Available online at www.sciencedirect.com
SCIENCE DIRECT®
Parallel Computing 30 (2004) 1195–1216
PARALLEL COMPUTING
www.elsevier.com/locate/parco

On performance analysis of heterogeneous parallel algorithms

Alexey Lastovetsky *, Ravi Reddy

Department of Computer Science, University College Dublin (UCD), Belfield, Dublin 4, Ireland

Received 5 March 2003; revised 15 April 2004; accepted 30 July 2004
Available online 15 September 2004

Abstract

The paper presents an approach to 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. A criterion of optimality of heterogeneous parallel algorithms is suggested. A parallel algorithm of matrix multiplication on heterogeneous clusters is used to illustrate the proposed approach.

Keywords: Performance analysis; Parallel algorithms; Heterogeneous networks of computers

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

* Corresponding author. Fax: +353 1 269 7262.
E-mail addresses: alexey.lastovetsky@ucd.ie (A. Lastovetsky), ravi.reddy@ucd.ie (R. Reddy).

0167-8191/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2004.07.007

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 show that the analysis is correct, and the analysed algorithm is really faster than its counterparts.

Theoretical performance analysis of heterogeneous parallel algorithms is a much more difficult task than that of homogeneous ones. While some research efforts in this direction have been made [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 of the heterogeneous algorithm on a heterogeneous network with the efficiency of 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 algorithms. Then we 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 of this algorithm on a heterogeneous network with the efficiency of 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.

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 aggregate communication characteristics are the same as that of the heterogeneous network;
- 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.

This approach is relatively easy to apply if the target architecture of the heterogeneous algorithm is a set of heterogeneous processors interconnected via a homogeneous communication network. In this case, all we need to do is to find a

(homogeneous) segment in the instrumental LAN and select two sets of processors in this segment so that

- Both sets consist of the same number of processors;
- All processors comprising the first set are identical;
- The second set includes processors of different speeds;
- The aggregate performance of the first set of processors is the same as that of the second one.

The first set of interconnected processors represents a homogeneous network of computers, which is equivalent to the heterogeneous network of computers represented by the second set of processors just by design. Indeed, these two networks of computers share the same homogeneous communication segment and, therefore, have the same aggregate communication characteristics. More reliable results are obtained if the intersection of the two sets of processors is not empty. This allows us to better control the accuracy of experiments by checking that the same processor has the same speed in the heterogeneous network running the heterogeneous algorithm and in the homogeneous network running its homogeneous prototype. Higher confidence of the experimental assessment can be achieved by experimenting with several different pairs of processor sets from different segments.

If the target architecture for the heterogeneous algorithm is a set of heterogeneous processors interconnected via a heterogeneous communication network, the design of experiments becomes much more complicated. A comprehensive solution of this problem is a subject for future research. We just outline how the problem can be approached in one simple but quite typical case. Let the communication network of the target heterogeneous architecture consist of a number of relatively fast homogeneous communication segments interconnected by slower communication links. Let fully parallel communications between different pairs of processors be enabled within each of the segments (for example, by using a switch, the number of ports of which is no less than the number of computers in the segment). Let communication links between different segments only support serial communication. Further design depends on the analysed heterogeneous algorithm. Assume that the communication cost of the algorithm comes mainly from relatively rare point-to-point communications separated by significant amount of computations, so that it is highly unlikely for two such communication operations to be performed in parallel. Also assume that each such a communication operation consists in passing a relatively long message. Those assumptions allows us to use a very simple linear communication model when time $t_{A\to B}(d)$ of transferring data block of size d from processor A to processor B is calculated as $t_{A\to B}(d) = s_{A\to B} \times d$, where $s_{A\to B}$ is the constant speed of communication between processors A and B and $s_{A\to B} = s_{B\to A}$. Thus, under all these assumptions, the only aggregate characteristic of the communication network, which has an impact on the execution time of the algorithm, is the average speed of point-to-point communications.

To design experiments on the instrumental LAN in this case, we need to select two sets of processors so that

- Both sets consist of the same number of processors;
- All processors comprising the first set are identical and belong to the same homogeneous communication segment;
- The second set includes processors of different speeds that span several communication segments;
- The aggregate performance of the first set of processors is the same as that of the second one;
- The average speed of point-to-point communications between processors of the second set is the same as the speed of point-to-point communications between processors of the first set.

The first set of interconnected processors will represent a homogeneous network of computers, equivalent to the heterogeneous network of computers represented by the second set.

In mathematical form, this problem can be formulated as follows. Let n be the number of processors in the first set, v be their speed, and s be the communication speed of the corresponding segment. Let the second set of processors, P, span m communication segments S_1, S_2,...,S_m. Let s_i be the communication speed of segment S_i, n_i be the number of processors of set P belonging to S_i, v_{ij} be the speed of the jth processor belonging to segment $S_i (i = 1,\ldots,m;\ j = 1,\ldots,n_i)$. Let s_{ij} be the speed of communication link between segments S_i and $S_j (i,j = 1,\ldots,m)$. Then,

$$\frac{\sum_{i=1}^{m} s_i \times \frac{n_i \times (n_i - 1)}{2} + \sum_{i=1}^{m} \sum_{j=i+1}^{m} n_i \times n_j \times s_{ij}}{\frac{n \times (n-1)}{2}} = s, \tag{1}$$

$$\sum_{i=1}^{m} n_i = n, \tag{2}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n_i} v_{ij} = n \times v. \tag{3}$$

Eq. (1) states that the average speed of point-to-point communications between processors of the second set should be equal to the speed of point-to-point communications between processors of the first set. Eq. (2) states that the total number of processors in the second set should be equal to the number of processors in the first set. Eq. (3) states that the aggregate performance of the processors in the second set should be equal to the aggregate performance of the processors in the first set.

