An Approach to Assessment of Heterogeneous Parallel Algorithms

Alexey Lastovetsky, Ravi Reddy

Department of Computer Science, University College Dublin, Belfield, Dublin 4, Ireland
{Alexey.Lastovetsky, manumachu.reddy}@ucd.ie

Abstract. The paper presents an approach to the performance analysis of heterogeneous parallel algorithms. As a typical heterogeneous parallel algorithm is just a modification of some homogeneous one, the idea is to compare the heterogeneous algorithm with its homogeneous prototype, and to assess the heterogeneous modification rather than to analyse the algorithm as an isolated entity. A criterion of optimality of heterogeneous parallel algorithms is suggested. A parallel algorithm of matrix multiplication on heterogeneous clusters is used to demonstrate the proposed approach.

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 synchronisation.

A natural approach to the problem is to distribute data across processors unevenly so that each processor performs the volume of computation proportional to its speed. Several authors have applied this approach to data parallel algorithms based on the two-dimensional block-cyclic distribution [1-4].

The methods of the performance analysis of homogeneous parallel algorithms are well studied. They are based on a number of models of parallel computers, including the parallel random access machine (PRAM) [5], the bulk-synchronous parallel model (BSP) [6], and the LogP model [7]. All the models assume a parallel computer to be a homogeneous multiprocessor. The PRAM is the most simplistic model. It assumes that all processors work synchronously and that interprocessor communication is free. The BSP allows processors to work asynchronously and models latency and limited bandwidth. Finally, the LogP is the most realistic model among them. It characterizes a parallel machine by the number of processors (P), the communication bandwidth (g), the communication delay (L), and the communication overhead (o). The LogP

model has been successfully used for the performance analysis of parallel algorithms for (homogeneous) supercomputers. The theoretical analysis of a homogeneous parallel algorithm is normally accompanied by a relatively small number of experiments on a homogeneous parallel computer system. The purpose of these experiments is to demonstrate that the analysis is correct, and the analysed algorithm is really faster than its counterparts.

Theoretical performance analysis of heterogeneous parallel algorithms is much more difficult task than that of homogeneous ones. While some research efforts have been made in this direction [8-9], there is no adequate and practical model of heterogeneous networks of computers yet, which would be able to predict the execution time of heterogeneous parallel algorithms with satisfactory accuracy. The problem of optimal heterogeneous data distribution has proved NP-complete even for such a simple linear algebra kernel as matrix multiplication on heterogeneous networks [4]. Therefore, most practical heterogeneous parallel algorithms are sub-optimal. A typical approach to assessment of a heterogeneous parallel algorithm is its experimental comparison with some homogeneous counterpart on one or several heterogeneous platforms. Different heterogeneous algorithms are also compared mostly experimentally. Due to the complex and irregular nature of heterogeneous networks, such experimental assessment of heterogeneous parallel algorithms is not as convincing as for homogeneous ones. One can easily argue that the demonstration of the advantage of one algorithm over other algorithm on one or several heterogeneous networks does not prove that the situation will not change if you run the algorithms on other networks of computers, with the different relative speed of processors, and the different structure and speed of the communication network.

In this paper, we present a new approach to the performance analysis of heterogeneous parallel algorithms. As a typical heterogeneous parallel algorithm is just a modification of some homogeneous one, the idea is to compare the heterogeneous algorithm with its homogeneous prototype, and to assess the heterogeneous modification rather than analyse the algorithm as an isolated entity. Namely, we propose to compare the efficiency demonstrated by the heterogeneous algorithm on a heterogeneous network with the efficiency demonstrated by its homogeneous prototype on a homogeneous network having the same aggregate performance as the heterogeneous one.

This paper is structured as follows. In Section 2, we briefly formulate our approach to assessment of heterogeneous parallel algorithm. Then we demonstrate how to apply this approach to the assessment of a concrete heterogeneous parallel algorithm. For this purpose we use an algorithm of matrix multiplication on heterogeneous networks based on the heterogeneous matrix distribution proposed in [3]. In Section 3, we describe a block cyclic algorithm of parallel matrix multiplication on homogeneous platforms. In Section 4, we introduce its heterogeneous modification. In Section 5, we assess this heterogeneous algorithm by comparing the efficiency demonstrated by this algorithm on a heterogeneous network with the efficiency demonstrated by its homogeneous prototype on a homogeneous network, which has the same aggregate performance as the heterogeneous one. We show that the heterogeneous algorithm is very close to the optimal one. In Section 6, we present some results of experiments with this application, which in particular confirm our theoretical analysis.

