Data Partitioning with a Realistic Performance Model of Networks of Heterogeneous Computers

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

Ravi Reddy
Department of Computer Science
University College Dublin, Belfield
Dublin 4, Ireland
Manumachu.Reddy@ucd.ie

Abstract

The paper presents a performance model of a network of heterogeneous computers that takes account of the heterogeneity of memory structure and other architectural differences. Under this model, the speed of each processor is represented by a function of the size of the problem whereas standard models use single numbers to represent the speeds of the processors. We prove that this model is more realistic than the standard ones when the network includes computers with significantly different memory structure. We formulate a problem of partitioning of an $\mathbf{n}$*-element set over* $\mathbf{p}$ *heterogeneous processors using this advanced performance model and give its efficient solution of the complexity* $O(\mathbf{p}^2 \times log_2\mathbf{n})$.

1. Introduction

Networks of heterogeneous computers are a promising parallel architecture attracting more and more researchers. Unlike traditional homogeneous parallel platforms, this heterogeneous parallel architecture uses processors running at different speeds. This makes programming networks a challenging task. Data and computations should be distributed unevenly to provide the best execution performance.

A number of algorithms of parallel solution of scientific and engineering problems on heterogeneous networks have been designed and implemented [1-4]. They use different performance models of networks of heterogeneous computers to distribute computations amongst the processors involved in their execution. But all the models use a single positive number to represent the speed of a processor, and computations are distributed amongst the processors such that their volume is proportional to this speed of the processor.

The paper presents a more advanced performance model of a network of heterogeneous computers. Under this model, the speed of each processor is represented by a function of the size of the problem. We start with motivation behind the conception of this model followed by its presentation and demonstrate that it is more realistic than the standard ones when the network includes computers with significantly different memory structure. Then we formulate a problem of partitioning of an $\mathbf{n}$-element set over $\mathbf{p}$ heterogeneous processors using this advanced performance model and give its efficient solution of the complexity $O(\mathbf{p}^2 \times log_2\mathbf{n})$.

2. Performance model

It is a well known fact that as the size of the problem increases, the data elements involved in the execution of the application move to lower levels of memory hierarchy thus resulting in the decrease of absolute speed of execution of the application. This is because more machine cycles are used to access the data elements at lower levels of hierarchy. Despite absolute speed of execution of an application being a decreasing function of the size of the problem, in many real life situations, the relative speeds of the processors involved in the execution of the application are a constant function of the size of the problem and thus can be approximated by a single number.

This is shown to be the case for a small network of two processors running matrix-matrix

Table 1. Specifications of the two computers

Machine Name	cpu MHz	Main Memory (KB)	Cache (KB)
Comp1	499	513960	512
Comp2	440	524288	1024

Table 2. Specifications of the four computers

Machine Name	cpu MHz	Main Memory (KB)	Cache (KB)
Comp1	499	513960	512
Comp2	440	524288	1024
Comp3	996	254576	256
Comp4	499	126176	512

multiplication and Cholesky factorization and whose sizes of the levels of memory hierarchy are shown in Table 1. The processors demonstrate absolute speeds that are decreasing functions of the size of the problem as shown in Figure 1 but demonstrate approximately the same relative speed as shown in Figure 2. One can see that the processors have almost the same size at each level of their memory hierarchies. However if the processors have significantly different sizes at each level of their memory hierarchies, they may not demonstrate relative speeds, which are constant functions of the size of the problem. This is shown in Figure 3 for a network of four computers. The sizes of the levels of memory hierarchy for these computers are shown in Table 2. As can be seen from Figure 3, each pair of processors with significantly different memory structure exhibits non-constant relative speeds as the size of the problem increases. If we use such networks of heterogeneous computers for execution of parallel or distributed algorithms, we cannot represent their speed by a single number. Realistically we must represent the speed by a function of the size of the problem.

In the experiments presented in Figure 3 we used straightforward serial algorithms. One may argue that this analysis is applicable to distributed computing rather than to parallel computing as parallel algorithms are normally based on serial algorithms much more efficiently using the memory hierarchy than the straightforward ones. Experiments presented in Figure 4 deal with serial algorithms derived from the two-dimensional block-cyclic parallel algorithms implemented in ScaLAPACK [5]. Indeed in this case relative speeds can be approximated by single numbers for much wider range of problem size but become far away from constants functions for larger problem sizes.

Another interesting output of the experiments is that even computers with similar structures of memory hierarchy (such as **Comp5** and **Comp7**) may demonstrate significantly different relative speeds for different sizes of the problem. Therefore straightforward extension of the standard performance model by adding such additional parameters as the number of memory levels and the size of each level will not work in this case.

We suggest a realistic performance model of networks of heterogeneous computers where each processor is represented by its absolute speed as a function of problem size. This integrated approach takes account of *all* architectural differences in computers having an impact on the performance of computers depending on the size of the problem. Our choice of the absolute speed is explained by the fact that the absolute speed is guaranteed to be a decreasing (or at least non-increasing) function of problem size, which may not be the case for a relative speed. This property of monotony is very important in designing efficient partitioning algorithms with the advanced performance model.

3. Algorithms of partitioning sets

One of the criteria to partitioning a set of **n** elements over **p** heterogeneous processors is that the number of elements in each partition should be

Table 3. Specifications of the seven computers

Machine Name	cpu MHz	Main Memory (KB)	Cache (KB)
Comp3	997	254576	256
Comp5	997	511800	256
Comp6	997	511800	256
Comp7	2793	513304	512
Comp8	1977	1030508	512
Comp9	2783	7933500	512
Comp10	499	7143360	512

proportional to the speed of the processor owning that partition. When the speed of the processor is represented by a single number as in the case of standard performance models of heterogeneous networks, the algorithm used to perform the partitioning is quite straightforward, of complexity O(***p***).

This problem of partitioning a set becomes non-trivial when the speeds of the processors are given as a non-increasing function of the size of the problem. Consider a small network of two processors, whose speeds as functions of problem size during the execution of the matrix-matrix multiplication are shown in Figure 5. If we use standard approach, we have to choose a point and use the absolute speeds of the processors at that point to partition the elements of the set such that the number of elements is proportional to the speed of the processor. If we choose the speeds (s_{00}, s_{01}) at points (x, s_{00}) and (x, s_{01}) to partition the elements of the set, the distribution obtained will be unacceptable for the size of the problem at points (y, s_{10}) and (y, s_{11}) where processors demonstrate different relative speeds compared to the relative speeds at points (x, s_{00}) and (x, s_{01}). If we choose the speeds (s_{10}, s_{11}) at points (y, s_{10}) and (y, s_{11}) to partition the elements of the set, the distribution obtained will be unacceptable for the size of the problem at points (x, s_{00}) and (x, s_{01}) where processors demonstrate different relative speeds compared to the relative speeds at points (y, s_{10}) and (y, s_{11}). In some such cases, the partitioning of the set obtained could be the worst possible distribution where the number of elements per processor obtained could be inversely proportional to the speed of the processor. In such cases, it is better to use an even distribution of equal number of elements per processor than the distribution based on using such wrong points.

The algorithms we propose are based on the following observation: If a distribution of the elements of the set amongst the processors is obtained such that the number of elements is proportional to the speed of the processor, then the points, whose coordinates are number of elements and speed, lie on a straight line passing through the origin of the coordinate system and intersecting the graphs of the processors with speed versus the size of the problem in terms of the number of elements. This is shown by the geometric proportionality in Figure 6.

Our general approach to finding the optimal straight line can be summarized as follows:

1. We assume that the speed of each processor is represented by a non-increasing continuous function of the size of the problem.
2. At each step, we have two lines both passing through the origin. The sum of the number of elements at the intersection points of the first line with the graphs is less than the size of the problem, and the sum of the number of elements at the intersection points of second line with the graphs is greater than the size of the problem.
3. The region between these two lines is divided by a line passing through the origin into two smaller regions, the upper region and the lower region. If the sum of the number of elements at the intersection points of this line with the graphs is less than the size of the problem, the optimal line lies in the lower region. If this sum is greater than the size of the problem, the optimal line lies in the upper region.
4. In general, the exact optimal line intersects the graphs in points with non-integer sizes of the problem. This line is only used to obtain an approximate integer-valued solution. Therefore, the finding of any other straight line, which is

close enough to the exact optimal one to lead to the same approximate integer-valued solution, will be an equally satisfactory output of the searching procedure. A simple stopping criterion for this iterative procedure can be the absence of points of the graphs with integer sizes of the problem within the current region. In this case, any of the two lines limiting this region can be used as such an optimal line.

