INFORMATICA, 2003, Vol. 14, No. 2, 167–180

Realistic Performance Prediction Tool for the Parallel Block LU Factorization Algorithm

Raimondas ČIEGIS, Vadimas STARIKOVIČIUS
Vilnius Gediminas Technical University
Saulėtekio al. 11, LT-2040 Vilnius, Lithuania
e-mail: rc@fm.vtu.lt, vs@sc.vtu.lt

Received: March 2003

Abstract. This work describes a realistic performance prediction tool for the parallel block LU factorization algorithm. It takes into account the computational workload, communication costs and the overlapping of communications by useful computations. Estimation of the tool parameters and benchmarking are also discussed. Using this tool we develop a simple heuristic for scheduling LU factorization tasks. Results of numerical experiments are presented.

Key words: performance prediction tools, parallel algorithms, LU factorization, heuristic for scheduling the tasks.

1. Introduction

Parallel computing is a powerful tool for the solution of the most complex real world problems. The development of parallel algorithms and codes is in great demand and becomes more and more popular. Recently, in order to assist in the creation of efficient parallel algorithms, a lot of attention are gaining such research topics as performance evaluation and prediction (Pease, 1991), benchmarking (Hockney, 1991), automatic or semi-automatic parallelization (Baravykaitė and Šablinskas, 2002; Baravykaitė *et al.*, 2002; Kennedy *et al.*, 1992), scheduling of tasks of parallel algorithms (Amoura *et al.*, 1998).

Simple and accurate performance prediction can greatly contribute to the parallel computing. Performance prediction tools are often used in parallel programming (see, Fahringer, 1995). Using such tools, parallel developers can omit costly coding and debugging stages and estimate the efficiency of parallel algorithms, ruling out inefficient ones. Kennedy *et al.* (1992) states that the accurate and reliable performance estimations are the most important features of any automatic or semi-automatic parallelization tool for scientific programs on parallel systems.

Finally, in order to use the parallel computers efficiently and not to waste the computational resources, we need simple in use and accurate performance prediction tools, which can assess the performance of the parallel codes before the real computations are done.

The approaches to performance evaluation are reviewed by Bhuyan and Zhang (1995). Although all *Massively Parallel Processor* (MPP) systems come up with some performance tools, virtually all of them are run-time supported for performance evaluation (see, Simons *et al.*, 1995). We also note recent joint issues of *IEEE Computer* (IEEE, 1995a) and *IEEE Parallel and Distributed Technology* (IEEE, 1995b) which are dedicated to parallel performance tools.

However, most of existing performance prediction tools lack the accuracy. This is mainly due to oversimplifications in characterizing the workload and communication overheads. Usually such prediction tools assume that every computational operation takes the same time, what is very questionable on modern processors with complex memory hierarchies (e.g., RISC processors). Another shortcoming is the estimation of collective communication costs as a sum of point-to-point operations costs. Most performance prediction tools are using theoretical models like the Phase Parallel Model (Xu and Hwang, 1996a), which are easier to implement and quite efficient for SPMD data-parallel programs. But such models can not estimate the overlapping of communications by useful computations.

In Section 2, we propose a performance prediction model as a set of rules and guidelines in order to provide a systematic approach to creation of performance prediction tool for specific parallel algorithm. We discuss consecutively how to characterize accurately computational workload, communication costs, and overlapping of communications by useful computations. We also discuss the issues of benchmarking and parameters evaluation.

As example, a performance prediction tool for parallel LU factorization algorithm is created. LU factorization was chosen because it and it's parallel block algorithm are well known (see, e.g., Golub and Van Loan, 1991). The structure of this algorithm is such that the accurate estimation of communication costs and accounting for overlapping of communications by useful computations are necessary features of the tool. Also, it is well known that linear algebra is the main computational part in most scientific computer applications. And finally, LU factorization is a part of well-known LINPACK and NASA benchmarks. The created tool and prediction results are presented in Section 2.

In Section 3, using the developed performance prediction tool and results of the theoretical analysis of parallel LU factorization algorithm, we propose a new heuristic for the scheduling of tasks in the parallel LU factorization algorithm. Performance results obtained with the new heuristic are presented and compared with well known and usually used cyclic and block partitioning schemes. Some concluding comments and remarks are made in Section 4.