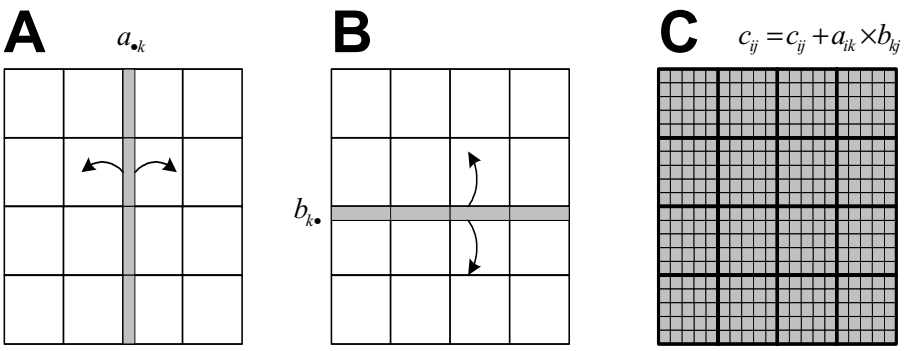


Fig. 1. One step of the algorithm of parallel matrix multiplication based on two-dimensional block distribution of matrices A, B, and C. First, the pivot column $a_{\bullet k}$ of $r \times r$ blocks of matrix A (shown shaded grey) is broadcast horizontally, and the pivot row $b_{k\bullet}$ of $r \times r$ blocks of matrix B (shown shaded grey) is broadcast vertically. Then, each $r \times r$ block c_{ij} of matrix C (also shown shaded grey) is updated, $c_{ij} = c_{ij} + a_{ik} \times b_{kj}$.

2 Assessment of Heterogeneous Algorithms

We propose to assess heterogeneous algorithms as follows. Typically, a heterogeneous algorithm is just a modification of some homogeneous one. Therefore, our proposal is to compare the heterogeneous algorithm with its homogeneous prototype and assess the heterogeneous modification rather than analyse the algorithm as an isolated entity.

Our basic postulate is that the heterogeneous algorithm cannot be more efficient than its homogeneous prototype. It means that the heterogeneous algorithm cannot be executed on the heterogeneous network faster than its homogeneous prototype on the *equivalent* homogeneous network. A homogeneous network of computers is equivalent to the heterogeneous network if

- Its communication characteristics are the same;
- It has the same number of processors;
- The speed of each processor is equal to the average speed of processors of the heterogeneous network.

The heterogeneous algorithm is considered *optimal* if its efficiency is the same as that of its homogeneous prototype.

3 Block Cyclic Algorithm of Parallel Matrix Multiplication on Homogeneous Platforms

Consider the following algorithm of parallel multiplication of two dense square $n \times n$ matrices *A* and *B* on a *p*-processor MPP:

- The *A*, *B*, and *C* matrices are identically partitioned into *p* equal $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$ squares, so that each row and each column contain $\sqrt{p}$ squares (for simplicity, we assume that *p* is a square number and *n* is a multiple of $\sqrt{p}$). There is one-to-one mapping between these squares and the processors. Each processor is responsible for computing its *C* square (see Figure 1).
- Each element in *A*, *B*, and *C* is a square $r \times r$ block and the unit of computation is the updating of one block, i.e., a matrix multiplication of size *r*. For simplicity, we assume that $\sqrt{p}$ is a multiple of *r*.
- The algorithm consists of $\frac{n}{r}$ steps. At each step *k*,
  - A column of blocks (the pivot column) of matrix *A* is communicated (broadcast) horizontally (see Figure 1);
  - A row of blocks (the pivot row) of matrix *B* is communicated (broadcast) vertically (see Figure 1);
  - Each processor updates each block in its *C* square with one block from the pivot column and one block from the pivot row, so that each block c_{ij} ($i, j \in \{1, \ldots, \frac{n}{r}\}$) of matrix *C* will be updated, $c_{ij} = c_{ij} + a_{ik} \times b_{kj}$ (see Figure 1).

  Thus, after $\frac{n}{r}$ steps of the algorithm, each block c_{ij} of matrix *C* will be

$$c_{ij} = \sum_{k=1}^{\frac{n}{r}} a_{ik} \times b_{kj} ,$$

i.e., $C = A \times B$.

