Performance Optimization of Parallel Algorithms

Martin Húdik and Michal Hodoň

Abstract: **The high intensity of research and modeling in fields of mathematics, physics, biology and chemistry requires new computing resources. For the big computational complexity of such tasks computing time is large and costly. The most efficient way to increase efficiency is to adopt parallel principles. Purpose of this paper is to present the issue of parallel computing with emphasis on the analysis of parallel systems, the impact of communication delays on their efficiency and on overall execution time. Paper focuses is on finite algorithms for solving systems of linear equations, namely the matrix manipulation (Gauss elimination method, GEM). Algorithms are designed for architectures with shared memory (open multiprocessing, openMP), distributed-memory (message passing interface, MPI) and for their combination (MPI + openMP). The properties of the algorithms were analytically determined and they were experimentally verified. The conclusions are drawn for theory and practice.**

Index Terms: **Collective communication operations, efficiency, Gauss elimination method, modeling, parallel algorithms, parallel architecture, parallel computation, performance prediction, pipelined broadcast, system of linear equations.**

I. INTRODUCTION

The high research intensity in fields of mathematics, physics, biology and chemistry, requires new powerful computing resources. For the large computational complexity of specific tasks computing, e.g., [14] in intelligent buildings automation or [12] in multirobot systems, the time is long and costly. The deployment of parallel principles proves to be the most effective solutions. The parallel principles open up new possibilities of harvesting computer processing power. The use of parallel systems brings new complex problems that must be addressed to achieve the desired increase in performance either in classic high performance computing (HPC) solutions or in specific low-cost embedded solutions, e.g., FPGA [15]. The ongoing transition to the parallel principles is already supported by hardware such as multi-processor, multi-core technology, symmetric multiprocessing (SMP) and also software such as message passing interface (MPI), open multiprocessing (OpenMP), parallel virtual machine (PVM), Java, C#, Intel TBB. Increasing throughput of interconnection networks helps development of parallel technology (Myrinet, Infiniband, giga ethernet, 10 gigabit ethernet, fibre channel).

The computer hardware is evolving fast. For good utilization of the new hardware also the software platform need to evolve. There are lot of software tools that the developer can choose from, but the question is which of them to pick for the best performance on a given hardware platform. The next parts of this paper discuss analytical approach to performance evaluation of parallel algorithm (PA) and analysis of PA for solving the system of linear equation is conducted on different hardware platform and software tools.

Manuscript received April 14, 2014.

The research is supported by the European Regional Development Fund and the Slovak state budget for the project "Research Centre of University of Žilina," ITMS 26220220183.

M. Húdik and M. Hodoň are with the Department of Technical Cybernetics, University of Žilina, Univerzitná 8215/1, 010 26 Žilina, Slovakia, email: {martin.hudik, martin.hudik}@fri.uniza.sk.

Digital object identifier 10.1109/JCN.2014.000074

II. ANALYSIS OF PARALLEL COMPUTERS

Computer architects have always seek to improve the performance of computer architectures. High performance can come from fast integrated circuits with high density of integration or using the parallel principles. The trend of single processor super computer ended, because of the physical boundaries that limit the computing power of single processor system. This paper is devoted to modern computer architectures, parallel environment using multiple computing nodes (processors, cores, computers).

A. Parallel Computers

Today trends in the world have been heading towards a substitution of conventional supercomputers with group of interconnected, highly specialized computers (clusters) or powerful workstations (NOW) [3]. The reason for the enlargement of clusters in the world is in particular their universality, mutual independence, price and scalability according to current requests of users. The main disadvantages are the complex management and lack of shared memory. Parallel systems achieve higher performance, but their high cost prevents them from wider dissemination.

The trends in recent years were symmetric multi-processor systems, the number of the processor architecture for SMP steadily increased. Multi-core CPUs as well as multicellular (cell) processors are currently starting to dominate [1], [16]. An example is TeraScale processor, which achieves performance among teraflops and the number of cores in the future will be in the order of hundreds. The main advantage of multi-core processors, in addition to increased performance, is a significant reduction in processor consumption and therefore more economical operation of computer. Another example of increasing influence of multi-core technology is new coprocessor card from Intel called Xeon Phi. The Intel Xeon Phi coprocessor consists of up to 61 cores connected by a high performance on-die bidirectional interconnect. The coprocessor runs a Linux operating system and supports all important Intel development tools, like C/C++ and Fortran compiler, MPI and OpenMP, high performance libraries like MKL, debugger and tracing tools like Intel VTune Amplifier XE. Traditional UNIX tools on the coprocessor are supported via BusyBox, which combines tiny versions of many common UNIX utilities into a single small executable [10].

