Data Partitioning with a Realistic Performance Model of Networks of Heterogeneous Computers with Task Size Limits

Alexey Lastovetsky, *Member, IEEE* and Ravi Reddy

***Abstract*—The paper presents a performance model that can be used to optimally schedule arbitrary tasks on a network of heterogeneous computers when there is an upper bound on the size of the task that can be solved by each computer. We formulate a problem of partitioning of an n-element set over p heterogeneous processors using this advanced performance model and give its efficient solution of the complexity $O(p^3 \times \log_2 n)$.**

***Index Terms*—Heterogeneous (hybrid) systems, Scheduling and task partitioning, Load balancing and task assignment**

I. Introduction

In this paper, we deal with the problem of optimal distribution or scheduling of arbitrary tasks on a network of heterogeneous computers when there is an upper bound on the size of the task that can be solved by each computer. These tasks are processed in parallel by the computers of the heterogeneous network. Examples of applications include search for patterns in text, audio, graphical files, processing of very large linear data files as in signal processing, image processing, and experimental data processing, linear algebra algorithms, simulation, combinatorial optimization algorithms, and many others.

Our efforts are mainly concentrated towards programming parallel and distributed applications on heterogeneous networks of computers, which are ubiquitous in university departments and companies, and programming distributed applications on computing grids, which are composed of nodes that may well be scattered around the world. Some of the issues with programming applications on such networks of heterogeneous computers have been explained in [1]. These are mainly:

- On such networks, all available resources, namely, slower machines in addition to faster machines must be used to execute the distributed application efficiently taking into account the memory structure of each processor. These processors may have significantly different sizes at each level of their memory hierarchies. It is quite likely that the subprograms assigned to some processors may not fit into their main memory leading to paging.
- Unlike dedicated distributed computer systems, such networks are not strongly centralized and consist of relatively autonomous and mainly general-purpose computers, where each one may be used and administered independently by its users. On the other hand, each computer is an integrated part of the network, which means that it periodically does some computations and communications just as such an integrated node of the network. One of the implications with the multi-user decentralized loosely-integrated nature of these networks is the unstable performance characteristics of processors during the execution of a parallel program as the computers may be involved in other computations and communications. As a result, the dependence of the speed of the processor on the problem size is not as sharp and distinctive as observed on dedicated distributed multiprocessor computer systems. In this case, the speed of the processors is more realistically represented by a continuous and relatively smooth function of the problem size.

The performance model discussed in [1] can be used to efficiently schedule arbitrary tasks on such network of heterogeneous computers when one or more arbitrary tasks do not fit into the main memory of the processors. This model particularly addresses 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. In this model, each processor is represented by its absolute speed as a continuous function of problem size.

However this model fails to address the problem of efficiently scheduling arbitrary tasks on a network of heterogeneous computers when there is an upper bound on the size of the task that can be solved by each computer. Hence we extend this model of networks of heterogeneous computers by introducing an additional parameter, namely, the upper bound on the size of the task that can be solved by each computer. The upper bound could signify one of the following cases:

- Allocation of a task whose size is beyond this bound could result in processor failure.
- Allocation of the task whose size is beyond this bound could result in unacceptable execution time to accomplish the task.

Consider a small network of three processors, whose speeds as functions of problem size during the execution of the matrix-matrix multiplication are shown in Figure 1. Consider the case of optimal distribution for problem sizes that lie between line 1 and line 2. Line 1 corresponds to the problem size b_1 that is the upper bound for processor represented by $s_2(x)$. Line 2 corresponds to the problem size b_2, which is the upper bound for the processors represented by speed functions $s_1(x)$ and $s_3(x)$. For these problem sizes, any distribution obtained by this model will most likely either crash the processor whose speed is represented by the speed function $s_2(x)$ or result in unacceptable execution time to execute the subtask assigned to this processor.

The advanced performance model retains the restrictions imposed by performance model [1] on the shape of the graph representing the speed function. However each processor is represented by its absolute speed as a continuous function of problem size only up till its upper bound on the problem size and beyond that, the absolute speed of the processor is assumed to be almost equal to zero.

We formulate a problem of partitioning of an **n**-element set over **p** heterogeneous processors using this advanced performance model and give its efficient solution of the complexity $O(\mathbf{p}^3 \times \log_2 \mathbf{n})$. This problem is a simple variant of the most advanced problem of partitioning a set with weighted elements [2]. We use the simple variant to explain how complex is the problem of scheduling arbitrary tasks amongst processors when (a) The processors have significantly different memory structure, (b) One or more tasks do not fit into the main memory of the processors, and (c) There is an upper bound on the size of task that can be solved by each processor. We also use this variant to explain in simple terms how the advanced model can be used to achieve better data partitioning on networks of heterogeneous computers before moving on to solve the most advanced problem.

To demonstrate the efficiency of the advanced performance model, we perform experiments using naïve parallel algorithms for linear algebra kernel, namely, matrix multiplication and Cholesky Factorization using horizontal striped partitioning of matrices on a local network of heterogeneous computers. Our main aim is not to show how matrices can be efficiently multiplied or efficiently factorized but to explain in simple terms how advanced model can be used to optimally schedule arbitrary tasks on networks of heterogeneous computers when one or more tasks do not fit into the main memory of the processors and when there is an upper bound on the size of task that can be solved by each processor. We also view these algorithms as good representatives of a large class of data parallel computational problems and a good testing platform before experimenting more challenging computational problems.