Consider this algorithm from the processor point-of-view. The processors of the MPP executing the algorithm are arranged into a two-dimensional $m \times m$ grid $\{P_{ij}\}$, where $m = \sqrt{p}$ and $i, j \in \{1, \ldots, m\}$. At each step *k* of the algorithm,

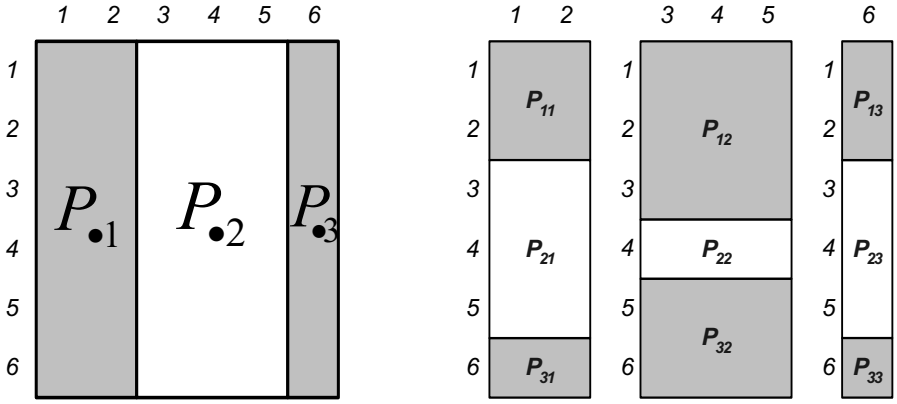
- The pivot column $a_{\bullet k}$ is owned by the column of processors $\{P_{iK}\}_{i=1}^{m}$ and the pivot row $b_{k\bullet}$ is owned by the row of processors $\{P_{Ki}\}_{i=1}^{m}$, where $K = \left\lceil \frac{r \times k}{\frac{n}{m}} \right\rceil$.
- Each processor P_{iK} (for all $i \in \{1, \ldots, m\}$) horizontally broadcasts its part of the pivot column $a_{\bullet k}$ to processors $P_{i\bullet}$.
- Each processor P_{Kj} (for all $j \in \{1, \ldots, m\}$) vertically broadcasts its part of the pivot row $b_{k\bullet}$ to processors $P_{\bullet j}$.
- Each processor P_{ij} receives the corresponding part of the pivot column and pivot row and uses them to update each $r \times r$ block of its C square.

Note that at each step k, each processor P_{ij} participates in two collective communication operations: a broadcast involving the row of processors $P_{i\bullet}$ and a broadcast involving the column of processors $P_{\bullet j}$. Processor P_{iK} is the *root* for the first broadcast, and processor P_{Kj} is the root for the second. As r is usually much less than m, in most cases at next step k+1 of the algorithm processor P_{iK} will be again the root of the broadcast involving the row of processors $P_{i\bullet}$, as well as processor P_{Kj} will be the root of the broadcast involving the column of processors $P_{\bullet j}$. Therefore, at step k+1, the broadcast involving the row of processors $P_{i\bullet}$ cannot start until processor P_{iK} completes this broadcast at step k. Similarly, the broadcast involving the column of processors $P_{\bullet j}$ cannot start until processor P_{Kj} completes that broadcast at step k. The root of the broadcast communication operation completes when its communication buffer can be reused. Typically, the completion means that the root has sent out the contents of the communication buffer to all receiving processors.

Thus, there is strong dependence between successive steps of the parallel algorithm, which hinders parallel execution of the steps. If at successive steps of the algorithm the broadcast operations involving the same set of processors had different roots, they could be executed in parallel. As a result, more communications would be executed in parallel and more computations and communications would be overlapped.

In order to break the dependence between successive steps of the algorithm, the way, in which matrices A, B and C are distributed over the processors, can be modified. The modified distribution is called a *two-dimensional block cyclic distribution* and can be summarized as follows:

- Each element in A, B, and C is a square $r \times r$ *block*.



(a) Partition between processor columns. (b) Partition inside each processor column.

Fig. 2. Example of two-step distribution of a 6×6 generalized block over a 3×3 processor grid. The relative speed of processors is given by matrix $s = \begin{pmatrix} 0.11 & 0.25 & 0.05 \\ 0.17 & 0.09 & 0.08 \\ 0.05 & 0.17 & 0.03 \end{pmatrix}$. (a) At the first step, the 6×6 square is distributed in a one-dimensional block fashion over processors columns of the 3×3 processor grid in proportion $0.33 : 0.51 : 0.16 \approx 2 : 3 : 1$. (b) At the second step, each vertical rectangle is distributed independently in a one-dimensional block fashion over processors of its column. The first rectangle is distributed in proportion $0.11 : 0.17 : 0.05 \approx 2 : 3 : 1$. The second one is distributed in proportion $0.25 : 0.09 : 0.17 \approx 3 : 1 : 2$. The third is distributed in proportion $0.05 : 0.08 : 0.03 \approx 2 : 3 : 1$

- The blocks are scattered in a cyclic fashion along both dimensions of the $m\times m$ processor grid, so that for all $i, j \in \{1,\ldots,\frac{n}{r}\}$ blocks a_{ij}, b_{ij}, c_{ij} will be mapped to processor P_{IJ} so that $I = (i-1) \bmod m + 1$ and $J = (j-1) \bmod m + 1$.

