ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2008, Vol. 37, No. 1

ANALYSIS OF PARALLEL CALCULATIONS IN COMPUTER NETWORK

Aleksandr Igumenov[1], Tomas Petkus[2]

[1]*Institute of Mathematics and Informatics, Akademijos str. 4, LT-08663, Vilnius, Lithuania*
[2]*Vilnius Pedagogical University, Studentu 39, LT-08106 Vilnius, Lithuania*

Abstract. In various spheres of activities, we often face the necessity to find the best or fastest way of solving a relevant problem. Nowadays, computer technologies are applied in solving most problems: in great flow of information processing as well as in scientific research. Complex problems, solution of which is problematic by using only one computer, are solved by several computers connected into a network. Seeking effective usage of computer network for parallel calculations conceptually different software can be applied.

In the article, the analysis of clusters by classical parallel solution algorithms is presented. An approximate value of π, matrix and vector and a system of linear equations and matrices product algorithms are analyzed. The present study consider a cluster that consists of 60 processors, as well as problem solution of cluster operating in the system with Windows XP at different speeds of networks.

1. Introduction

Speaking about parallel calculations, first of all data processing procedures are meant, when several operations of a computer system may be performed at the same time. Parallelism may be achieved if we have:

- Operation independence of separate calculating machines
- Adequacy of calculating system elements:
  - applying specialised equipment;
  - duplicating calculating machines.

Analysing organisation problems of parallel calculations, the following execution modes of independent program parts should be distinguished:

- Multitask (time distribution mode).
- Parallel execution.
- Distributed calculations.

In this article, systems working in the second mode are analysed (parallel execution).

2. Parallel calculations

Parallel algorithms

According to the tasks solved, parallel algorithms are distributed into several classes: data parallelism algorithms, functional parallelism algorithms, master-slave type algorithms.

Master-slave type algorithms may be applied to all tasks, where lists of tasks are not interdependent.

Master-slave type algorithms are analyses, when the computer-master has the list of tasks and distributes them to computer-slaves for solving.

The task distribution among computers is one of the major problems in the construction of effective parallel algorithms [1].

First, we have to stress that a master-slave algorithm is a "one-to-many" communication model type algorithm, i. e., one computer, called *master,* communicates with many other computers called *slaves.* Slaves do not exchange information among themselves. Such a limitation eliminates a large class of tasks, where the parallel solution of tasks requires the information from neighbouring tasks.

Master-slave type algorithms may not be applied to all tasks, because the peculiarities of some tasks allow speedup and good efficiency of the parallel algorithm only when special algorithms are created for that task.

Let us discuss the operational scheme of master-slave algorithms (Figure 1).

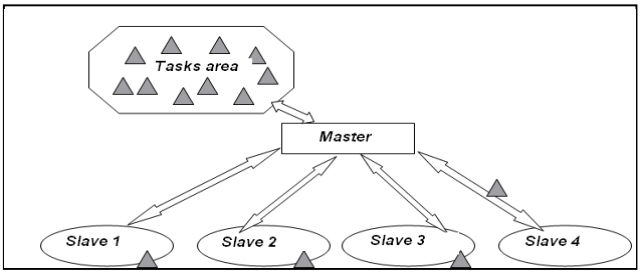

Figure 1. Executing scheme of master-slave algorithms

The computer-master generates and saves the list of tasks from which the tasks are distributed to computer-slaves. After calculating the received task, the computer-slaves send the results back to the master, which stores the common result and supplements the list of tasks. The arrows mark here data interchange among computers. Let us point out the fact that, at the same moment various computer-slaves may operate with absolutely different algorithms (it is functional parallelism). Most often there are such cases in practice, where all computer-slaves execute the same algorithm, but only with different data (data parallelism type algorithms) [2].

Criteria for evaluation of parallel algorithms

The first criterion that might be used to compare several parallel algorithms is *speedup of the parallel algorithm* when solving the problem with p processors:

$$S_p = \frac{T_1^*}{T_p}. \quad (1)$$

Here T_1^* is the time required for solving the task with the best known serial algorithm and T_p is the time needed for parallel algorithm to solve the problem with p processors.

