# Matrix-Matrix Multiplication on Heterogeneous Platforms

Olivier Beaumont, Vincent Boudet, Fabrice Rastello, Yves Robert

*Laboratoire de l'Informatique du Parallélisme*

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

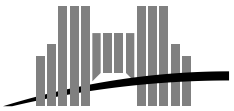


# *Matrix-Matrix Multiplication on Heterogeneous Platforms*

Olivier Beaumont
Vincent Boudet
Fabrice Rastello
Yves Robert

January 2000

Research Report N° 2000-02



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

# Matrix-Matrix Multiplication on Heterogeneous Platforms

Olivier Beaumont
Vincent Boudet
Fabrice Rastello
Yves Robert

January 2000

**Abstract**

In this paper, we address the issue of implementing matrix-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 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.

**Keywords:** heterogeneous resources, cluster, different-speed processors, load balancing, data distribution, data allocation

**Résumé**

Dans ce rapport, nous nous intéressons au problème de l'implémentation du produit matrice-matrice sur des plateformes hétérogènes. Nous considérons deux sortes de ressources de calculs hétérogènes : des réseaux de stations hétérogènes et des collections de clusters hétérogènes. Intuitivement, le problème est d'équilibrer la charge sur ces ressources de vitesses différentes tout en minimisant le volume des communications. Après avoir correctement formulé le problème, nous établissons sa NP-complétude. Ensuite nous présentons une heuristique (polynomiale) qui donne en pratique des résultats très satisfaisant : nous garantissons une performance théorique pour l'heuristique et nous prouvons son utilité pratique grâce à des expériences MPI.

**Mots-clés:** ressources hétérogènes, cluster, processeurs de vitesses différentes, distribution des données, équilibrage de charges

# 1 Introduction

In this paper, we deal with the implementation of a very simple but important linear algebra kernel, namely matrix-matrix multiplication (MMM for short), on heterogeneous platforms. Several parallel MMM algorithms are available for parallel machines or homogeneous networks of workstations or PCs (see [1, 18, 27] among others). The popular ScaLAPACK library [5] 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 MMM algorithms for homogeneous machines in Section 2.1.

Why extending parallel MMM algorithms to heterogeneous platforms? The answer is clear: future computing platforms are best described by the key-words *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 MMM algorithms on HNOWs, the idea is to make use of *all* available resources, namely slower machines *as well as* more recent ones. This is a challenging but very useful task, given the importance of MMM 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. Inter-nodes communications are typically an order of magnitude slower than intra-nodes communications. The need to design a MMM algorithm which would execute on a collection of clusters is less obvious. Are there actual applications which involve such huge matrices that their 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 [16] and Legion [22]. Huge linear algebra kernels often are at the core of these experiments, so investigating "metacomputing" MMM algorithms is quite natural. Anyway, we view MMM algorithms as a perfect case study for the implementation of tightly-coupled high-performance applications on the metacomputing grid [17]: indeed, such applications are much more difficult to tackle than loosely-coupled cooperative applications. Because MMM 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.

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 MMM problem with different-speed processors turns out to be surprisingly difficult. The main result of this paper is the NP-completeness of the MMM problem on heterogeneous platforms.

The rest of the paper is organized as follows. In Section 2 we summarize existing MMM algorithms for homogeneous platforms, and we discuss how to extend these to cope with heterogeneity. In Section 3 we formally state the MMM optimization problem for heterogeneous platforms, and we establish its NP-completeness (the long and technical proof of this important result is given in

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

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 MMM Algorithms

In this section we briefly describe how to implement a parallel (or distributed) MMM 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 MMM algorithm implemented in the ScaLAPACK library [5] on 2D homogeneous grids. For the sake of simplicity we restrict 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, 18, 27]. 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 $P_{i,*}$.
- 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 $c_{i,j} = c(i,j) + a_{i,k} \times b_{k,j}$.

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 [15, 11]. 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 $p_1 \times p_2$. 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.

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-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 column row, as illustrated in Figure 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.

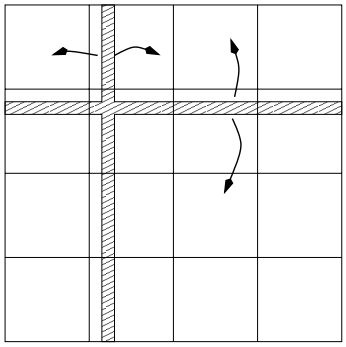


Figure 1: The MMM algorithm on a $4 \times 4$ homogeneous 2D-grid.

On Figure 1, we see that the total amount of communications performed by the MMM algorithm is proportional to the sum of the 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 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 MMM algorithms: for instance Cannon's algorithm [27] (whose main drawback 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 MMM 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-matrix product $C = AB$, we have to tile the $C$ matrix into $p$ non-overlapping rectangles, each rectangle being assigned to one processor. Figure 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 non-independent and maybe

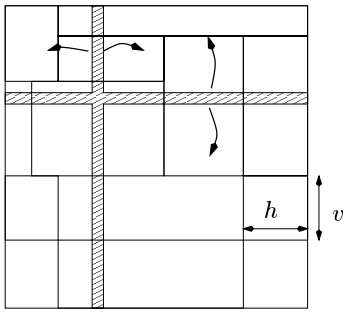



Figure 2: The MMM algorithm on a heterogeneous platform.

conflicting goals: (i) load-balancing computations; (ii) minimizing communication overhead. Goal (i) is related to the area of the rectangles that are allocated to the processors, while goal (ii) 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 MMM Optimization Problem

Consider a matrix-matrix product $C = A \times B$, 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 $s_i \times t_i$ 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 $s_i = h_i \times v_i$, 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 $s_i = h_i \times v_i$, $1 \leq i \leq p$, such that:

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

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.

### 3.2 Communication Overhead

At each step of the MMM 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 columns 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.

Minimizing $\hat{C}$ seems to be a very natural goal, because it represents the total volume of communications. However, other objective functions could be selected, because the target computing platform may influence the way communications are implemented. 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.

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 Figure 3: vertical communications are performed in parallel in all columns, and broadcasts or at least scatters can be performed horizontally.

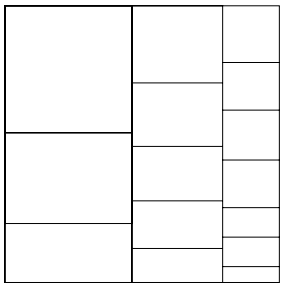


Figure 3: Tiling the unit square into columns of rectangles.