In last decade the Grid has starts to emerge as an alternative

of distributed parallel computers [11]. It represents an easily expandable architecture of a parallel computer that can include any device capable of communication [4] e.g., SMP computers, computers with Intel Xeon Phi coprocessor card or with GPUs, etc.

III. MODEL OF COMMUNICATION COST IN COMPUTERS WITH DISTRIBUTED MEMORY

The total time required to move a message consists of the following partial times:

- t_s – Starting time - the time required for processing / preparation of message on the receiver and transmitter
- t_h – Per-hop time – time required to move a message between two neighbouring nodes
- t_w – Per-word transmit time – the bandwidth of the transmission channel is *r* words per second, so the time to transfer one word is the inverse of the $t_w = 1/r$.

$$t_{\text{comm}} = lt_h + mt_w + t_s. \tag{1}$$

This equation represents the cost model for communication sending messages of size m between the nodes distant l jumps. Since in most systems the number of hops l is small, we ignored $l\ t_h$ (per-hop) time. The resulting time of communication "point to point" is then

$$t_{\text{comm}} = mt_w + t_s. \tag{2}$$

IV. MODEL OF COMMUNICATION COST IN COMPUTERS WITH SHARED MEMORY

Estimate the cost of communications for computers with shared memory is much more complex. Many variables need to be taking into account to produce a comprehensive communication model. Such a model would be very specific and dependent on particular computer architecture, and would not be generally applicable [13].

In a simplified model, all the delays associated with memory operations, network and other delays is added to t_s constant. The t_s constant is related to first access time to the coherent shared data the size of m words. Furthermore, we assume that access to shared data is more time-consuming as access to local data and therefore time access to one word t_w is assigned to shared data access. From the previous implies that for the simplified model we can write the equation:

$$t_{\text{comm}} = mt_w + t_s. \tag{3}$$

This equation is identical to the (2) describing a simplified cost model for computers with distributed memory. Constants t_s and t_w are for the model describing computers with shared memory much smaller.

V. PERFORMANCE EVALUATION

A. Classic Metrics

A.1 Parallel Cost

Let $T(s,p)$ represents a parallel algorithm for solving p processors (s defines the size of the problem). Then the price of this parallel algorithm can be defined as

$$C(s) = pT(s,p). \tag{4}$$

$C(s)$ represents the total work done by all processors involved in the calculation. Parallel program is called cost optional if $C(s) = T(s,1)$, i.e., when the total number of operations performed is same as for a fastest sequential algorithm, whose computation time is $T(s,1)$.

B. Specialized

B.1 Parallel Speed-Up

Let $O(s,p)$ is the total number of operations carried out by the p-processor system for application of size s, $T(s,p)$ is the time to perform the parallel task. In general, $T(s,p) < O(s,p)$ if $p \geq 2$. We assume $T(s,1) = O(s,1)$ for a single processor system (classical sequential system). Parallel acceleration (speed-up) is then defined as

$$S(s,p) = \frac{T(s,1)}{T(s,p)}. \tag{5}$$

B.2 Efficiency

Efficiency of the system for the p-processor system is defined as

$$E(s,p) = \frac{S(s,p)}{p} = \frac{T(s,1)}{pT(s,p)}. \tag{6}$$

VI. PARALLEL ALGORITHMS

A. Shared Memory

In parallel algorithms versus sequential algorithm is necessary for parallel access to shared data to use control mechanisms that cause additional delays. In modelling the performance of parallel algorithms for classical parallel systems with shared memory, these overheads neglected because it is assumed that in comparison with the workload of the calculation they are considerably lower [5]. Therefore, for the parallel algorithms for shared memory architecture, the time complexity is reduced to computation complexity. Interpreting this assumption in relation to the asymptotic function means:

$$\begin{aligned} \omega(s) &= \max[\text{computation}, h(s,p) < \text{computation}] \\ &= O(\text{computation}) \end{aligned} \tag{7}$$

where ω is a workload and h is a delay.