2. Performance Prediction Model for Parallel Algorithms

2.1. *Computational Workload*

In any performance prediction tool, first of all, we need to estimate the amount of work performed by each processor in application (referred as *computational workload*). In order to do that, some kind of metric and corresponding speed parameters are defined.

Metric. For scientific computing and signal processing applications where numerical calculation dominates, a natural metric is the number of *floating point operations* (flop) that need to be executed. This number should be determined by the inspection of the algorithm or programming code.

However, it cannot be assumed that every floating point operation takes the same amount of time. This uniform speed assumption does not hold on modern processors with complex memory hierarchies. For instance on RISC processor of SP2 the computational speed can differ forty times (Xu and Hwang, 1996a). Using this simple approach the accuracy of the prediction tool relies completely on the determination of one or several (for the most work-heavy parts of code) speed parameters. The origin of the problem is an attempt to avoid the detailed characterization of the memory hierarchy of the computer node. The alternatives are to try to estimate the cache misses (see, Fahringer, 1995), or to perform a detailed simulation of the specific processor architecture. For example, the tool PERFORM developed by Hey *et al.* (1997) provides "rapid simulation" of sequential program performance for RISC processors with complex memory hierarchies. It uses simulation methods at some intermediate level of abstraction with benchmarking of function calls, data movement operations(load/store) to produce accurate performance estimates in acceptable time.

Computing Speed Parameter γ. Benchmarking. Usually we do not have a possibility to make such a simulation of processor architecture. Then we need to provide for our tool the best possible computing speed parameter(s). Let denote the time of one flop by γ. It is obvious that an attempt to use vendor-supplied, LINPACK, NASA or other benchmarks for evaluation of speed parameter can lead to large accuracy errors in the predictions of such tool.

So, we need appropriate benchmarks, which include the same cache effects. The best possible solution is to use as benchmarks the main computational sequential parts of parallel codes or "close" counterparts of them in order to create similar use of caches and other memory layers. For example, in our performance prediction tool for parallel LU factorization algorithm, we use as a benchmark a sequential version this algorithm with matrix sizes equal to ones obtained after matrix distribution among processors.

In Table 1 we present the results of tests (benchmarking) for sequential LU factorization algorithm on three computer nodes with different processor architecture:

- RS – IBM RS/6000 43P-140 workstation, PowerPC 604e at 200 Mhz, 96 MB, peak 390 Mflop/s (vendor supplied),
- SP2 node – IBM RS/6000 SP2 thin node, Power2 at 120 Mhz, 128 MB, peak 480 Mflop/s (vendor supplied),
- PM – Power Mac workstation, dual Motorolla G4 processor at 450Mhz, 512 MB, peak 450 Mflop/s (single precision 660, double 1600) (vendor supplied).

Our sequential and parallel LU factorization algorithms are block algorithms (see, Choi *et al.*, 1995b). The most part of computations is performed in the multiple block matrix multiplications. This allows us to use subroutines for matrix multiplication from

Table 1
Block LU factorization algorithm. Speed benchmarking.

System (Library used)	N	r	Time	Speed
RS	3200	3200	2606,5	8,38
RS	3200	40	292,4	74,69
RS (ATLAS)	3200	50	175,9	124,2
SP2	3200	3200	217,1	100,64
SP2	3200	80	153,3	142,49
SP2 (ATLAS)	3200	80	57,47	380,72
PM	3200	3200	549,9	39,71
PM	3200	50	150,2	145,43
PM (ABSOFT)	3200	50	208,9	104,52
PM (ATLAS)	3200	40	51,9	420,98
PM	6400	50	1209,9	144,2
PM (ABSOFT)	6400	50	1687,9	103,52
PM (ATLAS)	6400	40	418,4	417,64
Single precision				
PM	6400	80	1091,6	160,1
PM (ABSOFT)	6400	80	309,6	564,49
PM (ATLAS)	6400	80	329,9	529,60

the linear algebra libraries, which are available on corresponding system. We have used our own implementation of the matrix multiplication and also the well-known and popular ATLAS library and ABSOFT library on computer node from PM cluster. ATLAS library automatically tunes to processor architecture and is very efficient. Among interesting features of ABSOFT library we mention its support of vector instructions on Motorolla G4 processor for single precision numbers.