Finally, in a metacomputing context, inter-cluster communications are typically one order of magnitude slower than intra-cluster communications, so we may want to adopt a two-level scheme:

we assign rectangles to clusters as above, while inside each cluster some master-slave mechanism could be provided.

It seems that minimizing the total communication volume is the most important optimization problem, because of its wide potential applicability. Also, forgetting about MMM 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.

### 3.3 The MMM Optimization Problem

We are ready to state the MMM 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 $R_i$ 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** *MMM-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 MMM-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 MMM-OPT(s):

**Lemma 1** *For all solutions of MMM-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, so this lower bound is not always tight. ■

### 3.4 NP-Completeness

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

**Definition 2** *MMM-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 MMM optimization problem:

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

Because the proof is both lengthy and technical, we provide it in the Appendix.

## 4 Related Results

We survey related results 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 MMM-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 [13, 12, 4]. 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.

To the best of our knowledge, there has been little work devoted to the implementation of dense linear algebra kernels on heterogeneous platforms. Extensions of parallel libraries such as ScaLAPACK are not yet available, even for simple HNOWs. Preliminary results on implementing MMM and linear system solvers on a HNOWs are reported in [8, 6, 9]. Load-balancing issues for heterogeneous 2D-grids are studied by Kalinov and Lastovetky [23]: in fact, they arrange the processors into columns. The load is first balanced inside each column independently; next the load is balanced between columns, weighting each column by the inverse of the harmonic mean of the cycle-times of the processors within the column. This leads to the so-called "heterogeneous block cyclic distribution", which ensures a perfect load balancing. It corresponds to a simple solution to conditions (1) and (2) of Section 3.1, but communications are not taken into account, and the number of horizontal neighbors of each processor is not bounded. Boudet et al [7] adopt a different strategy: they enforce the design of a true 2D-grid, where each processor communicates only with its four neighbors. The question is how to arrange the processors so that the load is best balanced? In that case, a perfect load-balancing is possible only if the processor cycle-times can be arranged into a rank-1 matrix. Heuristics are presented in [7] to obtain efficient solutions to that problem.

### 4.2 Problems Similar to MMM-OPT(s)

There are several problems related to MMM-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 by Kong et al. [26, 25]: 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 $\lfloor \frac{n}{m} \rfloor$ or $\lceil \frac{n}{m} \rceil$ 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 $p$ non-overlapping 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 [28, 20, 21], where approximation algorithms are given. The link with our problem is that the objective function is the same, but the original motivation in [20, 21] was a VLSI routing problem (and the constraints are quite different).

- There are several NP-complete geometric optimization problems that are listed in [14]. One example is the minimum rectangle tiling problem [24]: given an $n \times n$ array $A$ of non-negative numbers, and a positive integer $p$, find a partition of $A$ into $p$ non-overlapping 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 MMM-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.

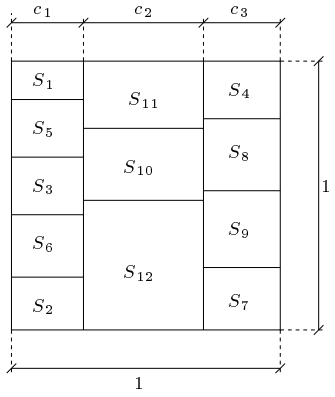



Figure 4: Column-based partitioning of the unit square: $\mathcal{C} = 3$, $k_1 = 5$, $k_2 = 3$ and $k_3 = 4$.

## 5.1 Optimal Column-based Tiling

As outlined in Section 3.3, the MMM-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 $p$ non-overlapping 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 MMM-DEC(s,K) is NP-complete (Section 3.4), we consider the more constrained problem MMM-COL(s) where we impose that the tiling is made up of processor columns, as illustrated in Figure 4. In other words, MMM-COL(s) is the restriction of MMM-OPT(s) to column-based partitions. In this section, we give a polynomial solution to the MMM-COL(s), which will be used as a heuristic to solve MMM-OPT(s).

**Framework** We describe the MMM-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** We describe the tiling algorithm; the optimality proof will be presented later. The main points are the following:

1. Re-index 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 $(\sum_{i=1}^{q} s_i)$ 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 following toy example: we have $p = 8$ areas of values $(0.02, 0.04, 0.06, 0.08, 0.2, 0.2, 0.2, 0.2)$. The results of the algorithm are given in Table 1. 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 3 columns ($f_3(8) = 5.4$). The last column of width $c_3 = s_7 + s_8 = 0.4$ is composed of 2 elements. The second column of width $c_2 = s_5 + s_6 = 0.4$ is also composed of 2 elements. Then the first column of width $c_1 = s_1 + s_2 + s_3 + s_4 = 0.2$ is made of the smallest 4 elements. Figure 5 represents this partitioning.

| | q=1 | q=2 | q=3 | q=4 | q=5 | q=6 | q=7 | q=8 |
|---|---|---|---|---|---|---|---|---|
| $\mathcal{C} = 1$ | 1.02 / 0 | 1.12 / 0 | 1.36 / 0 | **1.8 / 0** | 3 / 0 | 4.6 / 0 | 6.6 / 0 | 9 / 0 |
| $\mathcal{C} = 2$ | | 2.06 / 1 | 2.18 / 2 | 2.4 / 2 | 2.92 / 3 | **3.6 / 4** | 4.6 / 4 | 5.8 / 5 |
| $\mathcal{C} = 3$ | | | 3.12 / 2 | 3.26 / 3 | 3.6 / 4 | 4.12 / 5 | 4.72 / 5 | **5.4 / 6** |
| $\mathcal{C} = 4$ | | | | 4.2 / 3 | 4.46 / 4 | 4.8 / 5 | 5.32 / 6 | 5.92 / 7 |
| $\mathcal{C} = 5$ | | | | | 5.4 / 4 | 5.66 / 5 | 6 / 6 | 6.52 / 7 |
| $\mathcal{C} = 6$ | | | | | | 6.6 / 5 | 6.86 / 6 | 7.2 / 7 |
| $\mathcal{C} = 7$ | | | | | | | 7.8 / 6 | 8.06 / 7 |
| $\mathcal{C} = 8$ | | | | | | | | 9 / 7 |

Table 1: Table containing the values of the couples $f_{\mathcal{C}}(q)/r$ where $f_{\mathcal{C}}(q) = min_{q'} \left(1 + \left(\sum_{q'<i\leq q} s_i\right) \times (q - q') + f_{\mathcal{C}-1}(q')\right) = 1 + \left(\sum_{r<i\leq q} s_i\right) \times (q - r) + f_{\mathcal{C}-1}(r)$. Bold entries correspond to the optimal solution.

