On Grid-based Matrix Partitioning for Heterogeneous Processors

Alexey Lastovetsky, *Member, IEEE*
University College Dublin
alexey.lastovetsky@ucd.ie

Abstract

The problem of optimal matrix partitioning for parallel linear algebra on p heterogeneous processors is typically reduced to the geometrical problem of partitioning a unit square into rectangles. In the most general case, the problem has proved NP-complete. Therefore, restrictions of this problem allowing for polynomial solutions should be studied. So far, the only well-studied restriction has been a column-based geometrical partitioning problem obtained from the general problem by imposing the additional restriction that rectangles of the partitioning make up columns. This problem has a solution of the complexity $O(p^3)$. *This paper studies another restriction - a grid-based partitioning problem obtained from the general problem by imposing the additional restriction that the heterogeneous processors owing the rectangles of the partitioning form a two-dimensional grid. An algorithm of the complexity* $O(p^{3/2})$ *solving this problem is proposed, proved and experimentally validated.*

1. Introduction

Heterogeneous networks of computers are a promising distributed-memory parallel architecture. In the most general case, a heterogeneous network includes PCs, workstations, multiprocessor servers, clusters of workstations, and even supercomputers. Unlike traditional homogeneous parallel platforms, the heterogeneous parallel architecture uses processors running at different speeds. Therefore, traditional parallel algorithms, which distribute computations evenly across parallel processors, will not balance the load of different-speed processors of the heterogeneous network. Faster processors will quickly perform their portions of computation and will wait for slower ones at points of synchronization.

A natural approach to the problem is to distribute data across processors unevenly so that each processor will perform the volume of computation proportional to its speed. The simplest performance model, capturing the heterogeneity of processors, sees the heterogeneous network of computers as a set of interconnected processors, each of which is characterized by a single positive constant representing its speed. Two important parameters of the model include:

-- p, the number of the processors, and

-- $S=\{s_1, s_2, ..., s_p\}$, the speeds of the processors.

The speed of the processors can be *absolute* or *relative*. The absolute speed of the processors is understood as the number of computational units performed by the processor per one time unit. The relative speed of the processor can be obtained by normalization of its absolute speed so that $\sum_{i=1}^{p} s_i = 1$

This is typical in the design of heterogeneous parallel algorithms that the problem of distribution of computations in proportion to the speed of processors is reduced to the problem of partitioning of some mathematical objects, such as sets, matrices, graphs, etc.

Matrices are probably the most widely used mathematical objects in scientific computing. Therefore, no wonder that the studies of data partitioning problems mainly deal with partitioning matrices.

The problem of optimal matrix partitioning for parallel linear algebra on p heterogeneous processors is typically reduced to the geometrical problem of partitioning a unit square into rectangles. In the most general case, this problem has proved NP-complete [1]. Therefore, restrictions of this problem allowing for polynomial solutions should be studied. So far, the only well-studied restriction has been a column-based geometrical partitioning problem obtained from the general problem by imposing the additional restriction that rectangles of the partitioning make up columns.

This problem has a solution of the complexity $O(p^3)$ [1]. This paper studies another restriction - a grid-based partitioning problem obtained from the general problem by imposing the additional restriction that the heterogeneous processors owing the rectangles of the partitioning form a two-dimensional grid. An algorithm of the complexity $O(p^{3/2})$ solving this problem is proposed, proved and experimentally validated.

The paper is organized as follows. Section II outlines related work on matrix partitioning for parallel computing on heterogeneous platforms. Section III presents a grid-based partitioning problem and an algorithm of its solution of the complexity $O(p^{3/2})$. Section IV reports on some experimental results. Section V concludes the paper.

2. Matrix Partitioning for Heterogeneous Processors

In many cases the problem of optimal distribution of computations over a one-dimensional arrangement of heterogeneous processors can be reduced to the mathematical problem of partitioning of a set or a well-ordered set, even if the original problem is dealing with matrices but reduced to the problem of partitioning of a matrix in one dimension [2,3,4,5].

