Journal of Parallel and Distributed Computing **61**, 520–535 (2001)
doi:10.1006/jpdc.2000.1686, available online at http://www.idealibrary.com on IDEAL®

Heterogeneous Distribution of Computations Solving Linear Algebra Problems on Networks of Heterogeneous Computers

Alexey Kalinov and Alexey Lastovetsky

Institute for System Programming, *Russian Academy of Sciences*, *25*, *Bolshaya Kommunisticheskaya str.*, *Moscow 109004*, *Russia*
E-mail: ka@ispras.ru, lastov@ispras.ru

Received December 21, 1998; revised April 3, 2000; accepted September 18, 2000

This paper presents and analyzes two different strategies of heterogeneous distribution of computations solving dense linear algebra problems on heterogeneous networks of computers. The first strategy is based on heterogeneous distribution of processes over processors and homogeneous block cyclic distribution of data over the processes. The second is based on homogeneous distribution of processes over processors and heterogeneous block cyclic distribution of data over the processes. Both strategies were implemented in the mpC language—a dedicated parallel extension of ANSI C for efficient and portable programming of heterogeneous networks of computers. The first strategy was implemented using calls to ScaLAPACK; the second strategy was implemented with calls to LAPACK and BLAS. Cholesky factorization on a heterogeneous network of workstations is used to demonstrate that the heterogeneous distributions have an advantage over the traditional homogeneous distribution.

Key Words: parallel programming tools; parallel linear algebra software; ScaLAPACK; heterogeneous computing; parallel languages.

1. INTRODUCTION

Nowadays, high-performance scientific computations concentrate mostly on distributed—memory supercomputers consisting of identical processors. Therefore, it is no wonder that the main efforts of developers of parallel numerical libraries have been aimed at the achievement of the best performance on such machines. As a rule, in computing on such homogeneous computer systems, a strategy of homogeneous distribution of computations over processors is used [1, 6, 9]. The strategy will be referred to as the HoHo strategy, "homogeneous distribution of processes over processors–homogeneous distribution of data over the processes," with each

0743-7315/01 $35.00

physical processor running one process and data being evenly partitioned among the processes.

At the same time, progress in network technologies is making local and even global networks of computers (in particular, networks of PCs and workstations) more and more attractive for high-performance parallel computing. In developing applications for such networks, it is necessary to take into account their heterogeneity as the main peculiarity of common networks that differentiates them from supercomputers.

The heterogeneity is displayed at least in two forms—first in the form of heterogeneity of machine arithmetics of such parallel systems. Related challenges existing in writing reliable numerical library software for heterogeneous computing environments have been analyzed in [5].

Second, heterogeneity is displayed in the form of both the performances of individual processors and the speeds of data transfer between the processors. As a rule, to solve linear algebra problems on a heterogeneous network of computers, one uses numeric software originally developed for homogeneous distributed-memory machines and later ported to the network. Let us see what happens when a parallel linear algebra application, which provides a good distribution of computations and communications (due to the HoHo strategy) while running in homogeneous environments, runs on a heterogeneous network of computers. Since volumes of computations executed by different processors are approximately equal to each other, more powerful processors will wait for the weakest one at synchronization points. Therefore, the total time of computations will be determined by the time elapsed on the weakest processor.

We have done an experiment corroborating this statement. We considered two local subnetworks of our local network: homogeneous `abcd` consisting of SUN workstations `a`, `b`, `c`, and `d` of the same relative performance 1.9, and heterogeneous `aefg` consisting of SUN workstations `a`, `e`, `f`, and `g` of relative performances 1.9, 2.8, 2.8, and 7.1. The relative performances were estimated by a sequential LAPACK [2] Cholesky factorization routine. One can see that the total power of the heterogeneous subnetwork is about twice that of the homogeneous one. It could be expected that, for example, the parallel ScaLAPACK Cholesky solver [6] would be executed on the heterogeneous subnetwork about twice as fast as on the homogeneous one. But the real situation, shown in Fig. 1, turned out quite differently.

A natural solution to this problem is heterogeneous distribution of both processes over the processors and/or data among the processes, taking into account differences in performances of processors and speeds of communication links. As demonstrated in [3], such distribution allows us to achieve much better distribution of computations over processors of a heterogeneous computing network and hence to utilize its performance potential more efficiently.

Such heterogeneous distribution is a complex problem whose solution needs adequate tools. Designed especially to write efficient and portable parallel applications for heterogeneous networks of computers, the mpC language [12] is just such a tool. This language is an ANSI C superset that allows applications to be written that adapt to differences in performance of both processor and communication links of any particular executing network. In order to make passible the employment of software written with other parallel programming systems, mpC provides