In following sections, we illustrate application of this approach in the simplest case when the target architecture of the analysed heterogeneous algorithm is a set of heterogeneous processors interconnected via a homogeneous communication network.

3. Block cyclic algorithm of parallel matrix multiplication on homogeneous platforms

Consider the following algorithm of parallel multiplication of two dense square matrices A and B on a p-processor homogeneous distributed-memory multiprocessor (also known as MPP—massively parallel processor):

- 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 Fig. 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 Fig. 1);
  - A row of blocks (the pivot row) of matrix B is communicated (broadcast) vertically (see Fig. 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, $c_{ij} = c_{ij} + a_{ik} \times b_{ik}$ (see Fig. 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., $C = A \times B$.

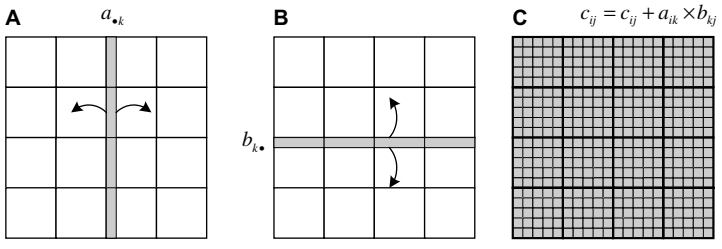

Fig. 1. One step of the algorithm of parallel matrix multiplication based on two-dimensional block distribution of matrices A,B, and C. First, the pivot column $a_{\bullet k}$ of $r \times r$ blocks of matrix A (shown shaded grey) is broadcast horizontally, and the pivot row $b_{k\bullet}$ of $r \times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

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 re-used. 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*.
- 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 $J = (j-1) \bmod m + 1$.

Fig. 2(a) illustrates this distribution from the matrix point-of-view. The matrix is now partitioned into $\frac{n^2}{r^2 \times m^2}$ equal squares, so that each row and each column contain $\frac{n}{r \times m}$ squares. All the squares are identically partitioned into m^2 equal $r \times r$ blocks, so that each row and each column contain m blocks. There is one-to-one mapping between these blocks and the processors. Thus, all the $m \times m$ squares of blocks are identically distributed over the $m \times m$ processor grid in a two-dimensional block fashion.

Fig. 2(b) shows this distribution from the processor point-of-view. Each square represents the total area of blocks allocated to a single processor.

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 (see Fig. 2(b)). 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 (see Fig. 2(a)). The modified, *heterogeneous*, distribution also partitions the matrix into generalized blocks of the same size, $(r \times l) \times (r \times l)$, where $m \leqslant l \leqslant \frac{n}{r}$. 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.

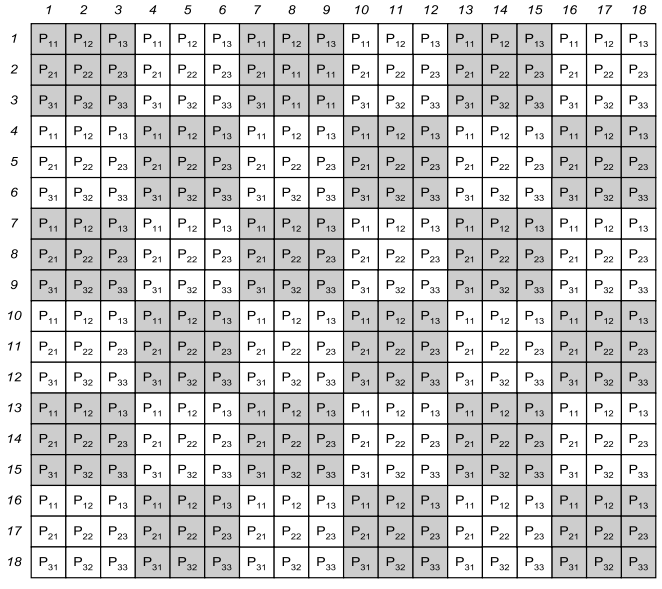

Fig. 2. A matrix with 18×18 blocks is distributed over a 3×3 processor grid. The numbers on the left and on the top of the matrix represent indices of a row of blocks and a column of blocks, respectively. (a) Block cyclic distribution over 3×3 grid. The labelled squares represent blocks of elements, and the label indicates at which location in the processor grid the block is stored—all blocks labelled with the same name are stored in the same processor. Each shaded and unshaded area represents different generalised blocks. (b) Data distribution from processor point-of-view. Each processor has 6×6 blocks.