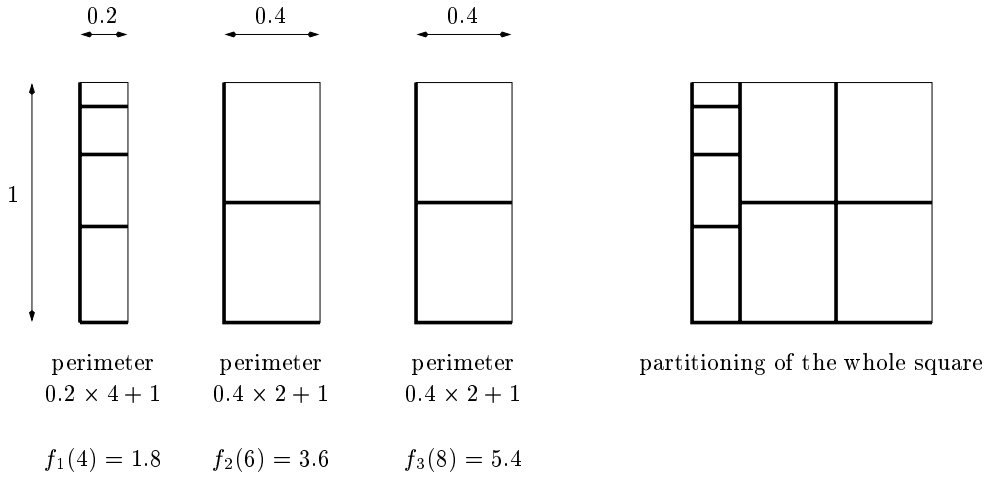



Figure 5: Optimal column-based partitioning for the example. Thicker lines correspond to the sum of the half perimeters.

**Algorithm** The algorithm is outlined as follows:

$S = 0$
**for** q=1 to p
  $S = S + s_q$
  $f_1^{perimeter}(q) = 1 + S \times q$
  $f_1^{cut}(q) = 0$
**endfor**
**for** $\mathcal{C}$=2 to p
  **for** q=$\mathcal{C}$ to p
    $f_{\mathcal{C}}^{perimeter}(q) = \min_{1 \leq r \leq q-\mathcal{C}+1} \left(1 + S \times r + f_{\mathcal{C}-1}^{perimeter}(q-r)\right)$
    $f_{\mathcal{C}}^{cut}(q) = q - r_{opt}$
  **endfor**
**endfor**

The worst-case complexity of the algorithm is $O(p^2 log(p))$: indeed, $f_{\mathcal{C}+1}(q)$ can be built from $f_{\mathcal{C}}$ in $O(\log p)$ steps, since the minimum in the algorithm can be searched by dichotomy: since for each $\mathcal{C}$, $f_{\mathcal{C}-1}$ is an non-decreasing function, $g_{\mathcal{C},q}(r) = 1 + S \times r + f_{\mathcal{C}-1}^{perimeter}(q-r)$ is a convex function of $r$. Hence, the minimum $\min_{1 \leq r \leq q-\mathcal{C}+1}(g_{\mathcal{C},q}(r))$ can be found by dichotomy in $O(\log(q - \mathcal{C} + 1)) = O(\log p)$ steps. 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, the expected cost will be $p\mathcal{C}_{opt} \log(p) \approx p\sqrt{p} \log(p)$.

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 following algorithm:

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

which corresponds to tracking (backwards) the bold entries in Table 1. The unit square is

partitioned into $\mathcal{C}_{opt}$ columns. The $i^{th}$ 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-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 figure 6 for an illustration.

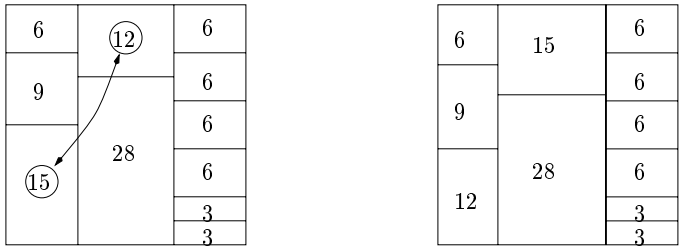



Figure 6: Two partitioning of the same problem instance. The left one is well-ordered while the right one is not.

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 $s_1 \leq s_2 \leq \ldots \leq s_p$ ; $\tau$ is a permutation of $\{1, 2, \ldots, p\}$ such that the $i^{th}$ 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 $1 + k_i \times \sum_{j=d+1}^{d+k_{i-1}} s_{\tau(j)}$. Hence, the total "perimeter" is

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

Since $k_1 \geq k_2 \geq \ldots \geq k_i$, this expression is minimized for $\tau = Identity$ which corresponds to a "well-ordered" partitioning. 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 MMM-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 $LB$. Figure 7 represents 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 in 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.

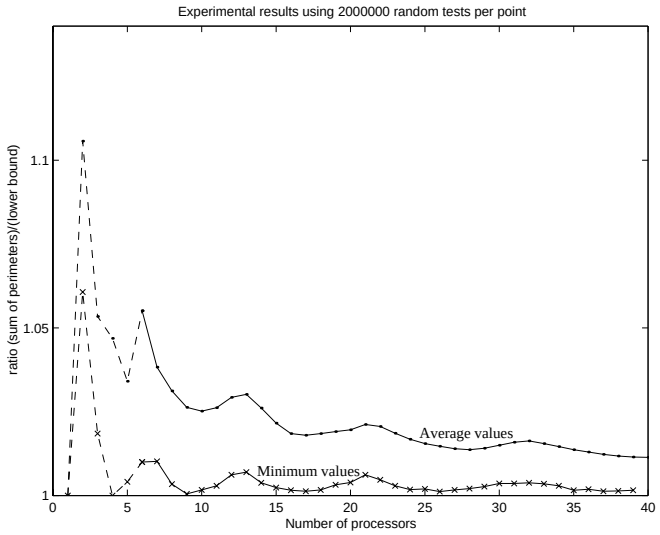



Figure 7: For each number of processors (varying from 1 to 40), 2,000,000 values for the $s_i$ have been randomly generated. For each case, we compute the ratio of the sum $\hat{C}$ of the half perimeters of our partitioning over the absolute lower bound $LB$. The average and minimum values of this ratio are reported in the two curves.

### 5.3 Theoretical Comparison with the Lower Bound

The column-based heuristic appears to be quite satisfactory in practice. Still, the following theoretical questions can be raised:

1. Is this "absolute lower bound" realistic? How far from the actual optimal solution is the optimal column-based algorithm?
2. How can we improve the column-based algorithm?