Note that T_1^* and T_1 are most often different, because not all serial algorithms may be completely parallelised and so the fastest serial algorithm differs from the parallel one. The inequality $T_1^* \le T_1$ is always true when a parallel algorithm is made up by modifying the serial one. The modification of constructions appends an additional program which slows down the algorithm. It is also noteworthy that, the different parallel algorithms solving the same task in different ways may gain an advantage when there is a certain number of a processor, but rather poor when with another quantity of processors. All that may depend on the parallel computer.

Another criterion that might help to evaluate the parallel algorithm is *the efficiency coefficient of processors* [3]:

$$E_p = \frac{S_p}{p}. \quad (2)$$

The efficiency shows how the processors are used. In solving certain problems, a high efficiency may be achieved only when there is a fixed number of processors, but with an increase in their number, the efficiency decreases. The efficiency depends not only on the problem, but also on the parallel algorithm and the parameters of parallel computers: granulation, calculation speed of processors, communication speed, etc. The efficiency of the same algorithm in different computers may vary very much. Traditionally, the higher communication speed of processors and the slower the processors the better efficiency is achieved. Amdahl Law [2] states that the efficiency of any parallel algorithm is lower than 1, and with an increase in number of processors it decreases.

3. Software and hardware

We used here the software package MPICH v.1.2.5 for solving parallel problems in a computer network. This kit may operate only with programs created using not lower than the Microsoft Visual C++ 6.0 version. MS Office programs were used for data processing and presentation.

Cluster of the Faculty of Mathematics and informatics, Vilnius Pedagogical University. This cluster consists of 60 computers, where 30 of them are Pentium III class and 30 are Pentium 4 class computers. The computer specifications are:

15 computers:
- Processor – Intel® Pentium 3,0 GHz (with Hyper Treading);
- RAM – DDR2 512 Mb RAM;
- HDD – SATA2 80 Gb, Cach 8 Mb, data transmission speed 300 Mb/s;
- 1 Gb integrated Ethernet adapter.

15 computers:
- Processor – Intel® Pentium 3,2 GHz (with Hyper Treading);
- RAM – DDR 512 Mb RAM;
- HDD – SATA 80 Gb, Cach 8 Mb, data transmission speed 150 Mb/s;
- 100 Mb integrated Ethernet adapter.
- 1 Gb D-Link Ethernet adapter

15 computers:
- Processor – Intel® Celeron 2,0 GHz;
- RAM – DDR 256 Mb RAM;
- HDD – IDA 40 Gb, Cach 8 Mb;
- 100 Mb integrated Ethernet adapter.

15 computers:
- Processor – Intel® Celeron 1 GHz;
- RAM – DDR 256 Mb RAM;
- HDD – IDA 40 Gb, cach 8 Mb;
- 100 Mb Ethernet adapter.

The total cluster RAM makes up to 22,5 Gb with standing memory 3,5 Tb.

4. Classical algorithms of parallel calculations

The parallel algorithm is decomposed into processes that can be executed in a parallel way. Several processes may be executed on one processor. Thus, the number of processes and processors can be different.

Calculation of π

There exists a function:

$$f(x) = \frac{1}{1+x^2}. \quad (3)$$

In geometry, a definite integral means an area under a curvilinear trapezoid, limited by the lines $x = a$, $x = b$, $y = 0$ and the function $f(x)$:

$$\int_0^1 \frac{1}{1+x^2}dx = \arctan(1) - \arctan(0) = \frac{\pi}{4}. \qquad (4)$$

From this expression we get the π value:

$$\pi = \int_0^1 \frac{4}{1+x^2}dx. \qquad (5)$$

One of the integral calculation methods is calculation by partial sums (Riemann sums, the left rectangle method).