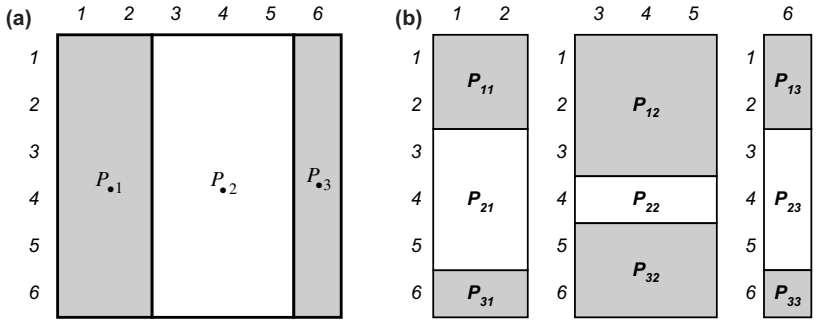

Fig. 3. Example of two-step distribution of a 6×6 generalised block over a 3×3 processor grid. The relative speed of processors is given by matrix $s = \begin{pmatrix} 0.11 & 0.25 & 0.05 \\ 0.17 & 0.09 & 0.08 \\ 0.05 & 0.17 & 0.03 \end{pmatrix}$. (a) Partition between processor columns. At the first 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) Partition inside each processor column. 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 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 jth slice is proportional to $\sum_{i=1}^{m} s_{ij}$ (see Fig. 3(a)). It is supposed that blocks of the jth slice will be assigned to processors of the jth 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.
- Then, each vertical slice is partitioned independently into m horizontal slices, so that the area of the ith horizontal slice in the jth vertical slice is proportional to s_{ij} (see Fig. 3(b)). It is supposed that blocks of the ith horizontal slice in the jth vertical slice will be assigned to processor P_{ij}. Thus, at this step, we balance the load of processors *within* each processor column independently.

Fig. 4(a) illustrates the heterogeneous two-dimensional block cyclic distribution from the matrix point-of-view.

Fig. 4(b) shows this distribution from the processor point-of-view. Each rectangle represents the total area of blocks allocated to a single processor.

Fig. 5 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,

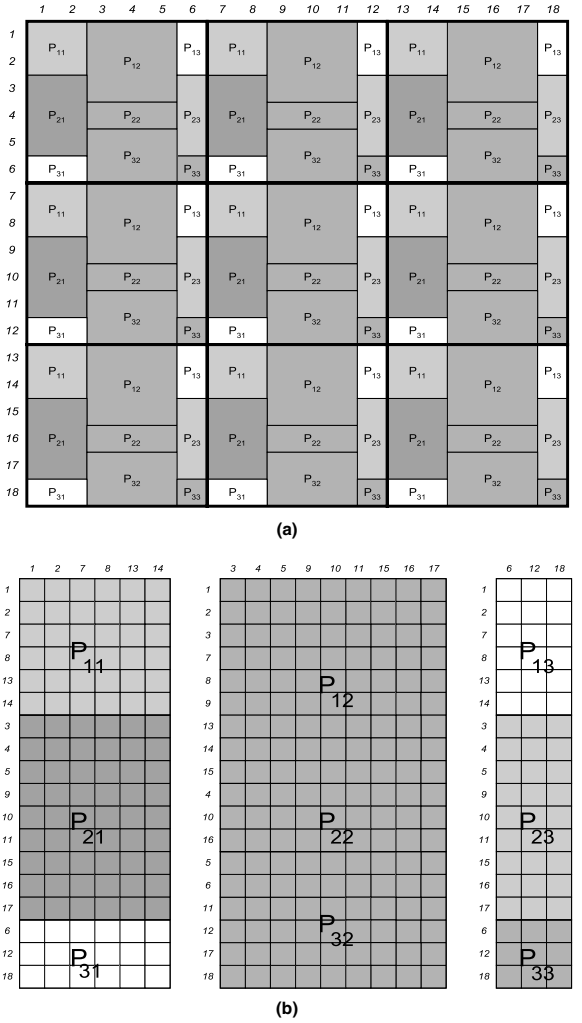

Fig. 4. A matrix with 18×18 blocks is distributed over a 3×3 processor grid. The relative speed of processors is given by matrix $s = \begin{pmatrix} 0.11 & 0.25 & 0.05 \\ 0.17 & 0.09 & 0.08 \\ 0.05 & 0.17 & 0.03 \end{pmatrix}$. The numbers on the left and on the top of the matrix represent indices of a row of blocks and a column of blocks, respectively. (a) Heterogeneous block cyclic distribution over 3×3 grid. 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) Data distribution from processor point-of-view. Each processor has the number of blocks approximately proportional to its relative speed, $\begin{pmatrix} 6 \times 6 & 9 \times 9 & 6 \times 3 \\ 9 \times 6 & 3 \times 9 & 9 \times 3 \\ 3 \times 6 & 6 \times 9 & 3 \times 3 \end{pmatrix} \approx \begin{pmatrix} 0.11 & 0.25 & 0.05 \\ 0.17 & 0.09 & 0.08 \\ 0.05 & 0.17 & 0.03 \end{pmatrix}$.