Some of the applications using distributed algorithms include Monte Carlo Simulations of Cellular Microphysiology [3], Grid Computing in high energy physics [4] that involves collecting petabyte-scale datasets and deployment of enormous computational, storage and networking resources to process, distribute and analyze these datasets, Parallel Simulated Annealing using Genetic Cross over to predict protein tertiary structures [5].

II. ALGORITHMS FOR PARTITIONING SETS

In this paper, we solve the simple variant of the most advanced problem of partitioning a set, which can be formulated as:

- Given: (1) A set of **n** elements, and (2) A well-ordered set of **p** processors whose speeds are functions of the size of the problem, $\mathbf{s_i=f_i(x)}$, with an upper bound $\mathbf{b_i}$ on the number of elements stored by each processor (i=0,…,p-1),
- Partition the set into **p** disjoint partitions such that the maximum of the execution times of the processors is minimized.

$$\max_{i=0}^{p-1}\left(\frac{x_i}{s_i}\right)$$

That is we solve the following min-max problem:

$$\min\left\{\max_{i=0}^{p-1}\left(\frac{x_i}{s_i}\right)\right\}$$

where x_i is the number of elements in each 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**). When there is an upper bound $\mathbf{b_i}$ on the number of elements stored by each processor (i=0,…,p-1), the algorithm used to solve the partitioning problem is of complexity $O(p^2)$. This algorithm can be summarized as follows:

1. Partition the set such that the number of elements in each partition is proportional to the speed of the processor and assuming no upper bound exists on the number of elements that can be stored by each processor. If the number of elements assigned to each processor is less than or equal to the upper bound on the number of elements that can be stored by each processor, we have the optimal distribution.
2. For each processor **i** (i=0,…,p-1), we check if the number of elements assigned to it is greater than the upper bound on the number of elements that it can store. For all the processors whose upper bounds are exceeded, we assign them the number of elements equal to their upper bounds. Now we solve the partitioning problem of a set with remaining elements over the remaining processors. We recursively apply this procedure until all the elements have been assigned.

The proof of optimality of the solution provided by this algorithm is given in [6].

When the speed of the processor is represented by a function of the size of the problem, **s=f(x)**, and when there is no upper bound on the number of elements stored by each processor, efficient algorithms used to perform the partitioning have been proposed of complexity $O(p^2 \times \log_2 n)$ [1].

When the speed of the processor is represented by a function of the size of the problem, **s=f(x)**, and when there is no upper bound on the number of elements stored by each processor, the problem of partitioning a set is non-trivial. Consider a small network of three processors, whose speeds as functions of problem size during the execution of the matrix-matrix multiplication are shown in Figure 1. Consider the case of optimal distribution for problem sizes that lie between line 1 and line 2. Line 1 corresponds to the problem size b_1 that is the upper bound for processor represented by $s_2(x)$. Line 2 corresponds to the problem size b_2, which is the upper bound for the processors represented by speed functions $s_1(x)$ and $s_3(x)$. For these problem sizes, the standard models that use single numbers to represent the speeds of the processors will fail to deliver optimal distribution. This is the case with any model that does not take into account the upper bound b_1 on the problem size that the processor represented by speed function $s_2(x)$ can solve. Allocation of problem size greater than b_1 would either result in failure of processor $s_2(x)$ or adverse execution performance of the distributed application. We show few non-optimal solutions in the figure. We prove subsequently why these solutions are non-optimal.

The algorithm we propose to solve this advanced partitioning problem is graphically illustrated in Figure 2 and has the following main points:

1. Partition the set such that the number of elements in each partition is proportional to the speed of the processor and assuming no upper bound exists on the number of elements that can be stored by the processor. The partitioning algorithm used to perform this task is discussed in [1]. If the number of elements in each partition assigned to each processor is less than the upper bound on the number of elements that can be stored by the processor, we have an optimal distribution.
2. For each processor **i** (i=0,…,p-1), we check if the number of elements assigned to it is greater than the upper bound on the number of elements that it can store. For all the processors whose upper bounds are exceeded, we assign them the number of elements equal to their upper bounds. Now we solve the partitioning problem of a set with remaining elements over the remaining processors. We recursively apply this procedure until all the elements have been assigned.

Proof. We prove the optimality of the solution provided by this algorithm using mathematical induction. We use the maximum time to solve the task assigned to each processor as the performance metric.

Before we proceed to prove the optimality of the algorithm, we make the following assumptions:

- We assume that the speed of each processor is represented by a continuous function of the size of the problem up till its upper bound on the problem size.
- The shape of the graph should be such that there is only one intersection point of the graph with any straight line passing through the origin. These assumptions on the shapes of the graph are representative of the most general shape of graphs observed for applications experimentally. That is the speeds of the processors must either be increasing or decreasing functions of problem size for the problem sizes for which the solutions are sought.
- We can safely assume that for each processor, for all $x \geq y$, where x and y are problem sizes, the execution times t_x and t_y are related by $t_x \geq t_y$.

