Heterogeneous Matrix-Matrix Multiplication or Partitioning a Square into Rectangles: NP-Completeness and Approximation Algorithms

O. Beaumont* V. Boudet* A. Legrand* F. Rastello* Y. Robert*

Abstract

In this paper, we deal with two geometric problems arising from heterogeneous parallel computing: how to partition the unit square into p rectangles of given area $s_1, s_2, \ldots, s_p$ (such that $\sum_{i=1}^{p} s_i = 1$), so as to minimize (i) either the sum of the p perimeters of the rectangles (ii) or the largest perimeter of the p rectangles. For both problems, we prove NP-completeness and we introduce approximation algorithms.

1 Introduction

In this paper, we deal with two simple geometric problems: how to partition the unit square into p rectangles of given area $s_1, s_2, \ldots, s_p$ (such that $\sum_{i=1}^{p} s_i = 1$), so as to minimize

- either the sum of the p half perimeters of the rectangles,
- or the largest half perimeter of the p rectangles.

Note that there always exist solutions to these problems: e.g. tile the unit square into p horizontal slices of height $s_1, s_2, \ldots, s_p$. The difficulty is to minimize the objective function.

Consider the following example with $p = 5$ rectangles $R_1, \ldots, R_5$ of areas $s_1 = 0.36$, $s_2 = 0.25$, $s_3 = s_4 = s_5 = 0.13$. A possible partition is shown in Figure 1. The size of each rectangle is the following: $0.61 \times \frac{36}{61}$ for R_1, $0.61 \times \frac{25}{61}$ for R_2, and $0.39 \times \frac{1}{3}$ for R_3, R_4, and R_5. The maximum half-perimeter is that of R_1, approximately 1.2002, which is very close to the absolute lower bound 1.2 obtained when the largest rectangle is a square (this is not achievable in this example). As for the second objective function, we compute that the sum of the half-perimeters is 4.39, while the absolute lower bound is $\sum_{i=1}^{p} 2\sqrt{s_i} \approx 4.36$ (obtained when all rectangles are squares, which is not achievable in this example either). Hence the partition turns out to be very satisfactory for both objective functions. The geometric interpretation for the sum of the half-perimeters is nice: it is the length of the lines drawn to make the partition, plus 2 corresponding to the right and bottom edge of the unit square.

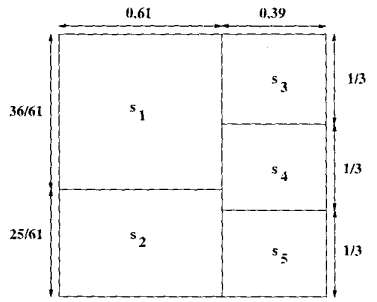


Figure 1. A simple example with 5 rectangles.

The main results of the paper are the proof of NP-completeness and approximation algorithms for both optimization problems. Beforehand, we explain the initial motivation for this work, which arises from minimizing communications in the design of parallel algorithms targeted to heterogeneous platforms. The rest of the paper is organized as follows. In Section 2 we explain the motivation from heterogeneous parallel computing. In Section 3 we formally state the optimization problems PERI-SUM (minimize the sum of the perimeters of the rectangles) and PERI-MAX (minimize the largest perimeter), and we establish their NP-completeness. Section 4 is devoted to the design of approximation algorithms for PERI-SUM; Section 5 is its counterpart for PERI-MAX. To demonstrate the practical usefulness of these heuristics, some MPI experi-

*LIP, UMR CNRS-ENS Lyon-INRIA 5668, Ecole Normale Supérieure de Lyon, 46, Allée d'Italie, 69364 Lyon Cedex 07, France. E-mail: Firstname.Lastname@ens-lyon.fr

1066-6192/01 $10.00 © 2001 IEEE

ments are reported in Section 6. In Section 7 we briefly survey related optimization problems. We give some final remarks and conclusions in Section 8.