Note that it is the continuity and monotony of the graphs representing the speed of the processors that make each step of this procedure possible. The continuity guarantees that any straight line passing through the origin will have at least one intersection point with each of the graphs, and the monotony guarantees no more than one such an intersection point.

Let us estimate the cost of one step of this procedure. At each step we need to find the points of intersection of $\mathbf{p}$ graphs $\mathbf{y=s_1(x)}$, $\mathbf{y=s_2(x)}$, ..., $\mathbf{y=s_p(x)}$, representing the absolute speeds of the processors, and the straight line $\mathbf{y=c\times x}$ passing through the origin. In other words, at each step we need to solve $\mathbf{p}$ equations of the form $\mathbf{c\times x = s_1(x)}$, $\mathbf{c\times x = s_2(x)}$, ..., $\mathbf{c\times x = s_p(x)}$. As we need the same constant number of operations to solve each equation, the complexity of this part of one step will be O($\mathbf{p}$). According to our stopping criterion, a test for convergence can be reduced to testing $\mathbf{p}$ inequalities of the form $\mathbf{l_i - u_i} < 1$, where $\mathbf{l_i}$ and $\mathbf{u_i}$ are the size coordinates of the intersection points of the $\mathbf{i}$-th graph with the lower and upper lines limiting the region respectively ($\mathbf{i}$=1,2,…,$\mathbf{p}$). This testing is also of the complexity O($\mathbf{p}$). Therefore, the total complexity of one step including the convergence test will still be O($\mathbf{p}$).

The simplest particular algorithm based on this approach bisects the region between the lines by a line passing through the origin at a slope equal to half of the sum of the slopes of the two lines as shown in Figure 7.

The use of bisection is shown in Figure 8. The first two lines drawn during step 1 are `line1` and `line2`. Then `line3` is drawn whose slope is half of the slopes of the lines `line1` and `line2`. Since the sum of the number of elements at the intersection points of this line with the graphs is less than the size of the problem, bisect the lower half of the region by drawing `line4` whose slope is half of the slopes of the lines `line3` and `line2`. Since the sum of the number of elements at the intersection points of this line with the graphs is greater than the size of the problem, bisect the upper half of the region by drawing a line whose slope is half of the slopes of the lines `line3` and `line4`. This line turns out to be the optimally sloped line.

In most real-life situations, this algorithm will demonstrate a very good efficiency. Obviously, the slope of the optimal line is a decreasing function of the size of the problem, $\theta_{opt} = \theta_{opt}(\mathbf{n})$. If $\theta_{opt}(\mathbf{n}) \sim \mathbf{n}^{-\mathbf{k}}$, where $\mathbf{k}$=const , then the maximal number of steps to arrive at the optimal line will be ~ $\mathbf{k}\times\log_2\mathbf{n}$. Correspondingly, the complexity of the algorithm will be O($\mathbf{p}\times\log_2\mathbf{n}$).

At the same time, in some situations this algorithm may be quite expensive. For example, if $\theta_{opt}(\mathbf{n}) \sim e^{-\mathbf{n}}$, then the number of steps to arrive at the optimal line will be ~$\mathbf{n}$. Correspondingly, the complexity of the algorithm will be O($\mathbf{p}\times\mathbf{n}$).

We modify this algorithm to achieve reasonable performance in all cases, independent on how the slope of the optimal line depends on the size of the problem. To introduce the modified algorithm, let us re-formulate the problem of finding the optimal straight line as follows:

1. The space of solutions consists of all straight lines drawn through the origin and intersecting the graphs of the processors so that the size coordinate of at least one intersection point is integer.
2. We search for a straight line from this space closest to the optimal solution.

At each step of the basic bisection algorithm, it is the region between two lines that is reduced, not the space of solutions. Our modified algorithm tries to reduce the space of solutions rather than the region where the solution lies as illustrated in Figure 9 and Figure 10. At each step of the algorithm, we find a processor, whose graph $\mathbf{s(x)}$ is intersected by the maximal number of lines from the current region of the space of solutions limited by the lower and upper lines. Then we detect a line, which divides the region into two smaller regions such that each region contains the same number of lines from the space of solutions intersecting this graph. To do it, we just need to draw a line passing through the origin and the point $((\mathbf{l}-\mathbf{u})/2,\ \mathbf{s}((\mathbf{l}-\mathbf{u})/2))$, where $\mathbf{l}$ and $\mathbf{u}$ are the size coordinates of the intersection points of this graph with the lower and upper lines limiting the current region of the space of solutions.

This algorithm guarantees that after $\mathbf{p}$ such bisections the number of solutions in the region is reduced at least by 50%. This means we need no more than $\mathbf{p}\times\log_2\mathbf{n}$ steps to arrive at the sought line. Correspondingly, the complexity of this algorithm will be O($\mathbf{p}^2\times\log_2\mathbf{n}$).

One can see that the modified bisection algorithm is not sensitive to the shape of the graphs of the processors, always demonstrating the same efficiency. The basic bisection algorithm is sensitive to their shape. It demonstrates higher efficiency than the modified one in better cases but much lower efficiency in worse cases.

An ideal bisection algorithm would be of the complexity O(**p**×log$_2$**n**) reducing at each step the space of solutions by 50% and being insensitive to the shape of the graphs of the processors. The design of such an algorithm is still a challenge. At the same time, we have designed a number of algorithms whose worst-case complexity is O(**p**^{2}×log$_2$**n**) but the best-case complexity is O(**p**×log$_2$**n**). We do not present the algorithms in the paper because of the restrictions on the length.

4. Experimental results

A small heterogeneous local network of 4 different Solaris and Linux workstations shown in Table 2 is used in the experiments. The network is based on 100 Mbit Ethernet with a switch enabling parallel communications between the computers.

There are two sets of experiments used to demonstrate the efficiency of the model. The first set is based on the parallel algorithm of matrix-matrix multiplication of two dense matrices using horizontal striped partitioning shown in Figures 14(a) and 14(b) and the second set is based on the parallel algorithm of matrix factorization of a dense matrix using horizontal striped partitioning shown in Figure 16. The matrices are horizontally sliced such that the number of slices is proportional to the speed of the processor.

The speed function for a processor is built using a set of 21 experimentally obtained points. We use piece-wise linear function approximation illustrated in Figure 12 to build the speed function. Such approximation of the speed function is compliant with the requirements of the model, which are that the speeds be non-increasing continuous functions of problem size. We detect the two lines, between which the solution lies, as shown in Figure 13. The absolute speed of the processor in number of floating point operations per second is calculated using the formula

$$Abs.\,speed = \frac{volume\,of\ computations}{time\,of\ execution} = \frac{MF \times n \times n \times n}{time\,of\ execution}$$

where **n** is the size of the matrix. **MF** is 2 for Matrix Multiplication and $\frac{1}{3}$ for Cholesky Factorization.

In each set of experiments, the speedups of the parallel application obtained by using the new flexible model are shown over the parallel applications using the standard model, which uses the speeds of the processors obtained by running the corresponding serial application. The size of the problem used to obtain the speed of each processor is the same on each processor. The experimental results show that the parallel applications using the advanced model demonstrate good speedup over parallel applications using the standard model. At a first glance, it may look strange that there is no problem size where the standard model demonstrates the same speed as the advanced model. Actually in heterogeneous environment, the distribution given by the standard model cannot in principle be better than the distribution given by the advanced model. This is because the speeds used in the standard model are obtained based on the fact that all the processors get the same number of elements and hence solve problems of the same size as in a homogeneous environment. Whatever problem size is used, it will give wrong estimation of distribution for at least one processor.

Figure 15 shows the speedup of the matrix-matrix multiplication executed on this network using the advanced model over the matrix-matrix multiplication using the standard model that determines the speed of the processor based on the multiplication of two dense 100×100 matrices and two dense 3000×3000 matrices. As can be seen from the figure, the advanced model performs better than the standard model for a network of heterogeneous computers that demonstrates relative speeds that are non-constant functions of the size of the problem.

