INFORMATICA, 1996, Vol. 7, No. 3, 281–294

LOAD BALANCING PROBLEM FOR PARALLEL COMPUTERS WITH DISTRIBUTED MEMORY

Raimondas ČIEGIS

Institute of Mathematics and Informatics
Akademijos 4, 2600 Vilnius, Lithuania
Email: raimondas.ciegis@fm.vtu.lt

Ramūnas ŠABLINSKAS, Juozas ŠIMKEVIČIUS

Vytautas Magnus University,
Vileikos 8, 3000 Kaunas, Lithuania

Jerzy WAŚNIEWSKI

The Danish Computer Center for Research and Education
UNI-C, DTH, Bldg. 305,
DK-2800 Lyngby, Denmark

Abstract. This paper deals with load balancing of parallel algorithms for distributed-memory computers. The parallel versions of BLAS subroutines for matrix-vector product and LU factorization are considered. Two task partitioning algorithms are investigated and speed-ups are calculated. The cases of homogeneous and heterogeneous collections of computers/processors are studied, and special partitioning algorithms for heterogeneous workstation clusters are presented.

Key words: parallel algorithms, load balancing, parallel virtual machine, distributed-memory computers.

1. Introduction. The load balancing is one of the most important problems for constructing efficient parallel algorithms (Golub and Van Loan, 1991; Freeeman and Phillips, 1991). For many problems we can assume that computation/communication ratio is sufficiently high and we expect to achieve linear speed-up on p processors (at least for sufficiently large problems). Parallel computation is load balanced when each processor has the same amount of useful calculation and communication during the whole computation time.

We consider local-memory computers when static task partitioning algorithm must be used because of redistribution of the workload during the computation is too expensive. Such algorithms are important in the case of parallel computers with slow communication in comparison with computation rate. This situation is typical for distributed computing on workstation clusters, for example by using Parallel Virtual Machine (PVM) software system (Geits *et al.*, 1993). In the latter case the load balancing problem is even harder as we must take into account the possibility of heterogeneous computer clusters.

Let us assume that the implementation of some numerical algorithm could be splitted into parts which are distributed among processors. Such work distribution in advance is called the *static scheduling*. We note that there are three different cases to be considered. In the first case the local subtasks have different computational complexity and we can not estimate this complexity a priori. Examples of such algorithms are numerical integration and the bisection iterative method for solving nonlinear equations (Ortega, 1988; Demmel *et al.*, 1993). In this case the load balancing of static task distribution could be very poor. Then the *pool-of-tasks* idea is used to improve the load balancing and the mixed approach of dynamic scheduling and static scheduling could be employed to reduce the amount of communication. We will not consider this case in our paper.

In the second case all subtasks have equal computational complexity. To illustrate our analysis we consider Level 2 BLAS subroutine *sgemv* which produces the matrix-vector product (in fact we are investigating a slightly simplified version of *sgemv*). Let us assume that A is an $m \times n$ matrix. We can express the matrix-vector product as m dot products

$$y_i = a_i \cdot X,$$

where a_i denotes the ith row of A. For parallel computers the rows of the matrix A are predistributed among all processors.

In the third case the computational complexity of subtasks is not equal but it could be estimated in advance. To illustrate the analysis we consider Level 3 BLAS subroutine *sgetrf* which computes the LU decomposition of an $n \times n$ matrix A. During the ith stage of Gaussian elimination we deal with an $(n-i) \times (n-i)$ submatrix of A and the amount of computations is reduced at each stage.

2. The load balancing of parallel implementation of *sgemv*. Suppose that we wish to construct a parallel implementation of *sgemv* using p processors. We assume that $p \leqslant m$, where m is the number of rows of A. Let us denote w_i the perfomance rate of ith processor/workstation and M_i the number of rows assigned to the ith processor. The computation of m dot products is distributed among p processors. In order to get the equitable distribution of the work we must solve the following optimization problem

$$\min_{M_i} \max_{1 \leqslant j \leqslant p} t_j = t^*, \quad t_j = \frac{M_j}{w_j}, \tag{1.a}$$

subject to the constraints

$$M_1 + M_2 + \dots + M_p = m, \quad M_j \geqslant 0. \tag{1.b}$$

2.1. The case of homogeneous processors. First of all we consider the case of homogeneous workstation/processor cluster when perfomance rates of all processors are equal, so that we have