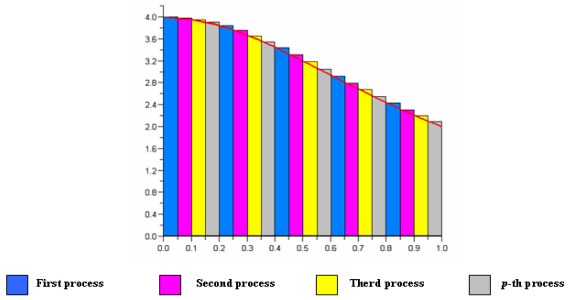

Figure 2. Illustration of the rectangle method.

The distribution of tasks among the processors is shown in Figure 2. The process-slaves send their own sums to the computer-master, after finishing the calculation of their sums. The computer-master calculates the total sum (Riemann) of process-slaves sums. In this way, an approximate value of π is obtained.

An algorithm for solving a system of linear equations applying the Gauss method

The general form of a linear equation is as follows:

$$a_0x_0 + a_1x_1 + \cdots + a_{n-1}x_{n-1} = b. \qquad (6)$$

The system of linear equations (*SLE*) has the following general form:

$$\begin{cases} a_0x_0 + a_0x_1 + \cdots + a_0x_{n-1} = b_0 \\ a_1x_0 + a_1x_1 + \cdots + a_1x_{n-1} = b_1 \\ \vdots \quad\quad \vdots \\ a_{n-2}x_0 + a_{n-2}x_1 + \cdots + a_{n-2}x_{n-1} = b_{n-2} \\ a_{n-1}x_0 + a_{n-1}x_1 + \cdots + a_{n-1}x_{n-1} = b_{n-1}. \end{cases} \qquad (7)$$

By the elementary rearrangement the system (7) is reduced to a triangular form, which is called an echelon form, and looks like this:

$$\begin{cases} a'_{0\,0}x_0 + a'_{0\,1}x_1 + \cdots + a'_{0\,n-1}x_{n-1} = b'_0 \\ 0 + a'_{1\,1}x_1 + \cdots + a'_{1\,n-1}x_{n-1} = b'_1 \\ \vdots \quad\quad \vdots \\ 0 + 0 + \cdots + a'_{n-1\,n-1}x_{n-1} = b'_{n-1}. \end{cases} \qquad (8)$$

That stage of the Gauss method is called direct process of algorithm (forward elimination) during which the variables x_{n-1} is calculated (Figure 3).

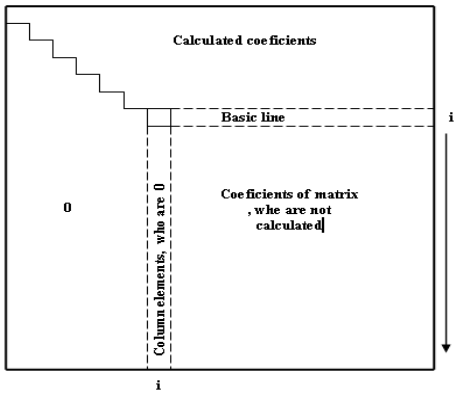

Figure 3. The scheme of the *i*-th step of algorithm of the Gaussian elimination method (direct process)

A reverse process of algorithm (backward elimination) starts (Figure 4), after *SLE* has acquired the form (8). The result is achieved by way of rearrangements. Thus, we have a rearranged *SLE* (9) (reduced echelon form). This is the way of computing all the variables x_i, where $i = 0, 1, 2, 3, \ldots, n\text{-}2$.

$$\begin{cases} a''_{0\,0}x_0 + 0 + \cdots + 0 = b''_0 \\ 0 + a''_{1\,1}x_1 + \cdots + 0 = b''_1 \\ 0 + 0 + \cdots + a''_{n-1\,n-1}x_{n-1} = b''_{n-1} \end{cases} \qquad (9)$$

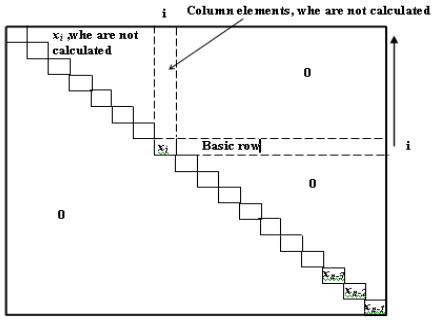

Figure 4. The scheme of the *i*-th step of algorithm of the Gauss method (reverse process)

The data distribution of a parallel algorithm of the Gauss method among processes is shown in Figure 5.

