s,K) to a geometric problem (ASP) that amounts to check if there exists a partition of the unit square into squares of given areas.
- Then, we prove the NP-completeness of ASP using a polynomial reduction to the 2-Partition-Equal problem which is NP-complete [21]. The draft of this last proof is the following:

We start from an arbitrary instance of the 2-Partition-Equal problem, i.e., from a set $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ of n integers, which we aim at partitioning into two subsets of same cardinal and same sum. The idea is to build an equivalent instance of this problem using a set $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$ of n integers such that $b_i > \frac{2}{3}\max b_i$. We simply define (polynomially) $b_i = 2(a_i + 2n\max_k a_k)$. Under a few technical assumptions, we show that there exists a solution to the initial 2-Partition-Equal problem if and only if $\mathcal{B}$ can be partitioned into two subsets of same sum (not necessarily of same cardinal). Finally, we build from $\mathcal{B}$ an instance of ASP using three kinds of squares: large squares (denoted as $A_{i,j}$ in Fig. 5), n squares of size $b_i \times b_i$ (denoted as A_{b_i} in Fig. 6), and a polynomial number of other squares (denoted as $A_{\overline{b_i},j}$ in Fig. 6). We show that the only possible configuration is the one shown in Fig. 5. In this configuration, there are two nonadjacent $M \times S$ rectangular zones (where $M = \frac{4}{3}\max_i b_i$ and $S = \sum_i \frac{b_i}{2}$), which are partitioned as shown in Fig. 6.

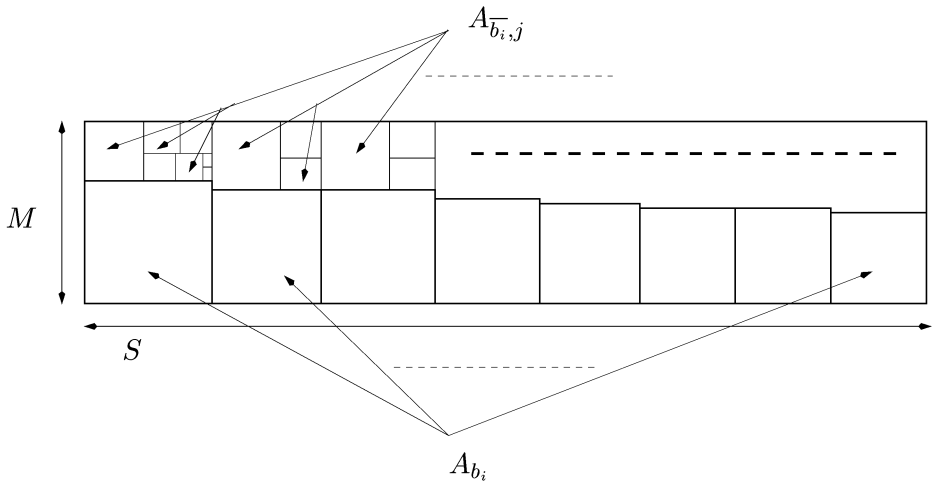

Fig. 6. Zoom on the $M \times S$ rectangle areas.

Because of the condition $b_i > \frac{M}{2}$, necessarily the A_{b_i} squares of size $b_i \times b_i$ are aligned. Therefore, for each rectangle the sum of the b_i is equal to S. Intuitively, the large rectangles are introduced to create the two nonadjacent rectangular zones of area $M \times S$; the n squares A_{b_i} must be aligned within these two zones, and the other squares are here to fill up the holes in the two rectangular zones.

3.5 Extensions of the Model

As pointed out in Section 3.2, minimizing the total volume of communications $\hat{C}$ seems to be a very natural goal. However, other objective functions could be selected, because the target computing platform may influence the way communications are implemented.

Objective Function for Parallel Communications. If *all* communications could be performed in parallel, the bottleneck would come from the processor which sends/receives the largest messages. To model such a situation, a possible objective function would be to minimize

$$\hat{M} = \max_{i=1}^{p}(h_i + v_i)$$

instead of

$$\hat{C} = \sum_{i=1}^{p}(h_i + v_i).$$

Unfortunately, the problem remains NP-complete with this objective function [3].

Objective Function for Heterogeneous Networks. Another extension of the model could come from the modeling of the network. It is natural to consider the case of a heterogeneous network, where processors communicate through different-speed links. This problem is difficult too: It is obviously NP-complete if we use different-speed processors and if we weigh the cost of each communication with a factor proportional to the bandwidth of the communication link (because it is more complicated than MM-OPT). But interestingly, the problem remains NP-complete, even when using homogeneous processors, i.e., a heterogeneous network linking processors computing with the same speed [32]. Still, we believe that it is possible to modify the column-based heuristics presented in Section 5 to design a MM algorithm targeted to a heterogeneous network linking different speed processors.

4 Related Results

We survey related papers from the literature in this section. They range into two categories: papers dealing with linear algebra on heterogeneous platforms on one hand and papers covering geometric optimization problems similar to MM-DEC(s,K) on the other hand.

4.1 Linear Algebra on Heterogeneous Platforms

Load balancing strategies for heterogeneous platforms have been widely studied. Distributing the computations (together with the associated data) can be performed either dynamically or statically or a mixture of both. Some simple schedulers are available, but they use naive mapping strategies such as master-slave techniques or paradigms based upon the idea *"use the past predict the future"*, i.e., use the currently observed speed of computation of each machine to decide for the next distribution of work [14], [13], [6]. Dynamic strategies, such as *self-guided scheduling* [34], could be useful too. 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 cluster at its best capabilities. However, dynamic strategies are outside the scope of this paper (and mentioned here for the sake of completeness). Because we have a library designer's perspective, we concentrate on static allocation schemes, which are less general and more difficult to design than dynamic approaches, but are better suited for the implementation of fixed algorithms such as linear algebra kernels, such as those of the ScaLAPACK library [7].

Several authors have dealt with the *static* implementation of MM algorithms on heterogeneous platforms. One simple approach is given by Kalinov and Lastovetky [26]. Their idea is to achieve a perfect load-balance as follows: First,

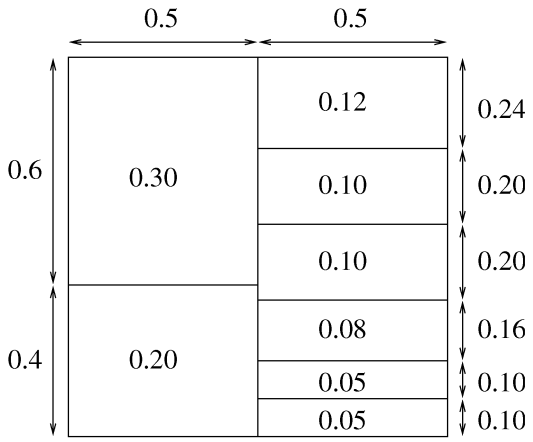

Fig. 7. The distribution of Kalinov and Lastovetky. Both columns are partitioned independently. Then, the final partition is made according to the total column weights: 0.5 versus 0.5. The total cost is $\hat{C} = 6$, which is to be compared to the cost $\hat{C} = 8$ of a partitioning into eight horizontal slices.

they take a fixed layout of processors arranged as a collection of processor columns; then, the load is evenly balanced *within* each processor column independently; next, the load is balanced *between* columns; this is the "heterogeneous block cyclic distribution" of [26], which we illustrate in Fig. 7, where we have $p = 8$ areas of values $(0.05, 0.05, 0.08, 0.1, 0.1, 0.12, 0.2, 0.3)$. This simple scheme is likely to give better results than a straightforward partitioning into horizontal slices.

Another approach is proposed by Crandall and Quinn [15]. First, they compare a contiguous block allocation (see Fig. 8) to horizontal slicing; next, they introduce a better processor arrangement: They introduce a recursive algorithm to tile the iteration space (i.e., partitioning the unit square) into p rectangles of prescribed area $s_1, s_2, \ldots, s_p$ with $\sum_{i=1}^{p} s_i = 1$ so that the total communication volume is kept small. The algorithm works recursively as follows: If at some stage their remains some rectangle R of size $h.v$ to partition into q rectangles of prescribed area $s_{i_1}, s_{i_2}, \ldots, s_{i_q}$, where $\sum_{j=1}^{q} s_{i_j} = h \times v$, then partition R along its shortest dimension into two rectangles R_1 and R_2, where R_1 contains the largest $\lceil \frac{q}{2} \rceil$ rectangles and R_2 the other ones. For instance, if $h \leq v$ and $s_{i_1} \geq s_{i_2} \geq \ldots \geq s_{i_q}$, we obtain a rectangle R_1 for the $\lceil \frac{q}{2} \rceil$ largest rectangles of area $h \times v_1$, where

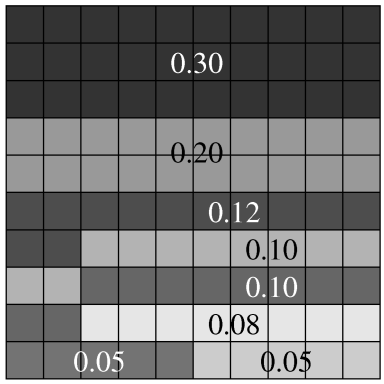

Fig. 8. Illustrating contiguous block allocation. The cost is as high as $\hat{C} = 8.2$.

$$v_1 = \frac{\sum_{j=1}^{\lceil \frac{q}{2} \rceil} s_{i_j}}{\sum_{j=1}^{q} s_{i_j}} \times v;$$

rectangle R_2 is for the remaining rectangles and of area $h \times v_2$, where $v_2 = v - v_1$ (see Fig. 9 for an illustration).

Kaddoura et al. [25] refine the previous recursive algorithm and provide several variations. They report several numerical simulations.