B. Distributed Memory

Distributed PA (DPA) for existing parallel computers based NOW, SMP, and Grid requires for performance modelling a complex analysis of all relevant components for modelling performance [9]:

- The impact of parallel system architecture,
- the impact of inter-process communication (IPC),
- communication initialization (start-up time),
- data transfer,
- switching (transmission through more communication nodes),

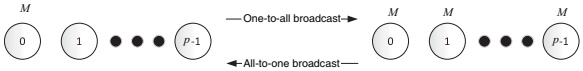

Fig. 1. All-to-one and one-to-all.

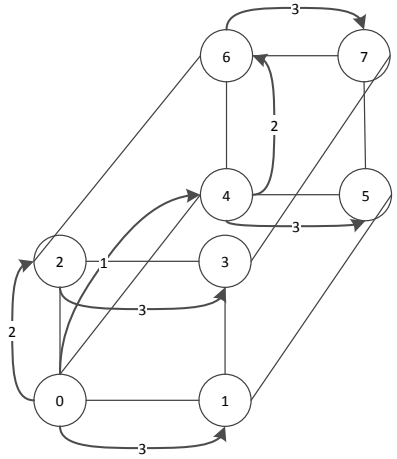

Fig. 2. One-to-all communication on the hypercube.

- and computation itself.

As a result of these effects the analyse of the total delay is

$$T(s,p) = T_{\text{comp}} + T_{\text{par}} + T_{\text{interact}} \tag{8}$$

where T_{comp}, T_{par}, and T_{interact} give the individual delay to perform the computation, overhead for parallelism and communication overhead.

B.1 Collective Communication: One-to-All (broadcast) and All-to-One Reduction

Parallel algorithm often needs to send identical data to all processes or a subset of processes. This operation is called a broadcast.

For one-to-all broadcast only one process has data of size of m, which will be distributed. At the end of the operation there will be p copies of the original data, which were sent to each process. The opposite operation is called an all-to-one reduction. In all-to-one reductions each p contributing process comprises a buffer containing M data with the size of m. Data from all processes are combined through associative operation and subsequently accumulated in the target process. This communication method is used for example to find the maximum, minimum, sum, etc.

Algorithm: The basic broadcast algorithm uses method of recursive doubling. When recursive doubling starts the process sends a message to one process and on the next step both process sends a message to another process. At each step doubles the number of message senders. To complete the broadcast it is necessary to done $\log_2 p$ steps.

Hypercube: We imagine hypercube as extended mesh network in d dimensions. Algorithms used for the mesh can be applied and used consecutively for each dimension of the hypercube. Communication begins at node 0.

B.1.a Evaluation. Table 1 provides an overview of communication operations and time requirements for the hypercube communication structure. Hypercube is one of the best possible communication links, but in practice is often not realized. When building a financially less demanding parallel systems like NOW most commonly used is interconnected interconnection (linear list).

VII. THE SYSTEM OF LINEAR EQUATIONS

Consider a system of linear equations (SLQ) [2] in the form:

$$\begin{aligned} a_{0,0}x_0 + a_{0,1}x_1 + \cdots + a_{0,n-1}x_{n-1} &= b_{0,} \\ a_{1,0}x_0 + a_{1,1}x_1 + \cdots + a_{1,n-1}x_{n-1} &= b_{1,} \\ a_{n-1,0}x_0 + a_{n-1,1}x_1 + \cdots + a_{n-1,n-1}x_{n-1} &= b_{n.} \end{aligned} \tag{9}$$

In matrix notation given system can be written as

$$\mathbf{A}\,\mathbf{x} = \mathbf{b} \tag{10}$$

where **A** is matrix of coefficients, **b** is vector of right side and **x** is vector of unknowns. Trying to solve it means to find all solutions of this system. The solution for system of linear equations 9 means to find the vector $\mathbf{r} = (r_1, r_2, \cdots, r_n)^T$ for which applies $\mathbf{A}\times\mathbf{r} = \mathbf{b}$. To calculate the system of linear equations there are many methods that are divided into two basic groups; finite methods and iterative methods [8].

- Direct methods (finite),
- Cramer's rule,
- elimination methods,
  - Gauss elimination,
  - Gauss-Jordan elimination,
- other (LU decomposition),
- and iterative methods [7].

VIII. PARALLEL ALGORITHMS GEM

For solving the system of linear equation we choose a simple gauss elimination method (GEM), because of its easy implementation and it is a good algorithm for analysis communication impact on overall performance.