The cases for **p**=1 and **p**=2 are trivial. For **p**=3, let us assume the upper bounds of the processors 1, 2, and 3 on the number of elements that they can store are b_1, b_2, and b_3 respectively. Suppose the optimal distribution assuming there are no upper bounds on the number of elements is (x_1, x_2, x_3) such that $x_1+x_2+x_3=n$ where n is the size of the problem. Consider the case where $x_1 > b_1$ and $x_2 > b_2$. Let us assign the number of elements equal to b_1 for processor 1. The remaining distribution has to satisfy the equality $x_2^{'} + x_3^{'} = n - b_1$ where $x_2^{'}$ and $x_3^{'}$ are to be chosen such that the speed of the processor is proportional to the number of elements assigned to it. If the speeds of the processors 2 and 3 are non-increasing functions of problem size, it can be proved that $x_2^{'} > x_2$ and $x_3^{'} > x_3$. This gives us the inequality $x_2^{'} > x_2 > b_2$. Therefore we have to necessarily assign b_2 number of elements to processor 2. If the speeds of the processors 2 and 3 are non-decreasing functions of problem size, there are three possibilities, $(x_2^{'} > x_2, x_3^{'} > x_3)$, $(x_2^{'} < x_2, x_3^{'} > x_3)$ and $(x_2^{'} > x_2, x_3^{'} < x_3)$. The first and the third possibility give us the inequality $x_2^{'} > x_2 > b_2$. For the second possibility, any allocation $x_2^{''}$ such that $x_2^{''} < b_2$ would result in an allocation of $x_3^{''}$ number of elements to processor 3 such that $x_3^{''} > x_3^{'}$ thus resulting in a larger execution time. Therefore we have to necessarily assign b_2 number of elements to processor 2. If the speed of the processor 2 is a non-decreasing function of problem size and speed of processor 3 is a non-increasing function of problem size, there are two possibilities, $(x_2^{'} < x_2, x_3^{'} > x_3)$ and $(x_2^{'} > x_2, x_3^{'} < x_3)$. In the first possibility, any allocation $x_2^{''}$ such that $x_2^{''} < b_2$ would result in an allocation of $x_3^{''}$ number of elements to processor 3 such that $x_3^{''} > x_3^{'}$ thus resulting in a larger execution time. The second possibility gives us the inequality $x_2^{'} > x_2 > b_2$. Therefore we have to necessarily assign b_2 number of elements to processor 2. Consider the case of optimal distribution where $x_1 > b_1$ is true. For processor 1, we assign the number of elements equal to b_1. The remaining elements are allocated such that $x_2^{'} + x_3^{'} = n - b_1$ where $x_2^{'}$ and $x_3^{'}$ are to be chosen such that the speed of the processor is proportional to the number of elements assigned to it. Any other allocation $x_1^{''}$ such that $x_1^{''} < b_1$ would result in an allocation where one of the inequalities ($x_2^{''} > x_2^{'}$), ($x_3^{''} > x_3^{'}$) is satisfied thus resulting in a larger execution time. It can be proved similarly for the case when $x_2 > b_2$.

Assuming this to be true for **p**=**k** processors, we have to prove the optimality for **p**=**k**+1 processors. For a given problem size **n**, let us assume the distribution given by our algorithm to be $x_0, b_1, b_2, \cdots, b_m, x_{m+1}, \cdots, x_k$ such that $x_0 + b_1 + \cdots + x_k = n$, where without loss of generality processors 1,…,**m** are allocated their upper bounds. It can be inferred that the execution times for the rest of the processors 0,**m**+1,…,**k** satisfy the equality $t_0 = t_{m+1} = \cdots = t_k$. It can also be inferred that $(t_0, t_{m+1}, \cdots, t_k) \geq t_i$ for all i=1,…,**m**. The execution time for the problem size is equal to $t_{mp} = \max_{i=0}^{k}(t_i) = (t_0, t_{m+1}, \cdots, t_k)$. Consider an alternative solution with the distribution $x_0^{'}, x_1^{'}, \cdots, x_k^{'}$ where $x_0^{'} + x_1^{'} + \cdots + x_k^{'} = n$ and $x_1^{'} \leq b_1, \cdots, x_m^{'} \leq b_m$. It can be easily seen that for atleast one processor i (i=0,**m**+1,…,**k**), $x_i^{'} \geq x_i$, thus giving an execution time $t_i^{'}$, which is greater than the execution time given by our algorithm t_{mp}.