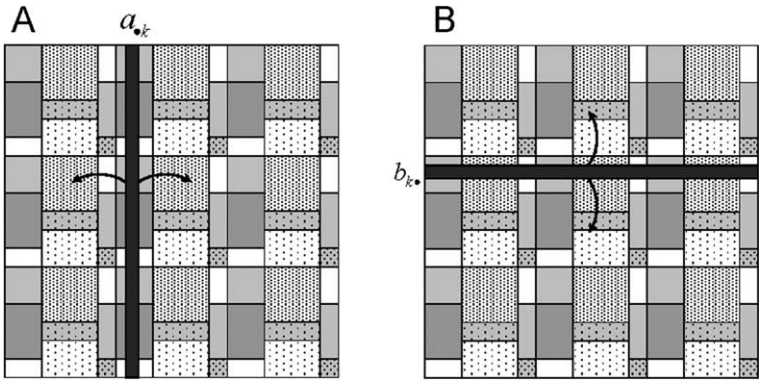

Fig. 5. One step of the algorithm of parallel matrix–matrix multiplication based on heterogeneous two-dimensional block distribution of matrices A,B, and C. First, each $r \times r$ block of the pivot column $a_{\bullet k}$ of matrix A (shown shaded dark grey) is broadcast horizontally, and each $r \times r$ 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, $c_{ij} = c_{ij} + a_{ik} \times b_{kj}$.

- 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.

The heterogeneous algorithm of matrix multiplication presented in this chapter was proposed in [12]. The core of this algorithm is the partitioning of a generalised block into uneven rectangles. More general problem of optimal partitioning a square into rectangles with no restrictions on the shape and arrangement of the rectangles was studied by Beaumont et al in [4,13] and proved to be NP-complete. They also proved that the optimal column-based partitioning that minimises the sum of the perimeters of the rectangles could be achieved in polynomial time. The partitioning used in this paper is just a version of this optimal column-based partitioning obtained under the additional restriction that the rectangles must be arranged into a two-dimensional $m \times m$ grid. This restriction is inherited from the prototype homogeneous algorithm of matrix multiplication. We decided not to relax the restriction to keep the heterogeneous algorithm closer to its homogeneous prototype.

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 = 16** and generalized block size **l = 1536**, which have appeared optimal for both homogeneous and heterogeneous block cyclic distributions.

A small local heterogeneous network of nine different Solaris, FreeBSD and Linux workstations is used in the experiments presented in Figs. 6 and 7. The relative speed of the workstations is as follows: 26, 20, 14, 14, 14, 14, 14, 9, and 1. We measure their relative speed with the core computation of the algorithm (updating of a matrix). Note that the relative speed does not depend on the size of problem for the wide range of matrix sizes used in our experiments.

The communication network interconnecting the workstations is based on 100 Mbit Ethernet with a switch enabling parallel communications between them.

For experiments presented in Fig. 8, we use the same set of heterogeneous workstations and a set of nine identical Linux workstations with the same relative speed, 14. The two sets of workstations belong to the same homogeneous communication segment of the local network specified above. Five Linux workstations of relative speed 14 are also shared by these two sets. Note that the aggregate performance

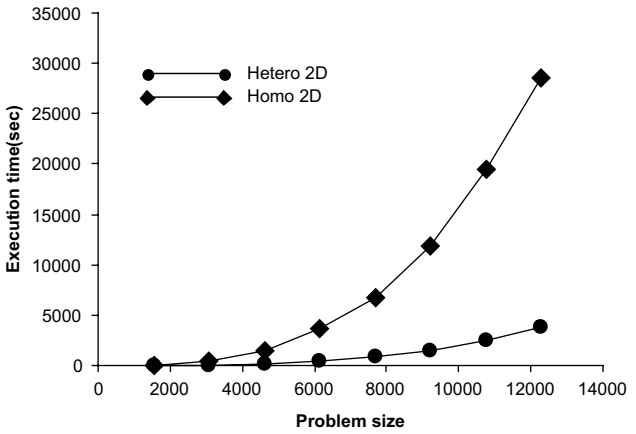

Fig. 6. Execution times of the heterogeneous and homogeneous algorithms on the same heterogeneous network.

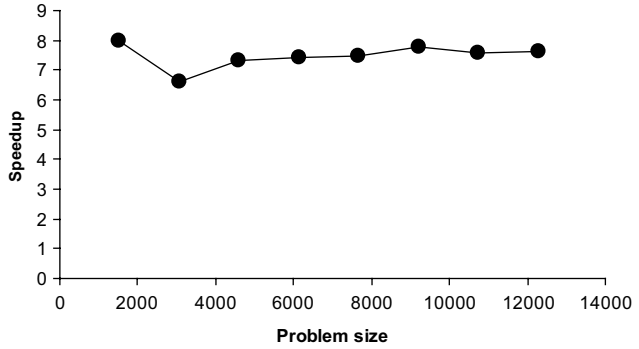

Fig. 7. The speedup of the heterogeneous algorithm over the homogeneous one. Both algorithms are performed on the same heterogeneous network.