In Table 1, we present the run times in seconds and the speed of computations given in the usual for benchmarks (Hockney, 1991) unit Mflop/s (*Millions of flops per second*), which is inverse of γ. On each computer node, we have computed the tests first for non-block version of the algorithm, when the block size r is equal to the matrix size N, and then for block versions with the optimal block sizes.

As was expected, even on the same system the values of speed parameter can differ up to 15 times. Hence, our first conclusion is that the speed parameter for the performance prediction tool should be evaluated running the same code version and block size as in parallel code. Then the main cache effects are included into our model.

2.2. *Communication Costs*

Next, we need to estimate communication costs. This is a very important feature of parallel performance prediction tools, which is not needed in sequential tools. Numerous

metrics and benchmarks have been proposed for various parallel systems from Massively Parallel Processors (MPPs) to workstation clusters (see, Fahringer, 1995; Hockney, 1991; Xu and Hwang, 1996b).

A simple traditional method for quantifying communication costs is to obtain a cost of communication operation as a sum of performed point-to-point operations costs. The main problem is that the accuracy can be reduced significantly due to the lack of attention to the topologies of interconnection networks on MPPs and dedicated clusters. Another shortcoming is the use of vendor-supplied latency and bandwidth values.

In our performance prediction tool, we use an expression suggested by Hockney (1991) to estimate the cost of broadcasting n items of data along a row or column of the mesh of processors by

$$B(n,p) = K(p)(\alpha + \beta n), \tag{1}$$

where α denotes the latency (the start-up time or the time needed to send a 0 byte message), β denotes the time for transmission of an elementary data (in our case 1 Byte) (the inverse of the bandwidth defined in the communication benchmarks), p is the number of processors in a row/column, and $K(p)$ is the cost function, which depends on the topology of interconnection networks. For completely connected network we use the cost function $K(p) = 1$, for hypercube network – $K(p) = \log p$, and for the LAN type network – $K(p) = p - 1$.

In order to determine communication parameters α and β, we use our own ping-pong style benchmarks. Here the important factor is to use in the tests data packets of the same size as in the sent/received operations of the parallel application. We obtain parameters α and β using least-square fitting of the measured timing data. Several sets of α and β can be used for different ranges of message sizes if needed.

In Table 2, we present the results of our communication benchmarks on different parallel systems from previous section.

Table 2

Communication systems benchmarking

Parallel System	α	β	Bandwidth
RS Cluster (Ethernet Hub, 1MB/s)	1000 μs	8 μs	0,95 MB/s
SP2 (HP Switch, US, 110 MB/s)	35 μs	0,11 μs	70 MB/s
PM cluster (Baystack switch, 10 MB/s)	78 μs	0,8 μs	9,5 MB/s

2.3. *Performance Simulation*

The main goal of our performance prediction tool is to characterize accurately the overlapping of communications by useful computations. This is a key feature of the prediction

tool for many parallel algorithms, without which even very accurate predictions of computational workload and communication costs will not produce good final results. Parallel LU factorization algorithm is exactly of that kind. Therefore, we chose it as an example.

Most of the other performance prediction tools are lacking this property. Usually, abstract theoretical models like the Phase Parallel Model (Xu and Hwang 1996a) are proposed to cover important parallel programming paradigms (see, Brinch Hansen 1995). Such tools are quite efficient for SPMD data-parallel programs and easier to implement.

In order to estimate the overlapping of communications by useful computations, our tool simulates the performance of each processor, which participates in parallel run. The tool has the same structure as the parallel algorithm. Instead of real computations and communications, it uses above-defined metrics and parameters γ, α, β. Such a simulation model enables us to make accurate performance predictions not only on homogeneous processors systems, but also on heterogeneous systems with different γ, α, β. It allows us also to use more sophisticated models for the estimation of communication overhead. They can take into account the order of communications between processors in collective operations (e.g., broadcast, scatter). We also can simulate asynchronous non-blocking and other communications.