Finding solutions to systems of linear equations is divided into two parts – the matrix transformation and the calculation of unknowns [6]. We will discuss only part of the matrix transformation, because it has a dominant influence on the performance of the algorithm.

A. Parallelization above the Shared Memory

Algorithm working over the shared memory works with a whole matrix of coefficients $n \times n$. Number of processes are created, each of which operates only over their common matrix (block of rows). GEM steps are shown in Fig. 3. The first process calculates and synchronizes the remaining processes. Using synchronization superior process announces that it carried out the calculation and stored data are valid. After the synchronization processes load row from memory of superior process and carry out the elimination and computation. Next in order of overarching process synchronizes other processes and so it continued until complete elimination of matrix is done.

Table 1. Overview of time complexity for individual collective communication methods.

Comunication operation	Time
One-to-all broadcast, all-to-one reduction	$\min[(t_s + t_w m)\log_2 p,\ 2(t_s \log_2 p + t_w m)]$
All-to-all broadcast, all-to-all reduction	$t_s \log_2 p + t_w m(p-1)$
All-reduce	$\min[(t_s + t_w m)\log_2 p,\ 2(t_s \log_2 p + t_w m)]$
Scatter, gather	$t_s \log_2 p + t_w m(p-1)$
All-to-all personalized	$(t_s + t_w m)(p-1)$

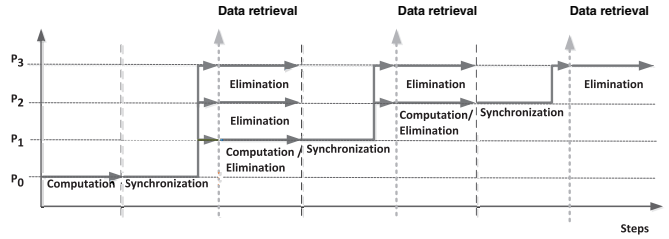

Fig. 3. GEM parallel algorithm above the shared memory.

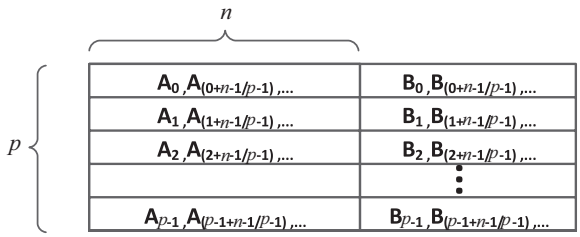

Fig. 4. Graphical representation of a cyclic decomposition.

A.1 Total Execution Time

In determining the total execution time of the parallel algorithm over the shared memory, the analysis of computational complexity is limited to the computation itself. Communication part can be neglected, given the fast communication link with the memory. Parallel complexity is derive as follows: Parallel Gaussian elimination performed approximately $\frac{1}{p}\sum_{j=1}^{n-1}\sum_{k=j}^{n-1} 1 = \frac{-n+n^2}{2p}$ dividing and approximately $\frac{1}{p}\sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=i}^{n} 2 = \frac{n+3n^2+2n^3}{3p}$ subtraction and multiplication. For simplicity, we assume that all arithmetic operations are performed in a unit time. Computational complexity can then be written as

$$T(s,p)_{\text{comp}} = t_c\left(\frac{-n+n^2}{2p} + \frac{n+3n^2+2n^3}{3p}\right) \qquad (11)$$

where the constant t_c represents average execution time of a single operation.

B. Parallelization over Distributed Memory

When designing parallel algorithm from sequential algorithm, it is important to choose the right decomposition strategy. Furthermore, it is important to choose the preferred model of communication. Communication takes place either synchronously or asynchronously. Because the synchronous communication is ineffective, in this paper we only analysis the asynchronous version of GEM algorithm.

B.1 Synchronous Implementation

For simplicity, row cyclic decomposition is selected. Each computing node is cyclically assigned with row of matrix coefficients **A** and **b** vector elements of right sides. Thus, the entire matrix **A** of size $n \times n$ and vector **b** of size n are divided among p processes, each process is assigned n/p rows

The GEM parallel algorithm based on the sequential version is complemented with a communication section, that is designed to send the currently processed row to the remaining processes

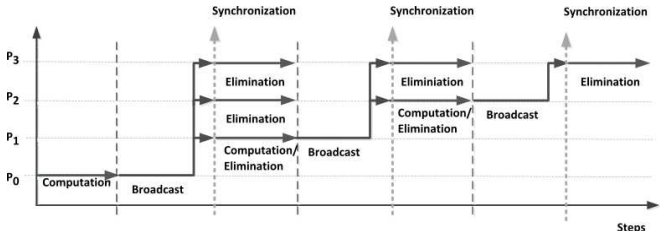

Fig. 5. Synchronous parallel implementation of GEM.

that are using it to carry out their part of the elimination of the matrix.