However, we are not aware of any theoretical result nor of any approximation bound stating some performance guarantee α (e.g., that the value $\hat{C}$ obtained by an algorithm is not α times larger than the optimal value $\hat{C}_{opt}$, where α is some constant). We have established the complexity of the problem in Section 3.4 and we provide approximation bounds in Section 5.3.

Finally, note that preliminary experimental results on implementing MM and linear system solvers on a heterogeneous network are reported in our previous papers [9], [8], [5], [10].

4.2 Problems Similar to MM-OPT(s)

There are several problems related to MM-OPT(s) in the literature:

- The most similar problem is the following: How to tile the unit square into p rectangles of same area so as to minimize the maximum perimeter of these rectangles? This problem is shown to be polynomial

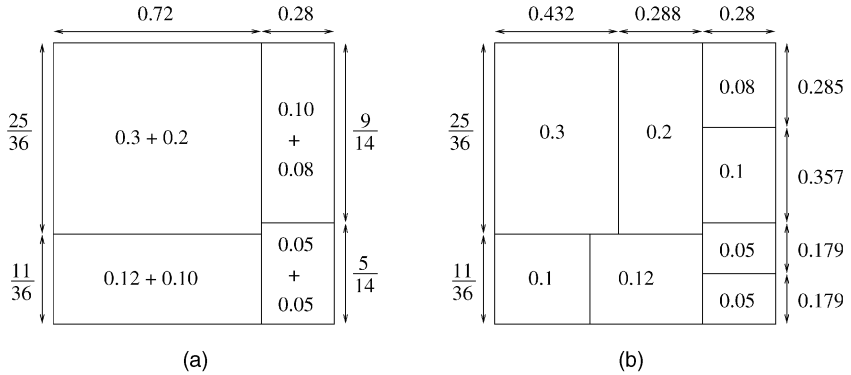

Fig. 9. Illustrating the recursive partitioning of Crandall and Quinn (obtained in three steps in this example). (a) First two steps. (b) Final partitioning : $\hat{C} = 5.56$.

by Kong et al. [30], [29]. The optimal solution is one of the following two arrangements: Let either $m = \lfloor \sqrt{p} \rfloor$ or $m = \lceil \sqrt{p} \rceil$ and use m columns composed of $\lfloor \frac{p}{m} \rfloor$ or $\lceil \frac{p}{m} \rceil$ rectangles. This problem is motivated by a data-allocation problem which is related to ours in the following sense: Assume that we have p equal-speed processors and that we aim at minimizing the largest amount of communications made by one processor. Because the above arrangements are optimal, we have a polynomial solution to this problem.

The heterogeneous counterpart of this problem is the following: Given p different-speed processors, how to allocate data so that the length of the largest communication is optimized? In terms of tiling, how to tile the unit square into p nonoverlapping rectangles of prescribed area $s_1, \ldots, s_p$ whose sum is 1 so that the largest perimeter is minimized? This interesting problem is NP-complete too [3], which again shows the intrinsic difficulty of designing heterogeneous parallel algorithms!

- Another related problem is to find the minimum partition of a rectangle with interior points. Given a rectangle R and a finite set P of points located inside R, find a set of line segments that partition R into rectangles such that every point in P is on the boundary of some rectangle. The goal is to minimize the total length of the introduced line segments. This problem is shown NP-complete in [33], [22], [23], where approximation algorithms are given. The link with our problem is that the objective function is the same, but the original motivation in [22], [23] was a VLSI routing problem (and the constraints are quite different).
- There are several NP-complete geometric optimization problems that are listed in [16]. One example is the minimum rectangle tiling problem [28]: Given an $n \times n$ array A of nonnegative numbers and a positive integer p, find a partition of A into p nonoverlapping rectangular subarrays such that the maximum weight of any rectangle in the partition is minimized (the weight of a rectangle is the sum of its elements).

5 Heuristics

In this section, we introduce a polynomial heuristic to solve the MM-OPT problem. After describing the heuristic and proving its optimality among all column-based approaches, we report experimental results that nicely demonstrate its efficiency. Finally, we provide a theoretical guarantee for the heuristic and we discuss possible extensions.

5.1 Optimal Column-Based Tiling

As outlined in Section 3.3, the MM-OPT(s) problem is the following: Given p real positive variables $s_1, \ldots, s_p$ such that $\sum_{i=1}^{p} s_i = 1$, tile the unit square into p nonoverlapping rectangles $R_1, \ldots, R_p$ of respective areas $s_1, \ldots, s_p$ so as to minimize the sum of the (half) perimeters of these rectangles. Because the associated decision problem MM-DEC(s,K) is NP-complete (Section 3.4), we consider the more constrained problem MM-COL(s), where we impose that the tiling is made up of processor columns, as illustrated in Fig. 10. In other words, MM-COL(s) is the restriction of MM-OPT(s) to column-based partitions. In this section, we give a polynomial solution to the MM-COL(s), which will be used as a heuristic to solve MM-OPT(s).

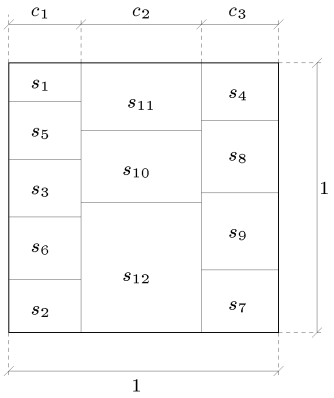

Fig. 10. Column-based partitioning of the unit square: $\mathcal{C} = 3$, $k_1 = 5$, $k_2 = 3$, and $k_3 = 4$.

Framework. We describe the MM-COL(s) problem more formally. We aim at tiling the unit square into $\mathcal{C}$ columns (where $\mathcal{C}$ is yet to be determined) of width $c_1, \ldots, c_{\mathcal{C}}$. Each column C_i is partitioned itself into k_i rows (to be determined too) of respective area $s_{\sigma(i,1)}, \ldots, s_{\sigma(i,k_i)}$. Of course, the final partitioning has $\sum_{i=1}^{\mathcal{C}} k_i = p$ rectangles and all the areas $s_1, \ldots, s_p$ are represented once and only once. The goal is to build such a partitioning subject to the minimization of the sum of the rectangle perimeters.

Algorithm 1. We describe our algorithm which is based upon the dynamic programming paradigm; the optimality proof will be presented later. The main points are the following:

1. Reindex the variables $s_1, \ldots, s_p$ such that
$$s_1 \leq s_2 \leq \ldots \leq s_p.$$
2. Iteratively build the function $f_{\mathcal{C}}$, by incrementing the value of $\mathcal{C}$ from 1 to the desired value. For $q \in \{1, \ldots, p\}$, $f_{\mathcal{C}}(q)$ represents the total perimeter of an optimal column-based partitioning of a rectangle of height 1 and width $\left(\sum_{i=1}^{q} s_i\right)$ into q rectangles of respective area $s_1, \ldots, s_q$ using $\mathcal{C}$ columns.

To help understand the derivation, we apply the algorithm on the example we have used throughout Section 4.1. We have $p = 8$ areas of values

$$(0.05, 0.05, 0.08, 0.1, 0.1, 0.12, 0.2, 0.3).$$

The results of the algorithm are described in Table 1 and the resulting partitioning is depicted in Fig. 11. Each column $\mathcal{C}_i$ contributes to the sum of the half-perimeters as follows: 1 for the vertical line and $k_i \times c_i$ for the k_i horizontal lines of length c_i.

In the example, the optimal partitioning is obtained for three columns ($f_3(8) = 5.5$). The first column of width $c_3 = s_7 + s_8 = 0.5$ is composed of 2 elements. The second column of width $c_2 = s_4 + s_5 + s_6 = 0.32$ is composed

TABLE 1

Table Containing the Values of the Couples $f_{\mathcal{C}}(q)/r$ where $f_{\mathcal{C}}(q) = \min_{r\in[\mathcal{C}-1,q-1]}\left(1 + \left(\sum_{r<i\leq q} s_i\right) \times (q-r) + f_{\mathcal{C}-1}(r)\right)$ and r is the Value Minimizing the Previous Expression

	q=1	q=2	q=3	q=4	q=5	q=6	q=7	q=8
$\mathcal{C}=1$	1.05 / 0	1.2 / 0	**1.54 / 0**	2.12 / 0	2.9 / 0	4 / 0	5.9 / 0	9 / 0
$\mathcal{C}=2$		2.1 / 1	2.28 / 2	2.56 / 2	2.94 / 3	**3.5 / 3**	4.38 / 4	5.76 / 5
$\mathcal{C}=3$			3.18 / 2	3.38 / 3	3.66 / 4	4 / 4	4.58 / 5	**5.5 / 6**
$\mathcal{C}=4$				4.28 / 3	4.48 / 4	4.78 / 5	5.2 / 6	5.88 / 7
$\mathcal{C}=5$					5.38 / 4	5.6 / 5	5.98 / 6	6.5 / 7
$\mathcal{C}=6$						6.5 / 5	6.8 / 6	7.28 / 7
$\mathcal{C}=7$							7.7 / 6	8.1 / 7
$\mathcal{C}=8$								9 / 7

of three elements. Then, the last column of width $c_1 = s_1 + s_2 + s_3 = 0.18$ is made up with the smallest three elements (see Fig. 11).

The algorithm is outlined in Table 2. ($f_{\mathcal{C}}^{perimeter}$ corresponds to the $f_{\mathcal{C}}(q)$ previously used. $f_{\mathcal{C}}^{cut}(q)$ corresponds to the total number of blocs in the first $\mathcal{C}-1$ columns so that there remains $q - f_{\mathcal{C}}^{cut}(q)$ blocs in the column $\mathcal{C}$).

The worst case complexity of the algorithm is $O(p^3)$. Note that, in practice, the complexity will be lower than the worst-case analysis shows because $f_{\mathcal{C}}(p)$ is a function that is first decreasing and then increasing as $\mathcal{C}$ varies. All the functions $f_{\mathcal{C}}$ will not be built and the expected cost will be $p^2\mathcal{C}_{opt} \approx p^{2.5}$.