A time complexity of such simulation model is $O(np)$. When the performance prediction tool is used $O(n)$ times, as it will be done in the next section, the running time for large n and p can be quite significant. The solution of this drawback can be a modification of the simulation model. For parallel LU factorization algorithm it is quite clear that there is no need to simulate each processor from the start to the finish. At the beginning all processors can be divided into groups with "identical" computation times. Thus we need to simulate only one member of a group and to control the number and the composition of each group. However, in the worst case we again can reach a situation when the number of groups is equal to the number of processors.

3. Performance Prediction Tool for the Parallel LU Factorization Algorithm

Following the rules and guidelines defined above, we have created a performance prediction tool for parallel LU factorization algorithm. To explain the ideas of performance simulation, we present below a simplified scheme of our tool for the parallel block LU factorization algorithm with 1D block data distribution among processors. Here we assume that communications are asynchronous.

Performance Prediction Tool

$T_i = 0,\ i = 1, 2, \ldots, p$ // `We use timers for each processor`
do $k = 1,\ M$ // $N = M \times r$
// `Processor with` k`-th pivot column performs`
`LU factorization of block` $A_{kk} = L_{kk}U_{kk}$
$T_{master} = T_{master} + \dfrac{2r(r^2-1)}{3}\,\gamma_{master}$

// Master computes k-th column of L matrix:
// $L_{ik} = A_{ik}U_{kk}^{-1}, \quad i = k+1, \ldots, M$
$T_{master} = T_{master} + (M-k)r^2\gamma_{master}$
$M_{master} = M_{master} - 1$ // The number of local columns

// Master broadcasts the pivot column to
// the other processors
$T_{master} = T_{master} + K(p)\Big(\alpha + \big((M-k)r^2 + \frac{r(r-1)}{2}\big)\beta\Big)$

// Slave processors receive the pivot column
do $i = 1, p, \ i \neq master$
  if $T_i < T_{master}$ **then** $T_i = T_{master}$
end do

// All processors compute U_{kj} blocks: $U_{kj} = L_{kk}^{-1}A_{kj}$
// and update their part of matrix A
do $i = 1, p$
  $T_i = T_i + M_i\left(r^2 + 2(M-k)r^2\right)$
end do
end do
$T = T_{master}$ // the predicted run time

In order to evaluate the effectiveness of this tool, we made numerical experiments. The parallel algorithm was executed on a cluster of RISC workstations connected with LAN, which was described in Sections 2.2 and 2.3. We collect here the parameters of parallel system, which were obtained from our benchmarks and used in the tool:

$$\alpha = 1000\,\mu s, \quad \beta = 8\,\mu s, \quad \gamma = 0,013\,\mu s.$$

In Table 3, we give the total CPU times T spent during realization of parallel LU factorization algorithm and predicted times T_{pr} obtained with the help of our prediction tool. Two series of numerical experiments were done with matrix sizes $N = 2400$ and $N = 3000$. Here p denotes the number of processors. The optimal block size $r = 40$ was chosen after preliminary computations. Matrix was distributed among processors using standard 1D cyclic partitioning scheme (Choi *et al.*, 1995a). As we can see, the predicted computation times are very close to real CPU times.

Remark 1. We get from Table 3 that the optimal number of processors is $p = 2$ for a smaller problem with $N = 2400$ and $p = 3$ for the larger problem with $N = 3000$. These results show that in many cases it is not optimal to use a maximum number of processors.