The algorithm is easily generalized for an arbitrary two-dimensional processor arrangement.

The two-dimensional block cyclic distribution is a general-purpose basic decomposition in parallel dense linear algebra libraries for MPPs such as ScaLAPACK [10]. The block cyclic distribution has been also incorporated in the HPF language [11].

4 Block Cyclic Algorithm of Parallel Matrix Multiplication on Heterogeneous Platforms

In an MPP, all processors are identical. Therefore, the load of the processors will be perfectly balanced if each processor performs the same amount of work. As all $r \times r$ blocks of the C matrix require the same amount of arithmetic operations, each processor executes an amount of work, which is proportional to the number of $r \times r$ blocks that are allocated to it, and, hence, proportional to the area of its rectangle. Therefore, to equally load all processors of the MPP, a rectangle of the same area must be allocated to each processor.

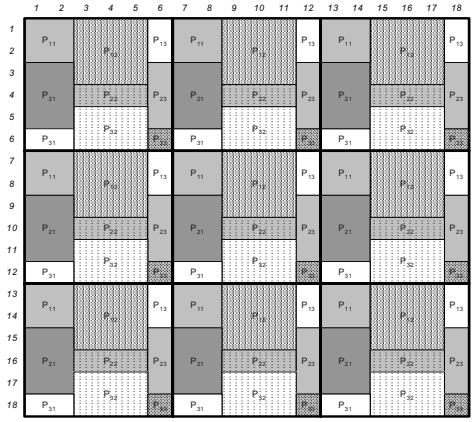
In a heterogeneous cluster, processors perform computations at different speeds. To balance the load of the processors, each processor should execute an amount of work that is proportional to its speed. In case of matrix multiplication, it means that the number of $r \times r$ blocks, which are allocated to each processor, should be proportional to its speed. Let us modify the two-dimensional block cyclic distribution to satisfy the requirement.

Suppose that the relative speed of each processor P_{ij} is characterised by a real positive number, s_{ij}, so that $\sum_{i=1}^{m}\sum_{j=1}^{m} s_{ij} = 1$. Then, the area of the rectangle allocated to processor P_{ij} should be $s_{ij} \times n^2$.

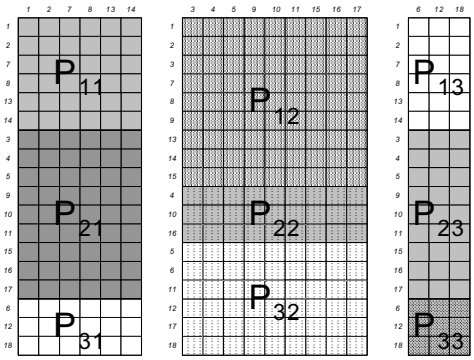
The homogeneous two-dimensional block cyclic distribution partitions the matrix into *generalized blocks* of size $(r \times m) \times (r \times m)$, each partitioned into $m \times m$ blocks of the same size $r \times r$, going to separate processors. The modified, *heterogeneous*, distribution also partitions the matrix into generalized blocks of the same size, $(r \times l) \times (r \times l)$, where $m \le l \le \frac{n}{r}$. The generalized blocks are identically partitioned into m^2 rectangles, each being assigned to a different processor. The main difference is that the generalized blocks are partitioned into unequal rectangles. The area of each rectangle is proportional to the speed of the processor that stores the rectangle.

The partitioning of a generalized block can be summarised as follows:

- Each element in the generalized block is a square $r \times r$ block of matrix elements. The generalized block is a $l \times l$ square of $r \times r$ blocks.
- First, the $l \times l$ square is partitioned into m vertical slices, so that the area of the j-th slice is proportional to $\sum_{i=1}^{m} s_{ij}$ (see Figure 2(a)). It is supposed that blocks of the j-th slice will be assigned to processors of the j-th column in the $m \times m$ processor grid. Thus, at this step, we balance the load *between* processor columns in the $m \times m$ processor grid, so that each processor column will store a vertical slice whose area is proportional to the total speed of its processors.