B.2 Analysis of the Parallel Algorithm

The total execution time of a parallel algorithm is the sum of the communication complexity and computational complexity

$$T(s,p)_{\text{par}} = t_c\left(\frac{-n+n^2}{2p} + \frac{n+3n^2+2n^3}{3p}\right) + \frac{1}{2}n(2t_s + t_w + nt_w)\log_2 p\,. \qquad (12)$$

B.3 Modification of Broadcast Communication Algorithm to Improve the Synchronous Version of GEM Algorithm for Implementation in a Multiprocessor Environment

The modern trend of SMP computers with n-integrated computing cores is the future of parallel systems. Today's multi-core workstations typically contain 4, 8, 16 and more compute cores. For efficient use of GRID, NOW or parallel computers, consisting of such a SMP computers, need to adapt existing software tools to take into account the characteristics of a such system.

From the perspective of optimization GEM synchronous algorithm it should focus on effective inter-process communication, which is one of the main limiting factors.

Based on the domain parting Fig. 7 we can split communication on inter-node communication (main domain) and intra-node communication (sub-domain).

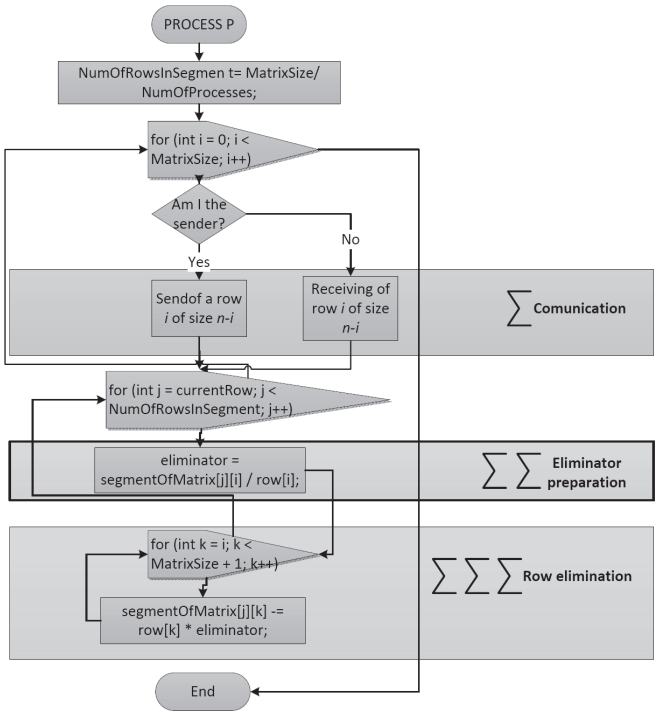

Fig. 6. Flowchart of the parallel algorithm GEM with an indication of computing and communication complexity.

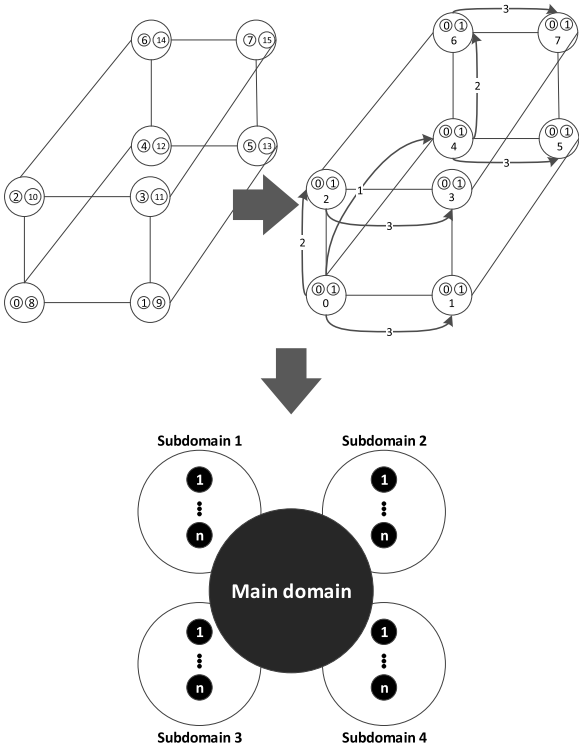

Fig. 7. Division of the communication domain on the sub-domains.