Complexity. There are **p** major steps in the algorithm. At each such major step **i**, we solve the problem of partitioning of a set amongst **p-i** processors such that the number of elements in each partition is proportional to the speed of the processor and assuming no upper bound exists on the number of elements that can be stored by the processor. The complexity of this step is O($\mathbf{p}^2 \times \log_2 \mathbf{n}$) [1]. Since there are **p** such steps, the overall worst-case complexity is O($\mathbf{p}^3 \times \log_2 \mathbf{n}$). Mathematically, the worst-case complexity is the summation of **p** terms:

$$\begin{aligned} C &= p^2 \times \log_2 n + (p-1)^2 \times \log_2(n-b_0) + \\ &\quad (p-2)^2 \times \log_2(n-b_0-b_1) + \cdots + 1 \\ &\cong p^2 \times \left(\log_2 n + \log_2(n-b_0) + \log_2(n-b_0-b_1) + \cdots\right) \\ &\cong p^2 \times \left(\log_2(n \times (n-b_0) \times (n-b_0-b_1) \times \cdots)\right) \\ &\cong p^2 \times (\log_2 n^p) \\ &\cong p^3 \times \log_2 n \end{aligned}$$

III. EXPERIMENTAL RESULTS

A small heterogeneous local network of 11 different Solaris and Linux workstations shown in Table I 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 [7, p.199] and the second set is based on the parallel algorithm of matrix factorization of a dense matrix using horizontal striped partitioning [7, p.293]. The matrices are horizontally sliced such that the total number of rows mapped to a processor is proportional to the speed of the processor.

The speed function for a processor is built using a set of few experimentally obtained points. The more the number of points used in building the speed functions, the more accurate the speed functions are. However it is prohibitively expensive to use large number of points to build the speed functions of the processors. Hence for each processor, an optimal set of few points needs to be chosen to build an efficient speed function. Such a speed function built gives the speed of the processor for any problem size with certain deviation from the ideal speed function and speed functions built with sets with more number of points. This deviation must be within acceptable limits, ideally not exceeding the inherent deviation of the performance of computers typically observed in the network. In our experiments, we set the acceptable deviation to be $\pm 5\%$. This implies that the speed function should give the speed of the processor for a problem size within $\pm 5\%$ accuracy from the speed given by an ideal speed function or the speed functions built with sets with more number of points. Figure 3 show speed functions for matrix multiplication obtained using three sets of 6, 7, and 8 points and speed functions for Cholesky Factorization obtained using three sets of 5, 7, and 8 points for the computer X7 whose specification is shown in Table 4. It can be seen that 6 points and 5 points are enough to build an efficient speed function that fall within acceptable limits of deviation for matrix multiplication and Cholesky Factorization respectively.

One naïve approach to select a set of **i** points is: If (x_{min}, s_{min}) is the point with minimal problem size experimentally obtained and (x_{max}, s_{max}) is the point with maximal problem size experimentally obtained, the remaining **i**-2 points experimentally tested have problem sizes (x_{min}+(x_{max}-x_{min})/(i-1)),…,(x_{min}+(i-2)*(x_{max}-x_{min})/(i-1))) respectively. The minimum problem size could be as low as a size of memory that fits into the top level of memory hierarchy of the computer and the maximum problem size is as high as the size of memory that can fit into the last level of the memory hierarchy of the computer.

We use piece-wise linear function approximation illustrated in Figure 4 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 continuous functions of problem size up till its upper bound on the problem size and shape requirements of the graph. 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 1/3 for Cholesky Factorization. In the case of matrix-matrix multiplication, the size of the task is the number of elements in resultant matrix C=A×B. In the case of Cholesky Factorization, the size of the task is the number of elements in the factorized matrix.

Figure 5 (a) shows the speedup of the matrix-matrix multiplication executed on this network using the advanced model over the matrix-matrix multiplication using the modified version of the standard model that determines the speed of the processor based on the multiplication of two dense 500×500 matrices and two dense 4000×4000 matrices. For problem sizes beyond 24000, the figure shows that the distribution given by the performance model [1] will result in failure of the application. For these problem sizes, the modified performance model is used to obtain optimal distribution.

Figure 5 (b) shows the speedup of the matrix factorization executed on this network using the advanced model over the matrix factorization using the modified version of the standard model that determines the speed of the processor based on the matrix factorization of a dense 2000×2000 matrix, and a dense 10000×10000 matrix. For problem sizes beyond 19000, the figure shows that the distribution given by the performance model [1] will result in failure of the application. For these problem sizes, the modified performance model is used to obtain optimal distribution.

The speedup calculated is the ratio of the execution time of the application using the advanced model over the execution time of the application using the modified version of the standard model. The modified version of the standard performance models is described briefly at the beginning of section II where the speed of the processor is represented by a single number and there is an upper bound on the number of elements stored by each processor.

A set of as few as 5 points is used to build the speed functions of the processors. As can be seen from the figures, the advanced model performs better than the currently existing models for a network of heterogeneous computers that demonstrates relative speeds that are non-constant functions of the size of the problem and when there is an upper bound on the size of the task that can be solved by each processor.

IV. Conclusion

In this paper, we address the problem of optimal distribution or scheduling of arbitrary tasks in heterogeneous environments when one or more tasks do not fit into the main memory of the processors and when there is an upper bound on the size of the task that can be solved by each computer. We have proposed an advanced performance model of a network of heterogeneous computers and designed efficient algorithms of data partitioning with this model.

In the presented research we do not take account of communication cost. Although we well understand the importance of its incorporation in our performance model, this is just out of scope of this paper. We also understand the importance of the problems of efficient building and maintaining of our model. These two problems are also out of scope of the paper and are subjects of our current research.