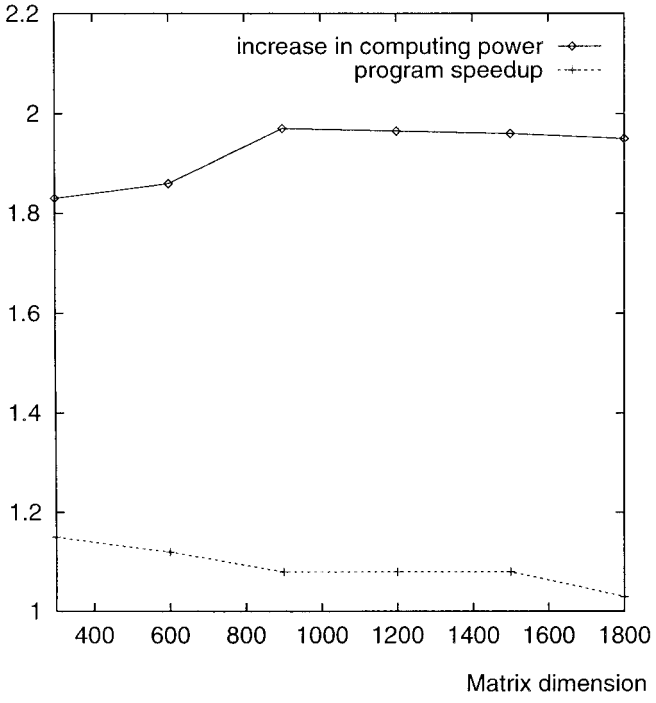

FIG. 1. Speedup achieved by the ScaLAPACK Cholesky solver on the heterogeneous network relative to the homogeneous one. The networks consist of four workstations. One workstation of the heterogeneous network belongs to the homogeneous one. Other workstations of the heterogeneous network are more powerful.

interfaces to such systems. In particular, mpC provides an interface to ScaLAPACK. More about mpC as well as the mpC free software can be found at http://www.ispras.ru/~mpc.

Different distribution strategies will be investigated using a typical linear algebra problem—the Cholesky factorization of square dense matrices. The problem was chosen as a well-known example of a practically important problem whose parallel solution needs careful balancing of computations and communications. For all the strategies, the same Cholesky factorization algorithm, namely the one implemented in ScaLAPACK, will be used. This algorithm provides very good scalability and even workload of processors and links for the traditional HoHo distribution on homogeneous networks.

In this paper, we consider only the heterogeneity of processor performances and represent an executing computer system as a set $\mathbf{L}$ of processors characterized by a vector $\mathbf{R}(\mathbf{L})$ of relative processor performances. A running parallel program, performing the Cholesky factorization of a processed matrix $\mathbf{M}$, consists of a number of processes with h_l processes running on processor $l \in \mathbf{L}$. For each $l \in \mathbf{L}$ only e_l processes ($e_l \leqslant h_l$) are involved in computations on the processed matrix $\mathbf{M}$. The set of all those involved processes is regarded as a process grid of P rows and Q columns ($\sum_{l \in \mathbf{L}} e_l = P \cdot Q$).

The paper considers only the following two extreme distribution strategies:

1. The HeHo strategy—heterogeneous distribution of involved $P \cdot Q$ processes over processors with homogeneous block cyclic distribution of matrix $\mathbf{M}$ over the processes. In this case, heterogeneous distribution of the involved processes is performed by an mpC program, while the latter calls a parallel ScaLAPACK function to perform the parallel Cholesky factorization itself.

2. The HoHe strategy—homogeneous distribution of involved $P \cdot Q$ processes over processors with each process running on a separate processor and heterogeneous block cyclic distribution of matrix $\mathbf{M}$ over the processes. In this case, both the distribution of the involved processes and the parallel Cholesky factorization itself are performed by an mpC program calling BLAS [7] and LAPACK functions for local computations. Note that in this case the parallel algorithm implemented by the mpC program for Cholesky factorization itself is, in general, the same as that implemented by the ScaLAPACK function for the HeHo strategy.

Section 2 describes the HeHo strategy. Section 3 introduces the heterogeneous block cyclic matrix distribution and describes the HoHe strategy. Section 4 shortly explains why we used the mpC language to implement the heterogeneous strategies. In Section 5 we try to analyze the performance of the strategies. Section 6 gives experimental results of Cholesky factorizations, using the heterogeneous strategies, on a network of heterogeneous workstations.

2. HETEROGENEOUS PROCESS DISTRIBUTION WITH HOMOGENEOUS DATA DISTRIBUTION

Let processor l of performance r_l execute h_l processes of the running parallel program. Let the total number of processes that should be involved in computations be equal to $P \cdot Q$; so $\sum_{l \in \mathbf{L}} h_l \geqslant P \cdot Q$.

For the ScaLAPACK algorithm of Cholesky factorization [6] and matrix $\mathbf{M}$ evenly distributed over the involved $P \cdot Q$ processes, each involved process executes approximately the same volume of computations. Therefore, in this case, the number e_l of processes, running on processor l and involved in computations, should be proportional to the performance r_l of the processor. Obviously, $\sum_{I \in \mathbf{L}} e_l = P \cdot Q$ and $e_l \leqslant h_l$.

For example, if $P = Q = 4$, $\mathrm{card}(\mathbf{L}) = 6$, and $\mathbf{R}(\mathbf{L}) = \{6, 5, 4, 3, 2, 1\}$, then the distribution $\{\mathbf{e}(\mathbf{l})\}$ may look as follows: $\{5, 4, 3, 2, 1, 1\}$.

The presented distribution strategy is based on the homogeneous two-dimensional data distribution, which can be briefly defined in accordance with [6] as follows.

A one-dimensional block cyclic data distribution is defined by the mapping $i_g \mapsto \langle p, b, i_l \rangle$ of a global index, i_g, where p is the logical process number, b is the block number in process p, and i_l is the local index within block b to which i_g is mapped.