In this paper, we address matrix-partitioning problems that do not impose the additional restriction of partitioning the matrix in one dimension. The partitioning problems occur, in particular, during design of many parallel algorithms for solution of linear algebra problems on heterogeneous platforms.

One such problem is matrix multiplication. Matrix multiplication C=A×B is a very simple but important linear algebra kernel. It also serves as a prototype for many other scientific kernels. Parallel algorithms for matrix multiplication on heterogeneous platforms have been well studied over the last decade. A typical heterogeneous matrix multiplication algorithm is designed as a modification of a well-known algorithm for matrix multiplication on homogeneous distributed-memory multiprocessors. Most often, the two-dimensional block-cyclic algorithm implemented in the ScaLAPACK library is used as a basis for the heterogeneous modifications [6].

Modifications of the algorithm for heterogeneous platforms typically use the following general design [7]:

- Matrices A, B, and C are identically partitioned into equal rectangular generalized blocks.
- The generalized blocks are identically partitioned into rectangles so that
  - There is one-to-one mapping between the rectangles and the processors.
  - The area of each rectangle is (approximately) proportional to the speed of the processor which owns the rectangle (see Figure 1).
  - Then, the algorithm follows the steps of its homogeneous prototype.

The motivation behind partitioning of the generalized blocks in proportion to the speed of the processors is as following. At each step of the algorithm, the amount of computations needed to update one $r \times r$ block of matrix C is the same for all the blocks. Therefore, the load of the processors will be perfectly balanced if the number of blocks updated by each processor is proportional to its speed. The total number of blocks updated by the processor is equal to the number of blocks allocated to this processor in each generalized block multiplied by the total number of generalized blocks. The number of blocks in a partition of the generalized block is equal to the area of the partition measured in $r \times r$ blocks. Thus, if the area of each partition is proportional to the speed of the processor owing it, then the load of the processors will be perfectly balanced.

The problem of optimal partitioning of the generalized block has been studied. So far, the studies have been interested not in an exact solution but rather in an asymptotically optimal solution, which is an approximate solution approaching the optimal one with the increase of the block size. Such relaxation of the partitioning problem makes its formulation possible not only in terms of absolute speeds of the processors but also in terms of their relative speeds. The use of relative speeds instead of absolute speeds normally simplifies the design and the application of partitioning algorithms.

From the partitioning point of view, a generalized block is an integer-valued rectangular. Therefore, if we are only interested in an asymptotically optimal solution, the problem of its partitioning can be reduced to a geometrical problem of optimal partitioning of a real-valued rectangle. Indeed, given a real-valued optimal solution of the problem, its asymptotically optimal solution can be easily obtained by rounding off the real values of the optimal solution to integers.

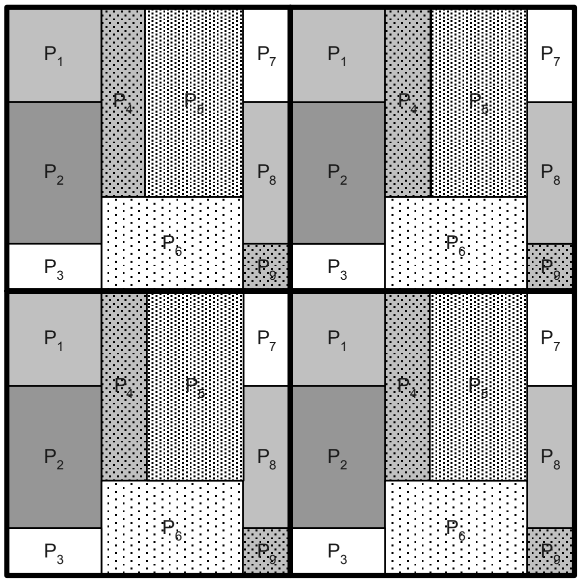


Figure 1. Data partitioning for parallel matrix multiplication on nine heterogeneous processors. A matrix is partitioned into four equal generalized blocks, each identically partitioned into nine rectangles. The area of each rectangle is proportional to the speed of the processors owing it.