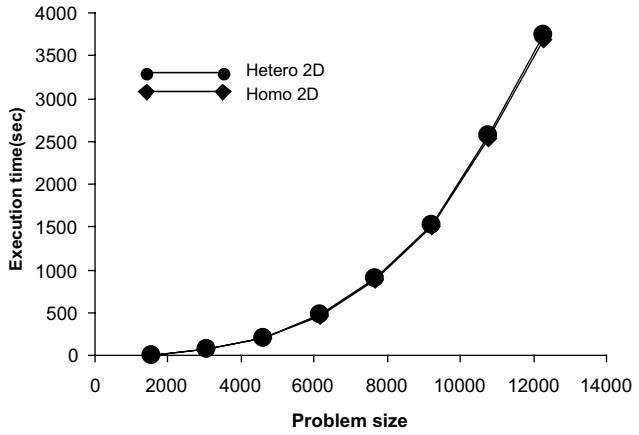

Fig. 8. Execution times of the heterogeneous algorithm on the heterogeneous network and of the homogeneous one on the homogeneous network. The two networks have approximately the same aggregate power of processors and share the same (homogeneous) communication network.

of the processors of the heterogeneous network is the same as that of the homogeneous one.

Fig. 6 shows the comparison of the execution times of 2 parallel algorithms of matrix multiplication:

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

One can see that the heterogeneous algorithm is approximately seven times faster than the homogeneous one.

Fig. 7 shows the speedup of the heterogeneous algorithm over the homogeneous one. The speedup is calculated as the execution time of the homogeneous algorithm divided by the execution time of the heterogeneous algorithm for the same problem size.

Fig. 8 shows the comparison of the execution times of the heterogeneous algorithm performed on the heterogeneous network and its homogeneous prototype performed on the homogeneous network. One can see that the algorithms show 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 experiments show 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 networks of computers consisting of reasonably heterogeneous processors interconnected via a homogeneous communication network, 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.

In our experiments with the heterogeneous algorithm of matrix multiplication we use a fixed 3×3 arrangement of processors. At the same time, given a shape of arrangement and a set of processors, there is still freedom in mapping of the processors onto the shape. In particular, different mappings of the processors onto the 3×3

Table 1
Specifications of 12 heterogeneous computers

Machine name	Architecture	cpu MHz	Main memory (KB)	Cache (KB)
Csserver	Linux 2.4.20-20.9bigmem Intel(R) Xeon(TM)	2783	7,933,500	512
pg1cluster01	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1,030,508	512
pg1cluster02	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1,030,508	512
pg1cluster03	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1,030,508	512
pg1cluster04	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1,030,508	512
pg1cluster05	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1,030,508	512
csultra01	SunOS 5.8 sun4u sparc SUNW,Ultra-5_10	440	524,288	2048
csultra02	SunOS 5.8 sun4u sparc SUNW,Ultra-5_10	440	524,288	2048
csultra03	SunOS 5.8 sun4u sparc SUNW,Ultra-5_10	440	524,288	2048
csultra04	SunOS 5.8 sun4u sparc SUNW,Ultra-5_10	440	524,288	2048
csultra05	SunOS 5.8 sun4u sparc SUNW,Ultra-5_10	440	524,288	2048
csultra06	SunOS 5.8 sun4u sparc SUNW,Ultra-5_10	440	524,288	2048

grid may influence the performance of the heterogeneous algorithm. This issue was addressed by Beaumont et al. in [14]. They proved that the optimal mapping should arrange processors in a nonincreasing order of their speed along each row and each column of the 2D arrangement. The 3×3 arrangement used in the experiments is selected to satisfy this criterion.

Another interesting question is how to select the shape of processors arrangement for our heterogeneous parallel algorithm if we have, say, 12 processors. Should we use 2×6, 3×4, 4×3, or 6×2 ? To answer this question we experimented with a