2 Problem Motivation

The motivation for this work is the design of parallel Matrix Multiplication (MM for short) algorithms targeted to heterogeneous platforms, such as heterogeneous clusters of workstations, or collections of such clusters. Parallel MM algorithms work as follows: let $C = A \times B$ the product to be computed, where A and B are square matrices of size $n \times n$. First, granularity is increased: matrix blocks rather than elementary matrix coefficients are allocated to processors, as in the ScaLAPACK library [4]. Hence, 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. Assume there are p processors. The three matrices A, B and C are partitioned into p (superposed) rectangles. There is a one-to-one mapping between these rectangles and the processors. Each processor is responsible for updating its rectangle: at each step, one pivot column and one pivot row are communicated to all processors, and independent updates take place; more precisely, each processor updates each block in its rectangle with one block from the pivot row and one block from the column row, as illustrated in Figure 2.

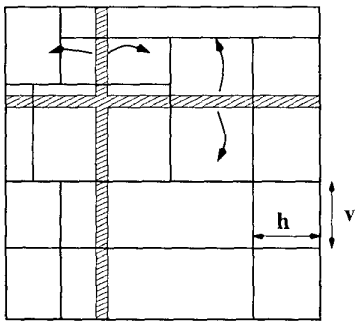


Figure 2. The MM algorithm on a heterogeneous platform.

Using different-speed processors, 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. In Figure 2, we have 13 different-speed computing resources. We let s_i the fraction of the total computing power represented by processor P_i, $1 \leq i \leq p$. Normalizing processor speeds, we have $\sum_{i=1}^{p} s_i = 1$. Normalizing the computing workload accordingly, we have to tile the unit square into p rectangles R_i of prescribed area s_i, $1 \leq i \leq p$. The question is: how to compute the *shape* of these p rectangles so as to minimize the total execution time?

Let $h_i \times v_i$ be the size of rectangle R_i, where $h_i v_i = s_i$. 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 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, 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. For instance it is natural to assume that communications will be mostly sequential on a heterogeneous network of workstations 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, 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 minimize the maximal amount of communications to be performed by each processor, so that the objective function becomes $\hat{M} = \max_{1 \leq i \leq p}(h_i + v_i)$.

Once a solution to either optimization problem has been found, we derive the allocation of data elements to processor P_i by rounding up the values $n \times h_i$ and $n \times v_i$. Finally, note that both optimization problems have a wide potential applicability. Forgetting about MM algorithms, 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, or the maximal amount of communications performed by one processor, while load-balancing the work, amounts to solving exactly the same optimization problems.

3 NP-Completeness

We formally state both optimization problems. We have to determine p rectangles R_i, of prescribed area s_i, $1 \leq i \leq p$ 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 solve the following optimization problems:

Definition 1
- *PERI-SUM(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.*
- *PERI-MAX(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 $\hat{M} = \max_{1 \leq i \leq p}(h_i + v_i)$ is minimized.*

There is an obvious lower bound for PERI-SUM(s) and for PERI-MAX(s):

Lemma 1 *For all solutions of PERI-SUM(s), $\hat{C} \geq 2\sum_{i=1}^{p} \sqrt{s_i}$. For all solutions of PERI-MAX(s), $\hat{M} \geq 2\max_{1 \leq i \leq p} \sqrt{s_i}$.*

The decision problems associated to the optimization problems PERI-SUM and PERI-MAX are the following:

Definition 2
- *PERI-SUM(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 $\sum_{i=1}^{p}(h_i + v_i) \leq K$?*
- *PERI-MAX(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 $\max_{1 \leq i \leq p}(h_i + v_i) \leq K$?*

Our main result states the intrinsic difficulty of the PERI-SUM and PERI-MAX optimization problems:

Theorem 1 *PERI-SUM(s,K) and PERI-MAX(s,K) are NP-complete.*

The proof is provided in [3, 15]. More important than the proof, the theorem itself clearly demonstrates the intrinsic difficulty of static load-balancing on heterogeneous platforms while minimizing communication cost.

4 Approximation Algorithms for PERI-SUM

There are several "natural" heuristics to approximate PERI-SUM. However, proving approximation bounds turns out to be very technical. We start in Section 4.1 with a column-based heuristic, very simple to implement, and which appears very efficient through extensive experimental comparisons. However, we have not been able to give a tight approximation bound: the bound of Section 4.1.3 depends on the relative size of the rectangles to be used in the tiling. In Section 4.2 we move to a recursive heuristic, much more complicated to describe, but for which a nice approximation bound is provided.