References

[1] A. Lastovetsky and R. Reddy, “Data Partitioning with a Realistic Performance Model of Networks of Heterogeneous Computers,” Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2004), 26-30 April 2004, New Mexico, France, CD-ROM/Abstracts Proceedings, IEEE Computer Society 2004.

[2] A. Lastovetsky and R. Reddy, “Classification of Partitioning Problems for Networks of Heterogeneous Computers,” Proceedings of the 5th International Conference on Parallel Processing and Applied Mathematics (PPAM 2003), Czestochowa, Poland, Lecture Notes in Computer Science, 3019, pp.-, 2003.

[3] J. Stiles, T. Bartol, E. Salpeter and M. Salpeter, “Monte Carlo simulation of neuromuscular transmitter release using MCell, a general simulator of cellular physiological processes,” Computational Neuroscience, pages 279--284, 1998.

[4] P. Avery and I. Foster, “The GriPhyN Project: Towards Petascale Virtual Data Grids,” Technical Report GriPhyN-2001 - 15, 2001.

[5] T. Yoshida, T. Hiroyasu, M. Miki, M. Ogura, and Y. Okamato, “Energy Minimization of Protein Tertiary Structure by Parallel Simulated Annealing using Genetic Crossover,” Proceedings of 2002 Genetic and Evolutionary Computation Conference (GECCO 2002) Workshop Program, (2002), pp.49-51.

[6] A. Lastovetsky and R. Reddy, “Classification of Partitioning Problems for Networks of Heterogeneous Computers,” Technical Report, University College Dublin, December 2003.

[7] A. Lastovetsky. Parallel Computing on Heterogeneous Networks. John Wiley & Sons, 423 pages, 2003, ISBN: 0-471-22982-2.

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.

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.

TABLE I
SPECIFICATIONS OF THE ELEVEN HETEROGENEOUS COMPUTERS

Machine Name	Architecture	cpu MHz	Main Memory (KB)	Maximum size of task (Matrix-Matrix Multiplication)	Maximum size of task (Cholesky Factorization)	Cache (KB)
X1	Linux 2.4.18-3 i686 Intel Pentium III	997	254576	24502500	30250000	256
X2	SunOS 5.5 Sun4m sparc SUNW,SPARCstation-5	110	65536	6000000	6250000	512
X3	Linux 2.4.20-20.9bigmem Intel(R) Xeon(TM)	2783	7933500	116640000	262440000	512
X4	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1030508	116640000	144000000	512
X5	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1030508	90250000	108160000	512
X6	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1030508	116640000	144000000	512
X7	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1030508	116640000	144000000	512
X8	SunOS 5.8 sun4u sparc SUNW,Ultra-5_10	440	524288	31360000	64000000	2048
X9	SunOS 5.8 sun4u sparc SUNW,Ultra-5_10	440	524288	30250000	59290000	2048
X10	SunOS 5.8 sun4u sparc SUNW,Ultra-5_10	440	524288	30250000	64000000	2048
X11	SunOS 5.8 sun4u sparc SUNW,Ultra-5_10	440	524288	30250000	59290000	2048