Figure 17 shows the speedup of the matrix factorization executed on this network using the advanced model over the matrix factorization using the standard model that determines the speed of the processor based on the matrix factorization of a dense 500×500 matrix and a dense 7000×7000 matrix. As can be seen from the figure, the advanced model performs better than the standard model for a network of heterogeneous computers that demonstrates relative speeds that are non-constant functions of the size of the problem.

5. Conclusions

In this paper, we address the problem of optimal data partitioning in heterogeneous environments when relative speeds of processors cannot be accurately approximated by constant functions of the problem size. We have proposed an advanced performance model of a network of heterogeneous computers and designed efficient algorithms of data partitioning with this model. We do not take account of communication cost. This is out of scope of this paper. The problems of efficient building and maintaining of our model are also out of scope of the paper.

References

[1] P. Crandall and M.Quinn, "Block Data Decomposition for Data-Parallel Programming on a Heterogeneous Workstation Network", Proceedings of the Second International Symposium on High Performance Distributed Computing, pp.42-49, 1993.

[2] P. Crandall and M. Quinn, "Problem Decomposition for Non-Uniformity and Processor Heterogeneity", Journal of the Brazilian Computer Society , vol. 2, no. 1, pp. 13-23. July 1995.

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

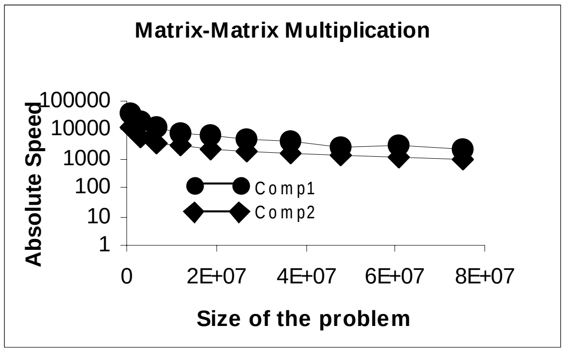
[4] O.Beaumont, V.Boudet, F.Rastello, and Y.Robert, "Matrix Multiplication on Heterogeneous Platforms", IEEE Transactions on Parallel and Distributed Systems, 12(10), pp.1033-1051, 2001.

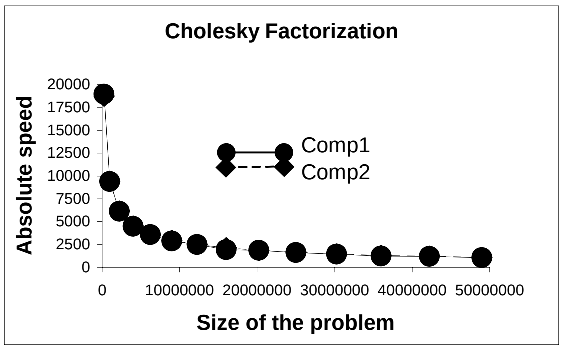
[5] L.Blackford, J.Choi, A.Cleary, J.Demmel, I.Dhillon, J.Dongarra, S.Hammarling, G.Henry, A.Petitet, K.Stanley, D.Walker, and R.Whaley, "ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers – Design Issues and Performance", Proceedings of Supercomputing'96, 1996.

BIOGRAPHIES

Alexey Lastovetsky received the PhD degree from the Moscow Aviation Institute in 1986, and the Doctor of Science degree from the Russian Academy of Sciences in 1997. He is currently a lecturer in the Computer Science Department at University College Dublin, National University of Ireland. His main research interests are parallel and distributed programming languages and systems for heterogeneous environments. He is a member of IEEE Computer Society.

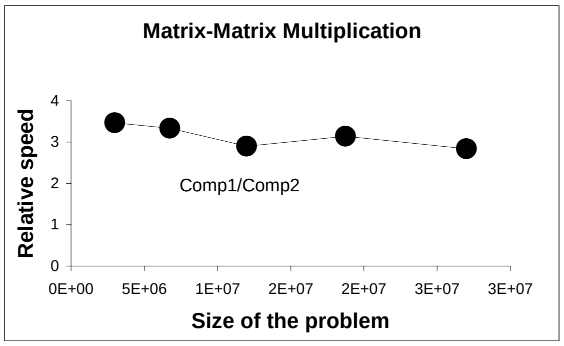
Ravi Reddy is currently a PhD student in the Computer Science Department at University College Dublin, National University of Ireland. His main research interests are design of algorithms and tools for parallel and distributed computing systems.





(a) (b)

Figure 1. (a) Absolute speeds of the computers shown in Table 1 against the size of the problem in Matrix-Matrix multiplication. (b) Absolute speeds of the computers shown in Table 1 against the size of the problem in Cholesky Factorization.



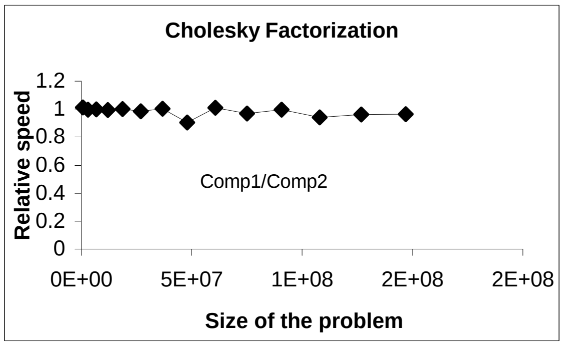
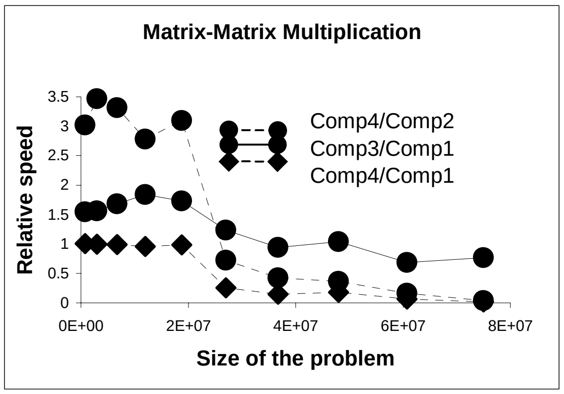
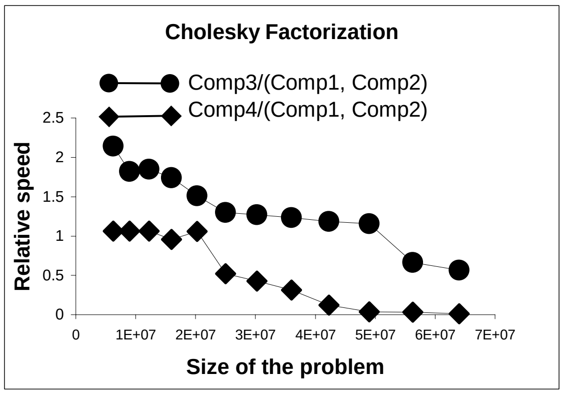


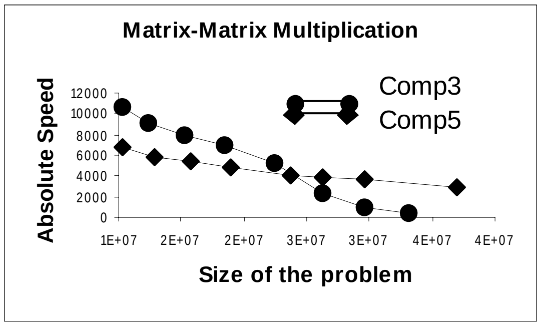
Figure 2. (a) Relative speeds of the computers shown in Table 1 against the size of the problem in Matrix-Matrix multiplication. (b) Relative speeds of the computers shown in Table 1 against the size of the problem in Cholesky Factorization.