In the most general form, the related geometrical problem has been formulated as follows [1]:

- Given a set of p processors P_1, P_2, ..., P_p, the relative speed of each of which is characterized by a positive constant, s_i, ($\sum_{i=1}^{p} s_i = 1$),
- Partition a unit square into p rectangles so that
    - There is one-to-one mapping between the rectangles and the processors.
    - The area of the rectangle allocated to processor P_i is equal to s_i ($i \in \{1,\ldots,p\}$).
    - The partitioning minimizes $\sum_{i=1}^{p}(w_i + h_i)$, where w_i is the width and h_i is the height of the rectangle allocated to processor P_i ($i \in \{1,\ldots,p\}$).

Partitioning the unit square into rectangles with the area proportional to the speed of the processors is aimed at the balancing of the load of the processors. As a rule, there are multiple different partitionings satisfying the criterion. Minimization of the sum of half-perimeters of the rectangles, $\sum_{i=1}^{p}(w_i + h_i)$, is aimed at minimization of the total volume of data communicated between the processors [1]. The use of a unit square instead of a rectangle in the formulation of the problem does not make it less general because the optimal solution of this problem for an arbitrary rectangle is obtained by straightforward scaling of the corresponding optimal solution for the unit square.

The general geometrical partitioning problem has proved NP-complete [1]. The NP-completeness means that, most likely, an efficient algorithm, solving the geometrical partitioning problem in its general form, does not exist. Therefore, restrictions of this problem allowing for polynomial solutions should be formulated and efficiently solved. So far, the only well-studied restriction is a column-based geometrical partitioning problem.

The column-based problem is obtained from the general one by imposing the additional restriction that rectangles of the partitioning make up columns, as illustrated in Figure 2. The column-based geometrical partitioning problem has a polynomial solution of the complexity $O(p^3)$. The corresponding algorithm is proposed and proved in [1].

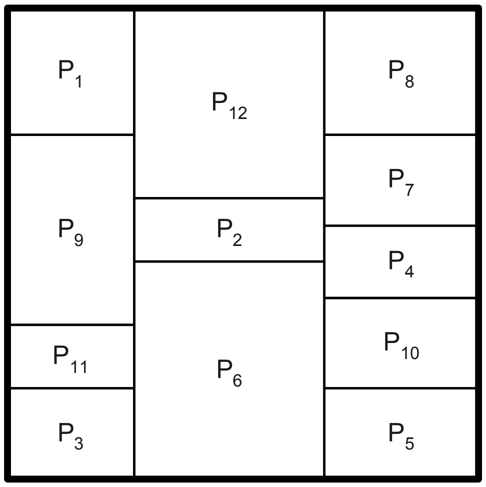


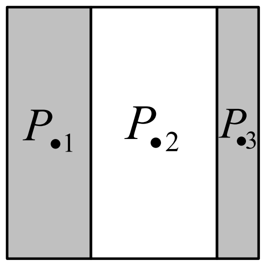
Figure 2. Column-based partitioning of the unit square into 12 rectangles. The rectangles of the partitioning form three columns.

Another restricted form of the column-based geometrical partitioning problem has been also addressed. Namely, the pioneering result in the field was an algorithm of the linear complexity solving the column-based geometrical partitioning problem under the additional assumption that the processors are already arranged into a set of processor columns [8] (that is, assuming that the number of columns c in the partitioning and the mapping of rectangles in each column to the processors are given). The algorithm is as follows.

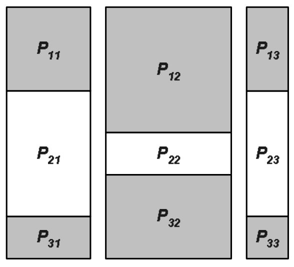
Algorithm 1: Optimal partitioning a unit square between p heterogeneous processors arranged into c columns, each of which is made of r_j processors, $j \in \{1,\ldots,c\}$:

- Let the relative speed of the i-th processor from the j-th column, P_{ij}, be s_{ij} ($\sum_{j=1}^{c}\sum_{i=1}^{r_j} s_{ij} = 1$).
- Then, we first partition the unit square into c vertical rectangular slices such that the width the j-th slice $w_j = \sum_{i=1}^{r_j} s_{ij}$.
    - This partitioning makes the area of each vertical slice proportional to the sum of speeds of the processors in the corresponding column.
- Second, each vertical slice is partitioned independently into rectangles in proportion with the speed of the processors in the corresponding processor column.