The final partitioning corresponding to the function $f_{\mathcal{C}_{opt}}(p) = \min_{1\leq\mathcal{C}\leq p} f_C(p)$ is found using the algorithm in Table 3, which corresponds to tracking (backwards) the bold entries in Table 1. The unit square is partitioned into $\mathcal{C}_{opt}$ columns. The ith column contains the rectangles $s_{d+1}, \ldots, s_{d+k_i}$ with $d = k_1 + k_2 + \ldots + k_{i-1}$.

Correctness. To prove the optimality of the algorithm, we show that the optimum solution can be achieved with a *well-ordered partitioning*. A partitioning is said to be well

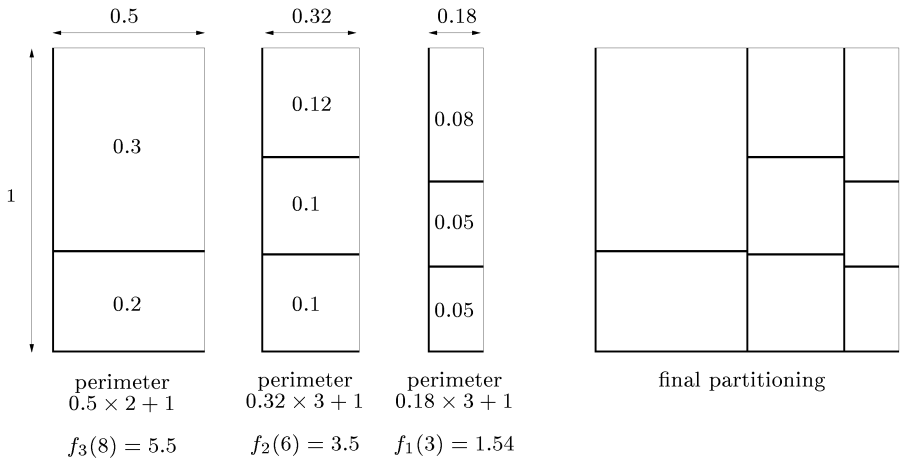

Fig. 11. Optimal column-based partitioning for the example. Thicker lines correspond to the sum of the half-perimeters. We obtain $\hat{C} = 5.5$.

TABLE 2
Algorithm 1

$S = 0$
for q=1 to p
  $S = S + s_q$
  $f_1^{perimeter}(q) = 1 + Sq$
  $f_1^{cut}(q) = 0$
endfor
for $\mathcal{C}$=2 to p
  for q=$\mathcal{C}$ to p
    $f_{\mathcal{C}}^{perimeter}(q) = \min_{r\in[\mathcal{C}-1,q-1]}\left(1 + \sum_{q-r<i\leq q} s_i(q-r) + f_{\mathcal{C}-1}^{perimeter}(r)\right)$
    $f_{\mathcal{C}}^{cut}(q) = r_{opt}$ {where r_{opt} reaches the minimum in the previous expression}
  endfor
endfor

TABLE 3
Algorithm 2

$q = p$
for $\mathcal{C} = \mathcal{C}_{opt}$ **downto** 2
  $k_{\mathcal{C}} = q - f_{\mathcal{C}}^{cut}(q)$
  $q = f_{\mathcal{C}}^{cut}(q)$
endfor
$k_1 = q$

ordered if for every pair of columns $\mathcal{C}_i$ and $\mathcal{C}_j$, either all the elements of $\mathcal{C}_i$ are smaller than (or equal to) all the elements of $\mathcal{C}_j$, or the other way round. See Fig. 12 for an illustration.

We start from a given partitioning made up of, say, $\mathcal{C}$ columns of size $k_1 \geq k_2 \geq \ldots \geq k_{\mathcal{C}}$. Suppose, for convenience, that $s_1 \leq s_2 \leq \ldots \leq s_p$; τ is a permutation of $\{1, 2, \ldots, p\}$ such that the ith column of this partitioning contains the rectangles $s_{\tau(d+1)}, \ldots, s_{\tau(d+k_i)}$ with

$$d = k_1 + k_2 + \ldots + k_{i-1}.$$

Now, recall that the cost of column C_i is $1 + k_i \sum_{j=d+1}^{d+k_{i-1}} s_{\tau(j)}$. Hence, the total "perimeter" is

$$\begin{aligned}
&\mathcal{C} + k_1 s_{\tau(1)} + k_1 s_{\tau(2)} + \ldots + k_1 s_{\tau(k_1)} \\
&+ k_2 s_{\tau(k_1+1)} + k_2 s_{\tau(k_1+2)} + \ldots + k_2 s_{\tau(k_1+k_2)} \\
&+ \ldots \\
&+ k_{\mathcal{C}} s_{\tau(k_1+\ldots+k_{\mathcal{C}-1}+1)} + k_{\mathcal{C}} s_{\tau(k_1+\ldots+k_{\mathcal{C}-1}+2)} + \ldots + k_{\mathcal{C}} s_{\tau(k_1+\ldots+k_{\mathcal{C}})}.
\end{aligned}$$

Since $k_1 \geq k_2 \geq \ldots \geq k_{\mathcal{C}}$, this expression is minimized for $\tau = Identity$, which corresponds to a "well-ordered" partitioning.[2] Hence, for each partitioning, there exists a corresponding better or equivalent partitioning that is "well ordered." This achieves the proof of correctness.

5.2 Experimental Comparison with the Lower Bound

As shown in Section 3.3, a lower bound for the sum $\hat{C}$ of the half-perimeters is twice the sum of the square roots of the areas $LB = 2\sum_{i=1}^{p} \sqrt{s_i}$. Of course, this bound cannot always be met. Consider an instance of MM-OPT(s) with only two processors, $s_1 = 1 - \epsilon$ and $s_2 = \epsilon$, where $\epsilon > 0$ is an arbitrarily small number. Partitioning into two rectangles requires to draw a line of length 1, hence, $\hat{C} = 3$. However, $LB = 2(\sqrt{1-\epsilon} + \sqrt{\epsilon}) > 2$ can be arbitrarily close to 2.

In this section, we experimentally compare, using a large number of random tests, the value $\hat{C}$ given by our partitioning against the absolute lower bound LB.

- Because the ratio between the processor speeds is not likely to be very large in practice (who would use a machine 100 times slower than another one?),[3] we assume that a uniform repartition of the processor speeds might be significant. The goal of Fig. 13 is to show that the column based partitioning is efficient in most reasonable situations. We randomly generate a large number of set of speeds with a uniform repartition. We represent two curves for a number of processors varying from 1 to 40. The first curve corresponds to the mean value of the ratio $\frac{\hat{C}}{LB}$ while the second curve gives the minimum values of this ratio. We see that on average, the optimal column-based tiling given by our algorithm gives a solution that is "almost" optimal, so that we can be satisfied with the results for all practical purposes.
- Next, we adopt a theoretical point of view and concentrate on worst cases. A uniform repartition is no longer acceptable with such a purpose. Hence, in Fig. 14, we generate a large number of sets of speeds using an exponential repartition. Because the ratio $r = \frac{\max s_i}{\min s_i}$ plays an important role in the cost of the worst case, we display the tests for different values of r varying from 2 to ∞.

2. The proof can easily be done by induction on the number of inversions in the permutation τ.

3. In fact, using 100 slower machines in conjunction to a fast one does make sense in some cases!

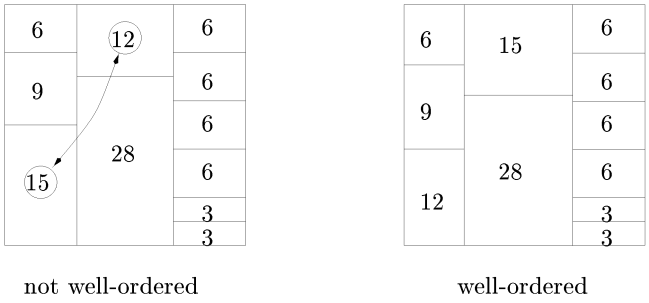

Fig. 12. Two partitionings of the same problem instance. The right one is well ordered while the left one is not.

5.3 Theoretical Comparison with the Lower Bound

The column-based heuristic appears to be quite satisfactory in practice. In this section, we prove it is (theoretically) not far from being optimal, especially when the ratio r between $\max s_i$ and $\min s_i$ is small. In other words, we are able to give the following guarantee to the column-based heuristic:

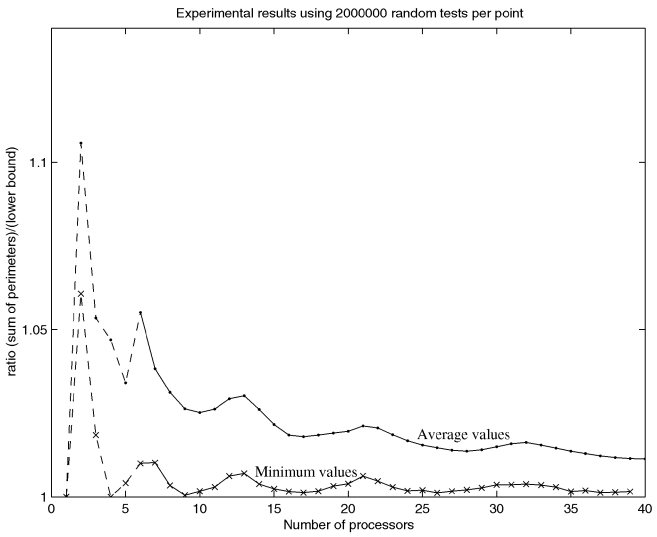

Fig. 13. For each number of processors (varying from 1 to 40), 2,000,000 values for the s_i have been randomly generated. For each case, we compute the ratio of the sum $\hat{C}$ of the half-perimeters of our partitioning over the absolute lower bound LB. The *average* and *minimum* values of this ratio are reported in the two curves.