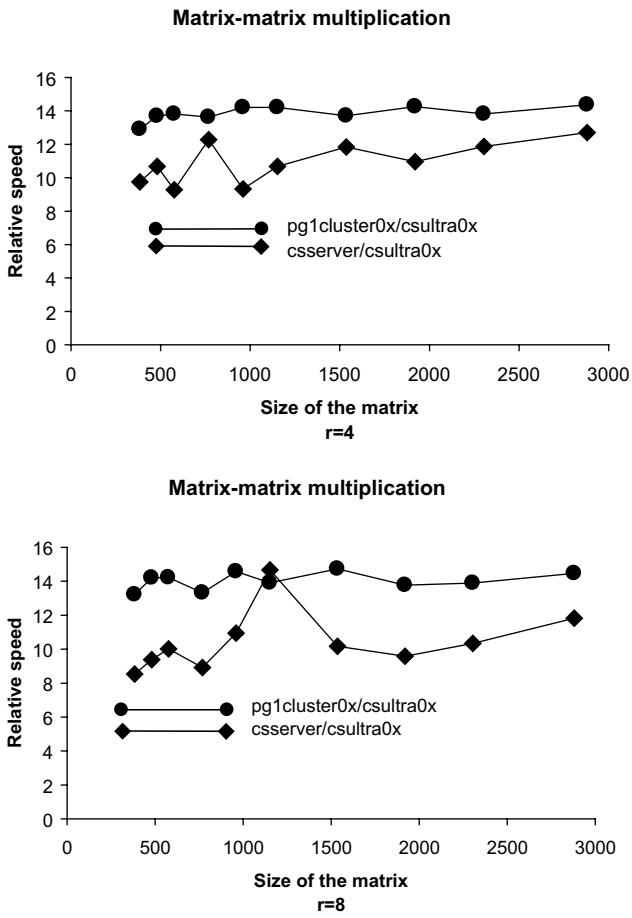

Fig. 9. Relative speeds of computers csserver, pg1cluster0x over the computers csultra0x. The relative speeds are shown for values of r equal to 4 and 8. For $r = 4$, the relative speed of pg1cluster0x computers is approximately 14 times that of the csultra0x computers and the relative speed of csserver computer is approximately 10 times that of the csultra0x computers. For $r = 8$, the relative speed of pg1cluster0x computers is approximately 14 times that of the csultra0x computers and the relative speed of csserver computer is approximately 10 times that of the csultra0x computers.

set of 12 different Solaris and Linux workstations, whose specifications are shown in Table 1, interconnected via a homogeneous communication network based on 100 Mbit Ethernet with a switch enabling parallel communications.

In the experiments, we vary the size of computation unit, **r**, the size of generalised block, **l**, and the shape of arrangement, $p \times q$. We observed that the relative speed of the processors depends on the size of computation block, **r**. Fig. 9 shows the relative speed of some of the processors against the problem size for **r = 4** and **r = 8**. Fig. 10 shows the relative speed of the same processors for **r = 16** and **r = 32** respectively.

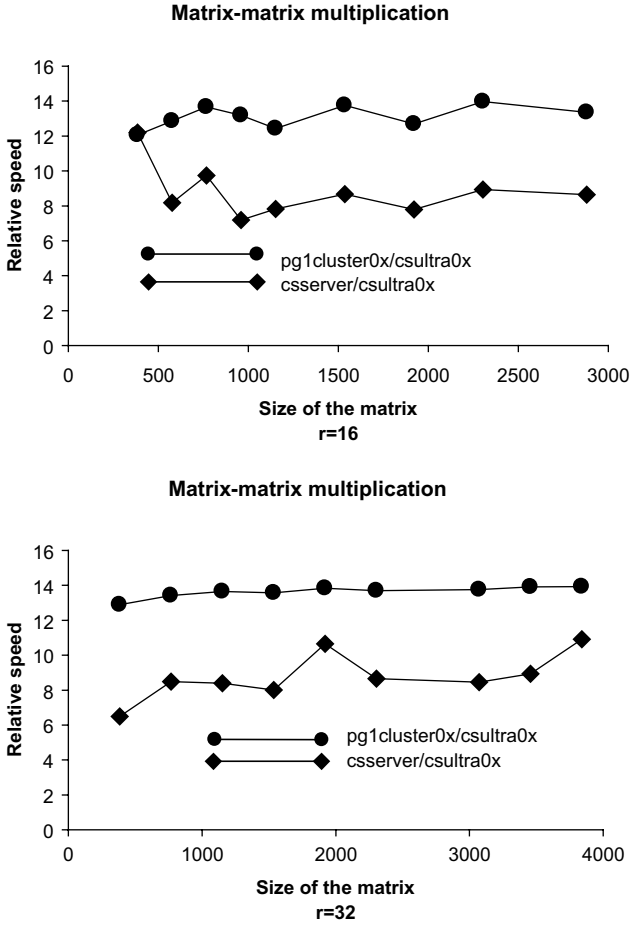

Fig. 10. Relative speeds of computers csserver, pg1cluster0x over the computers csultra0x. The relative speeds are shown for values of r equal to 16 and 32. For $r = 16$, the relative speed of pg1cluster0x computers is approximately 13 times that of the csultra0x computers and the relative speed of csserver computer is approximately 9 times that of the csultra0x computers. For $r = 32$, the relative speed of pg1cluster0x computers is approximately 13.5 times that of the csultra0x computers and the relative speed of csserver computer is approximately 8.5 times that of the csultra0x computers.

One can see that the relative speed of some pairs of processors differs by approximately 15% depending on the size of computation block, **r**. In our experiments, we always partition a generalised block according to the relative speed of the processors observed for the corresponding value of **r**. As in the previous experiments, the processors are mapped to the 2D arrangement in a nonincreasing order of their relative speed along each row and each column of the arrangement.

Tables 2–6 show the results of the experiments for **r = 4**, **r = 8**, **r = 16**, **r = 32** and **r = 64** respectively. In the tables, **N** is the size of matrix. Note that the size of generalised block, **l**, is given in matrix elements not in $r \times r$ blocks. One can see that the shape of arrangement has a visible (sometimes quite significant) impact on the efficiency of the algorithm if values of **r** and **l** are far away from the optimal ones, which are **r = 32** and **l = 3072**. At the same time, it practically does not influence the execution time of the algorithm if the parameter be optimal.