In this section, we prove that the column-based partitioning is 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:

**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,*

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

**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:

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

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,

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

and thus

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

Therefore,

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

Because the sum $\hat{C}$ of the half-perimeters of the optimal column-based partitioning satisfies to $\hat{C} \leq \hat{C}^*$, this 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.

### 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 6 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 Figure 8.

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 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 Figure 9 we give an example with 2 columns divided into 2 and 3 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 Figure 10. Note that this solution is neither column-based nor recursively defined. This partitioning is optimal amongst all rectangle-based partitionings.

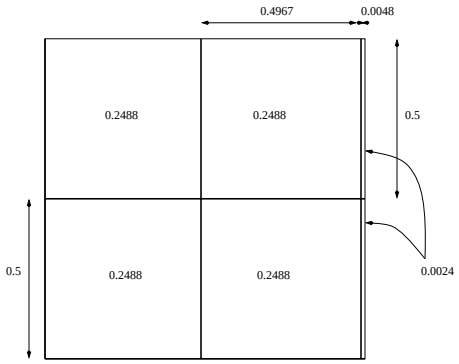



Figure 8: Optimal column-based partitioning. The sum of the half-perimeters is 5.

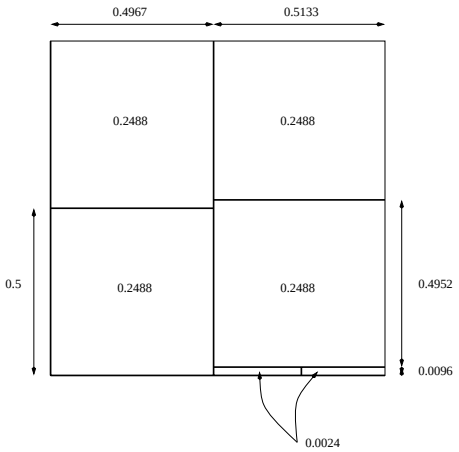



Figure 9: Optimal recursively defined partitioning. The sum of the half-perimeters is 4.51.

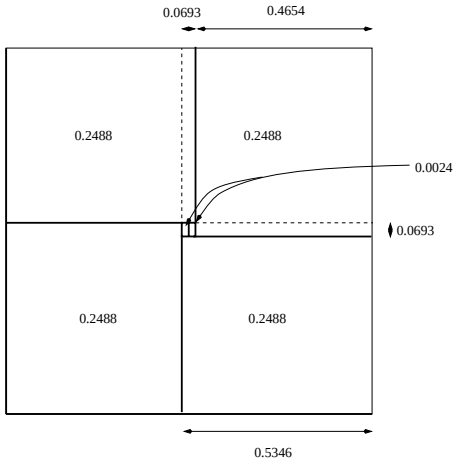



Figure 10: Optimal rectangle-based partitioning. The sum of the half-perimeters is 4.19.

Clearly, the less constrained the partitioning, the better the solution. Note that the improvement over the column-based partitioning can be important, roughly 16% 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.

# 6 MPI Experiments

To provide a preliminary experimental validation of our approach, we have implemented the heterogeneous MMM algorithm using the MPI library [29]. 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 7 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 7 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 Figure 11) with 4 different partitions of the same matrices which are shown in Figure 12.

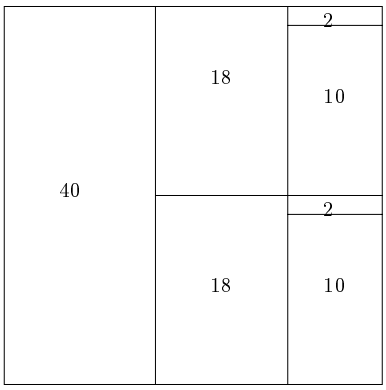
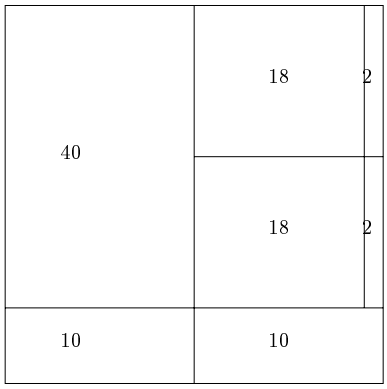



Figure 11: Partition given by the column-based heuristic (Cost $\hat{C} = 5.1$)

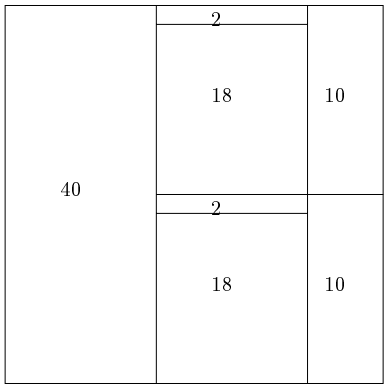
| Partition | n=640 r=32 | n=1280 r=32 | n=1280 r=64 |
|---|---|---|---|
| $\hat{C}$=5.1 | T=33 | T=269 | T=258 |
| $\hat{C}$=5.3 | T=34 | T=287 | T=265 |
| $\hat{C}$=5.4 | T=48 | T=289 | T=264 |
| $\hat{C}$=5.6 | T=34 | T=290 | T=267 |
| $\hat{C}$=6.4 | T=72 | T=367 | T=302 |

Table 2: Average times for a matrix-matrix multiplication.

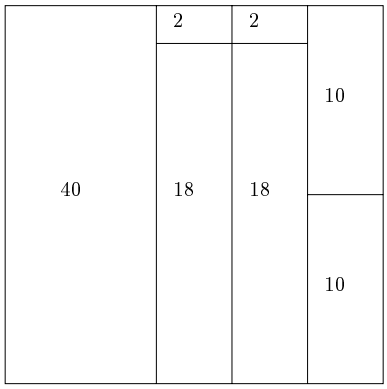
The measures were realized for matrices of size $n = 640$, using a blocksize $r = 32$, and for matrices of size $n = 1280$, using two blocksize values $r = 32$ and $r = 64$. Table 2 gives the average time to compute the MMM product for the five partitions. In the case of a matrix of size $n = 1280$, 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,



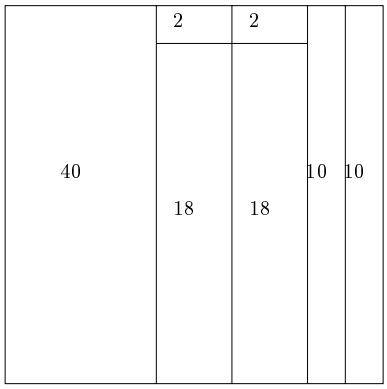

Partition of cost $\hat{C} = 5.3$




Partition of cost $\hat{C} = 5.4$




Partition of cost $\hat{C} = 5.6$




Partition of cost $\hat{C} = 6.4$

Figure 12: Four different column-based partitions

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.