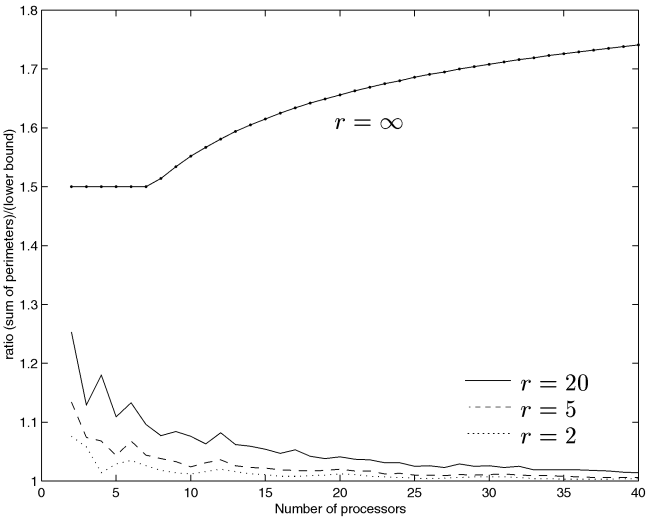

Fig. 14. For each number of processors (varying from 2 to 40) and for different values of $\frac{r=\max s_i}{\min s_i}$, 10,000 values for the s_i have been randomly generated using an exponential distribution. For each case, we compute the ratio of the sum $\hat{C}$ of the half-perimeters of our partitioning over the absolute lower bound LB. The *maximum* values of this ratio are reported in the four curves.

Proposition 1. *Let $r = \frac{\max s_i}{\min s_i}$ and let $\hat{C}$ denote the sum of the half-perimeters of the rectangles obtained with the optimal column-based partitioning. Then,*

$$\frac{\hat{C}}{2\sum\sqrt{s_i}} \leq \sqrt{r}\left(1 + \frac{1}{\sqrt{p}}\right).$$

Proof. Consider a column based partitioning with

$$\mathcal{C} = \lceil \sqrt{r} \sum \sqrt{s_i} \rceil$$

columns. Rectangles are evenly distributed amongst columns so that the numbers of rectangles in each column is either $\lfloor \frac{p}{\mathcal{C}} \rfloor$ or $\lceil \frac{p}{\mathcal{C}} \rceil$. Letting $\hat{C}^*$ denote the sum of the half-perimeters of the rectangles obtained with this column partitioning, we have:

$$\begin{aligned}\hat{C}^* &\leq \lceil \sqrt{r} \sum \sqrt{s_i} \rceil + \left\lceil \frac{p}{\lceil \sqrt{r} \sum \sqrt{s_i} \rceil} \right\rceil \\ &\leq 2 + \sqrt{r} \sum \sqrt{s_i} + \frac{p}{\sqrt{r} \sum \sqrt{s_i}}.\end{aligned}$$

Thus,

$$\frac{\hat{C}^*}{2\sum\sqrt{s_i}} \leq \frac{1}{\sum\sqrt{s_i}} + \frac{\sqrt{r}}{2} + \frac{p}{2\sqrt{r}\sum\sqrt{s_i}}.$$

Moreover,

$$\begin{aligned}\sum s_i = 1 &\Longrightarrow p \max s_i \geq 1 \\ &\Longrightarrow \min s_i \geq \frac{1}{pr}\end{aligned}$$

and, thus,

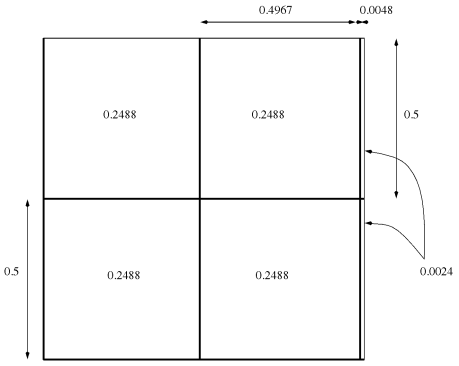

Fig. 15. Optimal column-based partitioning. The sum of the half-perimeters is 5.

$$\sum \sqrt{s_i} \geq p\sqrt{\min s_i} \geq \sqrt{\frac{p}{r}}.$$

Therefore,

$$\begin{aligned}\frac{\hat{C}^*}{2\sum\sqrt{s_i}} &\leq \sqrt{\frac{r}{p}} + \frac{\sqrt{r}}{2} + \frac{\sqrt{r}}{2} \\ &\leq \sqrt{r}(1 + \frac{1}{\sqrt{p}}).\end{aligned}$$

Since $\widehat{C}$ corresponds to the best solution among all possible column-based partitioning, $\hat{C}$ satisfies to $\hat{C} \leq \hat{C}^*$, which concludes the proof. □

If $r = 1$, i.e., all the processors have the same speed, the column-based partitioning is asymptotically optimal. On the other hand, if r is large, i.e., one processor is much faster than another, the bound is very pessimistic. In [4], we have considered a recursive heuristic, very complicated to describe, but for which a better approximation bound is provided. For all practical purposes, we believe that the column-based algorithm is the best trade-off between efficiency and simplicity.

5.4 Looking for a Better Solution

As already said, this section is mostly theoretical. We investigate new algorithms for the sake of improving the column-based solution, which is very satisfactory except in some "degenerate" artificial cases. To illustrate the point, consider the following partitioning problem into $p = 6$ rectangles of respective areas

$$(0.2488, 0.2488, 0.2488, 0.2488, 0.0024, 0.0024)$$

(the relative cycle-times of the six processors are approximately $(1, 1, 1, 1, 100, 100)$). The absolute lower-bound for this example is $LB = 2\sum_{i=1}^{6}\sqrt{s_i} = 4.19$. Consider the following solutions, which have different degrees of freedom:

1. The partitioning is constrained to be a column-based partitioning. Using the column-based algorithm, we obtain the solution depicted in Fig. 15.
2. The partitioning is constrained to be recursively defined as follows: The unit square is divided into several columns. Each column is in turn divided into

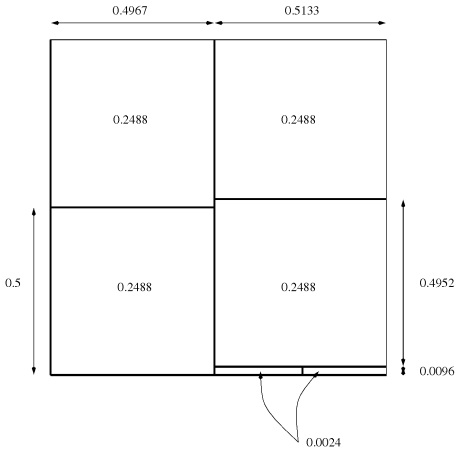

Fig. 16. Optimal recursively defined partitioning. The sum of the half-perimeters is 4.51.

several rows and so on. Of course, there are multiple choices for the number of columns and for the number of rows within each column and so on. In Fig. 16, we give an example with two columns divided into two and three rows, respectively. Finally, the last row of the second column is split into two rectangles. In the example, this partitioning is optimal amongst recursively defined partitionings (proof by exhaustive case analysis).

3. The partitioning is only constrained to be made out of rectangles. An example is given in Fig. 17. Note that this solution is neither column-based nor recursively defined. This partitioning is optimal among all rectangle-based partitionings.

Clearly, the less constrained the partitioning, the better the solution. Note that the improvement over the column-based partitioning can be important, roughly 16 percent for the rectangle-based solution. Again, in a realistic experiment, we would never use a processor in conjunction with another one which is 100 times faster! Also, with a library designer's perspective, a simple column-based partitioning has many advantages regarding code generation issues.

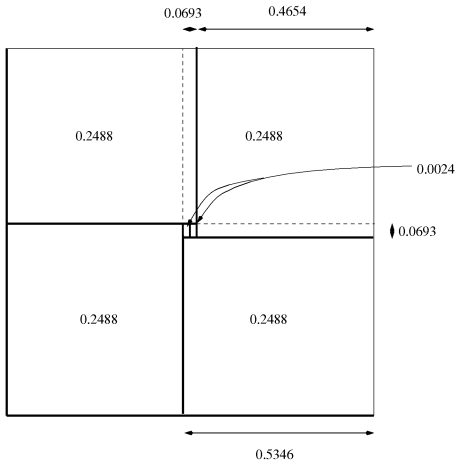

Fig. 17. Optimal rectangle-based partitioning. The sum of the half-perimeters is 4.19.

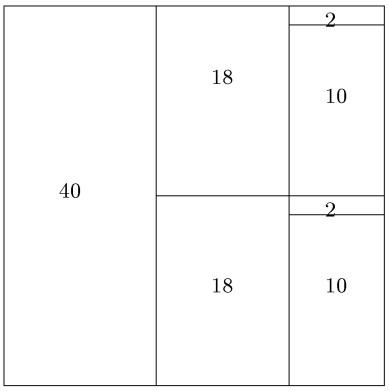

Fig. 18. Partition given by the column-based heuristic (Cost $\hat{C} = 5.1$).

6 MPI Experiments

To provide a preliminary experimental validation of our approach, we have implemented the heterogeneous MM algorithm using the MPI library [35]. In this section, we report a few experiments performed on a HNOW and on a (very small!) collection of two clusters.

6.1 Using a Single HNOW to Compare Different Partitions

In this section, we use a cluster of seven heterogeneous machines of relative cycle-times equal to $(1, 1, \frac{1}{5}, \frac{1}{5}, \frac{1}{9}, \frac{1}{9}, \frac{1}{20})$. These seven machines are SUN workstations of our laboratory, linked by a simple Ethernet network. We compare the partition given by the optimal column-based heuristic (see Fig. 18) with four different partitions of the same matrices which are shown in Fig. 19.