Table 2
Execution times of the heterogeneous parallel matrix multiplication for $r = 4$ (in seconds)

(l,l)	(384, 384)		(768, 768)		**(1536,1536)**		(3072, 3072)	
N	9216	12,288	9216	12,888	**9216**	**12,888**	9216	12,888
$p = 3$, $q = 4$	3456	11,533	3419	10,197	**3371**	**8820**	3840	8581
$p = 4$, $q = 3$	3401	10,790	3369	10,150	**3352**	**8658**	3316	9056
$p = 2$, $q = 6$	6000	9111	5849	8693	**3280**	**8287**	3261	8720
$p = 6$, $q = 2$	3525	8788	3518	8816	**3446**	**8846**	3530	10,578

Table 3
Execution times of the heterogeneous parallel matrix multiplication for $r = 8$ (in seconds)

(l,l)	(384, 384)		(768, 768)		(1536, 1536)		(3072, 3072)		(4608, 4608)
N	9216	12,288	9216	12,888	9216	12,888	9216	12,888	9216
$p = 3$, $q = 4$	2668	6967	2678	7229	**2717**	**6695**	3957	6751	4328
$p = 4$, $q = 3$	3830	7164	2570	6692	**2535**	**6748**	2569	7200	2769
$p = 2$, $q = 6$	2679	7273	2535	6897	2561	6782	**2469**	**6447**	2570
$p = 6$, $q = 2$	2627	6816	2629	6728	2640	6798	**2612**	**6573**	2812

Table 4
Execution times of the heterogeneous parallel matrix multiplication for $r = 16$ (in seconds)

(l,l)	(384, 384)		(768, 768)		(1536, 1536)		(3072, 3072)		(4608, 4608)
N	9216	12,288	9216	12,888	9216	12,888	9216	12,888	9216
$p = 3$, $q = 4$	3890	6837	2370	6218	**2350**	**5980**	3417	6127	4128
$p = 4$, $q = 3$	2344	6053	2368	6298	2306	6162	**2301**	**5802**	2769
$p = 2$, $q = 6$	2586	6685	2337	6386	2237	6206	**2200**	**5840**	2870
$p = 6$, $q = 2$	2591	6718	2388	6412	2381	6350	**2360**	**6280**	2910

Table 5
Execution times of the heterogeneous parallel matrix multiplication for $r = 32$ (in seconds)

(l,l)	(384,384)		(768,768)		(1536,1536)		**(3072,3072)**		(4608,4608)
N	9216	12,288	9216	12,888	9216	12,888	**9216**	**12,888**	9216
$p = 3$, $q = 4$	2616	6580	2286	6142	2212	5855	**2187**	**5702**	3128
$p = 4$, $q = 3$	2816	7180	2246	6085	2210	5842	**2173**	**5690**	2969
$p = 2$, $q = 6$	2670	6880	2346	6185	2252	5930	**2184**	**5890**	3170
$p = 6$, $q = 2$	2570	6180	2446	6042	2352	5942	**2195**	**5790**	3110

Table 6
Execution times of the heterogeneous parallel matrix multiplication for $r = 64$ (in seconds)

(l,l)	(768,768)	(1536,1536)	(3072,3072)	**(4608,4608)**	(9216, 9216)
N	9216	9216	9216	**9216**	9216
$p = 3$, $q = 4$	4456	3322	2722	**2514**	3812
$p = 4$, $q = 3$	4401	3401	2801	**2601**	4610
$p = 2$, $q = 6$	5000	4678	2878	**2569**	3588
$p = 6$, $q = 2$	5525	4323	2723	**2312**	4377

7. Related work

Design of heterogeneous parallel algorithms is typically reduced to the problem of optimal data partitioning of one or other mathematical object such as a set, a matrix, etc. As soon as the corresponding mathematical optimisation problem is formulated, the quality of its solution is assessed rather than the quality of solution of the original problem. As the optimisation problem is typically NP-hard, some sub-optimal solutions are proposed and analysed. The analysis is typically statistical: the sub-optimal solutions for a big number of generated inputs are compared to each other and the optimal one. This approach is used in many papers, in particular, in [1,2,4,9,13]. This approach estimates heterogeneous parallel algorithms indirectly and additional experiments are still needed to assess their efficiency in real heterogeneous environments.

Another approach is just to experimentally compare the execution time of the heterogeneous algorithm with that of its homogeneous prototype or a heterogeneous competitor. Some particular heterogeneous network is used for such experiments. In particularly, this approach is used in [3,15,16]. This approach directly estimates the efficiency of heterogeneous parallel algorithms in some real heterogeneous environment but still leaves an open question about their efficiency in other particular heterogeneous environments.

The approach presented in this paper is to carefully design a relatively small number of experiments in a real heterogeneous environment in order to experimentally compare the efficiency of the heterogeneous parallel algorithm with some experimentally obtained ideal efficiency (namely, the efficiency of its homogeneous prototype in an equally powerful homogeneous environment). This approach directly estimates

the efficiency of heterogeneous parallel algorithms providing relatively high confidence in the results of such an experimental estimation.