Figure 3 illustrates the algorithm for a 3×3 processor grid.



(a)



(b)

Figure 3. Example of two-step partitioning of the unit square between 9 heterogeneous processors arranged into a 3×3 processor grid. The relative speed of the processors is given by matrix $\begin{pmatrix} 0.11 & 0.25 & 0.05 \\ 0.17 & 0.09 & 0.08 \\ 0.05 & 0.17 & 0.03 \end{pmatrix}$. (a) At the first step, the unit square is partitioned in one dimension between processor columns of the 3×3 processor grid in proportion 0.33:0.51:0.16. (b) At the second step, each vertical rectangle is partitioned independently in one dimension between processors of its column. The first rectangle will be partitioned in proportion 0.11:0.17:0.05. The second rectangle will be partitioned in proportion 0.25:0.09:0.17. The third rectangle will be partitioned in proportion 0.05:0.08:0.03.

3. Grid-Based Heterogeneous Partitioning

We can see that Algorithm 1 also solves a more restricted problem when the given arrangement of processors forms a two-dimensional grid.

In the paper, we study a *grid-based* geometrical partitioning problem, obtained from the general problem by imposing the additional restriction that the heterogeneous processors owing the rectangles of the partitioning form a two-dimensional grid as illustrated in Figure 4. Equivalently, a grid-based partitioning can be defined as a partitioning of the unit square into rectangles such that there exist p and q such that any vertical line crossing the unit square will pass through exactly p rectangles and any horizontal line crossing the square will pass through exactly q rectangles.

The grid-based partitioning problem has some nice properties that allow for its efficient solution.

Proposition 1. Let a grid-based partitioning of the unit square between p heterogeneous processors form c columns, each of which consists of r processors, $p = r \times c$. Then, the sum of half-perimeters of the rectangles of the partitioning will be equal to $(r + c)$.

Proof. The sum of heights of the rectangles owned by each column of processors will be equal to 1. As we have in total c columns, the sum of heights of all rectangles of the partitioning will be equal to c. Similarly, the sum of widths of the rectangles owned by each row of processors in the processor grid will be equal to 1. As we have in total r rows in the grid, the sum of widths of all rectangles of the partitioning will be equal to r. Hence, the sum of half-perimeters of all rectangles will be equal to $(r + c)$.

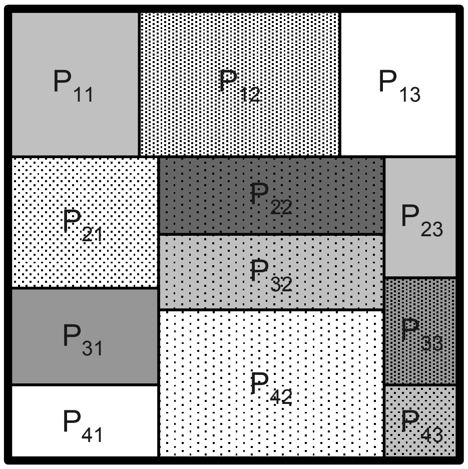


Figure 4. Grid-based partitioning of the unit square into 12 rectangles. The processors owing the rectangles of the partitioning form 4×3 processor grid.

There are two important corollaries from Proposition 1:

- The shape $r \times c$ of the processor grid formed by any optimal grid-based partitioning will minimize $(r+c)$.
- The sum of half-perimeters of the rectangles of the optimal grid-based partitioning does not depend on the mapping of the processors onto the nodes of the grid

The grid-based geometrical partitioning problem has a polynomial solution of the complexity $O(p^{3/2})$. The corresponding algorithm is as follows.