$$w_j = 1, \quad i = 1, 2, \dots, p.$$

Then the solution of (1) is well-known for real-valued M_i and it is defined as

$$M_1 = M_2 = \ldots = M_p = \frac{m}{p}.$$

The case in which m is not divisable by p is also handled simply. Let $r = \lfloor m/p \rfloor$, where $\lfloor z \rfloor$ denotes the greatest integer less than or equal to z, and let $q = m \bmod p$ be the reminder of the integer division of m by p. Then we allocate

$$M_i = r + 1, \quad i = 1, 2, \dots, q$$

rows of A to the first q processors and

$$M_i = r, \quad i = q + 1, q + 2, \dots, p$$

rows to the remaining $p - q$ processors. Notice that for the load-balancing of the *sgemv* operation it is not important which rows are assigned to each processor.

Next we give efficiency estimates of the obtained parallel implementation of *sgemv*. The communication cost is not included into the analysis. These results are well-known but we are giving a brief derivation for the self-completeness of our paper (see Golub and Van Loan, 1993; Demmel *et al.,* 1993).

The *efficiency* of a p-processor parallel algorithm is given by (Golub and Van Loan, 1993)

$$E_p = \frac{T(1)}{pT(p)},$$

where $T(k)$ is the time required to execute the program on k processors. A concept related to efficiency is *speed-up* S_p, which is defined as

$$S_p = \frac{T(1)}{T(p)}.$$

For some problems the fastest sequential algorithms do not parallelize and so a distinct parallel algorithm is used and the speed-up is defined as

$$\tilde{S}_p = \frac{T_{seq}}{T(p)} = \frac{T_{seq}}{T(1)} S_p.$$

The speed-up estimate from below for the *sgemv* algorithm is given by

$$S_p \geqslant \frac{m}{r+1} = \frac{rp+q}{r+1} \geqslant \frac{rp+1}{r+1} = p\left(1 - \frac{p-1}{p(r+1)}\right). \qquad (2)$$

This formula can also be obtained by noticing that during r stages all p processors are utilized and during the last stage only q processors are utilized and $p-q$ processors are idle. The worst case is when $r = 1$, i.e., $m = p+1$, then we have

$$S_p = \frac{p+1}{2},$$

and the efficiency of the parallel algorithm is equal to 0.5. This load imbalance becomes smaller with increasing r (see Table 1, where the efficiency E_p values are presented for $p = 10$ and different values of r).

Table 1. Effenciency E_{10} of the parallel *sgemv* version.

r	1	2	4	8	17
E_{10}	0.55	0.7	0.82	0.9	0.95

REMARK 1. We obtain the result of *sgemv* operation after two stages when $m = p+1$. The same speed-up can be achieved using only $p_1 = \lfloor (m+1)/2 \rfloor$ processors.

The load balance will be improved if we subdivide the last row among s processors, calculate partial dot products independently, send them to one of the processors and compute the required row-vector product. In practice the efficiency of this modification depends on the communication speed of the network used. If we do not take into account the communication time then the optimal number of processors s is obtained by solving the minimization problem

$$\min\left(\frac{2n}{s}+s-1\right)=R(s^*).$$

Then it follows from the necessary minimization condition that

$$-\frac{2n}{s^2}+1=0,$$

and we get the optimal number of processors $s^*=\sqrt{2n}$.

2.2. The case of heterogeneous workstation cluster. In this section we consider the case of heterogeneous workstation cluster. Let's assume that performance rates of processors satisfy the following inequalities

$$w_1\leqslant w_2\leqslant\ldots\leqslant w_p.$$

Then we get the real-valued solution of the minimization problem (1)

$$M_j=\frac{mw_j}{W_p},\quad j=1,2,\ldots,p,$$

where W_p is the total performance rate of parallel computer (or virtual parallel computer)

$$W_p=w_1+w_2+\cdots+w_p.$$

We assign to the jth proccesor the following number of rows

$$M_j=\left\lfloor\frac{mw_j}{W_p}\right\rfloor,\quad j=1,2,\ldots,p. \tag{3}$$

Then we only need to distribute the remaining q rows, where

$$q=m-(M_1+M_2+\cdots+M_p)\leqslant p-1.$$

The algorithm for solving this problem is given in Fig. 1.

```
For $j=1$ to $p$
    $t_j = \dfrac{M_j+1}{w_j}$
For $i=1$ to $q$
    $t_k = \min_{1\leqslant j\leqslant p}\{t_j\}$
    $M_k := M_k+1$
    $t_k = \dfrac{M_k+1}{w_k}$
```

Fig. 1. The distribution of remaining q rows of A.