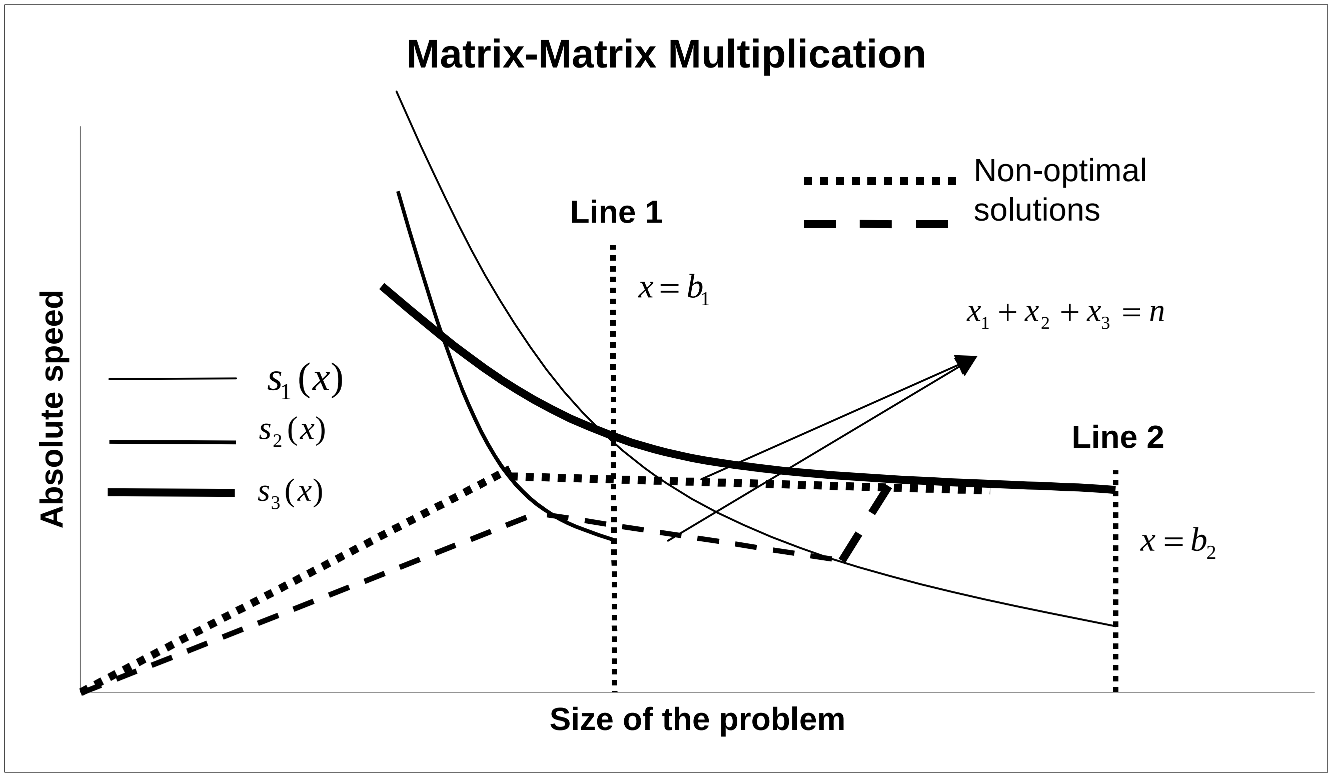


Fig. 1. A small network of three processors whose speeds are shown against the size of the problem. The Matrix-Matrix Multiplication used here uses a poor solver that does not use memory hierarchy efficiently. n is the size of the problem.

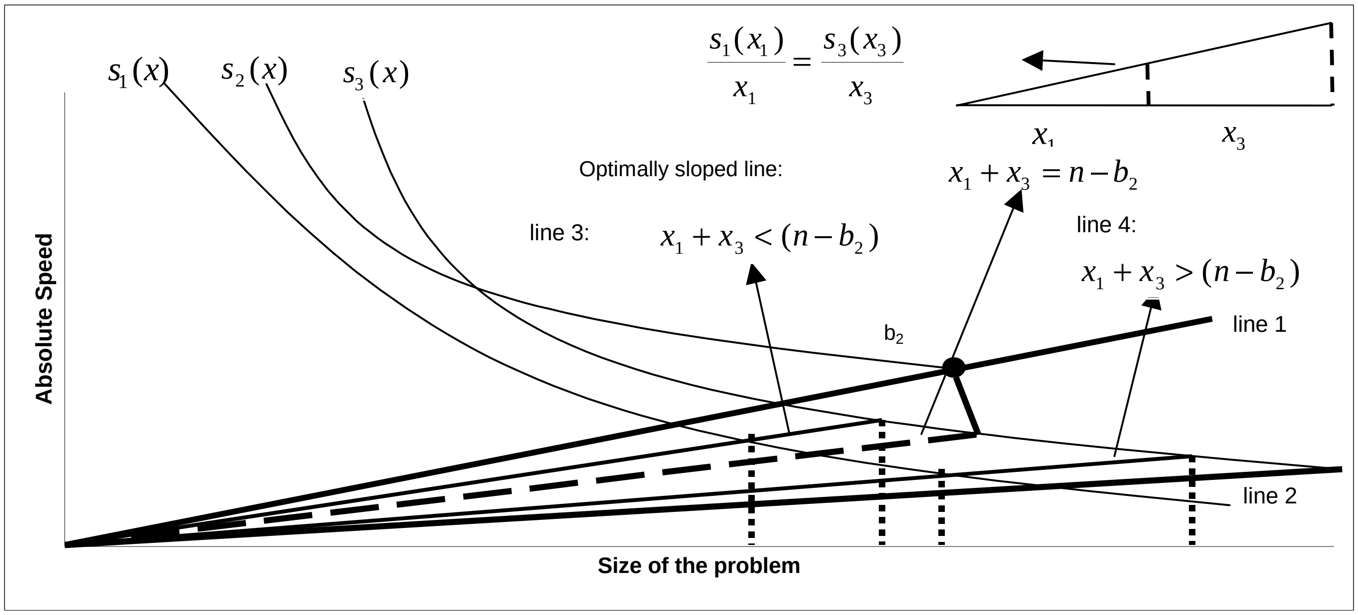
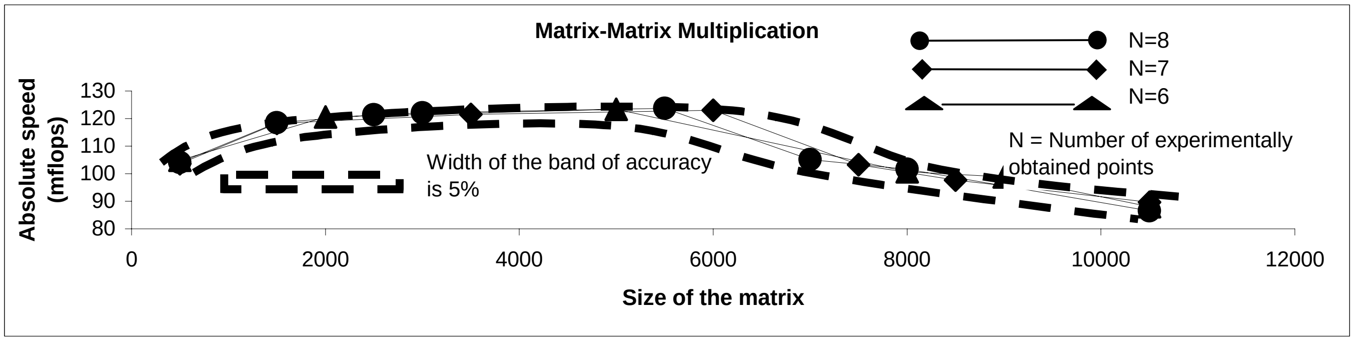