Algorithm 2: Optimal grid-based partitioning of a unit square between *p* heterogeneous processors:

- **Step 1:** Find the optimal shape $r \times c$ of the processor grid such that $p = r \times c$ and $(r+c)$ is minimal.
- **Step 2:** Map the processors onto the nodes of the grid.
- **Step 3:** Apply Algorithm 1 of the optimal partitioning of the unit square to this $r \times c$ arrangement of the *p* heterogeneous processors.

The correctness of Algorithm 2 is obvious. Step 1 of the algorithm finds the optimal shape of the processor grid that minimizes the sum of half-perimeters of any grid-based partitioning for any mapping of the processors onto the nodes of the grid. Step 3 just finds one of such partitioning, the rectangles of which are proportional to the speeds of the processors. Note that the returned partitioning is always column-based due to the nature of Algorithm 1 (see Figure 5).

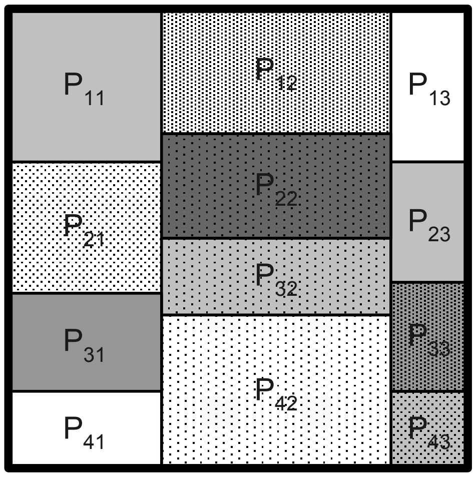


Figure 5. A 4×3 optimal grid-based partitioning returned by Algorithm 2. The rectangles of the partitioning form three columns.

Step 1 of Algorithm 2 can be performed by the following simple algorithm.

Algorithm 3: Finding *r* and *c* such that $p = r \times c$ and $(r+c)$ is minimal:

```
    $r = \lfloor \sqrt{p} \rfloor$;
    while($r>1$)
       if(($p \bmod r$) ==0))
       goto stop;
       else $r$--;
    stop: $c = p/r$;
```

Proposition 2. Algorithm 3 is correct.

Proof. If *p* is a square number then $r = c = \sqrt{p}$ minimizes $(r+c)$, and the algorithm works correctly in this case. Now let us assume that *p* is not a square number. Then, $r \neq c$. Due to symmetry, we can assume $r < c$ without loss of generality. We have $r + c = r + \frac{p}{r}$. It is easy to show that function $f(r) = r + \frac{p}{r}$ will be decreasing if $r < c$. Indeed, $\frac{df}{dr} = 1 - \frac{p}{r^2}$. Therefore, if $r < c$ then $r < \sqrt{p}$ and, hence, $1 - \frac{p}{r^2} < 0$. Thus, if $0 < r_1 < c_1$, $0 < r_2 < c_2$, $r_1 < r_2$ and $r_1 \times c_1 = r_2 \times c_2 = p$, then $r_1 + c_1 > r_2 + c_2$. Therefore, the algorithm will return the correct result if *p* is not a square number.

Proposition 3. The complexity of Algorithm 2 can be bounded by $O(p^{3/2})$.

Proof. The complexity of Step 1 of Algorithm 2 can be bounded by $O(p^{3/2})$. Indeed, we can use Algorithm 3 at this step. The number of iterations of the main loop of Algorithm 3 is no greater than $\sqrt{p}$. At each iteration, we test the condition $(p \bmod r) == 0$. A straightforward testing of this condition can be done in $\left\lceil \frac{p}{r} \right\rceil$ steps, each of the complexity $O(1)$ (for example, by repeated subtraction of *r* from *p*). Therefore, the overall complexity of Algorithm 3 can be comfortably bounded by $O(p^{3/2})$. As we can use an arbitrary mapping of the processors onto the nodes of the $r \times c$ grid, the complexity of Step 2 of Algorithm 2 can be bounded by $O(p)$. Algorithm 1 used at Step 3 has the complexity $O(p)$. Thus, the overall complexity of Algorithm 2 can be bounded by $O(p^{3/2}) + O(p) + O(p) = O(p^{3/2})$.