4.1 Column-Based Heuristic

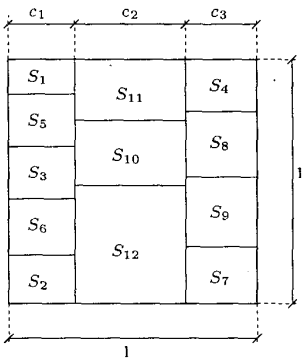


Figure 3. Column-based partitioning of the unit square: $C = 3$, $k_1 = 5$, $k_2 = 3$ and $k_3 = 4$.

4.1.1 Description

Since PERI-SUM(s) is NP-complete, we consider the more constrained problem COL-PERI-SUM(s) where we impose that the tiling is made up with processor columns, as illustrated in Figure 3. In other words, COL-PERI-SUM(s) is the restriction of PERI-SUM(s) to column-based partitions. In this section, we give a polynomial solution to COL-PERI-SUM(s), which will be used as a heuristic for PERI-SUM(s).

Framework We describe the COL-PERI-SUM(s) problem more formally: we aim at tiling the unit square into C columns (where C is yet to be determined) of width $c_1, \ldots, c_C$. 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 $\sum_{i=1}^{C} k_i = p$ 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 The main points of the column-based tiling 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_C, by incrementing the value of C from 1 to the desired value For $q \in \{1, \ldots, p\}$, $f_C(q)$ represents the total perimeter of an optimal column-based partitioning of a rectangle of height 1 and width $(\sum_{i=1}^{q} s_i) \times 1$ into q rectangles of respective area $s_1, \ldots, s_q$, using C columns.

To help understand the derivation, we apply the algorithm on the following example: 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. Each column 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.

	q=1	q=2	q=3	q=4
$C = 1$	1.05 / 0	1.2 / 0	**1.54 / 0**	2.12 / 0
$C = 2$		2.1 / 1	2.28 / 2	2.56 / 2
$C = 3$			3.18 / 2	3.38 / 3
$C = 4$				4.28 / 3

	q=5	q=6	q=7	q=8
$C = 1$	2.9 / 0	4 / 0	5.9 / 0	9 / 0
$C = 2$	2.94 / 3	**3.5 / 3**	4.38 / 4	5.76 / 5
$C = 3$	3.66 / 4	4 / 4	4.58 / 5	**5.5 / 6**
$C = 4$	4.48 / 4	4.78 / 5	5.2 / 6	5.88 / 7
$C = 5$	5.38 / 4	5.6 / 5	5.98 / 6	6.5 / 7
$C = 6$		6.5 / 5	6.8 / 6	7.28 / 7
$C = 7$			7.7 / 6	8.1 / 7
$C = 8$				9 / 7

Table 1. Table containing the values of the couples $f_C(q)/r$ where $f_C(q) = \min_{r \in [C-1, q-1]} \left(1 + \left(\sum_{r<i\leq q} s_i\right) \times (q - r) + f_{C-1}(r)\right)$ and r is the value minimizing the previous expression. Bold entries correspond to the optimal solution.

In the example, the optimal partitioning is obtained for 3 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 of 3 elements. Then the last column of width $c_1 = s_1 + s_2 + s_3 = 0.18$ is made up with the smallest 3 elements.

Algorithm The algorithm is outlined as follows ($f_C^{perimeter}$ corresponds to the $f_C(q)$ previously used. $f_C^{cut}(q)$ corresponds to the total number of blocs in the first $C - 1$ columns. So that, there remains $q - f_C^{cut}(q)$ blocs in the column C):

$S = 0$
for q=1 to p
  $S = S + s_q$
  $f_1^{perimeter}(q) = 1 + S \times q$
  $f_1^{cut}(q) = 0$
endfor
for C=2 to p
  for q=C to p
    $f_C^{perimeter}(q) = \min_{r \in [C-1, q-1]} 1 + \left(\sum_{r<i\leq q} s_i\right) \times (q - r) + f_{C-1}(r)$
    $f_C^{cut}(q) = r_{min}$
  endfor
endfor

Optimality This algorithm provides the optimal column-based partitioning. The proof is detailed in [2, 15]. The worst-case complexity of the algorithm is $O(p^3)$.