4. Task Scheduling for Parallel LU Factorization Algorithm

Scheduling of the tasks of weighted directed acyclic graphs is one of the most important problems in parallel computing. To schedule simply means to allocate a set of tasks

Table 3
Performance of the prediction tool for LU factorization algorithm

p	T N = 2400	T_{pr} N = 2400	T N = 3000	T_{pr} N = 3000
1	119.1	119.8	235	234
2	86.0	84.2	157	155
3	88.7	87.7	152	152
4	99.7	100.9	166	169
5	117.0	118.1	194	194
6	134.0	137.0	221	222

or jobs to resources in such a way that the optimum performance is obtained. Finding a scheduling algorithm that minimizes the parallel execution time is an NP-complete problem. The main research efforts in this area are focused on heuristic methods for obtaining near-optimal solutions in a reasonable time (see, e.g., Amoura *et al.*, 1998; Djordjević and Tošić, 1996). Recently an alternative approach, based on the genetic algorithms, to efficiently solve the scheduling problem becomes popular (see, Zomaya *et al.*, 1999). A task scheduling algorithm for a graph describing a nonlinear optics problem is considered by Čiegis and Šilko (2002).

In this section, using the developed performance prediction tool, we propose a new heuristics for the scheduling of tasks in parallel LU factorization algorithm. First, the results of the theoretical analysis of parallel LU factorization algorithm are given. They provide a basis for constructing better schedules, i.e., distributions of matrix among processors. In Subsection 4.1, we present results on the efficiency of parallel LU factorization algorithm, when only computational costs are taken into account. Then in Subsection 4.2, we study communication costs of the algorithm for the block and cyclic matrix mappings onto processors. Finally, in Subsection 4.3, we propose a new heuristic for the scheduling of tasks in parallel LU factorization algorithm. The new distribution scheme is compared with the well known cyclic and block partitioning schemes.

4.1. *Efficiency of Parallel LU Factorization Algorithm*

In this section we briefly summarize the well-known results on the efficiency of parallel LU factorization algorithm, when communication overheads are neglected. Detailed proofs of all lemmas and theorems of Subsection 4.1 and Subsection 4.2 are given in our paper (Čiegis *et al.*, 2000).

First, we assume that the coefficient matrix A is mapped onto processors using 1D and 2D block decomposition schemes. Block partitioning puts successive elements (columns/rows of the matrix) into the same processors, whereas cyclic partitioning allocates them round-robin across processors (Choi *et al.*, 1995b). For 2D partitioning the 2D mesh of processors is formed.

Let p be the number of processors, N the size of the matrix A. For a fixed number of processors p and sufficiently large N, we get the following asymptotical estimates of the efficiency.

Lemma 4.1. *In the case of block matrix decomposition the efficiency E_p of the parallel LU factorization is given by:*

$$E_{p,1D} \approx \frac{2}{3}\,, \qquad E_{p,2D} \approx \frac{1}{3}\,.$$

The inefficiency of LU factorization with block data distribution is due to processor idling resulting from an uneven load distribution during computations, despite the even initial load distribution.

The efficiency of the parallel LU factorization algorithm is increased if the matrix is partitioned among processors using the *cyclic* data mapping.

Lemma 4.2. *In the case of cyclic matrix decomposition the efficiency E_p of the parallel LU factorization is given by:*

$$E_{p,1D} \approx 1\,, \qquad E_{p,2D} \approx 1\,.$$

4.2. *Communication Complexity Analysis*

In this subsection we show that the block and cyclic distribution schemes have quite different communication costs. We assume that the processors work asynchronously, that is no processor waits for the others to finish an iteration of LU factorization algorithm before starting the next one. Each processor first computes and sends any data destined for the other processors and only then performs the remaining computations using the data it has. Hence, the processor is idle only if it waits to receive data to be used.

Let assume that we use the completely connected communication network. Then we put $K(p) = 1$ into the equation (1) and the cost of broadcasting n items of data along a row or column of the mesh of processors is estimated by

$$T_b = \alpha + \beta n.$$

The following two theorems give conditions when data communications are overlapped by useful computations (see, Čiegis *et al.*, 2000).

Theorem 4.1. *For 1D decomposition the broadcast communication is overlapped with computations if the following condition*

$$m \geqslant 1 + \frac{\beta}{2\gamma} + \frac{\alpha}{2(N-k)\gamma} \tag{2}$$

is satisfied, where k is the step number in LU factorization algorithm, $m = N/p$.