4. Experimental Results

In order to validate the algorithm presented in Section III, an algorithm of parallel matrix multiplication on heterogeneous processors based on the general scheme described in Section II and the grid-based matrix partitioning was implemented. The algorithm was implemented in HeteroMPI [9]. This section presents some results of experiments with this application. All presented results are obtained for r = 16 and the maximal size of generalized block providing the best balance of load of heterogeneous processors.

A heterogeneous cluster of 16 different Linux, Solaris and HP-UX workstations shown in Table 1 is used in the experiments. The network is based on 2 Gbit Ethernet with a switch enabling parallel communications between the computers. The absolute speeds of the processors are obtained using the BLAS routine `dgemm`. The absolute speeds in million floating point operations per second (MFlops) obtained by multiplication of two dense 1536×1536 matrices for the processors are shown in the last column in Table 1. It can be seen that the fastest processor is *hcl13* and the slowest processor is *hcl08*. It should be noted that one process is run per processor to obtain these

In the experiments, we use the one-process-per-workstation configuration of the application and vary the shape of arrangement, $r \times c$, of the processes. Table 2 shows the results of the experiments. As expected, 4×4 is the best arrangement minimizing the communication time. It can be also seen that the difference in total execution times for the same

TABLE 1.
SPECIFICATIONS OF SIXTEEN LINUX COMPUTERS ON WHICH THE MATRIX MULTIPLICATION IS EXECUTED

Com-puter	Cpu / Main memory / Cache (GHz/Mbytes/Kbytes)	Absolute speed (MFlops)
hcl01	3.60 / 256 / 2048	2171
hcl02	3.00 / 256 / 2048	2099
hcl03	3.40 / 1024 / 1024	1761
hcl04	3.40 / 1024 / 1024	1787
hcl05	3.40 / 256 / 1024	1735
hcl06	3.40 / 256 / 1024	1653
hcl07	3.40 / 256 / 1024	1879
hcl08	3.40 / 256 / 1024	1635
hcl09	1.00 / 1024 / 1024	3004
hcl10	1.00 / 1024 / 1024	2194
hcl11	3.00 / 512 / 1024	4580
hcl12	3.40 / 512 / 1024	1762
hcl13	3.40 / 1024 / 1024	4934
hcl14	2.80 / 1024 / 1024	4096
hcl15	3.60 / 1024 / 16	2697
hcl16	3.60 / 1024 / 2048	4840

TABLE 2.
EXECUTION TIMES OF THE PARALLEL MULTIPLICATION OF TWO DENSE N×N MATRICES ON 16 HETEROGENEOUS PROCESSORS FOR DIFFRENT ARRANGEMENTS R×C

Matrix size (n)	Processor grid (r×c)	Total execution time (sec)	Communication time (sec)
2048	1×16	54	27
2048	2×8	32	5
2048	**4×4**	**32**	**2**
3072	1×16	113	61
3072	2×8	68	19
3072	**4×4**	**62**	**8**
4096	1×16	218	114
4096	2×8	133	45
4096	**4×4**	**101**	**7**
5120	1×16	348	180
5120	2×8	211	61
5120	**4×4**	**179**	**23**
6144	1×16	537	258
6144	2×8	335	95
6144	**4×4**	**276**	**30**
7168	1×16	770	352
7168	2×8	514	150
7168	**4×4**	**420**	**51**
8192	1×16	1264	464
8192	2×8	709	181
8192	**4×4**	**582**	**50**
9216	1×16	1444	590
9216	2×8	969	233
9216	**4×4**	**828**	**93**
10240	1×16	1916	727
10240	2×8	1292	297
10240	**4×4**	**1100**	**110**

problem size is solely due to the difference in communication times.