Fig. 2. For processor represented by speed function $s_2(x)$, the upper bound on the number of elements that the processor can hold is b_2. We assign the number of elements equal to the upper bound b_2 for the processor represented by speed function $s_2(x)$. We then partition the set with n-b_2 elements amongst the processors represented by speed functions $s_1(x)$ and $s_3(x)$ respectively. The two lines, between which the optimal solution of partitioning the set with n-b_2 elements amongst the processors $s_1(x)$ and $s_3(x)$ respectively lies, are given by line1 and line2.



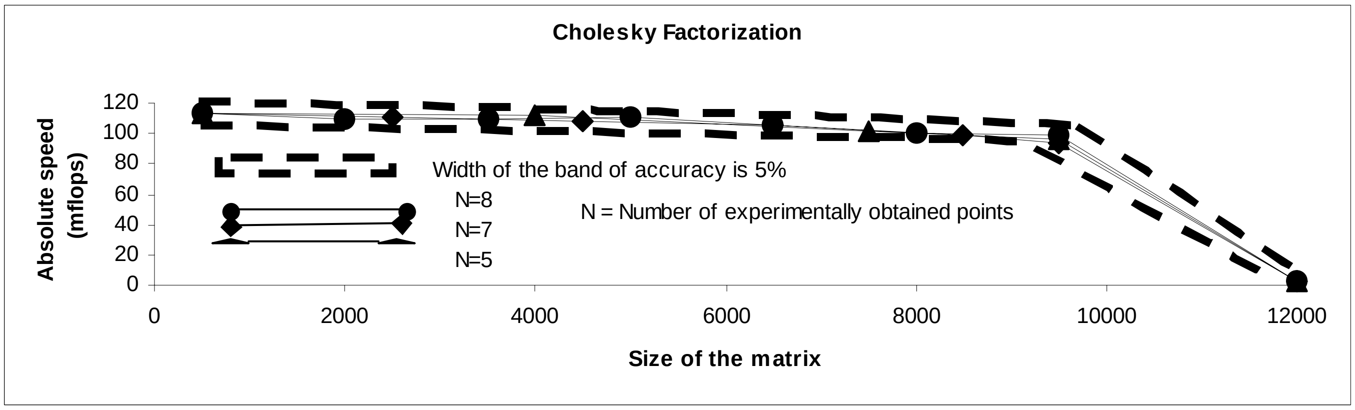


Fig. 3. Determination of a set with relatively few points used to build the speed functions of the processors. As few as 6 points and 5 points can be used to build an efficient speed function for matrix-matrix multiplication and Cholesky Factorization respectively with deviation approximately 5% from other speed functions built with more number of points.