An ideal approach would be to use a comprehensive performance model of heterogeneous networks of computers for analysis of the performance of heterogeneous parallel algorithms in order to predict their efficiency without real execution of the algorithms in heterogeneous environments. Some research efforts have been done in this direction [8,9,17,18], but the problem is still far away from its comprehensive solution.

Several authors consider scalability more important property of heterogeneous parallel algorithms than efficiency. They propose some approaches to analysis of scalability of heterogeneous parallel algorithms [19,20].

Many authors studied parallel algorithms of matrix multiplication on heterogeneous platforms [3,4,13,15,16].

8. Conclusion and future work

In this paper, we have presented a new approach to performance analysis of heterogeneous parallel algorithms, which is to compare the efficiency of the heterogeneous algorithm on the heterogeneous network with the efficiency of its homogeneous prototype on the homogeneous network having the same aggregate performance as the heterogeneous one.

We have also illustrated how to apply this approach to assessment of a concrete heterogeneous parallel algorithm, which implements matrix multiplication on heterogeneous networks of a particular type.

Our future research on this approach will focus on its application to heterogeneous parallel algorithms aimed at networks of computers with essentially heterogeneous communication layers.

References

[1] P. Crandall, M. Quinn, Block data decomposition for data-parallel programming on a heterogeneous workstation network, in: Proceedings of the Second International Symposium on High Performance Distributed Computing, pp. 42–49, 1993.

[2] M. Kaddoura, S. Ranka, A. Wang, Array decomposition for nonuniform computational environments, Journal of Parallel and Distributed Computing 36 (2) (1996) 91–105.

[3] A. Kalinov, A. Lastovetsky, Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers, Journal of Parallel and Distributed Computing 61 (4) (2001) 520–535.

[4] O. Beaumont, V. Boudet, F. Rastello, Y. Robert, Matrix multiplication on heterogeneous platforms, IEEE Transactions on Parallel and Distributed Systems 12 (10) (2001) 1033–1051.

[5] S. Fortune, J. Wyllie, Parallelism in random access machines, in: Proceedings of the 10th Annual Symposium on Theory of Computing, pp. 114–118, 1978.

[6] L.G. Valiant, A bridging model for parallel computation, Communications of the Association for Computing Machinery 33 (8) (1990) 103–111.

[7] D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, T. von Eicken, Log P: 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, May 1993.

[8] A. Lastovetsky, Adaptive parallel computing on heterogeneous networks with mpC, Parallel Computing 28 (10) (2002) 1369–1407.

[9] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, Y. Robert, Bandwidth-centric allocation of independent tasks on heterogeneous platforms, in: Proceedings of 16th International Parallel and Distributed Processing Symposium (IPDPS'2002), IEEE Computer Society, CD-ROM, 2002.

[10] L. Blackford, J. Choi, A. Cleary, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. Whaley, ScaLAPACK: a portable linear algebra library for distributed memory computers—design issues and performance, Proceedings of Supercomputing'96, 1996.

[11] High Performance Fortran Forum, High Performance Fortran Language Specification, version 2.0, 1997.

[12] A. Kalinov, A. Lastovetsky, Heterogeneous distribution of computations while solving linear algebra problems on networks of heterogeneous computers, in: Proceedings of the 7th International Conference on High Performance Computing and Networking Europe (HPCN Europe'99), Lecture Notes in Computer Science 1593 (1999) 191–200.

[13] O. Beaumont, V. Boudet, F. Rastello, Y. Robert, Heterogeneous matrix–matrix multiplication or partitioning a square into rectangles: NP-completeness and approximation algorithms, in: Proceedings of the 9th Euromicro Workshop on Parallel and Distributed Processing, pp. 298–302, IEEE Computer Society Press, 2001.

[14] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, Y. Robert, A proposal for a heterogeneous cluster ScaLAPACK (Dense linear solvers), IEEE Transactions on Computers 50 (10) (2001) 1052–1070.

[15] E. Dovolnov, A. Kalinov, S. Klimov, Natural block data decomposition for Heterogeneous Clusters, in: Proceedings of 17th International Parallel and Distributed Processing Symposium (IPDPS'2004), IEEE Computer Society, CD-ROM, 2003.

[16] Y. Ohtaki, D. Takahashi, T. Boku, M. Sato, Parallel implementation of Srassen's Matrix multiplication algorithm for heterogeneous clusters, Proceedings of 18th International Parallel and Distributed Processing Symposium (IPDPS'2004), IEEE Computer Society, CD-ROM, 2004.

[17] Y. Yan, X. Zhang, Y. Song, An effective and practical performance prediction model for parallel computing on nondedicated heterogeneous NOW, Journal of Parallel and Distributed Computing 38 (1) (1996) 63–80.

[18] A. Clematis, A. Corana, Modeling performance of heterogeneous parallel computing systems, Parallel Computing 25 (9) (1999) 1131–1145.

[19] X.-H. Sun, Scalability versus execution time in scalable systems, Journal of Parallel and Distributed Computing 62 (2) (2002) 173–192.

[20] A.Kalinov, Scalability analysis of matrix–matrix multiplication on heterogeneous clusters, in: Proceedings of ISPDC'2004/HeteroPar'04, IEEE Computer Society, 2004.