The idea of the algorithm is to assign an additional row to the processor which will finish this new work first. This process is repeated q times.

EXAMPLE 1. Suppose we have $p = 4$ processors and $m = 130$ rows. Let's assume that the relative perfomance rates of processors are the following

$$w_1 = 0.129, \quad w_2 = 0.202, \quad w_3 = 0.349, \quad w_4 = 0.620.$$

In the first stage of the algorithm we assign the number of rows given by (3) to each processor

$$M_1 = 12, \quad M_2 = 20, \quad M_3 = 34, \quad M_4 = 62.$$

The remainder is equal to $q = 2$. It is interesting to note that the ith processor will finish its part of work in time $\tilde{t}_i$, where

$$\tilde{t}_1 = 93, \quad \tilde{t}_2 = 99, \quad \tilde{t}_3 = 97, \quad \tilde{t}_4 = 100.$$

Additional row added to M_j would result in the following computation times for each jth processor

$$t_1 = 100.8\,, \quad t_2 = 104.0\,, \quad t_3 = 100.3\,, \quad t_4 = 101.6\,.$$

Hence, the additional row should be assigned to the third processor and we will have $M_3 = 35$. If the second additional row is also added to the third processor it will be busy for $t_3 = 103.6$. So we see, that this row must be assigned to the first processor and finally we have the following numbers of rows assigned to processors

$$M_1 = 13, \quad M_2 = 20, \quad M_3 = 35, \quad M_4 = 62.$$

The *sgemv* operation will be finished in time $t = 100.8$.

3. The load balancing of parallel implementation of *sgetrf*. Now we consider the parrallel implementation of LU factorization subroutine *sgetrf*. There should be taken into account two important details about the algorithm, which implementation is given in Fig. 2.

```
For $i = 1$ to $n-1$
  For $j = i+1$ to $n$
    $l_{ji} = a_{ji}/a_{ii}$
    For $k = j$ to $n$
      $a_{jk} := a_{jk} - l_{ji}a_{ik}$
```

Fig. 2. The ijk form of LU decomposition.

We can see that at the ith stage of elimination the ith row of A is used to modify all the remaining $n - i$ rows of the submatrix. This means that the number of modified rows is reduced by one at each stage. At the same time we are dealing with an $(n - i) \times (n - i)$ submatrix of A and computation involves $O((n - i)^2)$ operations. We can see that computation complexity of the ith subtask is also reduced.

The load balancing of parallel implementation of *sgetrf* depends on appropriate data distribution. We will consider two popular distributions of the matrix A (Golub and van Loan, 1993; Ortega, 1988)

- by blocks of rows,
- scattered row distribution.

We will investigate the efficiency of these distribution algorithms for homogeneous workstation clusters and will give modifications of the algorithms for heterogeneous clusters. In order to prove more general results we will also investigate the problem for which the number of subtasks is reduced by one at each stage (like in *sgetrf* operation) but the computational complexity of each subtask remains constant.

For convenience we assume that the number of rows n is divisible by p, i.e., $n = pr$.

3.1. Homogeneous cluster. Block distribution. Subtasks of constant complexity. In this section we assume that performance rates of all processors are equal. In a block row distribution consecutive rows of A are grouped into blocks and allocated to different processors. Then ith processor has rows $(i-1)r+1, (i-1)r+2, \dots, (i-1)r+r$. As it follows from the algorithm given in Fig. 2, if a block row distribution is employed then we will have a situation in which the first processor becomes idle after r stages, the second processor becomes idle after $2r$ stages, and so on. We will estimate the efficiency of such distribution. In this section we will assume that computational complexity of all subtasks is constant. Let the time unit be equal to the time used to modify one row by one processor.