(a)

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	1	1	2	2	0	0	1	1	2	2
1	0	0	1	1	2	2	0	0	1	1	2	2
2	3	3	4	4	5	5	3	3	4	4	5	5
3	3	3	4	4	5	5	3	3	4	4	5	5
4	0	0	1	1	2	2	0	0	1	1	2	2
5	0	0	1	1	2	2	0	0	1	1	2	2
6	3	3	4	4	5	5	3	3	4	4	5	5
7	3	3	4	4	5	5	3	3	4	4	5	5
8	0	0	1	1	2	2	0	0	1	1	2	2
9	0	0	1	1	2	2	0	0	1	1	2	2
10	3	3	4	4	5	5	3	3	4	4	5	5
11	3	3	4	4	5	5	3	3	4	4	5	5

(b)

	0	1	6	7	2	3	8	9	4	5	10	11
0	0	0	0	0	1	1	1	1	2	2	2	2
1	0	0	0	0	1	1	1	1	2	2	2	2
4	0	0	0	0	1	1	1	1	2	2	2	2
5	0	0	0	0	1	1	1	1	2	2	2	2
8	0	0	0	0	1	1	1	1	2	2	2	2
9	0	0	0	0	1	1	1	1	2	2	2	2
2	3	3	3	3	4	4	4	4	5	5	5	5
3	3	3	3	3	4	4	4	4	5	5	5	5
6	3	3	3	3	4	4	4	4	5	5	5	5
7	3	3	3	3	4	4	4	4	5	5	5	5
10	3	3	3	3	4	4	4	4	5	5	5	5
11	3	3	3	3	4	4	4	4	5	5	5	5

FIG. 2. Example of a homogeneous block cyclic distribution of a 12×12 matrix over 2×3 process grid with the block size 2×2—(a) matrix distribution over grid and (b) distribution from processor point of view.

Thus, if the number of processes is P, and the block size is m, then the block cyclic data distribution may be written as

$$i_g \mapsto \left\langle s \bmod P, \left\lfloor \frac{s}{P} \right\rfloor, i_g \bmod m \right\rangle,$$

where $s = \lfloor \frac{i_g}{m} \rfloor$.

Suppose we have a set of processes considered as a logical process grid with P rows and Q columns and a block-partitioned matrix with block size $m \times n$. Then the two-dimensional block cyclic distribution of the matrix can be regarded as a combination of two such one-dimensional block cyclic distributions: one that distributes the rows of the matrix over P processes and another that distributes the columns over Q processes. That is, the matrix element indexed globally by (i_g, j_g) can be written as

$$(i_g, j_g) \mapsto \langle (p, q), (b_p, b_q), (i_l, j_l) \rangle.$$

The distribution partitions the matrix into *generalized* blocks of size $(m \cdot P) \times (n \cdot Q)$, each partitioned into $(P \cdot Q)$ blocks of the same size, going to a separate process.

Figure 2 shows an example of the homogeneous block cyclic distribution of a 12×12 matrix, block-partitioned with the block size 2×2 ($m = 2$, $n = 2$), over a 2×3 process grid ($P = 2$, $Q = 3$). In this case, generalized blocks are of size 4×6.

3. HOMOGENEOUS PROCESSES DISTRIBUTION WITH HETEROGENEOUS DATA DISTRIBUTION

Let $\text{card}(\mathbf{L}) \geqslant P \cdot Q$. Let $P \cdot Q$ processes be distributed over $P \cdot Q$ most powerful processors in such a way that just one process goes to each of these processors and

let matrix **M** be distributed over the processes in accordance with the heterogeneous two-dimensional block cyclic distribution presented below.

Suppose that a positive real number is associated with each processor to characterize its relative performance. Then in addition to four numbers P, Q, m, and n parameterizing the homogeneous block cyclic distribution, the heterogeneous one is parameterized also by a $P \times Q$ matrix $\mathbf{R} = \{r_{ij}\}$, elements of which characterize relative performances of the corresponding processors. Its main difference from the homogeneous distribution lies in heterogeneous data distribution inside a generalized block. As in the case of the homogeneous distribution, a generalized $(m \cdot P) \times (n \cdot Q)$ block is partitioned into $(P \cdot Q)$ blocks. But in the case of the heterogeneous distribution, the blocks are not of the same size, but their sizes $m_{ij} \times n_{ij}$ depend on the performance of the processors. In general,

$$m_{ij} = \phi(i, j, m, P, \mathbf{R}), \qquad \sum_{i=0}^{P-1} m_{ij} = m \cdot P \quad \text{for all } j,$$

$$n_{ij} = \psi(i, j, n, Q, \mathbf{R}), \qquad \sum_{j=0}^{Q-1} n_{ij} = n \cdot Q \quad \text{for all } i,$$

and $m \times n$ is the average size of the uneven blocks.

In the paper, we consider the simplest form of functions ϕ and ψ deduced from the assumption that the part of matrix **M** processed by a separate processor is proportional to its performance. That is,

$$m_{ij} \cdot n_{ij} = \frac{m \cdot P \cdot n \cdot Q \cdot r_{ij}}{\sum_{i=0}^{P-1} \sum_{j=0}^{Q-1} r_{ij}}.$$

(a)

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	1	1	2	0	0	0	1	1	2
1	0	0	0	1	1	2	0	0	0	1	1	2
2	3	3	3	4	4	2	3	3	3	4	4	2
3	3	3	3	4	4	5	3	3	3	4	4	5
4	0	0	0	1	1	2	0	0	0	1	1	2
5	0	0	0	1	1	2	0	0	0	1	1	2
6	3	3	3	4	4	2	3	3	3	4	4	2
7	3	3	3	4	4	5	3	3	3	4	4	5
8	0	0	0	1	1	2	0	0	0	1	1	2
9	0	0	0	1	1	2	0	0	0	1	1	2
10	3	3	3	4	4	2	3	3	3	4	4	2
11	3	3	3	4	4	5	3	3	3	4	4	5