The measures were realized for matrices of size $n = 640$ using a blocksize $r = 32$ and for matrices of size $n = 1280$ using two blocksize values, $r = 32$ and $r = 64$. Table 4 gives the average time to compute the MM product for the five partitions. In the case of a matrix of size $n = 1280$, we see that the time is slightly smaller if we increase the blocksize because there are fewer communications. We check that the execution time does grow with the cost of the partition, which shows that our modeling of the communication costs is very reasonable and is in good adequation with these experiments. Note that (for fairness) we have not compared the results with the homogeneous block-cyclic distribution. Because the processor speeds are very different, the performances would have been disastrous.

6.2 Experimenting with Two Clusters

In this section, the target platform is made up of two clusters. The first cluster is a pile of Pentium Pros and the second cluster is a pile of Power PCs. The interconnection network within both clusters is a Myrinet network. There is also a Myrinet link between the two clusters. Hence, all communications are very fast. In the experiments, either we allocate to each cluster a fraction of the matrix which is proportional to its computing power, according to Section 3.1, or we give the same fraction to each processor, as in the homogeneous case. When we use the load-balancing strategy, we use each cluster as a farm of processors and equally distribute the workload inside the farm.

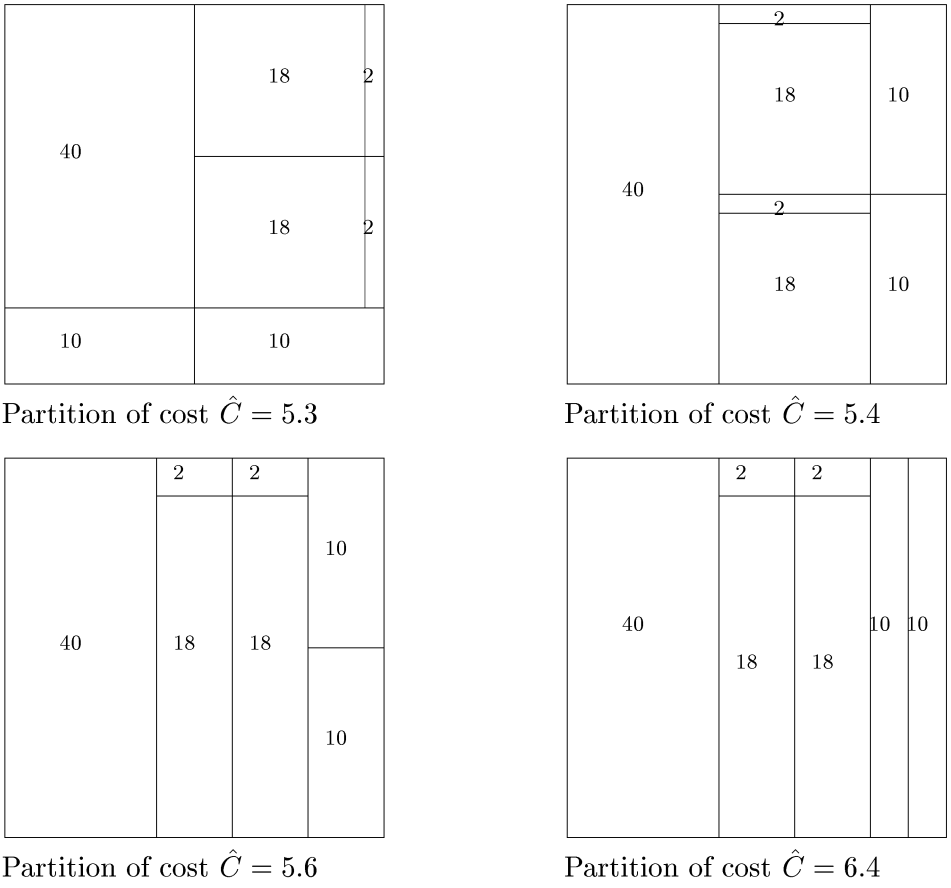

Fig. 19. Four different column-based partitions.

We use two configurations, one with five processors in the first cluster and three in the second one and the other one with only four processors in the first cluster and two processors in the second one. In both cases, the gain of the load-balancing heuristic over the homogeneous block-cyclic distribution (a meaningful comparison here because the processor speeds are rather similar) is very important (see Fig. 20).

7 Conclusion

In this paper, we have dealt with the implementation of MM algorithms on heterogeneous platforms. The bad news is that minimizing the total communication volume is NP-complete. The good news is that efficient polynomial heuristics can be provided, as we have shown, both theoretically (by guaranteeing their performance) and through simulations and MPI experiments. At the very least, the MPI experiments fully demonstrate the importance of using a good load-balancing strategy.

Future work could aim at testing a larger collection of clusters with slower intercluster links. The Globus system [18] provides a perfect framework for such experiments because hardware resources are used in a dedicated mode through a remote batch system so that static load-balancing strategies such as the one presented in this paper have all their significance.

The MM algorithm is the prototype of tightly-coupled kernels with a high spatial locality that need to be implemented efficiently on distributed and heterogeneous platforms. We view it as a perfect testbed before experimenting more challenging computational problems on the grid.

It is not clear which is the good level to program metacomputing platforms. Data-parallelism seems unrealistic due to the strong heterogeneity. Explicit message passing is too low-level. Despite their many advantages, object-oriented approaches (e.g., [24], [2]) still request the user to have a deep knowledge and understanding of both its application behavior and the underlying hardware and network. Remote computing systems such as NetSolve [11] face severe limitations to efficiently load-balance the work to processors. For the inexperienced user, relying on specialized but highly-tuned libraries of all kinds (communication, scheduling, application-dependent data decompositions) may prove a good trade-off until the programming environments evolve into high-level general-purpose yet efficient solutions.

TABLE 4
Average Times for a Matrix Multiplication

Partition	n=640 r=32	n=1280 r=32	n=1280 r=64
$\hat{C}$=5.1	T=33	T=269	T=258
$\hat{C}$=5.3	T=34	T=287	T=265
$\hat{C}$=5.4	T=48	T=289	T=264
$\hat{C}$=5.6	T=34	T=290	T=267
$\hat{C}$=6.4	T=72	T=367	T=302

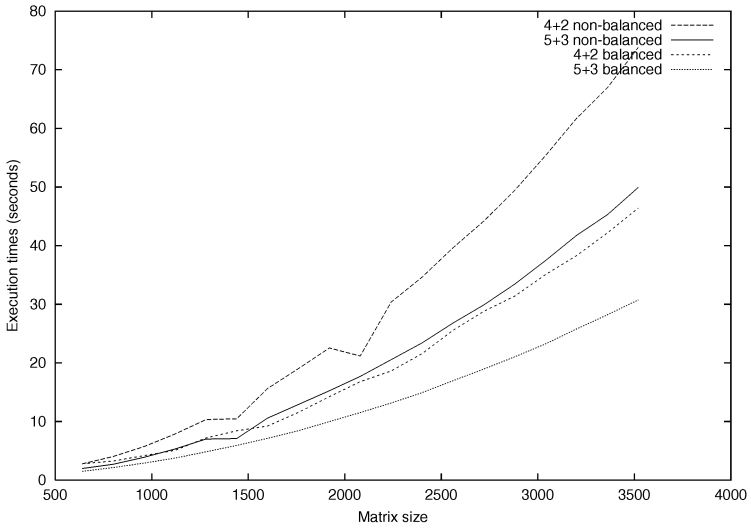

Fig. 20. MPI experiments with two clusters.

APPENDIX A
PROOF OF THEOREM 1

Definition 2 MM-DEC(s,K). *Given p real positive numbers $s_1, \ldots, s_p$ s.t. $\sum_{i=1}^{p} s_i = 1$ and a positive real bound K, is there a partition of the unit square into p rectangles R_i of area s_i and of size $h_i \times v_i$ so that $\sum_{i=1}^{p}(h_i + v_i) \leq K$?*

Theorem 1. *MM-DEC(s,K) is NP-complete.*

Proof. Obviously, MM-DEC(s,K) $\in$ NP. In this section, we prove the following lemma. □

Lemma 2. *$2P - eq \leq_P$ ASP $\leq_P$ MM − DEC, where 2P-eq and ASP are defined as follows:*

Definition 3. 2-Partition-Equal (2P-eq). *Given a set of p integers $\mathcal{A} = \{a_1, \ldots, a_p\}$, is there a partition of $\{1, \ldots, p\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that*

$$\sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i \text{ and } card(\mathcal{A}_1) = card(\mathcal{A}_2) \ ?$$

Definition 4. All-Squares-Partition (ASP). *Given a set*

$$\mathcal{L} = \{l_1, \ldots, l_p\}$$

of p real positive numbers such that $\sum_{i=1}^{p} l_i^2 = 1$, is there a partition of the unit square into p squares S_i of width l_i?

Since 2P-eq is known to be NP-Complete [21], Lemma 2 will complete the proof of Theorem 1.

A.1 Reduction: ASP $\leq_P$ MM-DEC(s,K)

We start by proving the easy part of Lemma 2, i.e., ASP $\leq_P$ MM − DEC(s, K). Let $\mathcal{L} = \{l_1, \ldots, l_p\}$ be a set of p real positive numbers s.t. $\sum_{i=1}^{p} l_i^2 = 1$. Solving ASP is equivalent to solving MM-DEC(s,K) with

$$\begin{cases} K = \sum_{i=1}^{p} 2l_i \\ \forall i,\ s_i = l_i^2 \end{cases}$$

and, therefore, ASP $\leq_P$ MM − DEC.

A.2 Reduction: 2P-eq $\leq_P$ ASP

In this section, we consider an arbitrary instance of the 2-Partition-Equal problem, i.e., a set $\mathcal{A} = \{a_1, \ldots, a_n\}$ of n integers. We assume that $n \geq 400$ without loss of generality. We have to polynomially transform this instance into an instance of the ASP problem which has a solution iff the original instance of 2-Partition-Equal has one solution.