Then the time required to execute the program on one processor is given by

$$T(1) = \sum_{i=1}^{n-1}(n - i) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}. \tag{4}$$

On a p processor computer the execution of stages $1, 2, \dots n - r$ requires r time units. Starting from the $(n - r + 1)$th stage only one processor continues calculations and the time estimate (4) can be used. In detail, we have

$$\begin{aligned} T(p) =& \sum_{i=1}^{n-r} r + \sum_{i=n-r+1}^{n-1} (n-i) \\ =& (n-r)r + \frac{r(r-1)}{2} = \frac{r(2n-r-1)}{2}. \end{aligned}$$

Then the speed-up of the parallel algorithm with a block row distribution is given by

$$S_p = \frac{T(1)}{T(p)} = \frac{n(n-1)}{r(2n-r-1)} = p\frac{n-1}{\left(2-\frac{1}{p}\right)n-1}.$$

If p is fixed and we choose n sufficiently large then

$$S_p \to \frac{p}{2-\frac{1}{p}} \approx \frac{p}{2}.$$

We can see that the efficiency of this distribution algorithm is only 0.5. The same result is also valid in the case when r is fixed and n is sufficiently large.

3.2. Homogeneous cluster. Block distribution. LU decomposition. In this section we will modify the estimates obtained in Sect. 3.1. For the case of LU decomposition we must take into account that the time t_i required to modify one row at ith stage is given by

$$t_i = \frac{(n+1-i)}{n}.$$

Then we have that the program execution on one processor requires the time

$$T(1) = \frac{1}{n}\sum_{i=1}^{n-1}(n-i)(n+1-i) = \frac{(n-1)(n+1)}{3}. \tag{5}$$

For p processors we obtain

$$\begin{aligned} T(p) =& \frac{r}{n-r}\sum_{i=1}^{n-i}(n+1-i) + \frac{1}{n}\sum_{i=n-r+1}^{n-1}(n-i)(n+1-i) \\ =& \frac{r}{2n}((n-r)(n+r+1) + \frac{2}{3}(r^2-1)). \end{aligned}$$

Hence, the speed-up of the parallel *sgetrf* algorithm with a block row distribution is given by

$$S_p = \frac{2p}{3} \frac{n^2 - 1}{(n-r)(n+r+1) + \frac{2}{3}(r^2 - 1)}.$$

If p is fixed and we choose n sufficiently large then

$$S_p \longrightarrow \frac{2p}{3 - \frac{1}{p^2}} \approx \frac{2p}{3}.$$

We can see that the efficiency of the *sgetrf* operation is 2/3. It is 4/3 times larger then the efficiency of the algorithm from Sect. 3.1.

3.3. Homogeneous cluster. Scattered row decomposition. In general scattered row distribution a block of consecutive p rows is allocated to different p processors. For example, if a wraparound approach is adopted then row k is assigned to processor $(k-1) \bmod p$. There, for conveniece, we have numbered processors $0, 1, \ldots, p-1$. For the scattered row distribution the difference between numbers of rows allocated to different processors remains not greater than one during all the stages of computations. Hence we have a satisfactory load balance and processors begin to become idle only after $n-p$ stages.

First we assume that computational complexity of all subtasks is constant. It is easy to see that each of the first $p-1$ stages will be accomplished in r time units, each of the next p stages will finish after $r-1$ time units, and so on. Hence the time required to execute the program on p processors is given by

$$T(p) = (p-1)r + \sum_{i=1}^{r-1} p(r-i) = \frac{n(r+1)}{2} - r.$$

The time $T(1)$ is calculated as in Sect. 3.1 (see (4)). Then the speed-up of the parallel algorithm with a scattered row distribution is given by

$$S_p = \frac{n(n-1)}{n(r+1) - 2r} = \frac{n-1}{r+1-\frac{2}{p}} = p\frac{r-\frac{1}{p}}{r+1-\frac{2}{p}}.$$

We observe that if p is fixed and we choose n sufficiently large then $S_p \longrightarrow p$. In Table 2 the efficiency values E_p are presented for $p = 10$ and increasing values of r.

Table 2. Effenciency E_{10} for a scattered row distribution

r	1	2	4	10	20
E_{10}	0.5	0.69	0.81	0.92	0.96

Similarly we can estimate the efficiency of the parallel *sgetrf* operation with a scattered row distribution. Execution time for one processor $T(1)$ is given in Sect. 3.2 by formula (5). For p processors we obtain

$$T(p) = \frac{r}{n} \sum_{i=n+2-p}^{n} i + \frac{1}{n} \sum_{j=1}^{r-1} (r-j) \left(\sum_{i=n+2-(j+1)p}^{n+1-jp} i \right).$$

In Table 3 we present efficiency values E_p for $p = 10$ and increasing values of r.