For the time analysis of algorithm, we start from Fig. 8, it is clear that sending one message to all recipients happens in three steps, which are performed sequentially. The first step and the third step represent internal communication, which is derived from point-to-point communication. The second step represents inter-node communication in the main domain and the time complexity is derived from the broadcast algorithm described in the previous sections. At the beginning we what to send message of size m to $(p \times j)$ process, where p is the number of SMP computers with multiple cores (processors) and j is the number of those cores (processors).

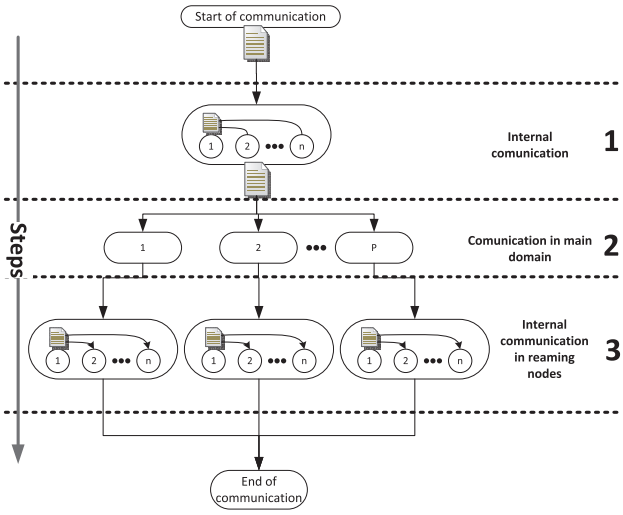

Fig. 8. The steps of proposed sub-domain broadcast communication algorithm.

Total time to deliver single message from one process to other is the sum of all partial times and applies

$$\begin{aligned} T &= T_1 + T_2 + T_3 \\ &= (t_{sc} + t_{wc}m)\,(n) + (t_s + t_w m)\log_2 p \; + (t_{sc} + t_{wc}m)(n) \\ &= 2(t_{sc} + t_{wc}m)\,(n) + (t_s + t_w m)\log_2 p\,. \end{aligned} \tag{13}$$

Assuming that the communication inside the computer (node) is of the order faster than communication between computers (nodes), i.e., $t_{sc} \ll t_s$ and $t_{wc} \ll t_w$, the resulting equation is simplified only on the transmission of messages between computers (nodes) in the main domain.

$$T = (t_s + t_w m)\log_2 p \tag{14}$$

from this assumption the following equation is valid for the time complexity of the GEM synchronous parallel algorithm:

$$\begin{aligned} T\,(s,p)_{\mathrm{par}} = t_c \left(\frac{-n + n^2}{2p} + \frac{n + 3n^2 + 2n^3}{3p}\right) \\ + \; \frac{1}{2} n(2t_s + t_w + nt_w)\log_2 p_{\mathrm{smp}} \end{aligned} \tag{15}$$

where p is the total number of processors (cores) parallel system and p_{smp} the actual number of SMP computers forming a parallel computer.

B.4 Asynchronous Implementation

With asynchronous GEM each process independently carries out operations until the whole algorithm performed all *n* iterations needed to transform matrix A on the upper triangular matrix. Processes carried out one of following steps:

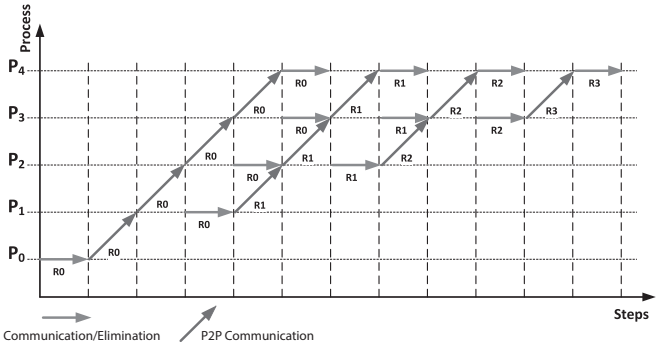

Fig. 9. Asynchronous parallel implementation of GEM.