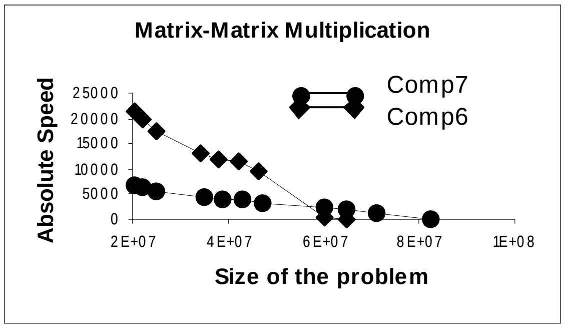




(a) (b)

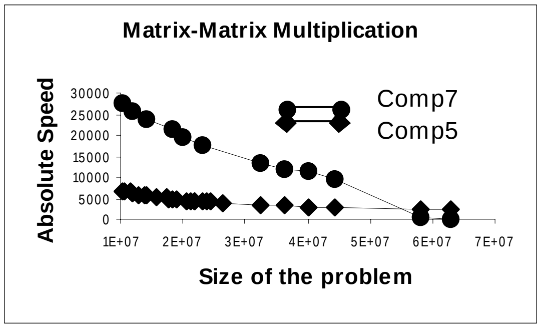
Figure 3. (a) Relative speeds of the computers shown in Table 2 against the size of the problem in Matrix-Matrix multiplication . (b) Relative speeds of the computers shown in Table 2 against the size of the problem in Cholesky Factorization.

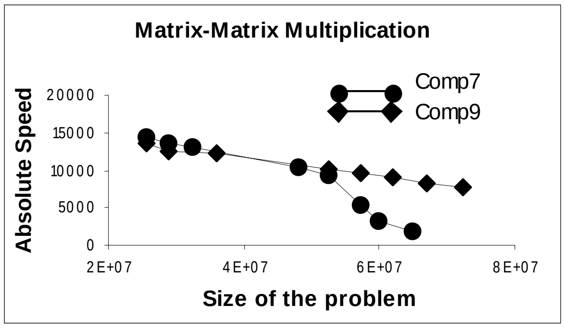




(a) (b)

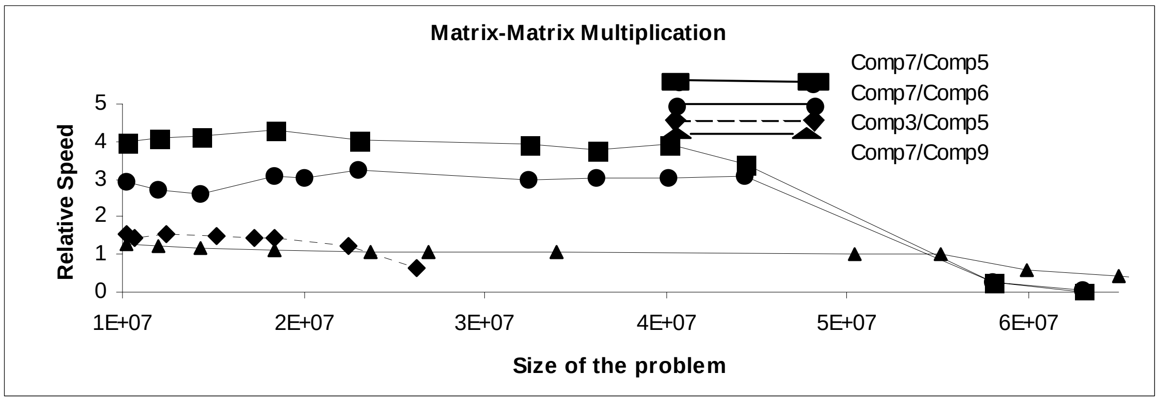
Figure 4. Absolute speeds of the computers shown in Table 3 against the size of the problem in serial Matrix-Matrix multiplication derived from two-dimensional block-cyclic parallel Matrix-Matrix multiplication used in ScaLAPACK. (a) Comp3/Comp5 (b) Comp7/Comp6.





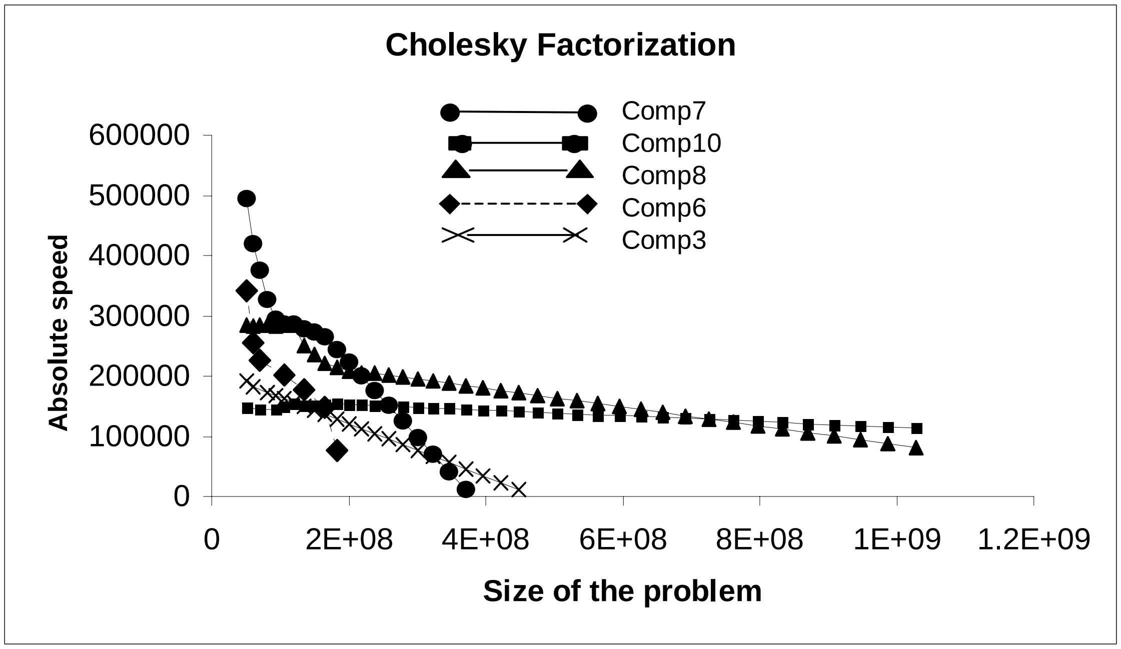
(c) (d)

Figure 4. Absolute speeds of the computers shown in Table 3 against the size of the problem in serial Matrix-Matrix multiplication derived from two-dimensional block-cyclic parallel Matrix-Matrix multiplication used in ScaLAPACK. (c) Comp7/Comp5 (d) Comp7/Comp9



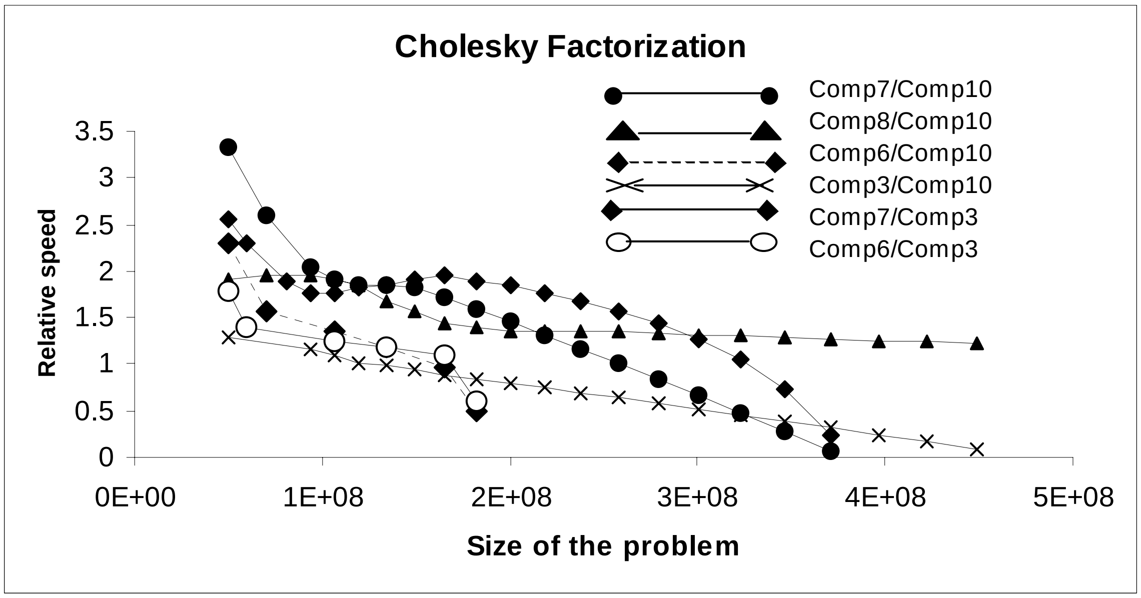
(e)

Figure 4. (e) Relative speeds of the computers shown in Table 3 against the size of the problem in serial Matrix-Matrix multiplication derived from two-dimensional block-cyclic parallel Matrix-Matrix multiplication used in ScaLAPACK.