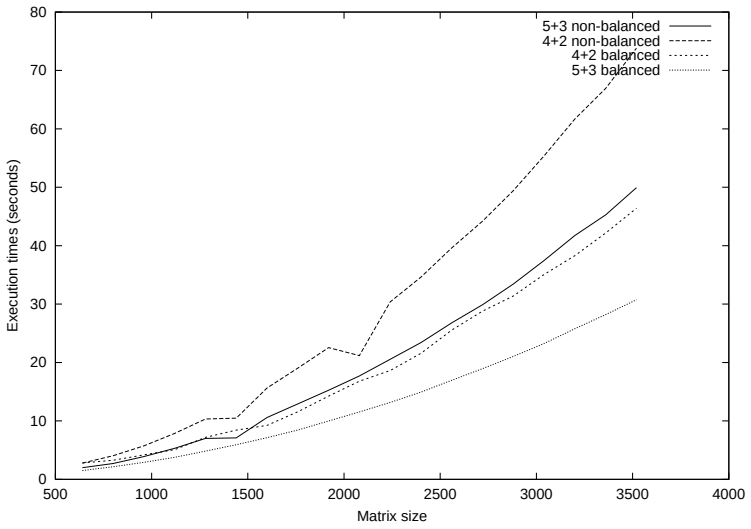



Figure 13: MPI experiments with two clusters.

We use two configurations, one with 5 processors in the first cluster and 3 in the second one, and the other one with only 4 processors in the first cluster and 2 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.

### 6.3 Future work

The preliminary MPI experiments reported in Sections 6.1 and 6.2 are promising. At the very least they fully demonstrate the importance of using a good load-balancing strategy.

Clearly, further and larger experiments must be performed. More experimental results will be provided in the final version of the paper. In particular, we aim at testing a larger collection of clusters with slower inter-cluster links. The Globus system [16] 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.

# 7 Conclusion

In this paper, we have dealt with the implementation of MMM 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.

The MMM algorithm is the prototype of tightly-coupled kernels 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. [22, 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 [10] 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-yet-general-purpose-and-efficient" solutions!

# 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*, 38(6):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. In *Extreme Linux Workshop*, pages 20–24, 1999.

[3] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. More np-complete results for heterogeneous parallel matrix-matrix multiplication. Technical Report RR-2000-XX, LIP, ENS Lyon, 2000. In preparation.

[4] F. Berman. High-performance schedulers. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 279–309. Morgan-Kaufmann, 1999.

[5] 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.

[6] Vincent Boudet, Fabrice Rastello, and Yves Robert. Algorithmic issues for (distributed) heterogeneous computing platforms. In Rajkumar Buyya and Toni Cortes, editors, *Cluster Computing Technologies, Environments, and Applications (CC-TEA'99)*. CSREA Press, 1999. Extended version available as LIP Technical Report RR-99-19.

[7] Vincent Boudet, Fabrice Rastello, and Yves Robert. A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers). In Hamid R. Arabnia, editor, *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*. CSREA Press, 1999. Extended version available as LIP Technical Report RR-99-17.

[8] Pierre Boulet, Jack Dongarra, Fabrice Rastello, Yves Robert, and Frédéric Vivien. Algorithmic issues on heterogeneous computing platforms. In *Clusters and Computational Grids Workshop*. to appear as a journal special issue, 1998. Extended version available as LIP Technical Report RR-98-49.

[9] Pierre Boulet, Jack Dongarra, Yves Robert, and Frédéric Vivien. Static tiling for heterogeneous computing platforms. *Parallel Computing*, 25:547–568, 1999.

[10] H. Casanova and J. Dongarra. Netsolve: A network server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997.

[11] 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 Communications*, 97:1–15, 1996. (also LAPACK Working Note #95).

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

[13] Michal Cierniak, Mohammed J. Zaki, and Wei Li. Scheduling algorithms for heterogeneous network of workstations. *The Computer Journal*, 40(6):356–372, 1997.

[14] P. Crescenzi and V. Kann. A compendium of NP optimization problems. World Wide Web document, URL: `http://www.nada.kth.se/~viggo/wwwcompendium/wwwcompendium.html`.

[15] J. J. Dongarra and D. W. Walker. Software libraries for linear algebra computations on high performance computers. *SIAM Review*, 37(2):151–180, 1995.

[16] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997.

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

[18] G. Fox, S. Otto, and A. Hey. Matrix algorithms on a hypercube i: matrix multiplication. *Parallel Computing*, 3:17–31, 1987.

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

[20] T.F. Gonzalez and S. Zheng. Improved bounds for rectangular and guilhotine partitions. *J. Symbolic Computation*, 7:591–610, 1989.

[21] T.F. Gonzalez and S. Zheng. Approximation algorithm for partitioning a rectangle with interior points. *Algorithmica*, 5:11–42, 1990.

[22] A.S. Grimshaw and W.A. Wulf. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, 1997.

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

[24] S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. In *Proc. 9th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 384–393. ACM Press, 1998.

[25] T.Y. Kong, D.M. Mount, and W. Roscoe. The decomposition of a rectangle into rectangles of minimal perimeter. *SIAM J. Computing*, 17(6):1215–1231, 1988.

[26] T.Y. Kong, D.M. Mount, and M. Wermann. The decomposition of a square into rectangles of minimal perimeter. *Discrete Applied Mathematics*, 16:239–243, 1987.

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

[28] A. Lingas, R.Y. Pinter, R.L. Rivest, and A. Shamir. Minimum edge length partitioning of rectilinear polygons. In *Proc. 20th Ann. Allerton Conference on Communication, Control and Computing*, 1982.

[29] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI the complete reference.* The MIT Press, 1996.

## Appendix: Proof of Theorem 1

**Definition 2** MMM-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** MMM-DEC(s,K) is NP-complete.

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

**Lemma 2**

$$\textit{2P-eq} \leq_P \textit{SSP} \leq_P \textit{MMM-DEC},$$

*where SSP and 2P-eq 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*

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

**Definition 4** *Square-Square-Partition (SSP)*
*Given a set $\mathcal{A} = \{s_1, \ldots, s_p\}$ of $p$ real positive numbers such that $\sum_{i=1}^{p} s_i = 1$, is there a partition of the unit square into $p$ squares $S_i$ of area $s_i$?*

Since 2P-eq is known to be NP-Complete [19], Lemma 2 will complete the proof of Theorem 1.