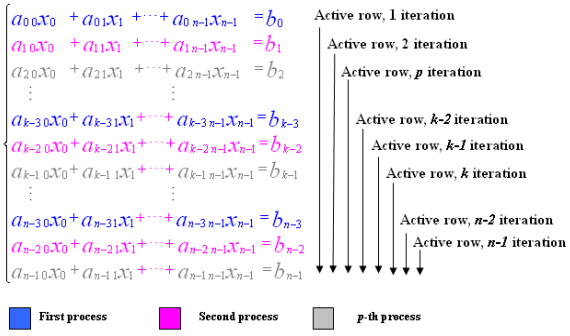

Figure 5. The scheme of data distribution of the parallel algorithm applying the Gauss method

An algorithm for the matrix and vector product

In the algebra and number theory the matrix and vector product is calculated as follows:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} * \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} a_{1\,i} * b_i \\ \sum_{i=1}^{n} a_{2\,i} * b_i \\ \sum_{i=1}^{n} a_{3\,i} * b_i \\ \vdots \\ \sum_{i=1}^{n} a_{m\,i} * b_i \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_m \end{pmatrix}. \quad (10)$$

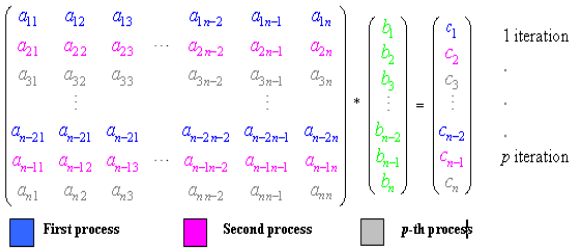

Figure 6. The scheme of data distribution among processes multiplying the matrix by a vector

Parallel multiplication of matrix and vector may be obtained by applying several methods: transmitting a row of matrix A and the vector $\vec{b}$ to the process-slaves or the fragments of matrix A and the vector $\vec{b}$ to the process-slaves.

In this article, the first method is analysed, where the process-master distributes a rows of matrix A and the vector $\vec{b}$ among process-slaves. Solving the task, each process multiplies two vectors $\vec{a}_i$ and $\vec{b}$. In experiments the matrix is quadratic with the rank $r(A)$=1000 and the vector $\vec{b}$ has 1000 components (generated at random). The distribution of data among the processes is illustrated in Figure 6.

An algorithm for multiplication of matrices

There are parallel realisations of matrices multiplying:

- bar data distribution
- data distribution by blocks:
  - *Fox* algorithm;
  - *Cannon* algorithm.

Fox and *Cannon* algorithms differ only by data distribution among processes. In this article, the *Fox* algorithm is analysed: data distribution by blocks when the number of processes is $p = s * q$. Besides, the number of matrix rows has to be iterative of number s, and the number of columns is iterative of number q:

$$\begin{pmatrix} A_{00} & \cdots & A_{0q-1} \\ \vdots & & \vdots \\ A_{q-10} & \cdots & A_{q-1q-1} \end{pmatrix} \times \begin{pmatrix} B_{00} & \cdots & B_{0q-1} \\ \vdots & & \vdots \\ B_{q-10} & \cdots & B_{q-1q-1} \end{pmatrix} = \begin{pmatrix} C_{00} & \cdots & C_{0q-1} \\ \vdots & & \vdots \\ C_{q-10} & \cdots & C_{q-1q-1} \end{pmatrix} \quad (11)$$

Here each block C_{ij} of matrix C is found as follows:

$$C_{ij} = \sum_{s=0}^{q-1} A_{is} B_{sj}. \quad (12)$$

The number of blocks is $k*k$, $k=n/q$. The program GRID topology has been created to facilitate data transmission and calculation.