4.1.2 Experimental Comparison with the Lower Bound

As shown in Section 3, a lower bound for the sum $\hat{C}$ of the half-perimeters is twice the sum of the square roots of the areas, i.e. $LB = 2\sum_{i=1}^{p} \sqrt{s_i}$. Of course this bound cannot always be met: consider an instance of PERI-SUM(s) with only two rectangles, $s_1 = 1 - \epsilon$ 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 4 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.

4.1.3 Theoretical Comparison with the Lower Bound

In this section, we prove that the column-based partitioning is a good approximation, especially when the ratio between $\max s_i$ and $\min s_i$ is small:

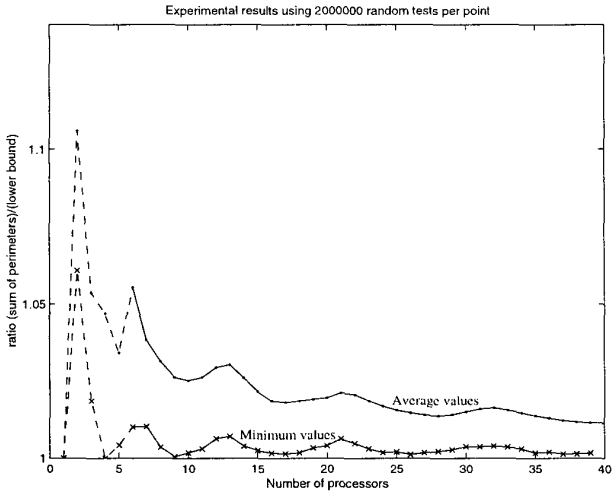


Figure 4. For each number of processors (varying from 1 to 40), 2,000,000 values for the s_i have been generated. For each case, we compute the ratio of the sum $\hat{C}$ of the half perimeters of our partitioning over the absolute lower bound LB. The worst case is a constant value equal to 1.5. The average and best cases are reported in the two curves.

Proposition 1 *Let $r = \frac{\max s_i}{\min s_i}$, let $\hat{C}$ denote the sum of the half perimeters of the rectangles obtained with the optimal column-based partitioning, and let $LB = 2\sum_{i=1}^{p}\sqrt{s_i}$. Then,*

$$\frac{\hat{C}}{LB} \leq \sqrt{r}(1 + \frac{1}{\sqrt{p}})$$

The proof can be found in [2]. It is straightforward when evaluating the cost of a very simple partitioning (with about $\sqrt{p}$ columns and $\sqrt{p}$ elements per column). 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. if one rectangle is much larger than another, the bound is *very* pessimistic.

4.2 Recursive Heuristic

We have derived a recursively defined heuristic that lead to a good approximation factor: letting $\hat{C}$ denote the sum of the half perimeters of the rectangles obtained with this heuristic, and $LB = 2\sum_{i=1}^{p}\sqrt{s_i}$, we have

$$\hat{C} \leq 1 + \frac{5}{4}LB$$

The construction of the heuristic, together with the proof of the approximation factor, are available in [3, 15].

5 Approximation algorithms for PERI-MAX

In this section, we introduce a polynomial heuristic to solve the PERI-MAX problem. Again, we consider a column based partitioning of the unit square. We consider two different heuristics, according to the area of the largest rectangle. Let $s_1 \geq s_2 \ldots \geq s_p$ denote the given areas of the rectangles.

If s_1 is greater than $\frac{1}{3}$, we use a first heuristic. In this case, one column is created for each element. Therefore, the half-perimeter of one rectangle of area s_i is $1 + s_i$. In this case,

$$\forall 1 \leq i \leq p, r_i = \frac{1 + s_i}{2\sqrt{s_1}} \leq r_1 \leq \frac{2}{\sqrt{3}}.$$

In the case where s_1 is less than $\frac{1}{3}$, we use a second heuristic, which ensures that

$$\forall 1 \leq i \leq p, r_i = \frac{h_i + v_i}{2\sqrt{s_1}} \leq \frac{2}{\sqrt{3}}.$$

The algorithm can be stated as follows:

Peri-max_column-based $(p, S = (s_1, \ldots, s_p))$
$c = 1$
for $i = 1$ **to** p
Scol(c)=Scol(c) $\bigcup\{i\}$
if $\sum_{i \in Scol(c)} s_i \geq$
$2\sqrt{\frac{s_1}{3}} - \sqrt{\frac{4s_1}{3} - \max_{i \in Scol(c)} s_i}$
$c = c + 1$
endif
endfor
$c_{max} = c$
if $\sum_{i \in Scol(c_{\max})} s_i \leq$
$2\sqrt{\frac{s_1}{3}} - \sqrt{\frac{4s_1}{3} - \max_{i \in Scol(c_{\max})} s_i}$
$Scol(1) = Scol(1) \bigcup Scol(c_{max})$
$c_{max} = c_{max} - 1$
endif
endPeri-max_column-based

The configuration of one of the columns Scol is depicted in Figure 5. The largest perimeter of the rectangles in the column Scol(c) is

$$\sum_{i \in Scol(c)} s_i + \frac{\max_{i \in Scol(c)} s_i}{\sum_{i \in Scol(c)} s_i}.$$

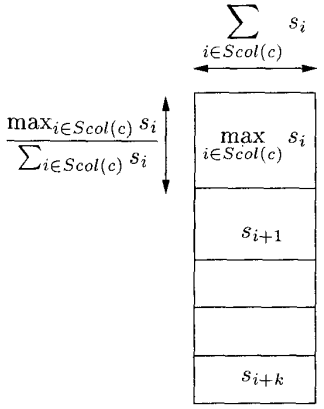


Figure 5. Scol(c).

Proposition 2 *Let $\hat{M}$ denote the maximum of the half perimeters of the rectangles obtained with the above heuristic, and let $LB = 2\sqrt{s_1}$. Then,*

$$\frac{\hat{M}}{LB} \leq \frac{2}{\sqrt{3}}$$

The proof is available in [3, 15]. Note that it is impossible to obtain a better guarantee (without taking into account the actual values of the s_i's). Indeed, if we consider the following situation with $s_1 = s_2 = s_3 = \frac{1}{3}$, then the optimal solution satisfies to

$$\hat{M} = \max_i(h_i + v_i) = \frac{2}{\sqrt{3}} 2\sqrt{s_1} = \frac{2}{\sqrt{3}} LB.$$

6 Preliminary Experimental Results Using MPI

To demonstrate the practical usefulness of our heuristics, some experiments have been conducted on an heterogeneous network of workstations using MPICH. Figure 6 shows the average execution time of the matrix-matrix multiplication algorithm for various matrix sizes and for different numbers of machines. Two Pentium III 550 MHz and two Pentium MMX 200 MHz linked by an Ethernet network have been used in the experiments reported in Figure 6(a). Heterogeneous distributions are obviously much better than classical homogeneous ones. The column-based approximation of the PERI-SUM heuristic leads to the best results. In the experiments reported in Figure 6(b), two additional Pentium II 350 have been used and, as communications are getting more important, the difference between 1D and 2D distributions begins to appear. Both homogeneous and heterogeneous 1D distributions are worse than 2D distributions. Lastly, for the Figure 6(c) experiments, a wider variety of machines (1 P-MMX 200 MHz, 2 P-II 350 MHz, 2 P-II 400 MHz, 3 P-II 500 MHz, and 2 P-III 550 MHz) has been used. In this case, optimizing communications gets crucial and the superiority of 2D distributions over 1D distributions becomes obvious. The PERI-SUM heuristic still leads to the best results.

7 Related Results

In this section, we survey results on geometric optimization problems similar to PERI-SUM or PERI-MAX:

Covering a square by small perimeter rectangles Alon and Kleitman [1] consider the tiling of the unit square into n rectangles. There is no constraint on the area of the rectangles. They show that one of the rectangle must have perimeter at least $4(2m + 1)/(n + m(m + 1))$, where m is the largest integer whose square is at most n. This result is exact for $n = m(m + 1)$ or $n = m^2$.

Decomposition of a square into rectangles of minimal perimeter Kong et al. [13] determine how to tile the unit square into p rectangles of same area so as to minimize the maximum perimeter of these rectangles. This is exactly our PERI-MAX problem constrained to same-area rectangles

($s_i = 1/p$ for $1 \leq i \leq p$). This problem is shown to be polynomial in [13]. 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 solution is extended to deal with the decomposition of a rectangle (instead of a square) onto same-area rectangles in [12].