(b)

	0	1	2	6	7	8
0	0	0	0	0	0	0
1	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
8	0	0	0	0	0	0
9	0	0	0	0	0	0
2	3	3	3	3	3	3
3	3	3	3	3	3	3
6	3	3	3	3	3	3
7	3	3	3	3	3	3
10	3	3	3	3	3	3
11	3	3	3	3	3	3

	3	4	9	10
0	1	1	1	1
1	1	1	1	1
4	1	1	1	1
5	1	1	1	1
8	1	1	1	1
9	1	1	1	1
2	4	4	4	4
3	4	4	4	4
6	4	4	4	4
7	4	4	4	4
10	4	4	4	4
11	4	4	4	4

	5	11
0	2	2
1	2	2
4	2	2
5	2	2
8	2	2
9	2	2
2	5	5
3	5	5
6	5	5
7	5	5
10	5	5
11	5	5

FIG. 3. Example of a heterogeneous block cyclic distribution of a 12×12 matrix over 2×3 processor grid with the average block size 2×2—(a) matrix distribution over grid and (b) distribution from processor point of view.

In particular, the above condition can be satisfied by the following choice of m_{ij} and n_{ij}:

$$n_{ij} = n_j = \frac{\sum_{i=0}^{P-1} r_{ij} \cdot n \cdot Q}{\sum_{i=0}^{P-1} \sum_{j=0}^{Q-1} r_{ij}}, \qquad m_{ij} = \frac{r_{ij} \cdot m \cdot P}{\sum_{i=0}^{P-1} r_{ij}}.$$

Figure 3 shows an example of the heterogeneous block cyclic distribution of a 12×12 matrix over a 2×3 processor grid ($P = 2$, $Q = 3$) with the average block size 2×2 ($m = 2$, $n = 2$), the generalized block size 4×6, and the matrix of processor performances

$$\mathbf{R} = \begin{pmatrix} 6 & 4 & 2 \\ 5 & 3 & 1 \end{pmatrix}.$$

4. IMPLEMENTATION OF HETEROGENEOUS DISTRIBUTION OF COMPUTATIONS IN MPC

While implementing the heterogeneous strategies we need to operate with such quantitative characteristics as relative speeds of processes. Traditional high-level tools for parallel programming networks, which we know of, do not provide facilities allowing the programmer to do so. Low-level tools, such as PVM and MPI, allow the programmer to write parallel applications adaptable to performances of processors. But the writing of such PVM/MPI applications is not supported directly by the tools and is extremely complex, tedious, and error-prone just due to their low level.

Therefore, we implemented the heterogeneous strategies in mpC [12]—a dedicated parallel language aimed at efficient portable modular programming heterogeneous networks of computers. It provides language constructs allowing the programmer to specify requirements on resources, necessary for efficient execution of the parallel application, and the mpC programming system tries to satisfy the requirements taking into account peculiarities of any particular heterogeneous network of computers.

The main idea underlying mpC is that an mpC application explicitly defines a dynamic abstract heterogeneous computing network and distributes data, computations, and communications over the network. The mpC programming system uses this information at run time to map the abstract computing network to any real executing network of computers in such a way that ensures efficient running of the application on this real network.

The mpC language is an ANSI C superset that introduces a new kind of managed resource, *computing space*, defined as a set of virtual processors of different performances connected with links of different communication speeds. At run time, the virtual processors are represented by actual processes of the particular running parallel application. The programmer manages the computing space by means of creating and discarding regions of the computing space, named *network objects*, just like he/she manages storage, creating and discarding data objects (regions of storage). At any moment of program execution, just a set of defined network objects represents the abstract computing network.

Our mpC implementation of the HeHo strategy uses dedicated parallel means of mpC for automatic process distribution while Cholesky factorization itself is performed by ScaLAPACK processes. More details about the implementation of the HeHo strategy can be found in [10].

The implementation of the HoHe strategy uses means of the mpC language both for process distribution and for Cholesky factorization itself with calls to LAPACK and BLAS routines for standard serial computations. More details about the implementation of the HoHe strategy can be found in [11].

5. PERFORMANCE ANALYSIS

For performance analysis of a parallel algorithm solving a problem of size n on p processors we use parallel efficiency $E(n, p)$ defined in [8] as

$$E(n, p) = \frac{T_{\text{seq}}(n)}{pT(n, p)},$$

where $T(n, p)$ is the run time of the parallel algorithm, and $T_{\text{seq}}(n)$ is the run time of the best sequential algorithm on a processor. A parallel algorithm is said to be scalable if its parallel efficiency depends on the problem size and the number of processes only through their ratio.

To estimate the times of local computations, we use a very rough approximation. It is assumed that the run time of the sequential Cholesky factorization on a processor with performance s is given by $N^3/3s$, and the time of local computations is calculated in a manner similar to the time of sequential computations. This means that if p processors of performance s are involved in parallel computations then the time of local computations is given by $N^3/3ps$. For the HoHo strategy, the time t_{comp}^{HoHo} of local computations is determined by the time of local computations on the slowest processor having performance $s_{\min}$, $t_{\text{comp}}^{HoHo} = N^3/3ps_{\min}$. For the HeHo strategy, the time of local computations t_{comp}^{HeHo} is determined by the time of local computations on the slowest process having performance $s'_{\min}$, $t_{\text{comp}}^{HeHo} = N^3/3ps'_{\min}$. The process performance is calculated as s/e, where s is the performance of the processor on which the process runs, and e is the number of processes involved in computations. For a very heterogeneous network $s'_{\min}$ is greater than $s_{\min}$, because the HeHo strategy tries not to involve processes running on slower processors in computations. For the HoHe strategy, the volume of data processed by every processor is proportional to its performance. Therefore, the time t_{comp}^{HoHe} of local computations is the same for all involved processes, $t_{\text{comp}}^{HoHe} = N^3/3ps_{\text{aver}}$, where s_{aver} is the average processor performance.