### 7.1 Reduction: SSP $\leq_P$ MMM-DEC(s,K)

We start by proving the easy part of Lemma 2, i.e. SSP $\leq_P$ MMM-DEC(s,K). Let $\mathcal{A} = \{s_1, \ldots, s_p\}$ be a set of $p$ real positive numbers s.t. $\sum_{i=1}^{p} s_i = 1$. Solving SSP is equivalent to solving MMM-DEC(s,K) with

$$K = 2\sum_{i=1}^{p} \sqrt{s_i}$$

and therefore,

$$\text{SSP} \leq_P \text{MMM-DEC}.$$

### 7.2 Reduction: 2P-eq $\leq_P$ SSP

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 $n > 400$ without loss of generality. We have to polynomially transform this instance into an instance of the SSP problem which has a solution iff the original instance of 2-Partition-Equal has one solution.

Define $\{b_1, \ldots, b_n\}$ as $\forall i, b_i = 2(a_i + 2n \max_k a_k)$. Thus, $b_i \geq \frac{\max_k b_k}{2}$, $b_i$ is even. Moreover, if we let $M = \max_k b_k$ and $S = \frac{\sum_i b_i}{2}$, then $S \geq 100M$. We build the following instance of the SSP problem (SSP$(b_1, \ldots, b_n)$): is there a partition of the unit square into $14 + n + \sum_k b_k(M - b_k)$ squares of respective area

$$\frac{(13S+11M)^2}{A} \quad (\times 1), \quad \frac{(7S+6M)^2}{A} \quad (\times 3), \quad \frac{(4S+3M)^2}{A} \quad (\times 2), \quad \frac{(3S+3M)^2}{A} \quad (\times 2),$$
$$\frac{(3S+2M)^2}{A} \quad (\times 2), \quad \frac{(2S+2M)^2}{A} \quad (\times 4), \quad \frac{b_i^2}{A} \quad (\forall i), \quad \frac{1}{A} \quad (\times \sum_k b_k(M - b_k)),$$

where $A = (20S + 17M)^2$ ?

For convenience, in what follows, we consider the (equivalent) scaled problem: is there a partition of the $(20S + 17M) \times (20S + 17M)$ square into $14 + n + \sum_k b_k(M - b_k)$ squares of respective area

$$\begin{aligned}
A_{13,11} &= (13S+11M)^2 \quad (\times 1),\\
A_{7,6} &= (7S+6M)^2 \quad (\times 3),\\
A_{4,3} &= (4S+3M)^2 \quad (\times 2),\\
A_{3,3} &= (3S+3M)^2 \quad (\times 2),\\
A_{3,2} &= (3S+2M)^2 \quad (\times 2),\\
A_{2,2} &= (2S+2M)^2 \quad (\times 4),\\
A_{bi} &= b_i^2 \quad (\forall i, 1 \leq i \leq n),\\
A_{1,1} &= 1 \quad (\times \sum_k b_k(M - b_k))?
\end{aligned}$$

In what follows, we prove that such a partition is necessarily the one depicted in Figure 14, where the two small $M \times S$ rectangle areas are shown by arrows in Figure 14 and fully described in Figure 15. The intuitive idea of the proof is the following: the large squares are used to prevent the two small $M \times S$ rectangle areas to be neighbors. Hence these areas must be filled separately by the remaining squares, those of area $b_i$ and those of area 1. This will be possible iff there the

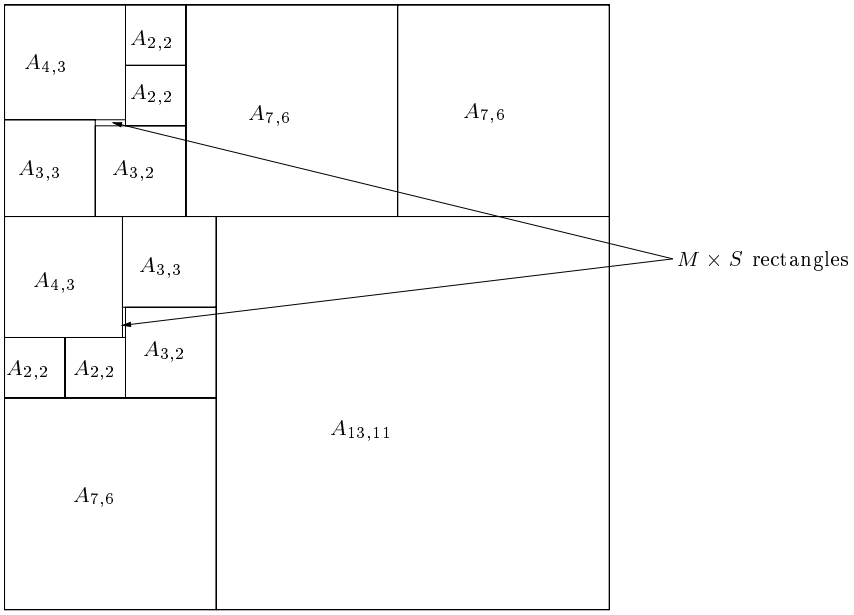



Figure 14: General position of the squares

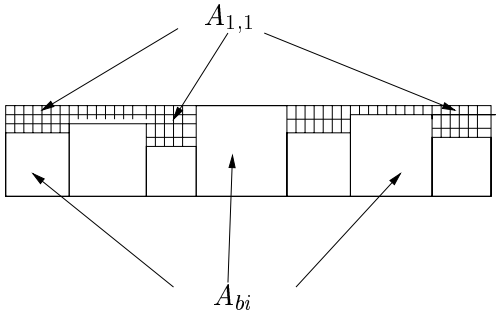



Figure 15: Zoom on the $M \times S$ rectangle areas

$b8i$ can be partitioned into two subsets of same cardinal and same sum. The number of squares of area 1 is computed so as to fill the holes and obtain a true tiling of the whole area.

### 7.2.1 Position of the Largest Four Squares

The general position of the largest four squares is shown in Figure 16a. Obviously, if we can tile the remaining area with the remaining squares, this will also be the case for the configuration shown in Figure 16b. Therefore, from now on, we assume (without loss of generality) that the largest four rectangles are arranged as shown in Figure 16b.