(f)

Figure 4. (f) Absolute speeds of the computers shown in Table 3 against the size of the problem in serial Cholesky Factorization derived from two-dimensional block-cyclic parallel Matrix-Matrix multiplication used in ScaLAPACK.



(g)

Figure 4. (g) Relative speeds of the computers shown in Table 3 against the size of the problem in serial Cholesky Factorization derived from two-dimensional block-cyclic parallel Matrix-Matrix multiplication used in ScaLAPACK.

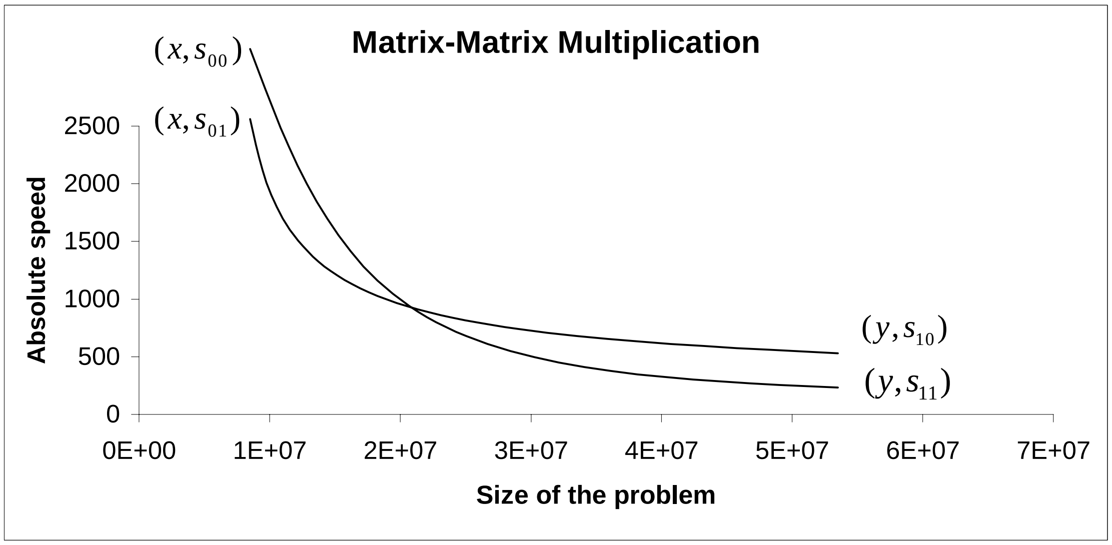


Figure 5. A small network of two processors whose speeds are shown against the size of the problem.

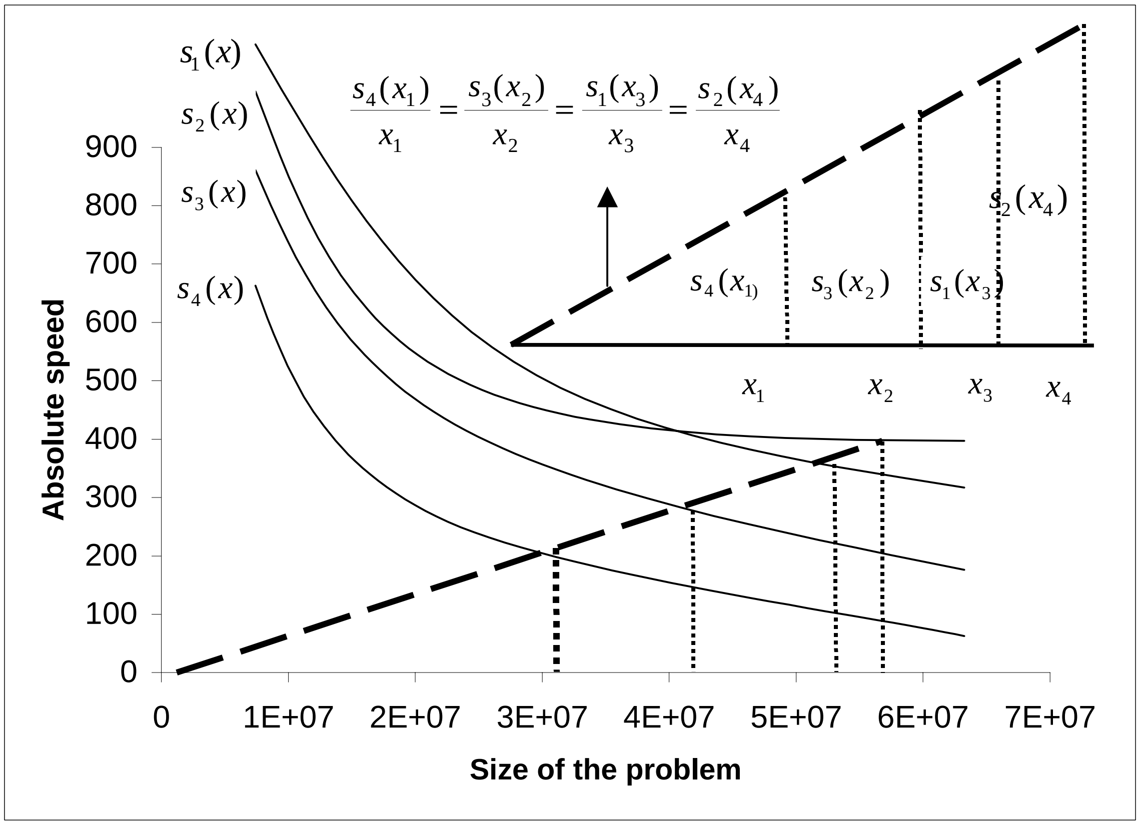


Figure 6. Optimal solution showing the geometric proportionality of the number of elements to the speed of the processor.

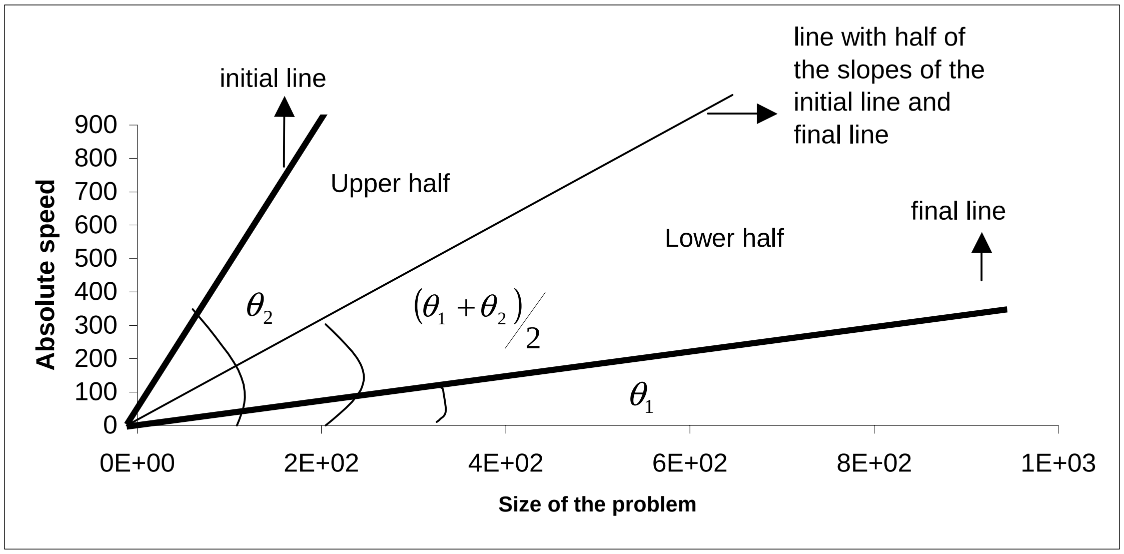


Figure 7. Determination of the slope of the line equal to half of the slopes of the initial and final lines.