Ideally, if concurrent overheads could be neglected, speedups provided by the heterogeneous strategies as compared to the homogeneous one would be approximated as is presented in Table 1.

The overwhelming majority of local networks are Ethernet based. Therefore, our estimation of communication overheads is based on the assumption that communications between processors are purely sequential, which is very typical for

TABLE 1

Ideal Speedups Provided by the Heterogeneous Strategies as Compared to the Homogeneous One

Strategy	Speed-up
HoHe	$s_{\text{aver}}/s_{\min}$
HeHo	$s'_{\min}/s_{\min}$

a shared Ethernet. In addition, we approximate the time to transfer a message between two processors by a linear function $a + bL$, where a is the time to prepare the message for transfer (communication latency) and bL is the time to transfer L doubles.

Cholesky factorization factors a symmetric, positive definite matrix A into a product of a lower triangular matrix L and its transpose; i.e., $A = L \cdot L^t$. One can partition the matrices A, L, and L^t and write the system as

$$\begin{bmatrix} A_{11} & A_{21}^t \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \cdot \begin{bmatrix} L_{11}^t & L_{21}^t \\ 0 & L_{22}^t \end{bmatrix} = \begin{bmatrix} L_{11}L_{11}^t & L_{11}L_{21}^t \\ L_{21}L_{11}^t & L_{21}L_{21}^t + L_{22}L_{22}^t \end{bmatrix}.$$

If L_{11}, the lower triangle Cholesky factor of A_{11}, is known, then the block equations can be rearranged as

$$L_{21} \leftarrow A_{21}(L_{11}^t)^{-1},$$
$$A_{22} \leftarrow A_{22} - L_{21}L_{21}^t = L_{22}L_{22}^t.$$

The factorization can be done by recursively applying the step outlined above to the updated matrix A_{22}. The computations involve the following operations:

1. The largest A_{11} belonging to one process is selected and this process computes L_{11};
2. L_{11} is broadcast to other processes of the grid column and the grid column processes compute L_{21};
3. L_{21} is broadcast to processes of the other grid columns;
4. L_{21} is transposed using the broadcast values of the L_{21} on the grid columns;
5. All processes update A_{22}.

This leads us to the estimate of the time of the communication

$$t_{\text{comm}} = t_2 + t_3 + t_4,$$

where t_i is the communication time of the ith operations. For the sake of simplicity we assume that $P = Q$, $n = m$, and N is divided by nP without a remainder. During the estimation of the HoHe strategy communication overhead we will proceed from the two obvious statements:

1. It takes not more than P^2 steps of the algorithm to decrease the dimension of the matrix A_{22} to nP (for the HoHo and HeHo strategies it takes P steps);

2. At every step of the L_{21} ring broadcast, not more than $2P-1$ messages are required (for the HoHo and HeHo strategies $P-1$ messages are required).

Thus the communication times of the HoHe strategy can be majored as

$$t_2 = (P-1) \sum_{i=1}^{N/(nP)} \frac{(nP)^2}{2} b + P^2 a$$

$$= (P-1)\, NP \left(\frac{n}{2} b + \frac{1}{n} a \right);$$

$$t_3 = (P-1) \sum_{i=1}^{N/(nP)} (N - (i-0.5)\, nP)\, nPb + P^2 (2P-1)\, a$$

$$= (P-1)\, N \left(\frac{N}{2} b + \frac{P(2P-1)}{n} a \right);$$

$$t_4 = (P-1) \sum_{i=1}^{N/(nP)} (N - (i-0.5)\, nP)\, nPb + P^2 (P-1)\, a$$

$$= (P-1)\, N \left(\frac{N}{2} b + \frac{P(P-1)}{n} a \right).$$

If P is large enough, the communication overhead can be approximated as

$$t_{\text{comm}}^{HoHe} \simeq PN^2 \left(b + \frac{3P^2}{Nn} a \right).$$

A similar analysis gives the estimate of communication overheads for the HoHo and HeHo strategies as

$$t_{\text{comm}}^{HoHo} \simeq t_{\text{comm}}^{HeHo} \simeq PN^2 \left(b + \frac{2P}{Nn} a \right).$$

They are the same in the case of MPI using only one protocol for communications between all processes. If a more efficient protocol is used for communication between processes running on the same processor, then $t_{\text{comm}}^{HoHo} > t_{\text{comm}}^{HeHo}$.

For simplicity, we assume that $T(n, p) = t_{\text{comp}}(n, p) + t_{\text{comm}}(n, p)$. Let s_{seq} be the performance of a processor on which sequential factorization has been carried out. Thus the estimated parallel efficiency for different strategies is presented in Table 2.

In our case, the problem size $n = N^2$ and the number of processors $p = P^2$. This means that for the algorithms to be scalable, their parallel efficiency should depend on the matrix dimension N and the grid dimension P only through their ratio. It is not true for all the considered strategies; hence these parallel algorithms are not scalable for Ethernet-like networks.