Table 3. Effenciency E_{10} for *sgetrf* with a scattered row distribution

r	1	2	4	10	20
E_{10}	0.61	0.75	0.86	0.94	0.97

4. Heterogeneous cluster. LU decomposition. Let's assume that processors are numbered in order of increasing perfomance rates, i.e.,

$$w_1 \leqslant w_2 \leqslant \ldots \leqslant w_p.$$

In the case of a block row distribution we use the algorithm proposed in Sect. 2.2. for *sgemv* operation. In order to illustrate the efficiency of such distribution method we will investigate the example of two processors with the performance rates $w_1 = 1, w_2 = s$. Then we assign to each processor the following number of rows

$$M_1 = \frac{n}{1+s}, \qquad M_2 = \frac{ns}{1+s}.$$

Suppose, for convenience, that M_1 and M_2 are integer numbers. The time required to execute *sgetrf* operation on one processor is given by formula (5).

For $p = 2$ processors we obtain

$$T(2) = \sum_{i=1}^{M_1} \frac{(n+1-i)M_2}{ns} + \sum_{i=M_1+1}^{n-1} \frac{(n+1-i)(n-1)}{ns}$$
$$= \frac{n^2(2s^2+6s+3)}{6(1+s)^3} + \frac{n}{2(1+s)^2} - \frac{1}{3(1+s)}.$$

Hence, the speed-up of the parallel *sgetrf* algorithm is given by

$$S_2 = (1+s)\frac{2(n^2-1)(1+s)^2}{n^2(2s^2+6s+3)+3n(1+s)-2(1+s)^2}.$$

If we choose n sufficiently large then

$$S_2 \to (1+s)\frac{2s^2+4s+2}{2s^2+6s+3}.$$

In Table 4 the efficiency values E_2 are presented for $n = 10, 100$ and increasing values of s.

Table 4. Effenciency E_2 for heterogeneous computer with a block row distribution

s	0.5	1.0	1.5	2	4
$E_2, n = 10$	0.645	0.688	0.723	0.751	0.825
$E_2, n = 100$	0.688	0.723	0.754	0.780	0.845

We conclude that processors numbering order is important for the efficiency of distribution algorithm.

Next we consider the scattered row distribution algorithm for heterogeneous workstation cluster. In this case we must define not only the total number of rows assigned to each processor but also specify which rows are assigned. The algorithm for solving the load balancing problem (1) is given in Fig. 3.

EXAMPLE 2. Suppose we have $p = 6$ processors and $n = 100$ rows. Let's assume that the relative performance rates of the processors are

$$w_1 = 1,\ w_2 = 1.5,\ w_3 = 2.5,\ w_4 = 3.11,\ w_5 = 3.6,\ w_6 = 4.3.$$

```
For $j = 1$ to $p$
    $M_j = 0$
    $t_j = \frac{1}{w_j}$
For $i = n$ to $1$
    $t_k = l \min_{1 \leqslant j \leqslant p} \{t_j\}$
    $M_k := M_k + 1$
    $i$th row is assigned to $k$th processor
    $t_k = \frac{M_k + 1}{w_k}$
```

Fig. 3. The scattered row distribution algorithm for heterogeneous workstation cluster.

After scattered row distribution the following rows are assigned to each of processors

$$
\begin{aligned}
R_1 =& \{9,\ 24,\ 40,\ 56,\ 72,\ 87\},\\
R_2 =& \{8,\ 18,\ 29,\ 39,\ 49,\ 61,\ 71,\ 82,\ 93\},\\
R_3 =& \{1,\ 7,\ 13,\ 20,\ 27,\ 34,\ 38,\ 46,\ 53,\ 58,\ 65,\ 70,\\
& 79,\ 84,\ 91,\ 97\},\\
R_4 =& \{4,\ 12,\ 16,\ 21,\ 26,\ 31,\ 35,\ 43,\ 47,\ 52,\ 57,\ 62,\ 67,\\
& 74,\ 78,\ 83,\ 88,\ 94,\ 98\},\\
R_5 =& \{2,\ 5,\ 15,\ 19,\ 23,\ 28,\ 32,\ 37,\ 42,\ 45,\ 50,\ 54,\ 60,\ 64,\\
& 68,\ 73,\ 76,\ 81,\ 86,\ 90,\ 95,\ 99\},\\
R_6 =& \{3,\ 6,\ 11,\ 14,\ 17,\ 22,\ 25,\ 30,\ 33,\ 36,\ 41,\ 44,\ 48,\ 51,\\
& 55,\ 63,\ 66,\ 69.\ 75,\ 80, 89,\ 96,\ 100\}.
\end{aligned}
$$