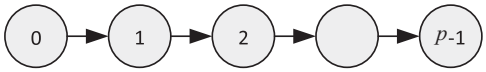

Fig. 10. Computers arranged in a threaded linear tree.

1. If the process has some data to send, then sends them to a particular process.
2. If the process has appropriate data to calculate then he carries out the calculation.
3. In other cases, the process is waiting to receive data in order to perform the previous two steps.

Such realized algorithm (see Fig. 9) is optimal with regard to the sequential equivalent. To implement this algorithm we use non-blocking communication methods and processes are logically arranged in a linear linked list. Communication is carried out in a manner similar one-to-all on a pipelined linear list. Some implementations of MPI provide non-blocking MPI collective operations. In that case, implement one-to-all (broadcast) method can be used.

C. Modification of Broadcast Communication Algorithm for Transforming the Synchronous Version of GEM Algorithm to the Asynchronous Version

The time complexity analysis shows that sending one message to all recipients happens in three steps, which are performed sequentially. At the beginning the message of size *M* is send to $p \times n$ processes, (where p is the number of SMP computers with multiple cores (processors) and *n* is the number of such cores (processors)). For modelling point-to-point communications between computers (nodes) we use (2) and for communication between the cores (CPUs) within a single computer (node) we use (3). Then, for step 1 and step 2 is valid equation

$$T_1 = T_2 = (t_{sc} + t_{wc}M)(n). \tag{16}$$

This equation describes the sequential sending one message to all processes within a single computer (node). Computers (nodes) communicating in step 2, are logically arranged in a threaded linear tree as shown in Fig. 10. The message is divided into smaller parts (packets) that are sent sequentially through a liner pipeline. For time complexity analysis of the algorithm the algorithm is divide into two steps. The first step is sent part of the message (packet) through linear tree until it reaches the last node. This step is called filling the linear tree and it's shown in Fig. 11, where p is the number of communicating processes, and

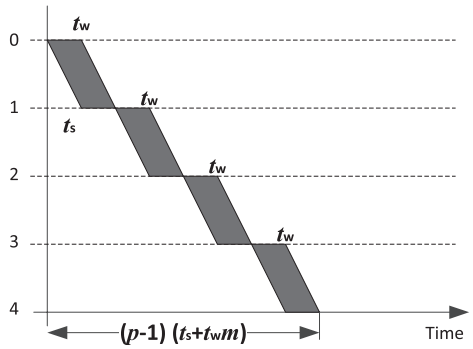

Fig. 11. Filling the linear tree.

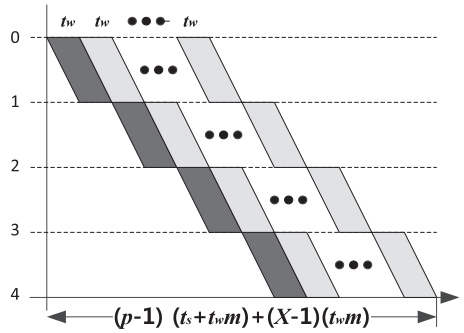

Fig. 12. Sending a message through a pipelined linear tree.

m is the size of the packet. After filling the linear tree the rest of the message go through the whole tree to the last node, as shown in Fig. 12, where *X* is the number of packets for which the message of size M divided and the size per packet, i.e., $X = M/m$. At the end of this process, all nodes have a copy of the message.

The total transmission time broadcast algorithm using linear tree is the sum of the time needed for the initial tree filling and the time to send a remaining packets

$$T_2 = (p-1)(t_s + t_w m) + (X-1)(t_w m). \tag{17}$$

The total time for sending one message from one process to all processes is given by the sum of partial times

$$\begin{aligned} T &= T_1 + T_2 + T_3 \\ T &= (t_{sc} + t_{wc}M)(n) + (p-1)(t_s + t_w m) + (X-1)(t_w m) \\ &\quad + (t_{sc} + t_{wc}M)(n)\,T \\ &= 2(t_{sc} + t_{wc}M)(n) + (p-1)(t_s + t_w m) + (X-1)(t_w m). \end{aligned} \tag{18}$$