TABLE 2

Estimated Parallel Efficiencies for Various Strategies

Strategy	Parallel efficiency
HoHo	$\left(s_{\text{seq}}\left(\frac{1}{s_{\min}}+P^2*\left(\frac{P}{N}b+\frac{2}{n}\left(\frac{P}{N}\right)^2\right)\right)\right)^{-1}$
HeHo	$\left(s_{\text{seq}}\left(\frac{1}{s'_{\min}}+P^2*\left(\frac{P}{N}b+\frac{2}{n}\left(\frac{P}{N}\right)^2\right)\right)\right)^{-1}$
HoHe	$\left(s_{\text{seq}}\left(\frac{1}{s_{\text{aver}}}+P^2*\left(\frac{P}{N}b+\frac{3P}{n}\left(\frac{P}{N}\right)^2\right)\right)\right)^{-1}$

The HoHe strategy concentrates on the difference in processor performance and would be most effective on small networks. The main disadvantage of this strategy is that it leads to non-Cartesian data distribution. This increases the number of algorithm steps in P times and leads to nonscalability even in the case of networks with parallel communications. The situation can be improved by restrictions on possible choices of n_{ij} and m_{ij} so that they construct a real 2D grid [4] ($n_{ij}=n_j$, $m_{ij}=m_i$). In this case it takes no more than $2P-1$ steps of the algorithm to decrease the dimension of the matrix A_{22} to nP, the number of algorithm steps increases only by two times, and the algorithm becomes scalable for networks with parallel communications. But in the case of fast and relatively small networks this restriction can lead to a loss of a possible speedup.

Strange as it may seem, the straightforward and easy-to-accomplish HeHo strategy turns out to be very attractive. It provides the same scalability of the algorithm as HoHo. Compared to the HoHo strategy its communication overheads are not greater and its time of local computation can be shorter. But there is a pitfall in a straightforward implementation of this strategy. The point is that this strategy does not take into account processor memory size. For a small application, the total size of which does not exceed the size of main memory, it works quite well. But for an application dealing with big matrices it can cause swapping, which in turn causes a slowing down in the parallel application. Therefore to use this strategy it is necessary to restrict the number h_l of processes running on lth processors in accordance with the estimated size of the application and the main memory available.

6. EXPERIMENTAL RESULTS

We compared three distribution strategies:

- The HeHo strategy—heterogeneous distribution of processes over processors–homogeneous distribution of data over the processes implemented in mpC with calls to ScaLAPACK.

TABLE 3

Relative Performances of Processors Demonstrated on Serial Cholesky Factorization

1	2	3	4	5	6	7	8
1.9	1.9	1.9	1.9	1	2.8	2.8	7.1

- The HoHo strategy—homogeneous distribution of processes over processors–homogeneous distribution of data over the processes, the traditional distribution strategy implemented in ScaLAPACK.
- The HoHe strategy—homogeneous distribution of processes over processors–heterogeneous distribution of data over the processes implemented in mpC with calls to BLAS and LAPACK.

The comparison was performed for the Cholesky factorization on a network of workstations. In our experiments we used different parts of a local network consisting of eight uniprocessor Sun workstations of different performances interconnected via 10 Mbit Ethernet. MPICH 1.0.13 was used as a particular communication platform. All workstations executed the same copy of code. Relative performances of the workstations, obtained by means of execution of the LAPACK routine `dpotf` performing serial Cholesky factorization, are shown in Table 3. In our experiments we used free BLAS and LAPACK.

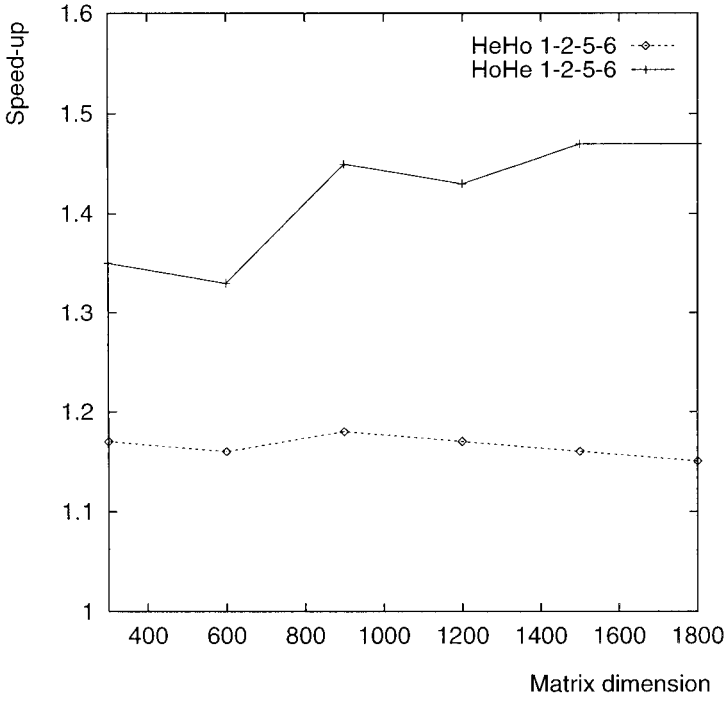

FIG. 4. Speedup achieved by the HeHo and HoHe distribution strategies relative to the homogeneous one on the heterogeneous networks consisting of four workstations, `1`, `2`, `5`, and `6`, 2×2 grid.