Define $\{b_1, \ldots, b_n\}$ as

$$\forall i, b_i = 2(a_i + 2n \max_k a_k).$$

Let $N = \max_k b_k$. Then, $b_i > \frac{2N}{3}$ and b_i is even. Moreover, if we let $M = \frac{4N}{3}$ and

$$S = \frac{\sum_i b_i}{2},$$

then $S \geq 100M$. We also have $\frac{M}{2} < b_i \leq \frac{3M}{4}$ for all i. The reason for introducing M is that we will tile the n rectangles $\overline{R_i}$ of size $b_i \times (M - b_i)$ into a minimal number of squares $KS(i)$, following the procedure of Kenyon [27]. Here, KS stands for *Kenyon's squares*. To get a logarithmic number of squares $KS(i)$, the rectangle $\overline{R_i}$ must not be too elongated, which is ensured by the inequality $M - b_i < b_i \leq 3(M - b_i)$. We obtain from [27] that $KS(i) \leq 3 + C \log b_i$, where C is a universal constant. In the following, for $1 \leq i \leq n$, we let $w(\overline{b_i}, j)$, $1 \leq j \leq KS(i)$, denote the widths of the $KS(i)$ squares obtained by the procedure in [27] to tile the rectangle $\overline{R_i}$ of size $b_i \times (M - b_i)$.

We build the following instance $\mathcal{L}$ of the ASP problem (ASP($b_1, \ldots, b_n$)): Is there a partition of the unit square into $14 + n + \sum_{i=1}^{n} KS(i)$ squares of respective width

$$\begin{array}{llll} \frac{(13S+11M)}{l} & (\times 1), & \frac{(7S+6M)}{l} & (\times 3), \\ \frac{(3S+2M)}{l} & (\times 2), & \frac{(2S+2M)}{l} & (\times 4), \\ \frac{(4S+3M)}{l} & (\times 2), & \frac{(3S+3M)}{l} & (\times 2), \\ \frac{b_i}{l} & (\forall i), & \frac{w(\overline{b_i},j)}{l} & (\forall i,\ \forall j \leq KS(i)), \end{array}$$

where $l = (20S + 17M)$?

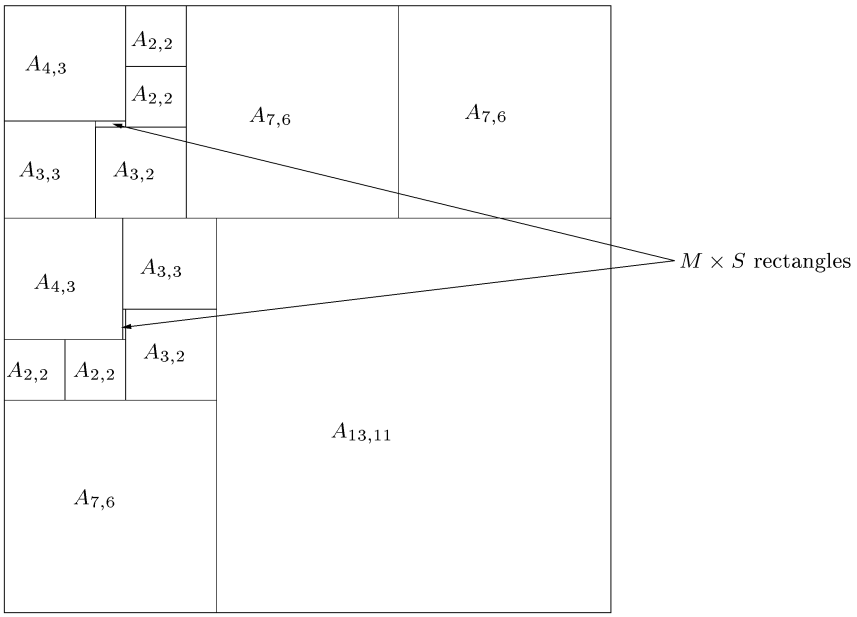

Fig. 21. General position of the squares.

For convenience, in what follows, we consider the (equivalent) scaled problem: Is there a partition of the $(20S + 17M) \times (20S + 17M)$ square into $14 + n + \sum_i KS(i)$ squares of respective width

$$\begin{cases} 13S + 11M & (\times 1), \\ 7S + 6M & (\times 3), \\ 4S + 3M & (\times 2), \\ 3S + 3M & (\times 2), \\ 3S + 2M & (\times 2), \\ 2S + 2M & (\times 4), \\ b_i & (\forall i,\ 1 \le i \le n), \\ w(\overline{b_i}, j) & (\forall i,\ 1 \le i \le n,\ \forall j,\ 1 \le j \le KS(i))\ ? \end{cases}$$

From now on, $A_{x,y}$ denotes a square of width $(xS + yM)$, A_{b_i} denotes a square of width b_i, and $A_{\overline{b_i},j}$ denotes a Kenyon's square of width $w(\overline{b_i}, j)$. In what follows, we prove that such a partition is necessarily the one depicted in Fig. 21, where the two small $M \times S$ rectangle areas are shown by arrows in Fig. 21 and fully described in Fig. 22. The intuitive idea of the proof is the following: The large squares are used to prevent the two small $M \times S$ rectangular zones to be neighbors. Hence, these areas must be filled separately by the remaining squares, namely the squares A_{b_i} and the Kenyon's squares. This will be possible iff the b_i can be partitioned into two subsets of same sum, which in turn will be possible iff the a_i can be partitioned into two subsets of same sum and same cardinal. The Kenyon's squares are introduced to fill the holes in the two rectangular zones and to obtain a true tiling of the whole area.

A.2.1 Position of the Largest Four Squares

The general position of the largest four squares is shown in Fig. 23a. Obviously, if we can tile the remaining area with the remaining squares, this will also be the case for the configuration shown in Fig. 23b. Therefore, from now on,

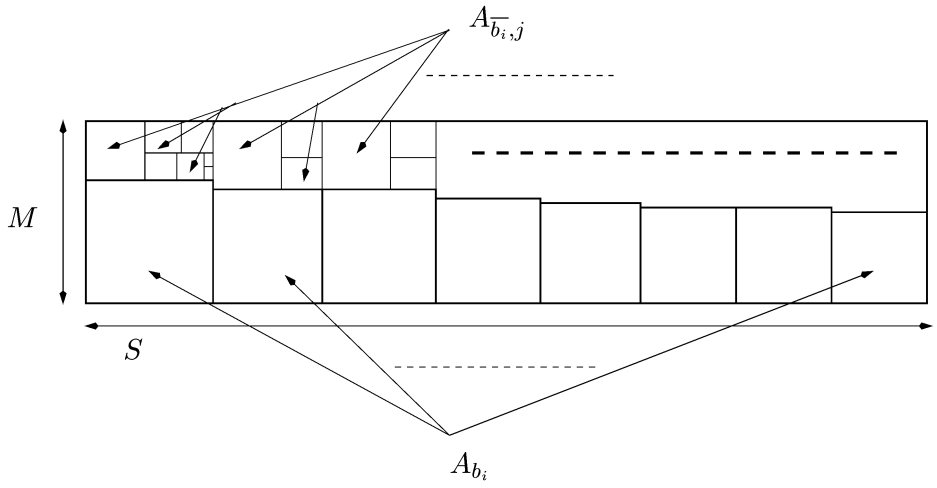

Fig. 22. Zoom on the $M \times S$ rectangular zones.

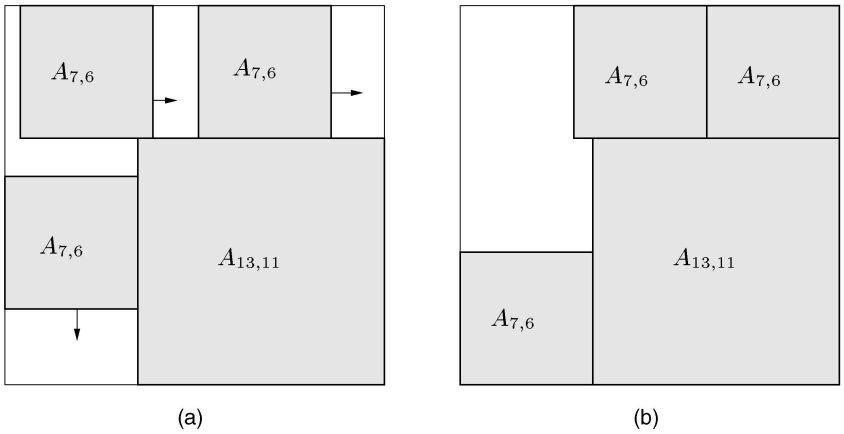

Fig. 23. General position of the largest four squares.

we assume (without loss of generality) that the largest four rectangles are arranged as shown in Fig. 23b.

A.2.2 Tiling the Remaining Surface

Now, we discuss the tiling of the remaining surface (the white area of Fig. 23b). We give all dimensions in Fig. 24). We proceed by an exhaustive case analysis:

- First case: We start by tiling the $6S + 5M$ basis with a $4S + 3M$ square.
  The different situations to consider are shown in Fig. 25, Fig. 26, and Fig. 27, respectively. The only correct configuration is the one depicted in Fig. 27.
- Second case: We start by tiling the $6S + 5M$ basis with a $3S + 3M$ square.
  The different situations to consider are shown in Fig. 28, Fig. 29, and Fig. 30. The only correct configuration is the one depicted in Fig. 30.
- Moreover, any solution requires either a $4S + 3M$ square or a $3S + 3M$ square to be on the $6S + 5M$ basis.

Therefore, Fig. 27 and Fig. 30 describe the only two possibilities to start the tiling of the remaining surface described in Fig. 24. By symmetry, we can complete these partial tilings into the solution described in Fig. 31 (other equivalent solutions are also possible).