Theorem 4.2. *In the case of 2D square grid of processors the overlapping of the broadcast communication with computations takes place if the following condition*

$$M \geqslant \frac{\beta}{2\gamma} + 1.5 + \sqrt{\left(\frac{\beta}{2\gamma} + 1.5\right)^2 + \frac{\alpha}{\gamma}} \tag{3}$$

is satisfied, where $M = N/\sqrt{p}$

The important consequence from Theorem 4.2 is that for 2D data partitioning the latency coefficient α can not be neglected even for large values of N.

Next, we study total communication costs during the implementation of all steps of the LU factorization algorithm in 1D case.

Theorem 4.3. *If conditions of Theorem 4.1 are not satisfied then the non overlapped communication overhead occurs*

- *during each step of LU factorization algorithm with the cyclic matrix decomposition scheme (in total* $N-1$ *times),*
- *only once per each block, i.e.,* $p-1$ *times, using the block data decomposition.*

It follows from Theorem 4.3 that the block partitioning of the matrix is better than cyclic one if communication overheads are compared. Thus the optimal data distribution must take into account properties of both these distributions and automatically tune to the paticular problem sizes and the computer parameters.

4.3. *Heuristic for Scheduling Tasks*

In this section, we propose a simple heuristic for scheduling the tasks of LU factorization algorithm onto processors. This heuristic uses essentialy the created performance prediction tool for the parallel LU factorization algorithm.

Taking into account the results of Theorem 4.1 and Theorem 4.3, we propose the following data distribution algorithm:

Matrix Distribution among Processors

1. The last J columns (and rows in 2D case) of the matrix are distributed according the block partitioning scheme.
2. The remaining $N-J$ columns (and rows in 2D) are distributed according the cyclic partitioning scheme.
3. Using the proposed performance prediction tool, we find the optimal value of J:

$$\min_{1 \leqslant J \leqslant N} T_{pr}(J) = T_{pr}(J_0)\,.$$

We note that this heuristic algorithm coincides with the block partitioning if $J_0 = N$, and with the cyclic partitioning if $J_0 = 1$.

In Table 4, we present the real T and predicted T_{pr} CPU times of the parallel LU factorization algorithm obtained with cyclic and heuristic partitionings on RS cluster. The matrix size was taken $N = 3000$.

Table 4

Comparison of the results for cyclic and heuristic partitionings

	cyclic		heuristic	
p	T	T_{pr}	T	T_{pr}
1	235	234	235	234
2	157	155	154	153
3	153	152	129	129
4	166	170	118	122
5	194	194	118	122

As we see, our heuristic produced partitioning schemes with a better results for two, three and four processors. It is interesting to note that for five processors the heuristic recommended to use only four processors.

We have also carried out a number of simulations by scheduling an 1024×1024 matrix in order to compare block, cyclic, and heuristic partitionings. The following computer parameters were used in our analysis (see, Amoura *et al.*, 1998):

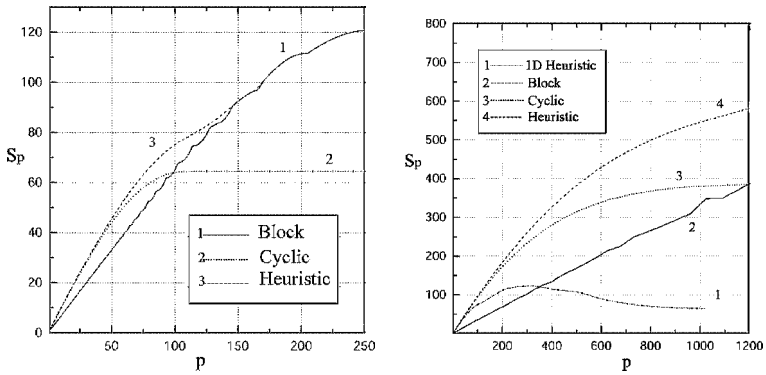
$$\alpha = 136\mu s, \quad \beta = 3.2\mu s, \quad \gamma = 0.18\mu s. \tag{4}$$

In Figs. 1, 2 we compare the speed-ups S_p for the block, cyclic and heuristic distributions for three networks: a) completely connected network, b) hypercube, c) LAN. 1D and 2D matrix partitionings are investigated in our experiments.