5. Application to Cartesian Matrix Partitioning

The grid-based partitioning problem is not the most restrictive partitioning problem that has been addressed by algorithm designers. A *Cartesian* partitioning problem is even more restrictive. The Cartesian problem can be obtained from the column-based problem by imposing the additional restriction that rectangles of the partitioning make up rows, as illustrated in Figure 6. A Cartesian partitioning can be also defined as a grid-based partitioning, rectangles of which make up both rows and columns.

Cartesian partitionings play an important role in design of parallel heterogeneous algorithms. In particular, *scalable* heterogeneous parallel algorithms are normally based on Cartesian partitionings. The point is that in a Cartesian partitioning no rectangle has more than one direct neighbor in any direction (left, up, right, or down), which results in scalable communication patterns for algorithms with communications only between direct neighbors.

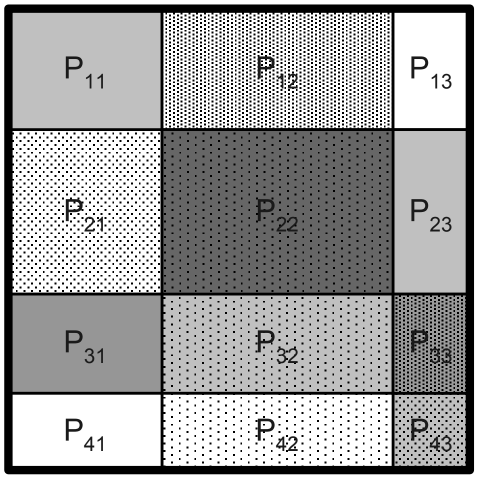


Fig. 6. Cartesian partitioning of the unit square into 12 rectangles. The rectangles of the partitioning form a 4×3 grid. No rectangle has more than one direct neighbor in any direction (left, up, right, or down).

Despite it is more restrictive than the column-based and grid-based problems, the Cartesian partitioning problem proves more difficult. The reason is that unlike optimal column-based and grid-based partitionings, an optimal Cartesian partitioning may not perfectly balance the load of processors (just because for some combinations of relative speeds there may be no Cartesian partitioning at all perfectly balancing their load). In other words, the areas of the rectangles of the partitioning may not be proportional to the speeds of the processors. Therefore, relative speeds of the processors become useless and the problem should be re-formulated in terms of absolute speeds. In a general form, the Cartesian partitioning problem can be formulated as follows:

- Given p processors, the speed of each of which is characterized by a given positive constant,
- Find a Cartesian partitioning of a unit square such that
    - There is one-to-one mapping between the rectangles of the partitioning and the processors.
    - The partitioning minimizes $\max_{i,j}\left\{\frac{h_i \times w_j}{s_{ij}}\right\}$, where h_i is the height of rectangles in the i-th row, w_j is the width of rectangles in the j-th column, s_{ij} is the speed of the processor owing j-th rectangle in the i-th row, $i \in \{1,\dots,r\}$, $j \in \{1,\dots,c\}$, $p = r \times c$.

This formulation is motivated by parallel matrix algorithms requiring the same amount of computation for calculation of each single element of the resulting matrix. The speed of a processor can be obtained by normalizing its absolute speed expressed in numbers of matrix elements computed per time unit (that is, by dividing the absolute speed by the total number of elements in the matrix). In this case, $\frac{h_i \times w_j}{s_{ij}}$ will give us the computation time of the processor owing j-th rectangle in the i-th row of the partitioning; and $\max_{i,j}\left\{\frac{h_i \times w_j}{s_{ij}}\right\}$ will give the total computation time assuming fully parallel execution of the algorithm. The cost of communication is not addressed in this formulation at all (to do it, additional parameters describing the communication network are needed).