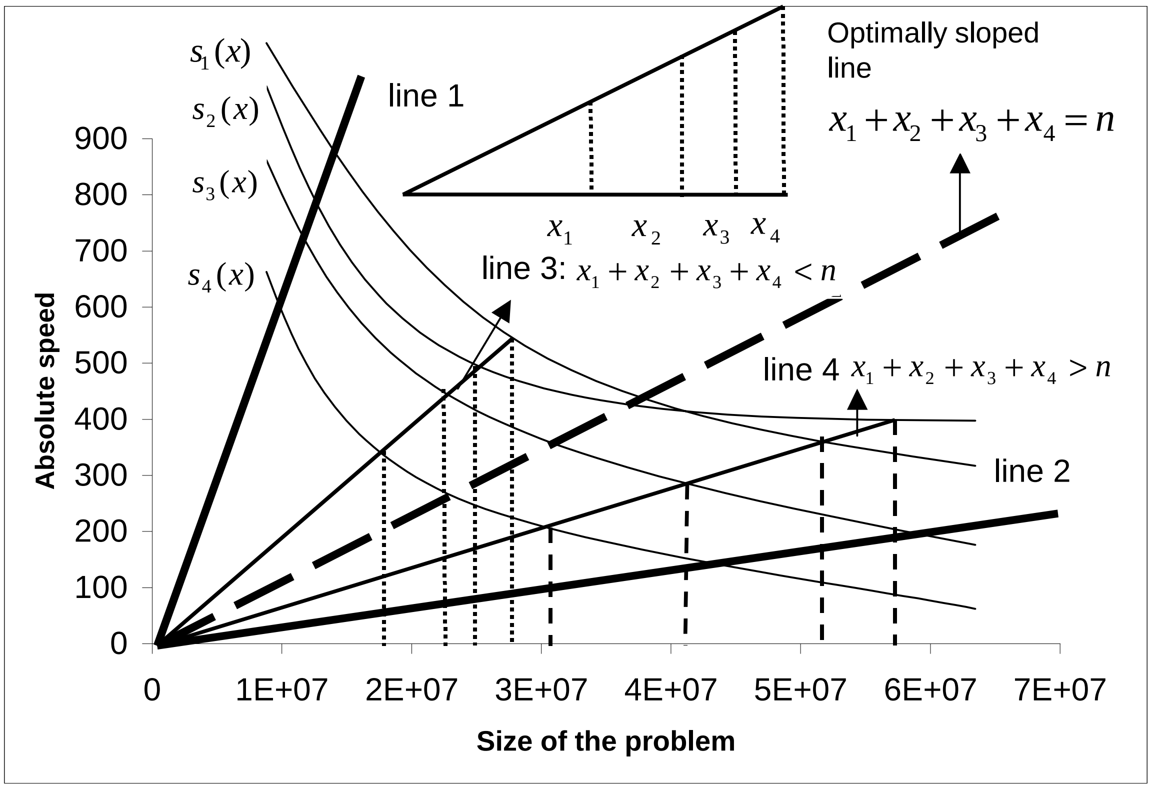


Figure 8. Use of bisection of the range to narrow down to the optimal solution satisfying the criterion that the number of elements should be proportional to the speed of the processor. n is the size of the problem.

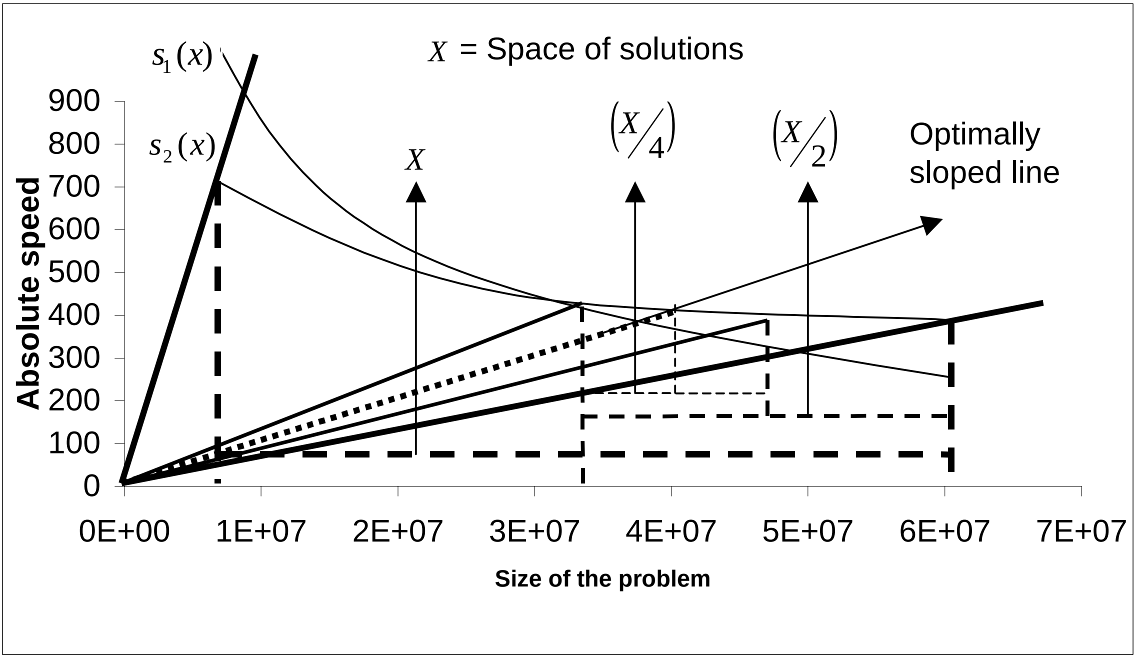


Figure 9. Bisection of the space of solutions in the modified algorithm.

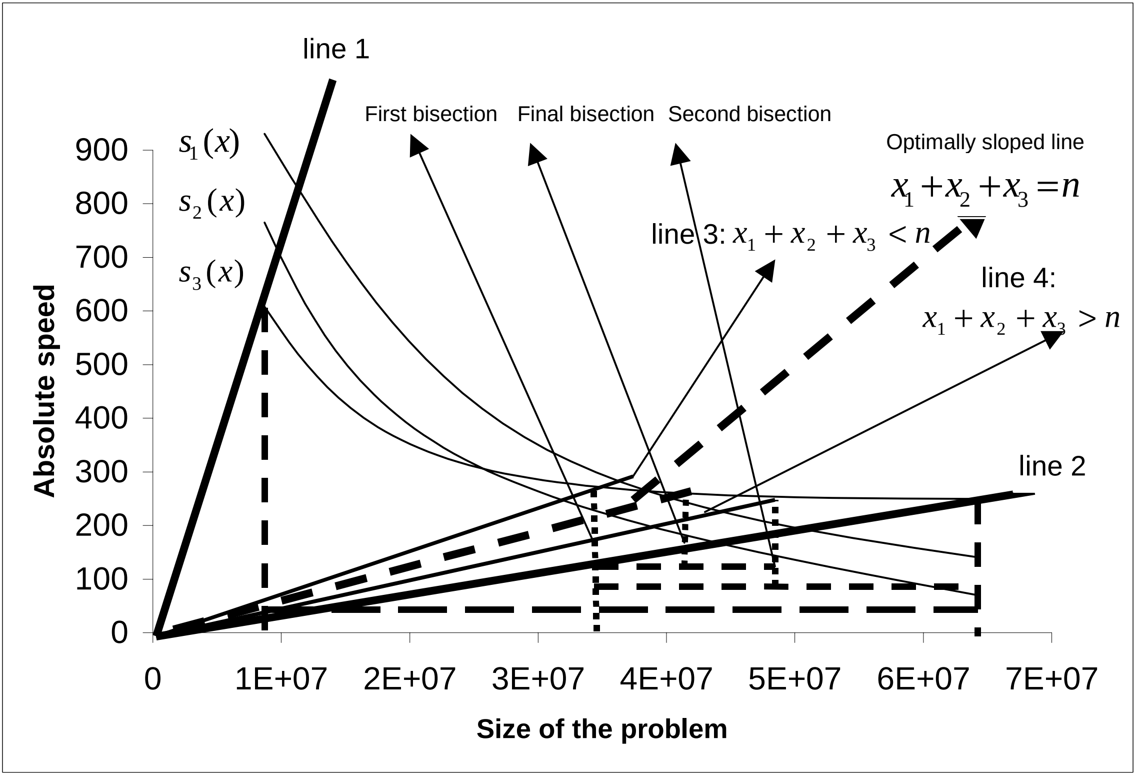


Figure 10. Modification of the algorithm shown in figure 7 where the bisection results in efficient solution. n is the size of the problem.