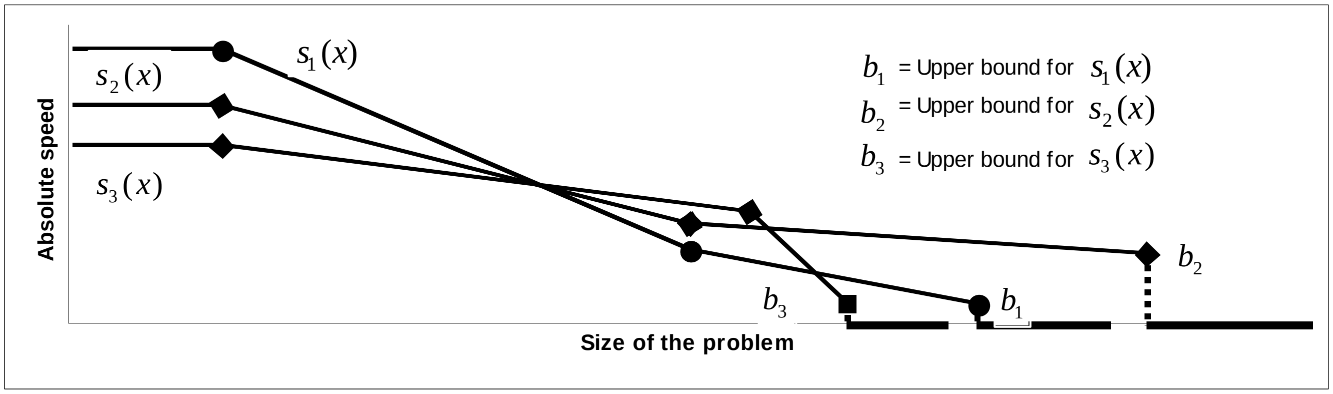
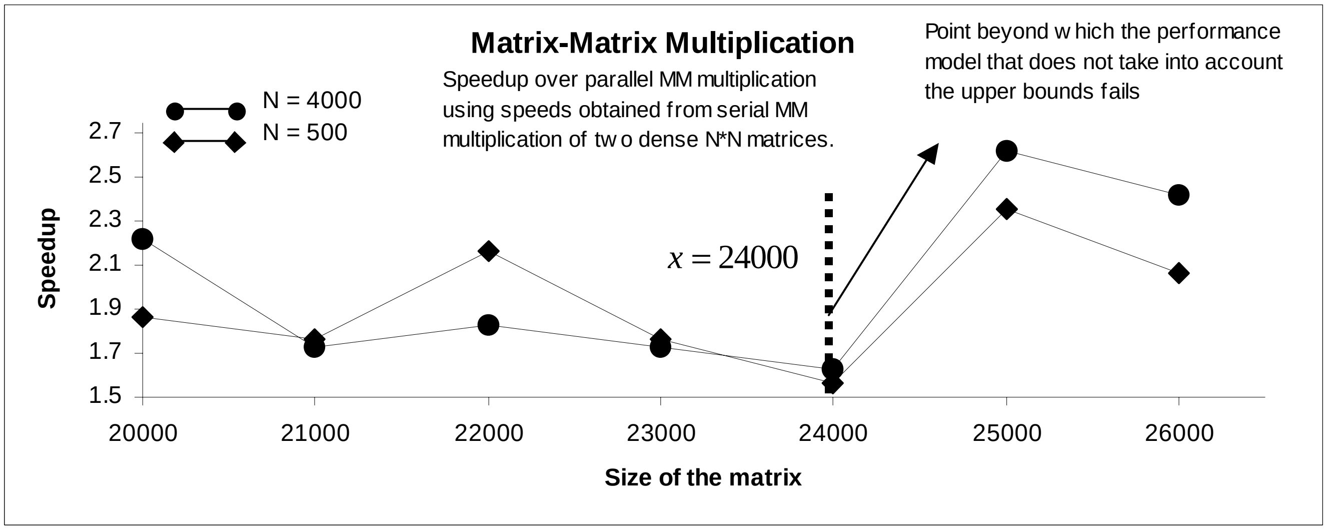
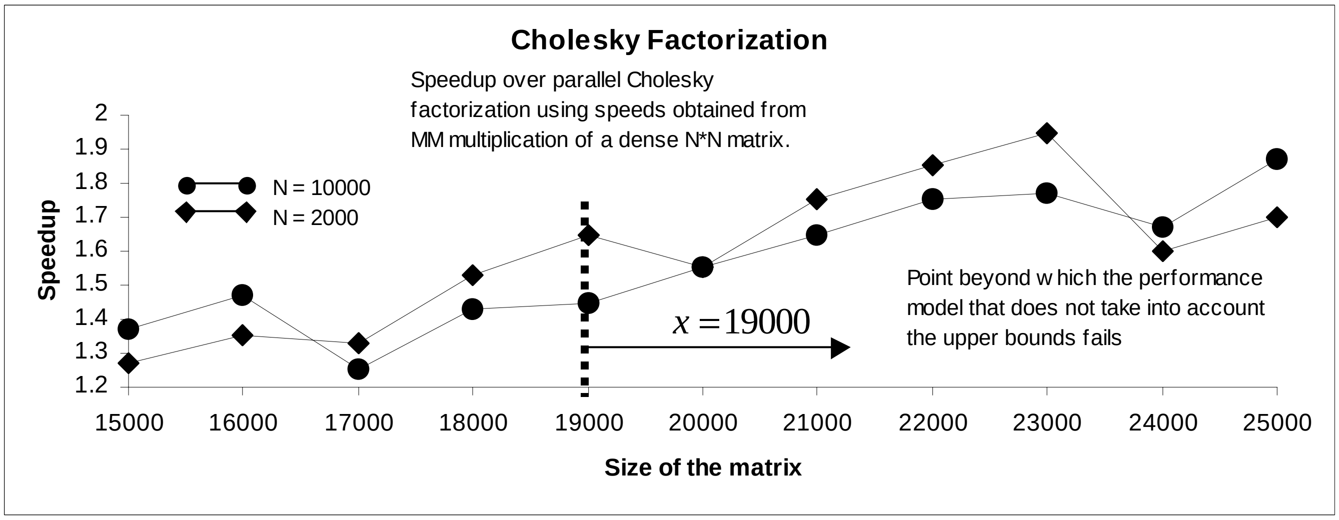


Fig. 4. Using piece-wise linear approximation to build speed functions for 3 processors. The speed functions are built from 3 experimentally obtained points. Speeds of the processors are assumed to be zero for problem sizes beyond their upper bounds.



(a)



(b)

Fig. 5. Results obtained using the network of heterogeneous computers shown in Table 1. (a) Comparison of speedups of Matrix-matrix multiplication using horizontal striped partitioning of matrices. (b) Comparison of speedups of Cholesky factorization using horizontal striped partitioning of matrices.