(a) Heterogeneous block cyclic distribution over 3x3 grid.



(b) Data distribution from processor point-of-view.

Fig. 3. A matrix with 18×18 blocks is distributed over a 3×3 processor grid. The relative speed of processors is given by matrix $s=\begin{pmatrix}0.11 & 0.25 & 0.05\\ 0.17 & 0.09 & 0.08\\ 0.05 & 0.17 & 0.03\end{pmatrix}$. The numbers on the left and on the top of the matrix represent indices of a row of blocks and a column of blocks, respectively. (a) Each labelled (shaded and unshaded) area represents different rectangles of blocks, and the label indicates at which location in the processor grid the rectangle is stored – all rectangles labelled with the same name are stored in the same processor. Each square in a bold frame represents different generalised blocks. (b) Each processor has the number of blocks approximately proportional to its relative speed, $\begin{pmatrix}6\times6 & 9\times9 & 6\times3\\ 9\times6 & 3\times9 & 9\times3\\ 3\times6 & 6\times9 & 3\times3\end{pmatrix}\approx\begin{pmatrix}0.11 & 0.25 & 0.05\\ 0.17 & 0.09 & 0.08\\ 0.05 & 0.17 & 0.03\end{pmatrix}$.

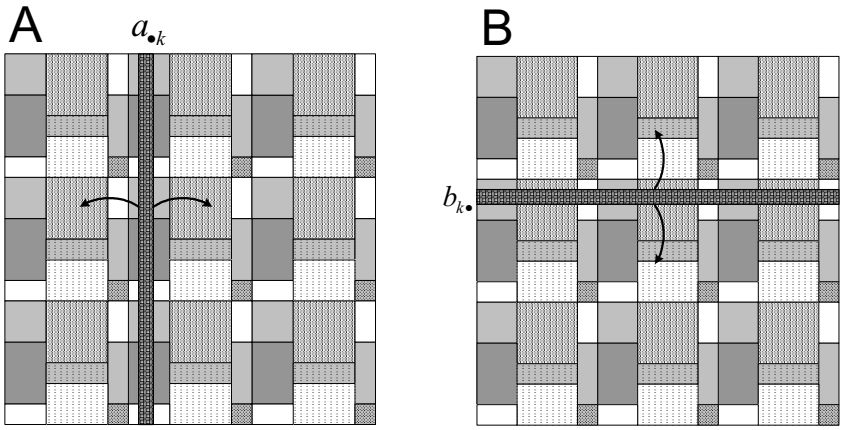


Fig. 4. One step of the algorithm of parallel matrix-matrix multiplication based on heterogeneous two-dimensional block distribution of matrices A, B, and C. First, each $r \times r$ block of the pivot column $a_{\bullet k}$ of matrix A (shown shaded dark grey) is broadcast horizontally, and each $r \times r$ block of the pivot row $b_{k\bullet}$ of matrix B (shown shaded dark grey) is broadcast vertically. Then, each $r \times r$ block c_{ij} of matrix C is updated, $c_{ij} = c_{ij} + a_{ik} \times b_{kj}$.

- Then, each vertical slice is partitioned independently into *m* horizontal slices, so that the area of the *i*-th horizontal slice in the *j*-th vertical slice is proportional to s_{ij} (see Figure 2(b)). It is supposed that blocks of the *i*-th horizontal slice in the *j*-th vertical slice will be assigned to processor P_{ij}. Thus, at this step, we balance the load of processors *within* each processor column independently.

Figure 3(a) illustrates the heterogeneous two-dimensional block cyclic distribution from the matrix point-of-view.

Figure 3(b) shows this distribution from the processor point-of-view. Each rectangle represents the total area of blocks allocated to a single processor.

Figure 4 depicts one step of the algorithm of parallel matrix-matrix multiplication on a heterogeneous $m \times m$ processor grid. Note that the total volume of communications during execution of this algorithm is exactly the same as that for a homogeneous $m \times m$ processor grid. Indeed, at each step *k* of both algorithms,

- Each $r \times r$ block a_{ik} of the pivot column of matrix *A* is sent horizontally from the processor, which stores this block, to *m*-1 processors;
- Each $r \times r$ block b_{kj} of the pivot row of matrix *B* is sent vertically from the processor, which stores this block, to *m*-1 processors.

The size *l* of a generalized block is an additional parameter of the heterogeneous algorithm. The range of the parameter is $[m, \frac{n}{r}]$. The parameter controls two conflicting aspects of the algorithm:

- The accuracy of load balancing.
- The level of potential parallelism in execution of successive steps of the algorithm.