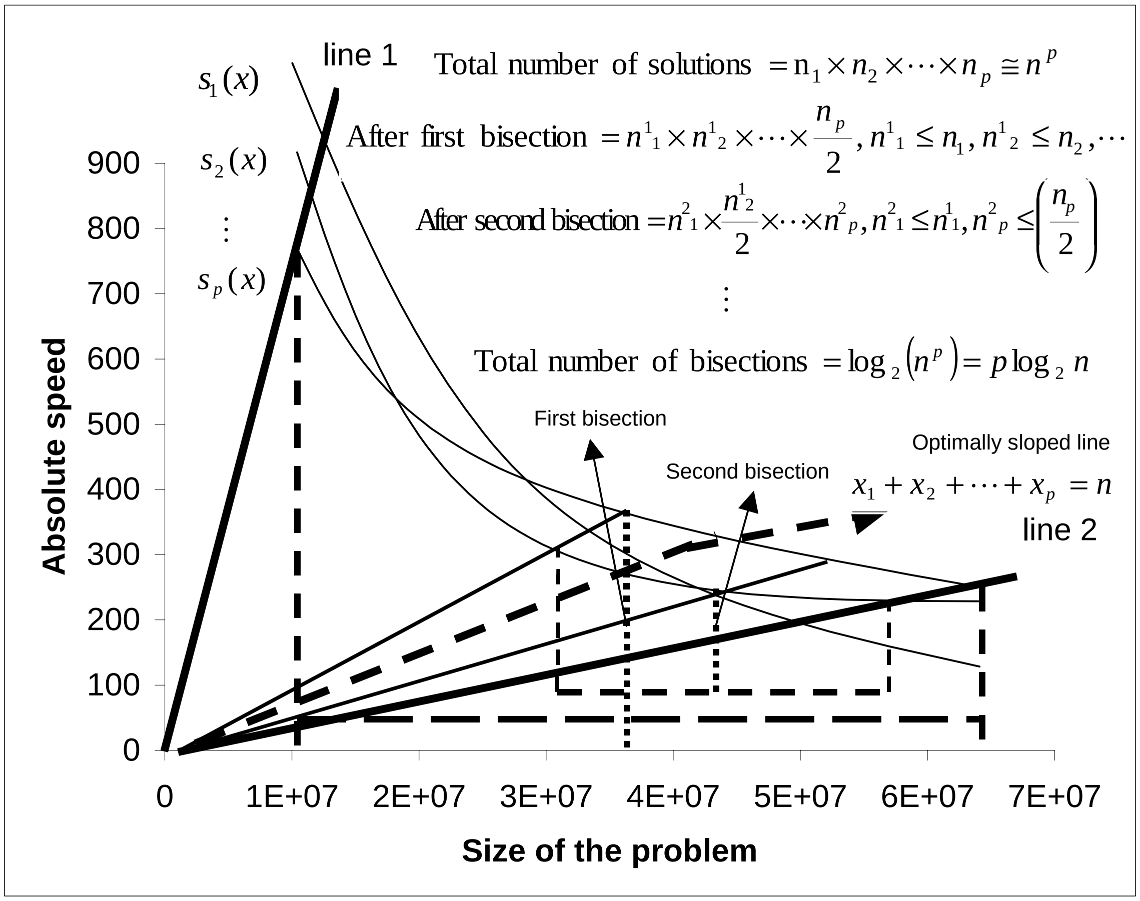


Figure 11. Schematic proof of the complexity of the modified algorithm. The total number of bisections are *p*log$_2$*n*. At each step of bisection, p intersection points are obtained giving a total complexity of O(*p*^{2}log$_2$*n*).

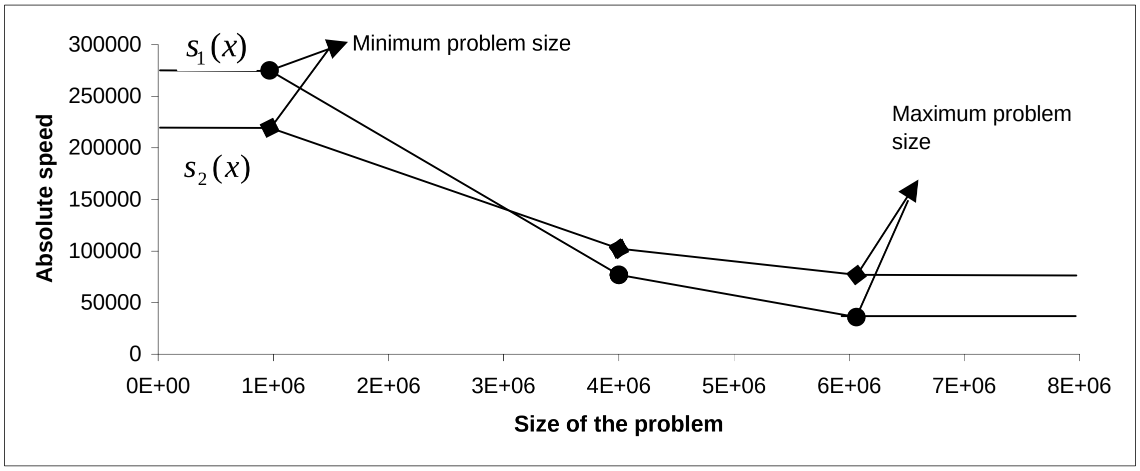
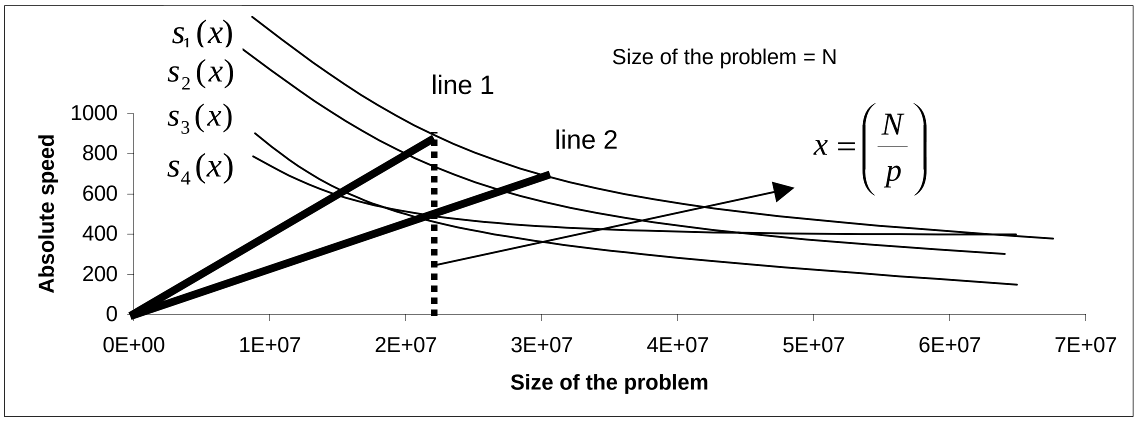


Figure 12. Using piece-wise linear approximation to build speed functions for 2 processors. The speed functions are built from 3 experimentally obtained points.