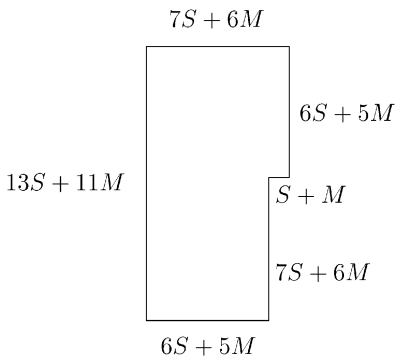

Fig. 24. Remaining surface.

A.2.3 Partial Conclusion

We have proved that any tiling of the remaining surface (see Fig. 24) is similar to the one depicted in Fig. 31. After using all the large rectangles $A_{x,y}$, there remains two nonadjacent rectangle areas of area $M \times S$ to be tiled. Therefore, we can solve the ASP problem iff we can tile these two areas with the remaining squares, i.e., n squares A_{b_i} of width b_i and $\sum_{i=1}^{n} KS(i)$ Kenyon's squares of width $w(\overline{b_i}, j)$. Since $\min_i b_i > \frac{M}{2}$, we cannot superpose two rectangles A_{b_i} and A_{b_j}, $i \neq j$, on top of each other. Since $\sum_i b_i = 2S$, one can easily check that both $M \times S$ rectangle areas have to be tiled as depicted in Fig. 22. Necessarily the A_{b_i} squares of size $b_i \times b_i$ have to be aligned. Therefore, for each rectangle the sum of the b_i is equal to S. Consequently, our instance $\mathcal{L}$ of the ASP problem has a solution iff there exists a partition of $\{b_1, \ldots, b_n\}$ into two subsets of same sum S.

A.2.4 Final Reduction

To complete the reduction, we have to show that there exists a partition of $\{b_1, \ldots, b_n\}$ into two subsets of same sum iff the original instance $\mathcal{A}$ of the 2-Partition-Equal problem has a solution.

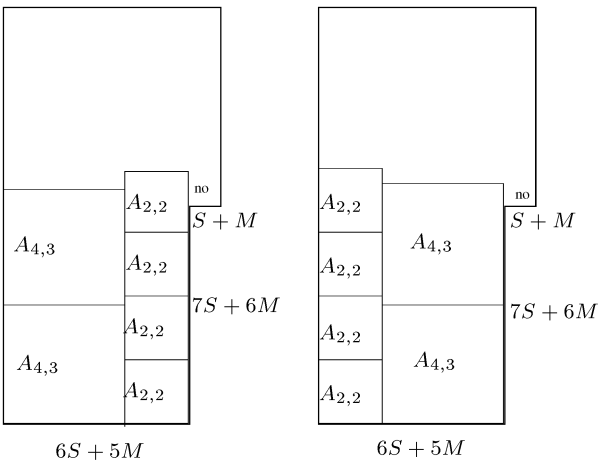

Fig. 25. Some impossible configurations with a $4S + 3M$ square.

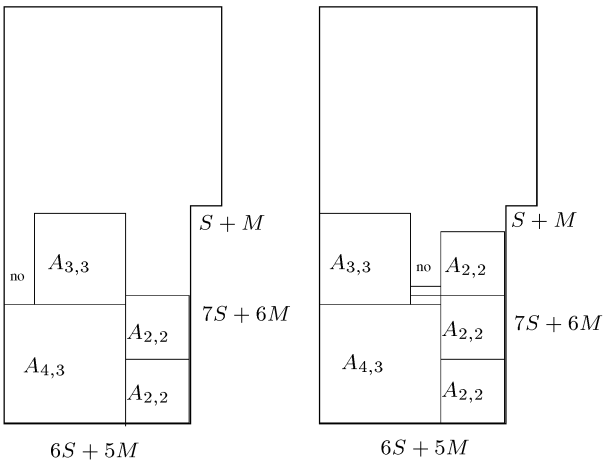

Fig. 26. Some impossible configurations with a $4S+3M$ square.

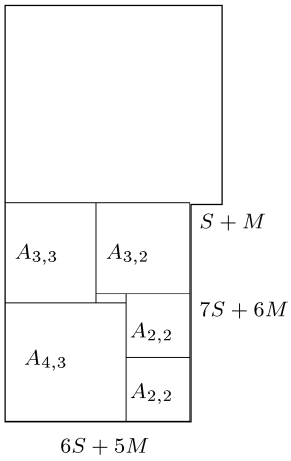

Fig. 27. The only correct configuration with a $4S+3M$ square.

First, suppose that the original instance of the 2-Partition-Equal problem has a solution, i.e. there exists a partition of $\{1,\ldots,n\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ satisfying

$$\sum_{i\in\mathcal{A}_1} a_i = \sum_{i\in\mathcal{A}_2} a_i \text{ and card } (\mathcal{A}_1) = \text{ card } (\mathcal{A}_2).$$

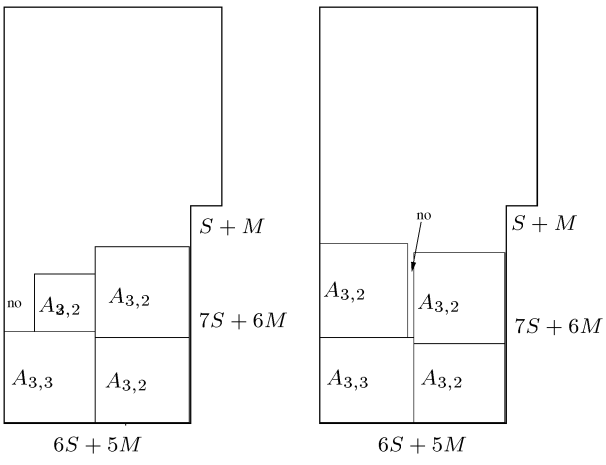

Fig. 28. Some impossible configurations with a $3S+3M$ square.

Recall that $b_i = 2(a_i + 2n \text{ MAX})$, where $\text{MAX} = \max_i a_i$. Then,

$$\begin{aligned}\sum_{i\in\mathcal{A}_1} b_i &= \sum_{i\in\mathcal{A}_1} a_i + 2n \text{ MAX card } (\mathcal{A}_1)\\ &= \sum_{i\in\mathcal{A}_2} a_i + 2n \text{ MAX card } (\mathcal{A}_2)\\ &= \sum_{i\in\mathcal{A}_2} b_i.\end{aligned}$$

Therefore, there exists a suitable partition of $\{b_1,\ldots,b_n\}$.

Conversely, suppose that there exists a partition of $\{1,\ldots,p\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that

$$\sum_{i\in\mathcal{A}_1} b_i = \sum_{i\in\mathcal{A}_2} b_i.$$

Thus,

$$\begin{aligned}\sum_{i\in\mathcal{A}_1} a_i &= \sum_{i\in\mathcal{A}_1} b_i - 2n \text{ MAX card } (\mathcal{A}_1)\\ \sum_{i\in\mathcal{A}_2} a_i &= \sum_{i\in\mathcal{A}_2} b_i - 2n \text{ MAX card } (\mathcal{A}_2)\\ \sum_{i\in\mathcal{A}_1} a_i - \sum_{i\in\mathcal{A}_2} a_i &= 2n \text{ MAX card } (\mathcal{A}_2 - \mathcal{A}_1).\end{aligned}$$

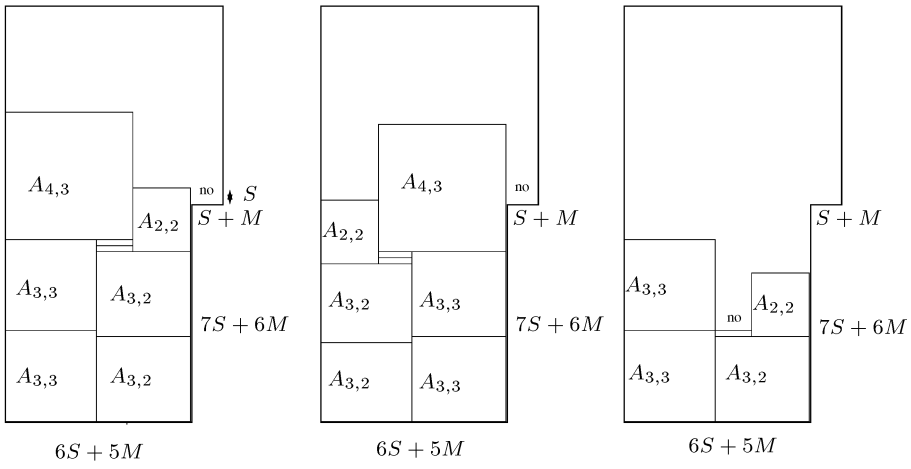

Fig. 29. Some impossible configurations with a $3S+3M$ square.

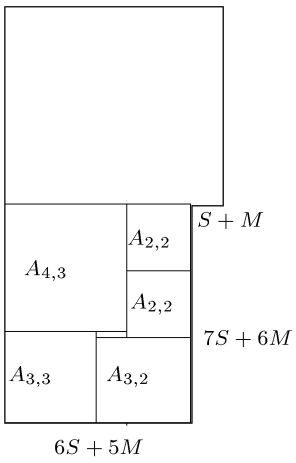

Fig. 30. The only correct configuration with a $3S+3M$ square.

Moreover, since

$$\sum_{a_i \in \mathcal{A}_1} a_i - \sum_{a_i \in \mathcal{A}_2} a_i \leq n \text{ MAX},$$

we obtain

$$\text{card}(\mathcal{A}_1) = \text{ card } (\mathcal{A}_2) \text{ and} \sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i.$$

Therefore, the original instance of the 2-Partition-Equal problem has a solution.

A.3 Conciseness of the Transformation

The last element of the proof is the conciseness of the transformation. We have to proof that our instance of the ASP problem has a size polynomial in the size of the original instance of the 2-Partition-Equal problem.