The greater is this parameter, the greater is the total number of $r \times r$ blocks in a generalized block, and, hence, the more accurately this number can be partitioned in a proportion given by positive real numbers. Therefore, the greater is this parameter, the better the load of processors is balanced. On the other hand, the greater is this parameter, the stronger the dependence between successive steps of the parallel algorithm is, which hinders parallel execution of the steps.

Consider two extreme cases. If $l = \frac{n}{r}$, the distribution provides the best possible balance of the load of processors. At the same time, the distribution turns into a pure two-dimensional block distribution resulting in the lowest possible level of parallel execution of successive steps of the algorithm.

If $l=m$, then the distribution is identical to the homogeneous distribution, which does not bother about load-balancing at all. At the same time, it provides the highest possible level of parallel execution of successive steps of the algorithm. Thus, the optimal value of this parameter lies in between of these two, being a result of trade-off between load-balancing and parallel execution of successive steps of the algorithm.

The algorithm is easily generalized for an arbitrary two-dimensional processor arrangement.

5 Assessment of the Heterogeneous Algorithm

Let us compare the heterogeneous algorithm presented in Section 4 with its homogeneous prototype presented in Section 3. We assume that parameters n, m and r are the same. Then, both algorithms consist of $\frac{n}{r}$ successive steps.

At each step, equivalent communication operations are performed by each of the algorithms, namely:

- Each $r \times r$ block of the pivot column of matrix A is sent horizontally from the processor, which stores this block, to m-1 processors;
- Each $r \times r$ block of the pivot row of matrix B is sent vertically from the processor, which stores this block, to m-1 processors.

Thus, the per-step communication cost is the same for both algorithms.

If l is big enough, then at each step each processor of the heterogeneous network will perform the volume of computation approximately proportional to its speed. In this case, the per-processor computation cost will be approximately the same for both algorithms.

Thus, the per-step cost of the heterogeneous algorithm will be approximately the same as that of the homogeneous one. So the only reason for the heterogeneous algorithm to be less efficient than its homogeneous prototype is the lower level of potential overlapping of communication operations at successive steps of the algorithm. Obviously, the bigger is the ratio between the maximal and minimal

processor speed, the lower is this level. Note that if the communication layer serializes data packages (for example, plain Ethernet), then the heterogeneous algorithm has approximately the same efficiency as the homogeneous one. Therefore, in that case the presented heterogeneous algorithm is the optimal modification of its homogeneous prototype. In section 6, we present some experimental results that allow us to estimate the significance of the additional dependence between successive steps of the algorithm if the communication layer allows multiple data packages.

6 Experimental Results

This algorithm of parallel matrix multiplication on heterogeneous clusters was implemented in the mpC language [8].

This section presents some results of experiments with this application. All presented results are obtained for **r** = **8** and generalized block size **l = 9**, which have appeared optimal for both homogeneous and heterogeneous block cyclic distributions.