The Cartesian problem has not been studied in the above general form. An algorithm solving this problem has to find an optimal combination of the shape $r\times c$ of the processor grid, the mapping of the processors onto the nodes of the grid, and sizes of rectangles allocated to the processors. However, simplified versions of the problem were studied and proved its difficulty. For example, under the additional assumption that the shape $r \times c$ of the partitioning is given, the problem proved NP-complete [2]. Moreover, it is still unclear if there exists a polynomial algorithm solving the problem even when both the shape $r \times c$ of the grid and the mapping of the processors onto the nodes of the grid are given. The only positive result states that there exists an optimal Cartesian partitioning such that the processors will be arranged in the grid in a non-increasing order of their speeds (in any direction – from left to right and from top to bottom) [2].

At the same time, an efficient algorithm returning an approximate solution of the simplified version of the Cartesian partitioning problem, when the shape $r \times c$ of the partitioning is given, has been proposed [2]. An approximate solution of the general Cartesian partitioning problem can be found by applying this algorithm [2] to the shape $r \times c$ returned by Algorithm 3, that is, to the shape optimal for grid-based partitionings.

6. Conclusion

The problem of optimal matrix partitioning for parallel linear algebra on p heterogeneous processors is typically reduced to the geometrical problem of partitioning a unit square into rectangles. In the most general case, this problem has proved NP-complete [1]. So far, the only well-studied restriction of the general problem has been a column-based geometrical partitioning problem that has a solution of complexity $O(p^3)$ [1]. This paper has studied another restriction - a grid-based partitioning problem obtained from the general problem by imposing the additional restriction that the heterogeneous processors owing the rectangles of the partitioning form a two-dimensional grid. An algorithm of the complexity $O(p^{3/2})$ solving this problem has been proposed, proved and experimentally validated. We have also demonstrated how the results can be used in approximate solution of the general Cartesian matrix-partitioning problem.

ACKNOWLEDGMENT

The author would like to thank Ravi Reddy for his help in conducting the experiments.

REFERENCES

[1] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, “Matrix Multiplication on Heterogeneous Platforms”, IEEE Transactions on Parallel and Distributed Systems 12(10), pp. 1033-1051, 2001.

[2] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert, “A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers)”, IEEE Transactions on Computers, Volume 50, No. 10, pp.1052-1070, October 2001.

[3] P. Boulet, J. Dongarra, F. Rastello, Y. Robert, and F. Vivien, “Algorithmic issues on heterogeneous computing platforms”,, Parallel Processing Letters, 9(2), pp.197-213, 1999.

[4] A. Lastovetsky and R. Reddy, “A Variable Group Block Distribution Strategy for Dense Factorizations on Networks of Heterogeneous Computers”, Proceedings of the 6th International Conference on Parallel Processing and Applied Mathematics (PPAM 2005), 11-14 September, Poznan, Poland, Lecture Notes in Computer Science 3911, pp.1074-1081, Springer, 2006.

[5] J. Barbosa, J. Tavares, and A. J. Padilha, “Linear Algebra Algorithms in a Heterogeneous Cluster of Personal Computers”, Proceedings of the 9th Heterogeneous Computing Workshop (HCW 2000), Cancun, Mexico, IEEE Computer Society Press, May 2000, pp.147-159.

[6] J. Choi, J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley, “The Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines”, Scientific Programming, 5(3), pp.173–184, 1996.

[7] A. Kalinov and A. Lastovetsky, "Heterogeneous Distribution of Computations Solving Linear Algebra Problems on Networks of Heterogeneous Computers", Journal of Parallel and Distributed Computing 61(4), pp. 520-535, Academic Press, 2001.

[8] A. Kalinov and A. Lastovetsky, “Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Networks of Heterogeneous Computers”, Proceedings of the 7th International Conference on High Performance Computing and Networking Europe (HPCN'99), 12-14 April 1999, Amsterdam, The Netherlands, Lecture Notes in Computer Science 1593, pp.191-200, Springer-Verlag, 1999.

[9] A. Lastovetsky and R. Reddy, “HeteroMPI: Towards a Message-Passing Library for Heterogeneous Networks of Computers”, Journal of Parallel and Distributed Computing, 66(2), pp.197-220, Elsevier, 2006.