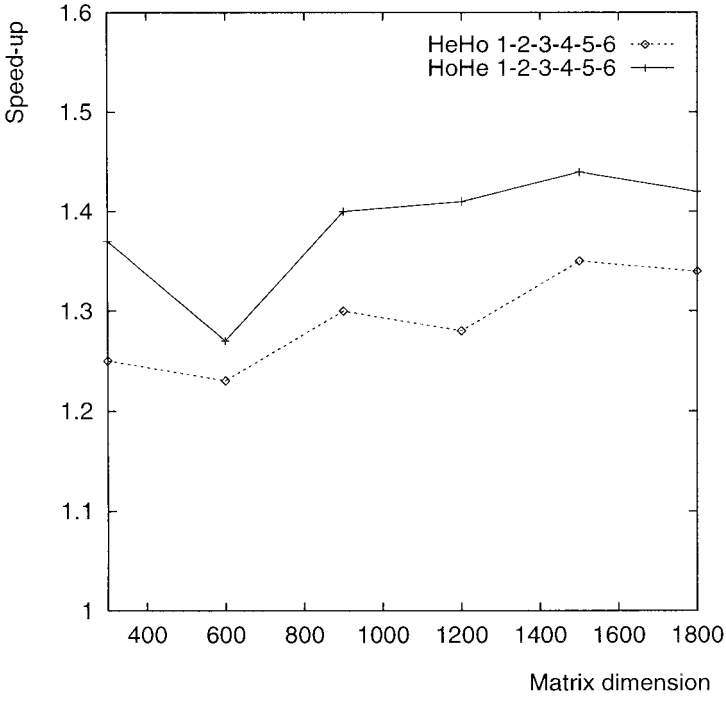

FIG. 5. Speedup achieved by the HeHo and HoHe distribution strategies relative to the homogeneous one on the heterogeneous networks consisting of six workstations, `1`, `2`, `3`, `4`, `5`, and `6`, 2×3 grid.

We compared speedups achieved by the heterogeneous strategies as compared to the homogeneous one on the following three heterogeneous networks:

- network `1-2-5-6` consisting of four workstations `1`, `2`, `5`, and `6`, 2×2 grid;
- network `1-2-3-4-5-6` consisting of six workstations `1`, `2`, `3`, `4`, `5`, and `6`, 2×3 grid;
- network `1-2-3-4-5-6-7-8` consisting of all eight workstations of the local network, 2×4 grid.

Figures 4, 5, and 6 present the experimental results. While experimenting, we used the block sizes found experimentally to provide better run time: $n = m = 6$ for the HoHo and HeHo strategies and $n = m = 20$ for the HoHe strategy. For the HeHo strategy two processes were running on each processor.

One can see that the two heterogeneous strategies provided very good speedups. For our network the HoHe strategy appeared more efficient than the HeHo one. Just as it was predicted by our performance analysis, the advantage of HoHe over HeHo as well as both of them over HoHo fell off as the network size grew. For a small network the run time is determined by the time of local computations. As the network size grows the contribution of communications in the total run time increases and the profit of the heterogeneous strategies decreases because it is due to better distribution of computations the weight of which lowers.

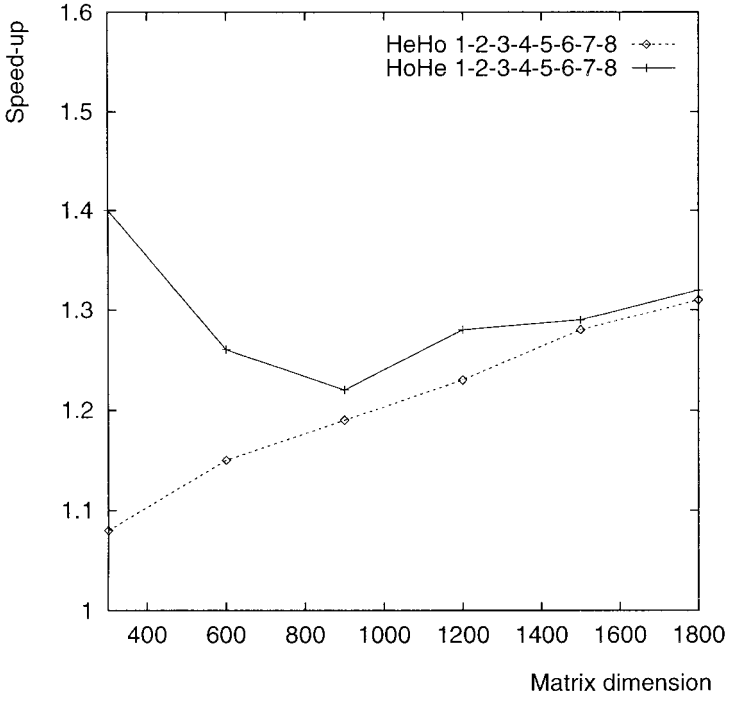

FIG. 6. Speedup achieved by the HeHo and HoHe distribution strategies relative to the homogenous one on the heterogeneous networks consisting of eight workstations, 1–8, 2×4 grid.

While estimating these results, it is necessary to take into account that the ScaLAPACK implementation of the Cholesky factorization is more efficient than the mpC one. For example, on the homogeneous network of workstations 1, 2, 3, and 4, the ScaLAPACK application is about 10% faster than the mpC one. Note, that the result is not due to the non-Cartesian nature of the heterogeneous block cyclic matrix distribution, because the same result is also obtained for a pure mpC application (that is, without calls to ScaLAPACK) implementing the homogeneous block cyclic matrix distribution. So it could be explained by a more rational communication pattern used in ScaLAPACK.