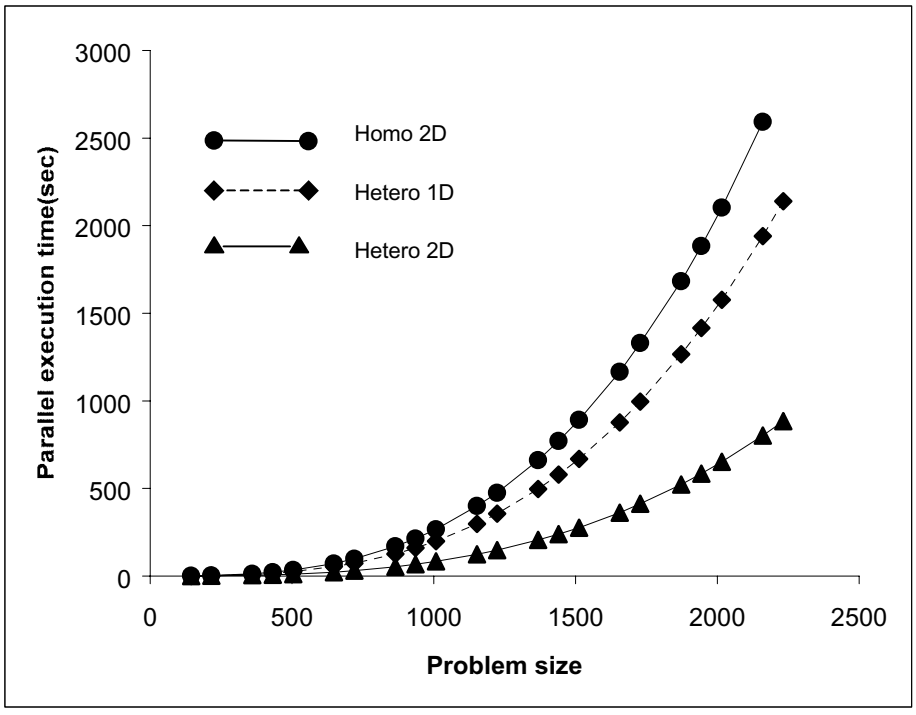


Fig. 5. Execution times of the heterogeneous and homogeneous 2D algorithms and the heterogeneous 1D algorithm. All algorithms are performed on the same heterogeneous network.

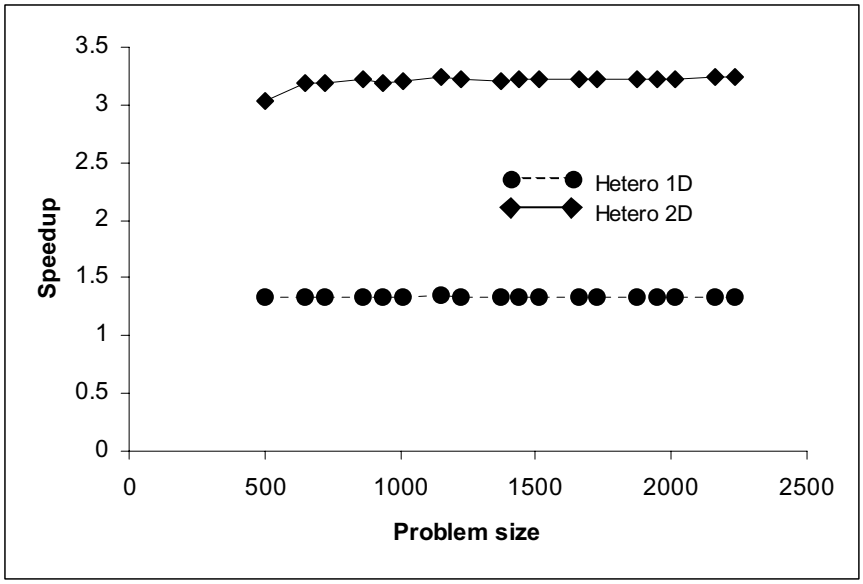


Fig. 6. The speedup of the heterogeneous 1D and 2D algorithms compared to the homogeneous 2D block cyclic algorithm. All algorithms are performed on the same heterogeneous network.

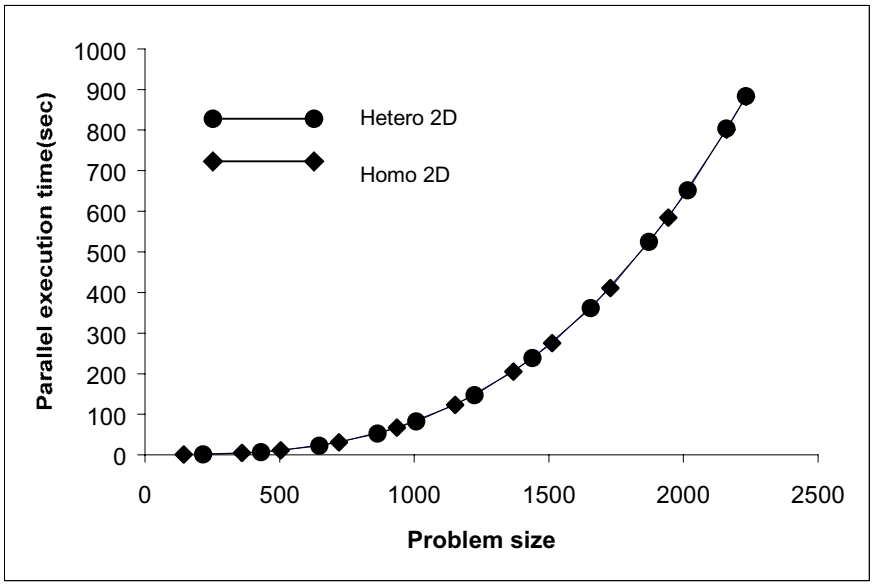


Fig. 7. Execution times of the 2D heterogeneous block cyclic algorithm on a heterogeneous network and of the 2D homogeneous block cyclic algorithm on a homogeneous network. The networks have approximately the same aggregate power of processors and share the same communication network.