5. Experimental results

The master-slave strategy has been applied in experiments. The task was formed by the process-master and distributed to process-slaves. The answer is formed in the process-master by collecting the data from process-slaves.

Approximate calculations of the π value

The interval [0, 1] was divided into 1.000.000.000 parts for execution of experiments.

In experiments, we calculate π with 1 Gb and 100 Mb computer networks. The achieved results are compared. The results are illustrated in Figure 7 where one computer solves one process, and the results where one computer solves two processes are illustrated in Figure 8.

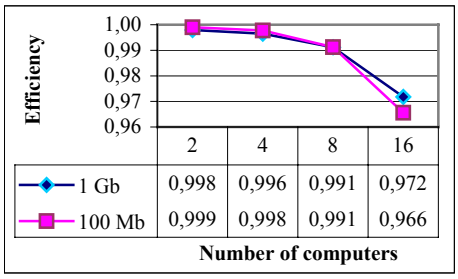

Figure 7. The algorithm efficiency in different networks, where one computer solves one process

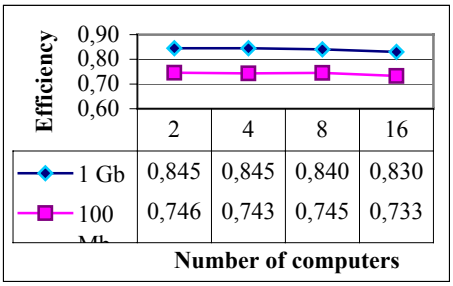

Figure 8. The algorithm efficiency in different networks, where one computer solves two processes

Parallel solution of SLE applying Gauss method

To carry out experiments, an *SLE* of 1000 equations was generated. The experiments were done with 1 Gb and 100 Mb computer networks.

The experiments were carried out to solve the *SLE* by applying the Gauss method with 1 Gb and 100 Mb computer networks and the results obtained were compared. The results are illustrated in Figure 9 where one computer solves one process, and the results

where one computer solves two processes are illustrated in Figure 10.

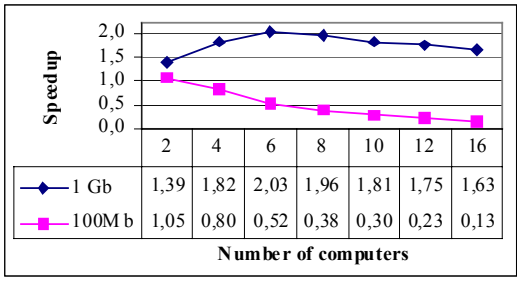

Figure 9. Speedup comparison in different networks where one computer solves one process

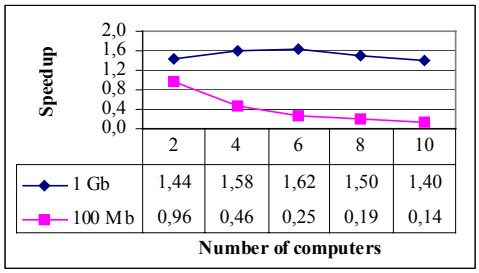

Figure 10. Speedup comparison in different networks where one computer solves two processes

The multiplication of matrix and vector

In experiments, we used a square matrix with the rank $r(A) = 1000$ and a vector of 1000 components. The primary data (matrix A and vector $\vec{b}$) were randomly generated.

The experiments have been carried out by multiplying the matrix by the vector, using 1 Gb and 100 Mb computer networks. The obtained results were compared. The results are illustrated in Figure 11 where one computer solves one process, and the results where one computer solves two processes are illustrated in Figure 12.