Lemma 3. *Define* $\text{MAX} = \max_i a_i$ *as above and let* $c(\mathcal{A})$ *and* $c(\mathcal{L})$ *denote respectively the encoding of the data* $\mathcal{A}$ *and* $\mathcal{L}$. *Then,*

$$Length(c(\mathcal{L})) = O(Length(c(\mathcal{A}))^3).$$

Proof. We write $f(x) = O(g(x))$ if there exists a constant c such that $f(x) \leq c.g(x)$ for x large enough. Similarly, we write $f(x) = \Omega(g(x))$ if there exists a constant c such that $g(x) \leq c.f(x)$ for x large enough.

For the encoding of the initial instance $\mathcal{A}$, we have $\text{Length}(c(\mathcal{A})) = \Omega(\sum_i \log a_i)$. Since

$$\sum_i \log a_i \geq \log \text{ MAX} + (n-1)\log(\min_i a_i)$$
$$\geq (n-1)\log 2 + \log \text{ MAX},$$

we derive that

$$\text{Length}(c(\mathcal{A})) = \Omega(n + \log \text{ MAX}).$$

For the encoding of the ASP instance $\mathcal{L}$, we have

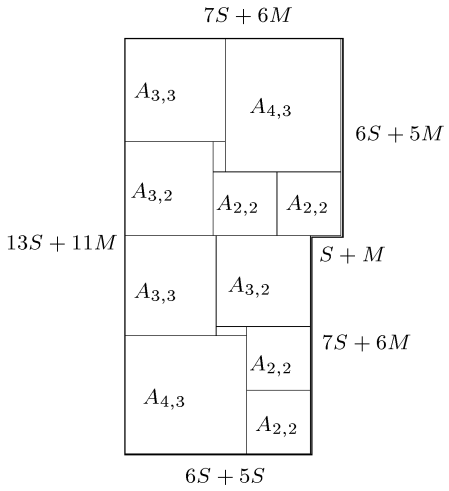

Fig. 31. One possible tiling of the remaining surface.

$$\begin{cases} \log b_i \leq \log((4n+2)\text{ MAX}) = O(\log n + \log \text{ MAX}), \\ \log M = O(\log n + \log \text{ MAX}), \\ \log S = O(n(\log n + \log \text{ MAX}), \\ \sum_i KS(i)\log b_i \leq \sum_i (3 + C\log b_i)\log b_i \\ \qquad = O(n(\log n + \log \text{ MAX})^2), \end{cases}$$

where C is the universal constant given by Kenyon [27]. Therefore,

$$\text{Length}(c(\mathcal{L})) = O(\text{Length}(c(\mathcal{A}))^3).$$

□

This achieves the proof of the NP-completeness of ASP and, therefore, the proof of the NP-completeness of MM-DEC. □

ACKNOWLEDGMENTS

The authors would like to thank the reviewers whose comments and suggestions have greatly improved the presentation of the paper. They would also like to thank Claire Kenyon for her careful reading of the proofs.

REFERENCES

[1] R. Agarwal, F. Gustavson, and M. Zubair, "A High Performance Matrix Multiplication Algorithm on a Distributed-Memory Parallel Computer, Using Overlapped Communication," *IBM J. Research and Development*, vol. 38, no. 6, pp. 673-681, 1994.

[2] H.E. Bal, A. Plaat, T. Kielmann, J. Maassen, R. van Nieuwpoort, and R. Veldema, "Parallel Computing on Wide-Area Clusters: The Albatross Project" *Proc. Extreme Linux Workshop*, pp. 20-24, 1999.

[3] O. Beaumont, V. Boudet, A. Legrand, F. Rastello, and Y. Robert, "Heterogeneity Considered Harmful to Algorithm Designers," Technical Report RR-2000-24, LIP, ENS Lyon, June 2000. Also available at www. ens-lyon. fr/LIP/.

[4] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix-Matrix Multiplication on Heterogeneous Platforms," Technical Report RR-2000-02, LIP, ENS Lyon, Jan. 2000.

[5] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix-Matrix Multiplication on Heterogeneous Platforms," *Proc. 2000 Int'l Conf. Parallel Processing (ICPP 2000)*, 2000.

[6] F. Berman, "High-Performance Schedulers," *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, eds., Morgan-Kaufmann, pp. 279-309, 1999.

[7] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley, *ScaLAPACK Users' Guide.* SIAM, 1997.

[8] V. Boudet, F. Rastello, and Y. Robert, "Algorithmic Issues for (Distributed) Heterogeneous Computing Platforms," *Proc. Cluster Computing Technologies, Environments, and Applications (CC-TEA '99),* R. Buyya and T. Cortes, eds., 1999.

[9] P. Boulet, J. Dongarra, F. Rastello, Y. Robert, and F. Vivien, "Algorithmic Issues on Heterogeneous Computing Platforms," *Proc. Clusters and Computational Grids Workshop,* 1998.

[10] P. Boulet, J. Dongarra, F. Rastello, Y. Robert, and F. Vivien, "Algorithmic Issues on Heterogeneous Computing Platforms," *Parallel Processing Letters,* vol. 9, no. 2, pp. 197-213, 1999.

[11] H. Casanova and J. Dongarra, "Netsolve: A Network Server for Solving Computational Science Problems," *Int'l J. Supercomputer Applications and High Performance Computing,* vol. 11, no. 3, pp. 212-223, 1997.

[12] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley, "ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers—Design Issues and Performance," *Computer Physics Comm.,* vol. 97. pp. 1-15, 1996.

[13] 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.

[14] M. Cierniak, M.J. Zaki, and W. Li, "Scheduling Algorithms for Heterogeneous Network of Workstations," *Computer J.,* vol. 40, no. 6, pp. 356-372, 1997.

[15] 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.

[16] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation,* Springer, 1999.

[17] J.J. Dongarra and D.W. Walker, "Software Libraries for Linear Algebra Computations on High Performance Computers," *SIAM Rev.,* vol. 37, no. 2, pp. 151-180, 1995.

[18] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *Int'l J. Supercomputer Applications,* vol. 11, no. 2, pp. 115-128, 1997.

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

[20] G. Fox, S. Otto, and A. Hey, "Matrix Algorithms on a Hypercube i: Matrix Multiplication," *Parallel Computing,* vol. 3, pp. 17-31, 1987.

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

[22] T.F. Gonzalez and S. Zheng, "Improved Bounds for Rectangular and Guilhotine Partitions," *J. Symbolic Computation,* vol. 7, pp. 591-610, 1989.

[23] T.F. Gonzalez and S. Zheng, "Approximation Algorithm for Partitioning a Rectangle with Interior Points," *Algorithmica,* vol. 5, pp. 11-42, 1990.

[24] A.S. Grimshaw and W.A. Wulf, "The Legion Vision of a Worldwide Virtual Computer," *Comm. ACM,* vol. 40, no. 1, pp. 39-45, 1997.

[25] M. Kaddoura, S. Ranka, and A. Wang, "Array Decomposition for Nonuniform Computational Environments," *J. Parallel and Distributed Computing,* vol. 36, pp. 91-105, 1996.

[26] A. Kalinov and A. Lastovetsky, "Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Networks of Heterogeneous Computers," *Proc. High Performance Computing and Networking Europe 1999,* P. Sloot, M. Bubak, A. Hoekstra, and B. Hertzberger, eds., pp. 191-200, 1999.

[27] R.W. Kenyon, "Tiling a Rectangle with the Fewest Squares," *J. Combination Theory A,* vol. 76, pp. 272-291, 1996.

[28] S. Khanna, S. Muthukrishnan, and M. Paterson, "On Approximating Rectangle Tiling and Packing," *Proc. Ninth Ann. ACM-SIAM Symp. Discrete Algorithms,* pp. 384-393, 1998.

[29] T.Y. Kong, D.M. Mount, and W. Roscoe, "The Decomposition of a Rectangle into Rectangles of Minimal Perimeter," *SIAM J. Computing,* vol. 17, no. 6, pp. 1215-1231, 1988.

[30] T.Y. Kong, D.M. Mount, and M. Wermann, "The Decomposition of a Square into Rectangles of Minimal Perimeter," *Discrete Applied Mathematics,* vol. 16, pp. 239-243, 1987.

[31] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing.* Benjamin/Cummings Inc., 1994.

[32] A. Legrand, "Algorithmique Parallèle: Environnements Hétérogènes et Non-Dédiés," Master's thesis, École Normale Supérieure de Lyon, June 2000. Also available at www.ens-lyon.fr/~yrobert.

[33] A. Lingas, R.Y. Pinter, R.L. Rivest, and A. Shamir, "Minimum Edge Length Partitioning of Rectilinear Polygons," *Proc. 20th Ann. Allerton Conf. Comm., Control and Computing,* 1982.

[34] C.D. Polychronopoulos, "Compiler Optimization for Enhancing Parallelism and Their Impact on Architecture Design," *IEEE Trans. Computers,* vol. 37, no. 8, pp. 991-1004, Aug. 1988.

[35] M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, and J. Dongarra, *MPI: The Complete Reference.* MIT Press, 1996.

Olivier Beaumont received the PhD degree from the Université de Rennes in January 1999. He is currently an associate professor in the Computer Science Laboratory LIP at École Normale Supérieure de Lyon. His main research interests are parallel algorithms on distributed memory architectures.

Vincent Boudet is currently a PhD student in the Computer Science Laboratory LIP at École Normale Supérieure de Lyon. He is mainly interested in algorithm design and in compilation-parallelization techniques for distributed memory architectures.

Fabrice Rastello received the PhD degree from École Normale Supérieure de Lyon in September 2000. He is currently working as an engineer for ST Microelectronics in Grenoble. His main research interests are parallel algorithms for distributed memory architectures and automatic compilation/parallelization 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 École Normale Supérieure de Lyon. He is the author of three books, more than 60 papers published in international journals, and more than 70 papers published in international conferences. His main research interests are parallel algorithms for distributed memory architectures and automatic compilation/parallelization techniques. He is a member of the ACM and of the IEEE.