Let's compare different distribution algorithms. The total performance rate of the given workstation cluster is $W_6 = 16$. Suppose that the time required to execute *sgetrf* operation with $n = 100$ rows on the first processor is $T_1 = 160$ time units. Let's denote $T_{bl,d}, T_{bl,i}, T_{sc}$ the times required to execute the same program on $p = 6$ processors for a block row distribution with processors numbered in order of decreasing performance rates, for a block row distribution with reversed processor numbering order, and for the scattered row distribution, respectively. Then the following execution time and speed-up results for our model problem are obtained

$$T_{bl,d} = 16.93\,, \quad T_{bl,i} = 14.73\,, \quad T_{sc} = 10.38\,,$$
$$S_{bl,d} = 9.45\,, \quad S_{bl,i} = 10.86\,, \quad S_{sc} = 15.42\,.$$

REFERENCES

Demmel, J.W., M.T. Heath and H.A. van der Vorst (1993). Parallel numerical linear algebra. *Acta Numerica,* **7**, 111–197.

Freeman, T.L., and C. Phillips (1991). *Parallel Numerical Algorithms.* Prentice Hall. 237pp.

Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam (1993). *PVM 3.0 User's Guide and Reference Manual.* Tenessee. 278pp.

Golub, G.H., and Ch. F. Van Loan (1991). *Matrix Computations.* The Johns Hopkins University Press, Baltmore and London. 636pp.

Ortega, J.M. (1988). *Introduction to Parallel and Vector Solution of Linear Systems.* Plenum Press, New York and London.

Received May 1996

R. Čiegis has graduated from the Vilnius University (Faculty of Mathematics) in 1982, received the Degree of Candidate of Physical and Mathematical Sciences from the Institute of Mathematics of Byelorussian Academy of Sciences in 1985 and the Degree of Habil. Doctor of Mathematics from the Institute of Mathematics and Informatics, Vilnius in 1993. He is a senior researcher at the Numerical Analysis Department, Institute of Mathematics and Informatics. R. Čiegis is also a Professor at the Kaunas Vytautas Magnus University and a Professor and a head of Mathematical Modelling Department of Vilnius Technical University. His research interests include numerical methods for nonlinear PDE, parallel numerical methods and numerical modelling in physics, biophysics, ecology.

R. Šablinskas was born in 1971. After having received his master's degree in VMU he has been admited as an engineer in telecommunications company Omnitel. In 1995 he has been admited as a visit graduate student in Kaunas Vytautas Magnus University. His research interest covers distributed and parallel computing, optimization, neutral network models.

J. Šimkevičius is a researcher at the Department of Mathematics and Statistics of Vytautas Magnus University, Kaunas, Lithuania. Scientific interest include parallel computing, wavelet transform, digital signal processing.

J. Waśniewski is a senior researcher at the Danish Computer Center for Research and Education. He has the Degree of Doctor of Mathematics. His scientific interests include parallel computing, mathematical modelling in ecology.

LYGIAGREČIŲ KOMPIUTERIŲ SU PASKIRSTYTA ATMINTIMI TOLYGAUS APKROVIMO KLAUSIMU

Raimondas ČIEGIS, Ramūnas ŠABLINSKAS,
Juozas ŠIMKEVIČIUS, Jerzzy WAŚNIEWSKI

Šis darbas skirtas lygiagrečių algoritmų, realizuojamų kompiuteriais su paskirstyta atmintimi, tolygaus apkrovimo analizei. Išnagrinėti BLAS bibliotekos paprogramių *sgemv* ir *sgetrf* lygiagretūs variantai. Analizuojami du duomenų paskirstymo algoritmai, t.y. blokinis ir ciklinis paskirstymas, ir apskaičiuoti jų pagreitėjimo koeficientai. Pateikti šių duomenų paskirstymo algoritmų apibendrinimai heterogeniškiems lygiagretiesiems kompiuteriams. Teorinė analizė iliustruojama skaitiniais rezultatais.