5. Conclusions

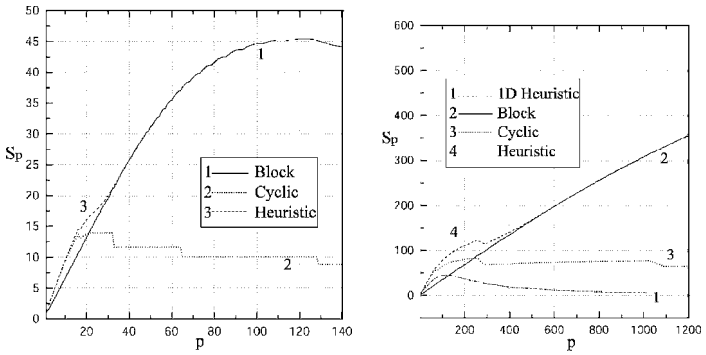
A performance prediction model is proposed as a set of rules and guidelines in order to provide a basis in creation of performance prediction tool for parallel algorithms. The use of performance simulation is promoted. As example, a performance prediction tool for the parallel LU factorization algorithm is created, which shows good agreement between predicted and real run times. The created tool enables us to answer the following questions:

1. What performance can be achieved with the parallel algorithm?
2. What data distribution should be used?
3. How many processors should be used for a given size of the matrix?

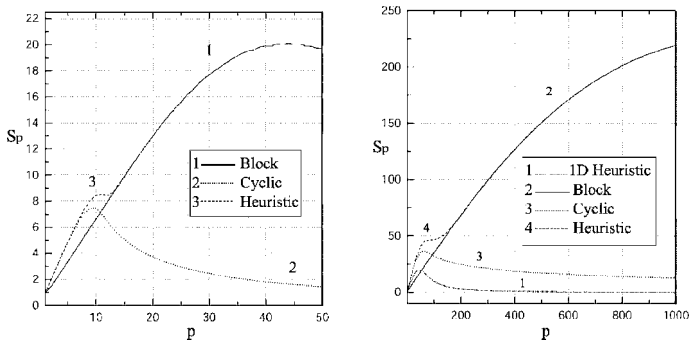


1D connected network 2D connected network

Fig. 1. Predicted speed-ups S_p of the parallel LU factorization algorithm as a function of number of processors p for the completely connected network.



1D hypercube network 2D hypercube network



1D LAN network 2D LAN network

Fig. 2. Predicted speed-ups S_p of parallel LU factorization algorithm as a function of number of processors p for the LAN and hypercube networks.

The tool is used to create a new heuristic for the scheduling of tasks of the parallel LU factorization algorithm. It is shown that the proposed heuristic is better than the well known and usually used cyclic and block partitioning schemes.

References

Amoura, A., E. Bampis and J. König (1998). Scheduling algorithms for parallel gaussian elimination with communication costs. *IEEE Transactions on Parallel and Distributed systems*, **9**, 679–686.

Baravykaitė, M., R. Belevičius and R. Čiegis (2002). One application of the parallelization tool of master–slave algorithms. *Informatica*, **13**(4), 393–404.

Baravykaitė, M., and R. Šablinskas (2002). The template programming of parallel algorithms. *Mathematical Modelling and Analysis*, **7**(1), 11–20.

Bhuayan, L., and X. Zhang (1995). Tutorial on on multiprocessor performance measurement and evaluation. *IEEE Computer Society Press*, **9**, 679–686.

Brinch Hansen, P. (1995). *Studies in Computational Science: Parallel Programming Paradigms*. Prentice Hall, New Jersey.

Čiegis, R., V. Starikovičius and J. Waśniewski (2000). On the efficiency of scheduling algorithms for parallel Gaussian elimination with communication delays. *Lecture Notes in Computer Science*, **1947**, 75–82.

Čiegis, R., and G. Šilko (2002). On the efficiency of scheduling algorithms for parallel Gaussian elimination with communication delays. *Lecture Notes in Computer Science*, **2028**, 75–82.

Choi, J., J. Demmel, I. Dhillon and J. Dongarra (1995a). Scalapack: a portable linear algebra library for distribution memory computers – design issues and performance. *LAPACK Working Note 95*.