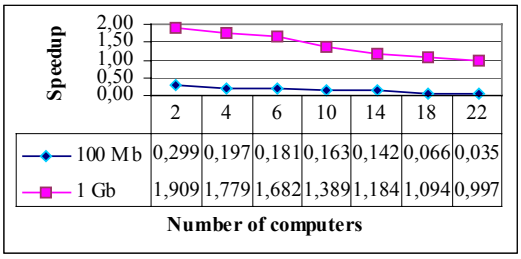

Figure 11. Speedup comparison in different networks where one computer solves one process

The multiplication of matrices

The experiments were done with square matrices of different rank. The primary data were randomly generated (matrices A and B). The experiments were carried out with 1 Gb and 100 Mb computer networks. A series of experiments done with the 1 Gb computer network are illustrated in Figure 13. Other experiments carried out with the 100 Mb computer network are illustrated in Figure 14.

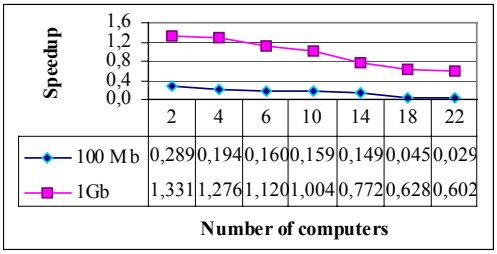

Figure **12.** Speedup comparison in different networks where one computer solves two processes

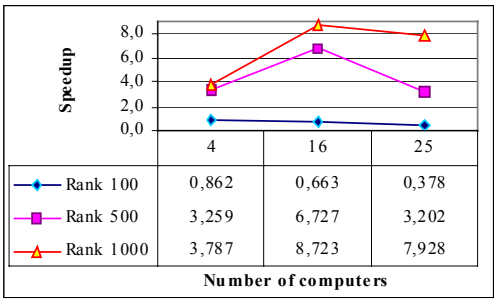

Figure 13. Speedup comparison in the 1 Gb network

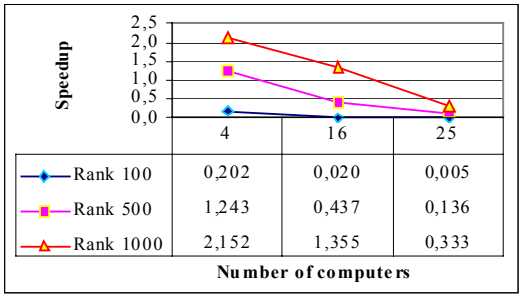

Figure 14. Speedup comparison in the 100 Mb network

The experiments were performed with the 1 Gb and 100 Mb computer networks. The results were compared, when the rank of the matrix was 100 (Figure 15), 500 (Figure 16), and 1000 (Figure 17).

Analysis of the experimental results

All the experiments have been carried out at the Vilnius Pedagogical University cluster that operates on the basis of Windows XP. Windows XP has one very serious shortcoming: there are too many system transmissions in the local network. It may be experimentally illustrated very well by multiplication of matrices with the rank $r = 100$ and by solving the *SLE*. Besides, the formulation of a primary task is predetermined by computer-master resources. For example, by multiplying matrices with the rank $r = 1000$, the required memory is about 22 MB, and with the rank $r = 10000$ it requires about 2 GB. Furthermore, as experiments have shown, application of the 100 Mb network is reasonable for solving tasks with a low network load, e.g., approximate calculation of the value π (Figures 7 and 8).

Analysing the results of experiments of *SLE* (Figures 9 and 10) it has been found out that application of both the 100 Mb network and the 1 Gb network is not reasonable. The maximum speedup with the 1 Gb network is reached only if 6 computers are available,

i.e. only 2.03 times faster than 1 computer is used. It happens due to a very frequent exchange of data among processes (this is very well illustrated in Figure 10). In order to increase the efficiency of algorithm in the network, it is necessary to change the data exchange strategy: to use a faster network or to reduce the portions of primary data.