### 7.2.2 Tiling the Remaining Surface

Now we discuss the tiling of the remaining surface (the white area of Figure 16b). We give all dimensions in Figure 17). In the following figures, $A_{x,y}$ denotes a square of size $xS + yM$. We proceed by an exhaustive case analysis:

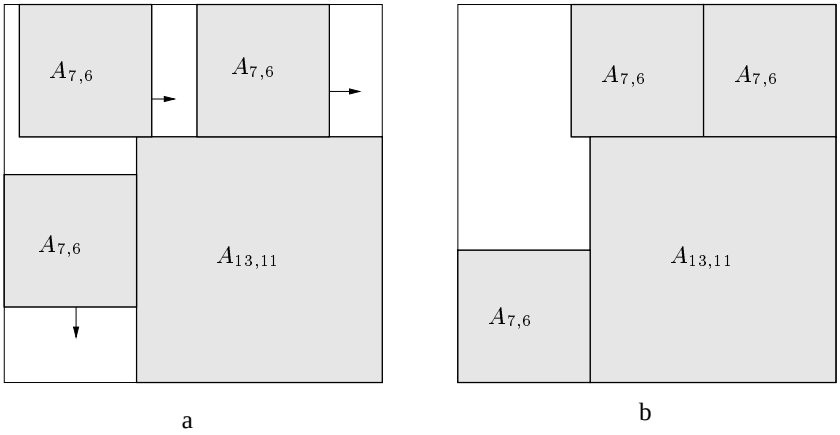



Figure 16: General position of the largest four squares

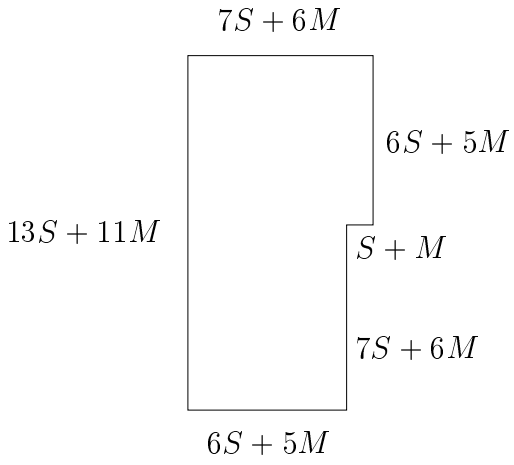



Figure 17: Remaining surface

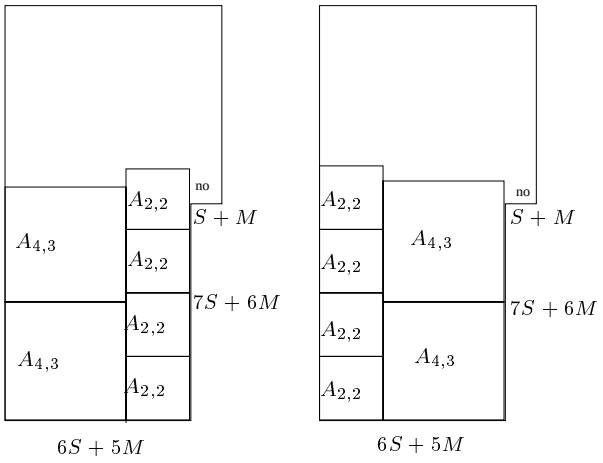



Figure 18: Some impossible configurations with a $4S+3M$ square.

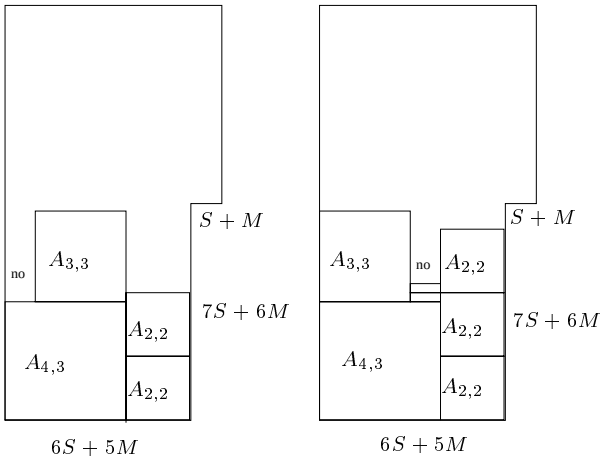



Figure 19: Some impossible configurations with a $4S+3M$ square.

- First case: we start by tiling the $6S+5M$ basis with a $4S+3M$ square.
  The different situations to consider are shown in Figures 18, 19 and 20 respectively. The only correct configuration is the one depicted in Figure 20.
- Second case: we start by tiling the $6S+5M$ basis with a $3S+3M$ square.
  The different situations to consider are shown in Figures 21, 22 and 23. The only correct configuration is the one depicted in Figure 23.
- Moreover, any solution requires either a $4S+3M$ square or a $3S+3M$ square to be on the $6S+5M$ basis.

Therefore, Figures 20 and 23 describe the only two possibilities to start the tiling of the remaining surface described in Figure 17. By symmetry, we can complete these partial tilings into the solution described in Figure 24 (other equivalent solutions are also possible).

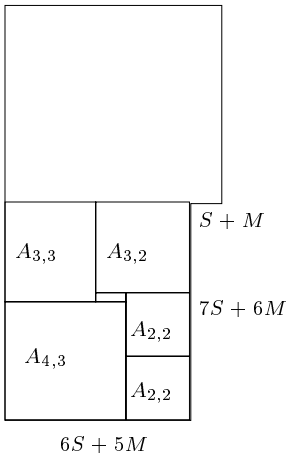



Figure 20: The only correct configuration with a $4S + 3M$ square.

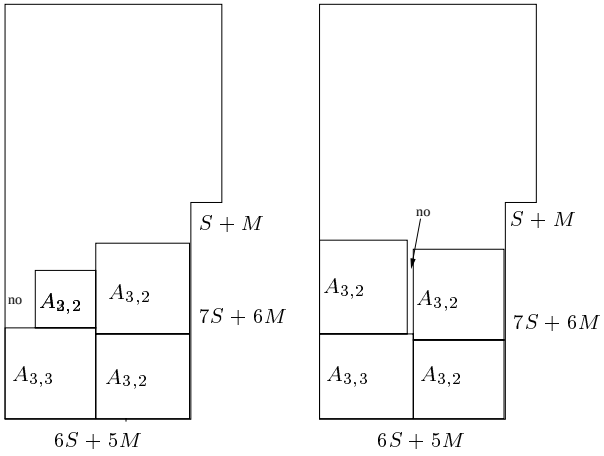



Figure 21: Some impossible configurations with a $3S + 3M$ square.