7. CONCLUSION

This paper has presented two different strategies of heterogeneous distribution of computations for linear algebra problems taking into account processor performances. It has been shown that for heterogeneous parallel environments the heterogeneous strategies are more efficient than the traditional homogeneous strategy.

The HoHe strategy (homogeneous distribution of involved processes over processors with each process running on a separate processor and heterogeneous block cyclic distribution of data over the processes) has turned out to be more efficient than the HeHo strategy (heterogeneous distribution of involved processes

over processors with homogeneous block cyclic distribution of data over the processes) on the networks on which we experimented. The main disadvantage of the HoHe strategy is non-Cartesian nature of the data distribution. This leads to additional communications that can be essential in the case of large networks. Moreover the non-Cartesian nature leads to nonscalability of the linear algebra algorithms scalable in case of Cartesian data distribution and networks provided parallel communications. Note that on Ethernet-like networks all these algorithms are not scalable even in the case of Cartesian data distributions.

The HeHo strategy is easy to accomplish. It allows us to utilize a lot of high-quality software, such as ScaLAPACK, developed for homogeneous distributed memory systems in heterogeneous environments and to obtain a good speedup with minimal expenses.

Implementation of the heterogeneous strategies was facilitated by the mpC language which supported an advanced approach to the problem of distribution of computations over processors in heterogeneous parallel environments. In particular, mpC provided means for rather easy and handy distribution of both processes involved in computations over processors and data over the processes taking into account their performances.

Profit of the heterogeneous strategies depends on ratios of the network equipment speed and processor performances. The faster network and the slower processors, the more profitable the heterogeneous strategies are. Since the trend is that the increase of network equipment speed surpasses the increase of processor performance, in the future, fast network expediency of the heterogeneous strategies will be warranted.

ACKNOWLEDGMENTS

We thank Jack Dongarra and Antoine Petitet for their valuable ideas and useful comments on preliminary versions of the paper.

REFERENCES

1. P. Alpatov, G. Baker, C. Edwards, J. Gunnels, G. Morrow, J. Overfelt, R. van de Geijn, and Y.-J. J. Wu, PLAPACK: Parallel linear algebra libraries design overview, *in* "Proc. of the SC97 Conference," ACM, San Diego, CA, 1997.
2. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, S. McKenney, S. Octrouchov, and D. Sorensen, "LAPACK Users' Guide," 2nd ed., SIAM, Philadelphia, 1995.
3. D. Arapov, A. Kalinov, A. Lastovetsky, I. Ledovskih, and T. Lewis, A programming environment for heterogeneous distributed memory machines, *in* "Proceedings of the Sixth Heterogeneous Computing Workshop (HCW'97)," pp. 32–45, IEEE Comput. Soc., Geneva, Switzerland, 1997.
4. O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert, "Data Allocation Strategies for Dense Linear Algebra Kernels on Heterogeneous Two-Dimensional Grids," Technical Report RR-99-31, LIP, ENS, Lyon, 1999.
5. L. S. Blackford, A. Cleary, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, A. Petitet, H. Ren, K. Stanley, and R. C. Whaley, "Practical Experience in the Dangers of Heterogeneous Computing UT," Technical Report CS-96-330, University of Tennesee, 1996.

6. J. Choi, J. J. Dongarra, S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley, "The Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines," Technical Report CS-94-246, University of Tennesee, 1994.
7. J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, A set of level 3 basic linear algebra subprograms, *ASM Trans. Math. Software* **16,** No. 1 (1990), 1–17.
8. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, "Solving Problems on Concurrent Processors," Vol. 1, Prentice–Hall, Englewood Cliffs, NJ, 1988.
9. B. Hendrickson and D. Womble, The torus-wrap mapping for dense matrix calculations on massively parallel computers, *SIAM SSC* **15**, No. 5 (1994).
10. A. Kalinov and A. Lastovetsky, mpC + ScaLAPACK = Efficient solving linear algebra problems on heterogeneous networks, *in* "Proceedings of the 5th International Euro-Par Conference," Lecture Notes in Computer Science, Vol. 1685, pp. 1024–1029, Springer-Verlag, Toulouse, 1999.
11. A. Kalinov and 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, Vol. 1593, pp. 191–200, Springer-Verlag, Amsterdam, 1999.
12. A. Lastovetsky, "The mpC Programming Language Specification," Technical Report, ISPRAS, Moscow, 1994.

ALEXEY YA. KALINOV is a senior researcher at the Institute for System Programming, Russian Academy of Sciences. His research interests are in parallel and distributed programming in heterogeneous environments, compilers, and computer modeling of human-steering vehicles. He received his MS in mathematics and engineering from the Moscow Aviation Institute in 1980 and his Ph.D. in engineering from the Institute for Research and Development of Tractors in 1990.

ALEXEY L. LASTOVETSKY is a leading researcher at the Institute for System Programming, Russian Academy of Sciences. His research interests include parallel and distributed programming, programming languages, compilers, and theory of programming languages. He received his MS in mathematics and engineering and Ph.D. in computer science from the Moscow Aviation Institute in 1980 and 1985, respectively, as well as his Doctor of Sciences in physics and mathematics from the Institute for System Programming in 1997. He has previously developed an algebraic approach to semantics of programming languages and an ANSI C superset for vector and superscalar computers. He teaches at the Moscow State University and at the Moscow Institute for Physics and Technology. He is on the editorial board of Programmirovanie (a journal of the Russian Academy Sciences on computer science also distributed as Programming and Computer Software by Plenum Publishing). He was on the advisory committee of the software track of HICSS'30 and HICSS'31 and on the program committee of PDPTA'99.