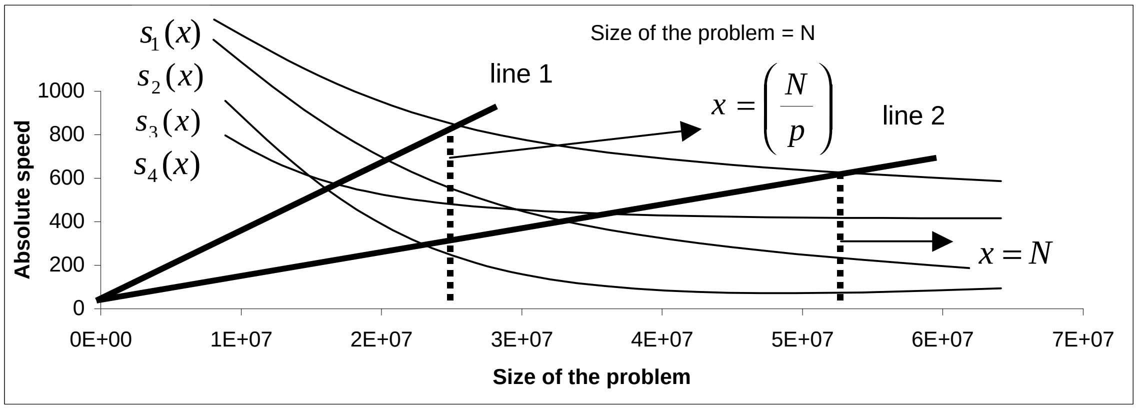


Figure 13. Two cases where line1 and line2, between which the solution lies, are detected.

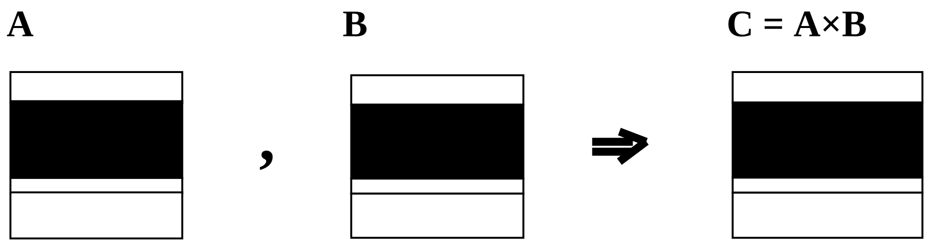


Figure 14(a). Matrix operation C=A×B with matrices A, B, and C unevenly partitioned in one dimension. The slices mapped onto a single processor are shaded in black.

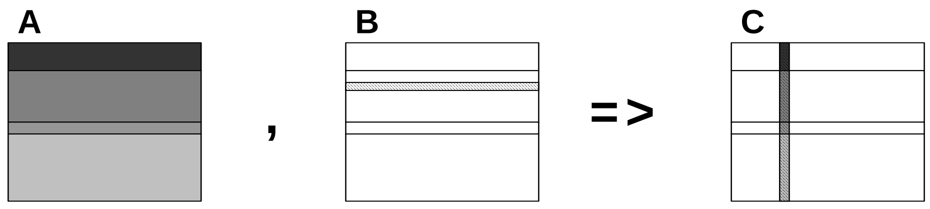


Figure 14(b). One step of parallel multiplication of matrices A and B. The pivot row of blocks of matrix B (shown slashed) is first broadcast to all processors. The each processor computes, in parallel with the others, its part of the corresponding column of blocks of the resulting matrix C.

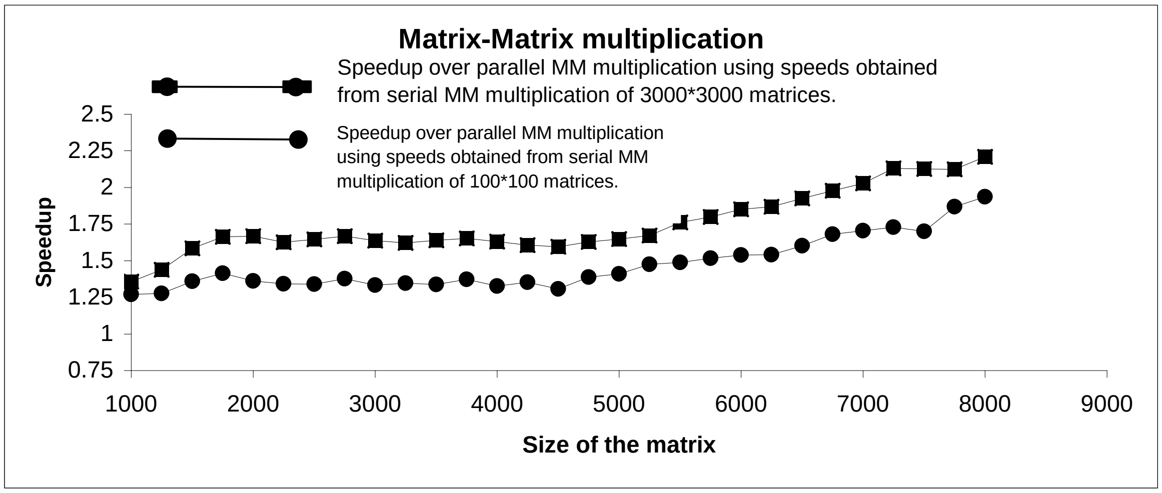


Figure 15. Results obtained using the network of heterogeneous computers shown in Table 2. Comparison of speedups of MM algorithm using horizontal striped partitioning of matrices.

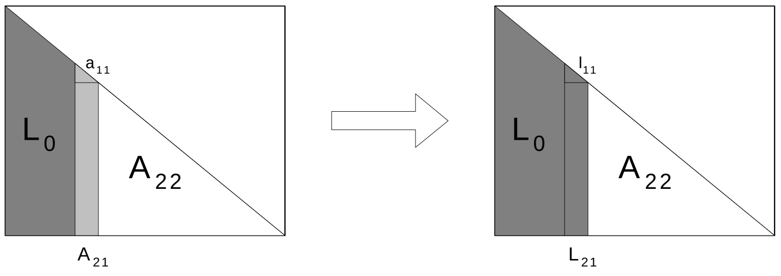


Figure 16. One step of the Cholesky factorization algorithm: the column panel $L = \begin{pmatrix} l_{11} \\ L_{21} \end{pmatrix}$ **is computed and the trailing submatrix** A_{22} **is updated.**

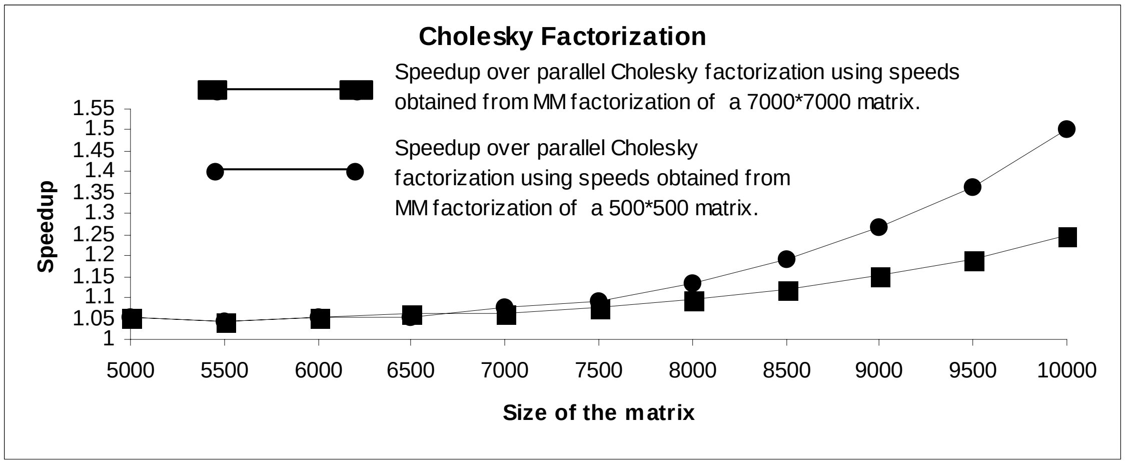


Figure 17. Results obtained using the network of heterogeneous computers shown in Table 2. Comparison of speedups of Cholesky Factorization using horizontal striped partitioning of matrices.