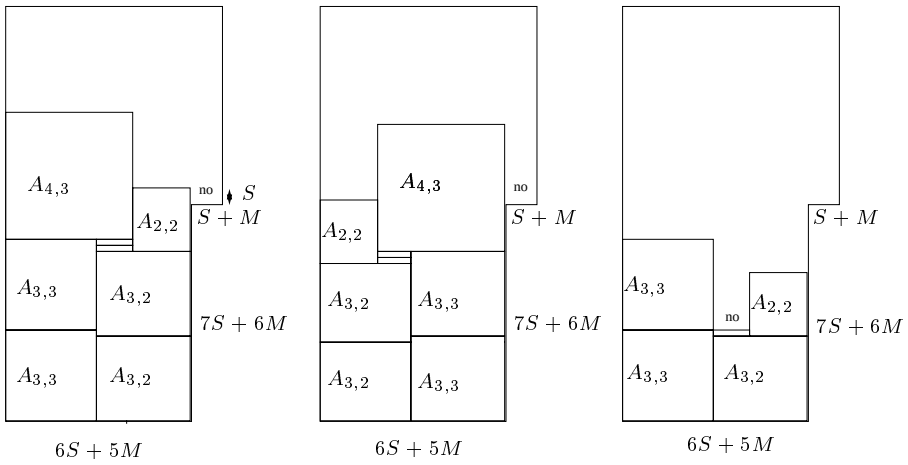



Figure 22: Some impossible configurations with a $3S + 3M$ square.

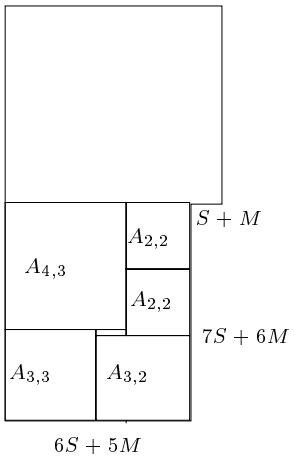



Figure 23: The only correct configuration with a $3S + 3M$ square

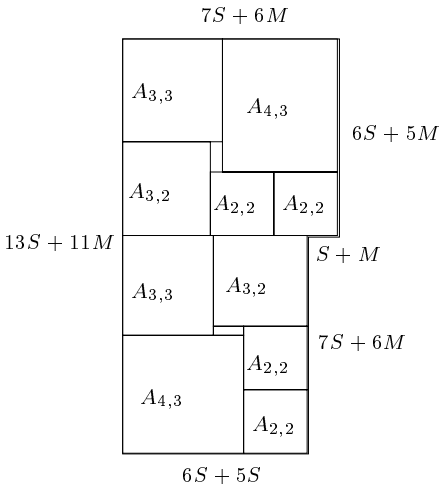



Figure 24: One possible tiling of the remaining surface.

### 7.2.3 Partial conclusion

We have proved that any tiling of the remaining surface (see Figure 17) is similar to the one depicted in Figure 24: after using all the large rectangles $A_{x,y}$, there remains two non-adjacent rectangle areas of area $M \times S$ to be tiled. Therefore, we can solve the SSP problem iff we can tile these two areas with the remaining squares, i.e. $n$ squares of area $b_i^2, 1 \leq i \leq n$, and $\sum_k b_k(M - b_k)$ squares of area 1. Since $\min_k b_k \geq \frac{\max b_k}{2}$ and $\sum_k b_k = 2S$, one can easily check that both $M \times S$ rectangle areas have to be tiled as depicted in Figure 15. Therefore, our instance of the SSP problem has a solution iff there exists a partition of $\{b_1, \ldots, b_n\}$ into two subsets of same sum.

### 7.2.4 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 of the 2-Partition-Equal problem has a solution.

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

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

Recall that $b_k = 2(a_k + 2n \text{ MAX})$, where $\text{MAX} = \max_k a_k$. Then,

$$\begin{aligned} \sum_{k \in \mathcal{A}_1} b_k &= \sum_{k \in \mathcal{A}_1} a_k + 2n \text{ MAX } \text{card}(\mathcal{A}_1) \\ &= \sum_{k \in \mathcal{A}_2} a_k + 2n \text{ MAX } \text{card}(\mathcal{A}_2) \\ &= \sum_{k \in \mathcal{A}_2} b_k \end{aligned}$$

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

$$\sum_{k \in \mathcal{A}_1} b_k = \sum_{k \in \mathcal{A}_2} b_k.$$

Thus,

$$\begin{aligned} \sum_{k \in \mathcal{A}_1} a_k &= \sum_{k \in \mathcal{A}_1} b_k - 2n \text{ MAX } \text{card}(\mathcal{A}_1) \\ \sum_{k \in \mathcal{A}_2} a_k &= \sum_{k \in \mathcal{A}_2} b_k - 2n \text{ MAX } \text{card}(\mathcal{A}_2) \\ \sum_{k \in \mathcal{A}_1} a_k - \sum_{k \in \mathcal{A}_2} a_k &= 2n \text{ MAX } \text{card}(\mathcal{A}_2 - \mathcal{A}_1) \end{aligned}$$

Moreover, since

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

we obtain

$$\text{card}(\mathcal{A}_1) = \text{card}(\mathcal{A}_2) \text{ and } \sum_{k \in \mathcal{A}_1} a_k = \sum_{k \in \mathcal{A}_2} a_k$$

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

### 7.3 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 SSP problem has a size polynomial in the size of the original instance of the 2-Partition-Equal problem.

**Lemma 3** *Define* $MAX = \max_k a_k$ *as above, and let* $c(a)$ *and* $c(b)$ *denote respectively the encoding of the data* $a$ *and* $b$*. Then,*

$$Length(c(b)) = O(Length(c(a))^2).$$

**Proof**

$$\text{Length}(c(a)) = \sum_k \log(a_k) \geq \log(\text{MAX}) + (n-1)\log(\min_k a_k) \geq (n-1)\log 2 + \log \text{MAX}.$$

$$\text{Length}(c(b)) = \sum_k \log(b_k) = \sum_k \log(2n\ \text{MAX}(1 + \frac{a_k}{n\text{MAX}})) \leq 1 + n(\log n + \log 2) + n \log \text{MAX}.$$

Therefore,

$$\text{Length}(c(b)) = O(\text{Length}(c(a))^2).$$

■

This achieves the proof of the NP-completeness of SSP, and therefore the NP-completeness of MMM-DEC. ■