A small heterogeneous local network of 9 different Solaris and Linux workstations is used in the experiments presented in Figures 5 and 6. The relative speed of the workstations is as follows: 46, 46, 46, 46, 46, 46, 46, 84, and 9. We measure their relative speed with the core computation of the algorithm (updating of a matrix). The network is based on 100 Mbit Ethernet with a switch enabling parallel communications between the computers.

For experiments presented in Figure 7, we use the same heterogeneous network and a homogeneous local network of 9 Solaris with the following relative speeds: 46, 46, 46, 46, 46, 46, 46, 46, 46. The two sets of workstations share the same network equipment. Note that the aggregate performance of the processors of the heterogeneous network is practically the same as that of the homogeneous one.

Figure 5 shows the comparison of the execution times of 3 parallel algorithms of matrix multiplication:

- The algorithm based on 2D heterogeneous block cyclic distribution;
- The algorithm based on 1D heterogeneous block cyclic distribution;
- The algorithm based on 2D homogeneous block cyclic distribution.

One can see that the 2D heterogeneous algorithm is almost twice faster than the 1D heterogeneous algorithm and almost 3 times faster than the 2D homogeneous one.

Figure 6 shows the speedup demonstrated by the heterogeneous algorithms compared to the homogeneous one.

Figure 7 shows the comparison of the execution times of the 2D heterogeneous block cyclic algorithm performed on the heterogeneous network and the 2D homogeneous block cyclic algorithm performed on the homogeneous network. One can see that the algorithms demonstrate practically the same speed, but each on its network. As the two networks are practically of the same power, we can conclude that the heterogeneous algorithm is very close to the optimal heterogeneous modification of the basic homogeneous algorithm. The experiment shows that the additional dependence between successive steps introduced by the heterogeneous modification has practically no impact on the efficiency of the algorithm. This may be explained by the following two factors:

- The speedup due to the overlapping of communication operations performed at successive steps of the algorithm is not very significant;
- The speed of processors in the heterogeneous network does not differ too much. Actually, the network is moderately heterogeneous. Therefore, for this particular network, the additional dependence between steps is very weak.

Thus, for reasonably heterogeneous networks, the presented heterogeneous algorithm has proved to be very close to the optimal one significantly accelerating matrix multiplication on such platforms compared to its homogeneous prototype.

References

[1] Crandall, P., Quinn, M.: Block Data Decomposition for Data-Parallel Programming on a Heterogeneous Workstation Network. In: Proceedings of the Second International Symposium on High Performance Distributed Computing. Spokane WA USA (1993) 42-49

[2] Kaddoura, M., Ranka, S., Wang, A.: Array Decomposition for Nonuniform Computational Environments. Journal of Parallel and Distributed Computing 3 (1996) 91-105

[3] Kalinov, A., Lastovetsky, A.: Heterogeneous Distribution of Computations Solving Linear Algebra Problems on Networks of Heterogeneous Computers. Journal of Parallel and Distributed Computing 61 (2001) 520-535
[4] Beaumont, O., Boudet, V., Rastello, F., Robert, Y.: Matrix Multiplication on Heterogeneous Platforms. IEEE Transactions on Parallel and Distributed Systems 12 (2001) 1033-1051
[5] Fortune, S., Wyllie, J.: Parallelism in Random Access Machines. In: Proceedings of the 10th Annual Symposium on Theory of Computing. San Diego CA USA (1978) 114-118
[6] Valiant, L.G.: A Bridging Model for Parallel Computation. Communications of the Association for Computing Machinery 33 (1990) 103-111
[7] Culler, D.E., Karp, R.M., Patterson, D.A., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., von Eicken, T.: LogP: Towards a Realistic Model of Parallel Computation. In: Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. San Diego CA USA (1993)
[8] Lastovetsky, A.: Adaptive parallel computing on heterogeneous networks with mpC. Parallel Computing 28 (2002) 1369-1407
[9] Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Robert, Y.: Bandwidth-Centric Allocation of Independent Tasks on Heterogeneous Platforms. In: Proceedings of 16th International Parallel and Distributed Processing Symposium. IEEE Computer Society, CD-ROM/Abstracts Proceedings, Fort Lauderdale FL USA (2002)
[10] Blackford, L., Choi, J., Cleary, A., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.: ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers – Design Issues and Performance. In: Proceedings of the 1996 ACM/IEEE Supercomputing Conference. IEEE Computer Society, CD-ROM/Abstracts Proceedings, Pittsburgh PA USA (1996)
[11] High Performance Fortran Language Specification, Version 2.0. High Performance Fortran Forum (1997)