Partitioning a rectangle with interior points 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 [14] and approximation algorithms are given in [6, 7]. The link with our PERI-MAX problem is that the objective function is the same, but the original motivation in [6, 7] was a VLSI routing problem (and the constraints are quite different).

Array partitioning The minimum rectangle tiling problem [10] is partially related to our optimization problems PERI-MAX: 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). This problem is NP-complete, and approximation algorithms are given in [11, 10]

Finally, note that Crandall and Quinn [5], Kaddoura, Ranka and Wang [8] and Kalinov and Lastovetky [9] have proposed several heuristics for a formulation of the heterogeneous matrix-matrix multiplication problem that is very similar to the PERI-MAX problem. Both papers report several numerical simulations. However, we are not aware of any theoretical result, nor of any approximation bound stating some performance guarantee.

8 Conclusion

In this paper, we have dealt with two geometric problems arising from heterogeneous parallel computing. Because both problems have been shown NP-complete, we have introduced approximation algorithms. Preliminary MPI experiments demonstrate the practical usefulness of these heuristics. The original motivation for this work is very important: the MM 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 computational grid.

References

[1] N. Alon and D. Kleitman. Covering a square by small perimeter rectangles. *Discrete Computational Geometry*, 1:1–7, 1986.

[2] 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. Short version appears in the proceedings of ICPP'2000.

[3] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. Partitioning a square into rectangles: NP-completeness and approximation algorithms. Technical Report RR-2000-10, LIP, ENS Lyon, Feb. 2000.

[4] 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.

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

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

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

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

[9] 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.

[10] 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.

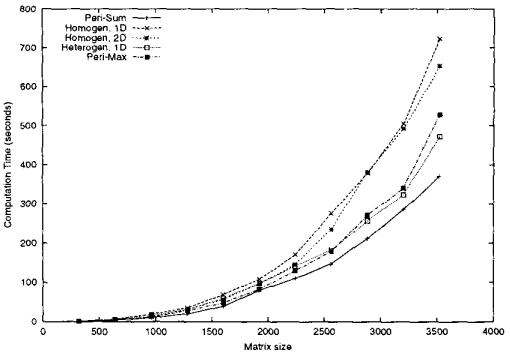
[11] S. Khanna, S. Muthukrishnan, and S. Skiena. Efficient array partitioning. In *Proc. 24th Int. Colloquium on Automata, Languages and Programming*, LNCS 1256, pages 616–626. Springer-Verlag, 1997.

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

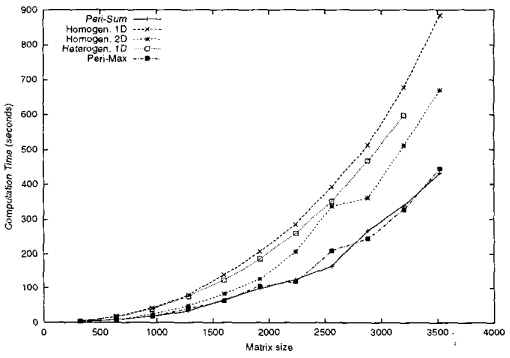
[13] T. Kong, D. Mount, and M. Wermann. The decomposition of a square into rectangles of minimal perimeter. *Discrete Applied Mathematics*, 16:239–243, 1987.

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

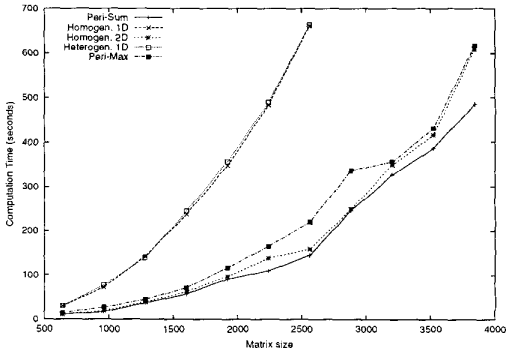
[15] F. Rastello. *Partitionnement et distribution des données: des nids de boucles à l'algorithmique hétérogène*. PhD thesis, Ecole Normale Supérieure de Lyon, Sept. 2000.



(a) 4 nodes



(b) 6 nodes



(c) 10 nodes

Figure 6. Comparison between various heuristics on a heterogeneous network of up to 10 workstations.