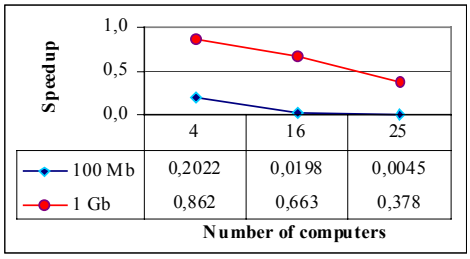

Figure 15. Speedup comparison where the matrix rank $r = 100$

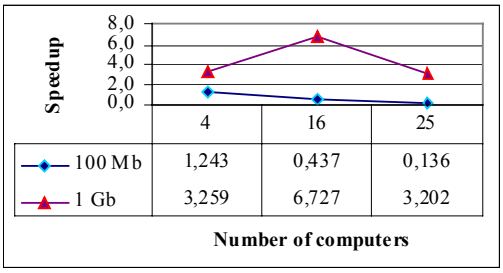

Figure 16. Speedup comparison where the matrix rank $r = 500$

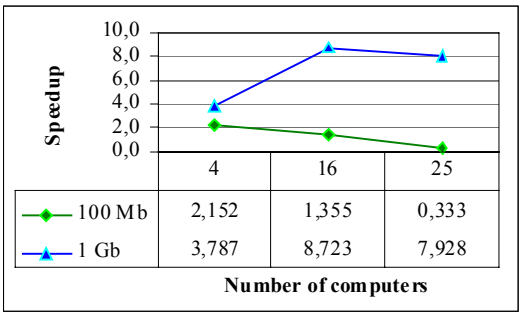

Figure 17. Speedup comparison where the matrix rank $r = 1000$

In the experiments, the data of matrix and vector multiplication (Figures 11 and 12) are analogous to the experiments of *SLE solving*. However these results have one exclusive feature: the maximum speedup is achieved when the task is solved by two computers and it is approximately equal to 2. Thus, application of the 100 Mb network is not reasonable here as well. This is due to the reasons mentioned above: low network capacity and a great amount of data to exchange among processes.

In the experiments of multiplying the data of two matrices illustrate (Figures 13-17) that the data transmission by blocks is more time-consuming than calculations themselves when the rank of matrices is $r = 100$. By increasing the rank of matrices up to $r = 500$ we can observe some speedup. The maximum speedup on the 100 Mb network is achieved with 4 computers and is approximately equal to 1.2 times. The maximum speedup has been achieved with 4 computers, and almost the same results have obtained (*2.15* times) with the *rank* $r = 1000$.

A qualitative change is obtained by multiplying matrices on the 1 Gb network. There is no sense to use the 1 Gb computer network when the rank of matrices is $r = 100$. The rank of matrices being $r = 500$ and $r = 1000$, the results are better. The maximum speedup, with $r = 1000$, is achieved using 16 computers and it is equal to *8.7* times. Furthermore, a considerable speedup is achieved with 4 computers. By carrying out further experiments it has been observed that the results are highly influenced by the network capacity.

Conclusions

Several classical parallel algorithms have been analysed in this article. After several experiments, we can draw the following conclusions:

- Realization of parallel algorithms accelerates solution of problems compared with serial algorithms.
- 100 Mb computer network with MS Windows XP is useless it a larger amount of data transmission is required for problem solution.
- The computer-master must have much higher characteristics than computer-slaves, if large-scale problems with matrices are solved. It is very important to evaluate the available computer resources.
- Multiplying large matrices or a large matrix and vector, it is useful to apply distribution by blocks.
- It is reasonable to use smaller portions of primary data in order to reduce network load, if a linear data distribution method is used for multiplying matrices or matrix and vector.

Acknowledgement

The research is partially supported by the Lithuanian State Science and Studies Foundation project "The research of human factors in multiple criteria optimization problems applying parallel computing (No. T-33/07)".

References

[1] Technology of paralleling. *http://www.spbcas.ru/cfd/techn/Parallel.htm.*

[2] **R. Ciegis, R. Sablinskas.** Analysis of the Efficiency of Master-Slave Parallel Algorithm (*In Lithuanian*), *Vilnius*, 1998, 390-392

[3] **T. Petkus.** Application of Computing Network in Interactive Optimization. *Vilnius, VPU*, 2001.

[4] **R. Ciegis.** Parallel algorithms (*In Lithuanian*), *Vilnius, Technika*, 2001, 145-156.

Received November 2007.