Choi, J., J. Dongara and D. Walker (1995b). The design of a parallel dense linear algebra software library. *Numerical Algorithms*, **10**, 379–399.

Djordjević, G., and M. Tošić (1996). A heuristic for scheduling task graphs with communication delays onto multiprocessors. *Parallel Computing*, **22** 1197–1214.

Fahringer, T. (1995). Estimating and optimizating performance for parallel programs. *IEEE Computer*, **28**(11), 47–56.

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

Hey, T., A. Dunlop and E. Hernandez (1997). Realistic parallel performance estimation. *Parallel Computing*, **23**, 5–21.

Hockney, R. (1991). Performance parameters and benchmarking on supercomputers. *Parallel Computing*, **17**, 1111–1130.

IEEE (1995a). Special issue on performance evaluation tools. *IEEE Computer*.

IEEE (1995b). Special issue on performance evaluation tools. *IEEE Parallel and Distributed Technology*.

Kennedy, K., C. MacIntosh and M.S. Kinley (1992). Static performace estimation in a parallelising compiler. *Technical Report*, Department of Computer Science, Rice University.

Pease, D. (1991). PAWS: A performance evaluation tool for parallel computing systems. *IEEE Computer*, **22**(1), 18–29.

Simons, M., C. Pancake and J. Yan (1995). Performance evaluation tools for parallel and distributed systems. *EEE Computer*, **28**, 16–19.

Xu, Z., and K. Hwang (1996a). Early prediction of mpp performance: the SP2, T3D, and Paragon experiences. *Parallel Computing*, **22**, 917–942.

Xu, Z., and K. Hwang (1996b). Modeling communication overhead: MPI and MPL performance on the SP2 system. *IEEE Parallel and Distributed Technology*, 9–23.

Zomaya, A. Y., C. Ward and B. Macey (1999). Genetic scheduling for parallel systems: comparative studies and performance issues. *IEEE Transactions on Parallel and Distributed Systems*, **10**(8), 795–812.

R. Čiegis has graduated from Vilnius University Faculty of Mathematics in 1982. He received the PhD degree from the Institute of Mathematics of Byelorussian Academy of Science in 1985 and the degree of Habil. Doctor of Mathematics from the Institute of Mathematics and Informatics, Vilnius in 1993. He is a professor and a head of Mathematical Modeling Department of Vilnius Gediminas Technical University. His research interests include numerical methods for solving nonlinear PDE, parallel numerical methods and mathematical modeling in nonlinear optics, biophysics, technology.

V. Starikovičius has graduated from Vilnius University Faculty of Mathematics in 1998. He received Master degree in mathematics. In 2002 he received the PhD degree from Vilnius University. Now he works in Vilnius Gediminas Technical University. His research interests include mathematical modelling of porous media materials, parallel algorithms, numerical methods for hyperbolic equations.

Lygiagrečiojo blokinio LU faktorizacijos algoritmo efektyvumo įvertinimo įrankis

Raimondas ČIEGIS, Vadimas Starikovičius

Nagrinėjamas naujas lygiagrečiųjų algoritmų vykdymo laiko prognozavimo įrankis. Pirmiausia aptarta skaičiuojamosios dalies ir duomenų persiuntimo kaštų įvertinimo metodika. Parodyta, kad nustatant modelio konstantas, būtina atsižvelgti į uždavinio dalies, tenkančios kiekvienam procesoriui, dydį. Vienas svarbiausių naujojo įrankio bruožų tas, kad įvertinama skaičiavimo ir duomenų persiuntimo procesų sinchroninio vykdymo galimybė. Tai pasiekama emuliuojant kiekvieno procesoriaus darbą. Įrankio efektyvumas iliustruojamas blokinio LU faktorizacijos algoritmo analize. Pasiūlytas naujas duomenų paskirstymo algoritmas ir ištirtas jo efektyvumas. Parodyta, kad šis duomenų paskirstymas efektyvesnis už klasikinius ir dažniausiai naudojamus blokinį ir ciklinį paskirstymus.