Assuming that the communication inside the computer (node) is of the order faster than communication between computers (nodes), i.e., $t_{sc} \ll t_s$ and $t_{wc} \ll t_w$, the resulting equation is simplified only for the transmission of messages between computers (nodes) in the main domain

$$T = (p-1)(t_s + t_w m) + (X-1)(t_w m). \tag{19}$$

D. Analysis of GEM Asynchronous Algorithm

D.1 Computational Complexity

Computational complexity is the same computational complexity of algorithm for architectures with shared memory and

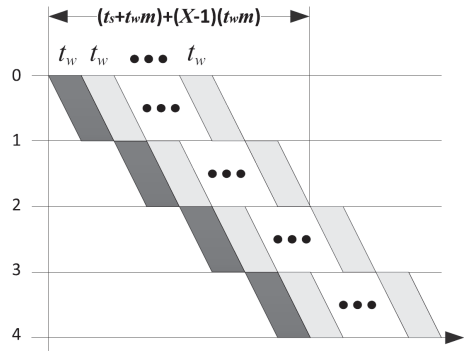

Fig. 13. Sending a message to the following process.

is given by (11).

D.2 Communication Complexity

With asynchronous GEM each process independently conduct operations until the whole algorithm performed all n iterations needed to transform matrix A to an upper triangular matrix. This means that after each elimination communication occurs, at which the calculated row is sent to the following process and the further processes. Processes are arranged in a row in a linear fashion. When communicating with use of pipelined broadcast algorithm the time to send a message to the next process is derived from the Fig. 13.

The total transmission time is the sum of the time needed for the initial linear tree filling and the time to transmission the remaining packets.

$$T = (p-1)(t_s + t_w m) + (X-1)(t_w m), \tag{20}$$

$$T = (t_s + t_w n). \tag{21}$$

Due to the pipeline character of computation and communication the total time of communication simplifies only to a data transfer to a nearest neighbor. In each iteration the amount of data to be transferred is reduced.

$$T(s,p)_{\text{comm}} = \sum_{i=0}^{n-1} (t_s + t_w (n-i)) = nt_s + \frac{\mathrm{n}t_w}{2} + \frac{n^2 t_w}{2}. \tag{22}$$

D.3 Execution Time

The total execution time of a parallel algorithm is the sum of the communication complexity and computational complexity

$$T(s,p)_{\text{par}} = t_c\left(\frac{-n+n^2}{2p} + \frac{n+3n^2+2n^3}{3p}\right) + nt_s + \frac{\mathrm{n}t_w}{2} + \frac{n^2 t_w}{2}. \tag{23}$$

E. The Hybrid Algorithm

This algorithm works over a distributed shared memory. This combination emerged from use of technology for communication using MPI messaging and communication via shared memory OpenMP. When designing the algorithm we modified an asynchronous version of GEM algorithm of distributed memory. Modification of the original algorithm is based on a division of cycles for row elimination of matrix. The divided cycles are carried out on separate cores of SMP computer.

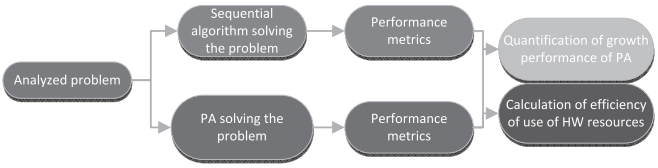

Fig. 14. Measurement methodology.

E.1 Execution Time

The total execution time of a parallel algorithm is the sum of the communication complexity, computational complexity and initialization time

$$T(s,p)_{\text{par}} = \frac{t_c}{p_{\text{core}}}\left(\frac{-n+n^2}{2p} + \frac{n+3n^2+2n^3}{3p}\right) + \frac{n\,t_s}{p} + \frac{\mathrm{n}t_w}{2} + \frac{n^2 t_{w2}}{2p} + \frac{1}{2}n\,(n-1)\,t_{\text{init}}\,p_{\text{core}}. \tag{24}$$

IX. EXPERIMENTAL RESULTS

A. Measurement Methodology

According to the method of measurement the methodologies can be divided into direct and indirect. For empirical performance measurement of algorithms more suitable methods are the direct because of ease of implementation. The overall methodology of measurement is shown in Fig. 14.

A.1 Measurement Methodology for Sequential Algorithm

Empirical tests of a sequential program was conducted on computer with multi-core processor. The tests examined the execution time, which is required for processing sequential program. Each test consists of five sessions, within each session the execution time was measured. Then overall execution time is calculated as the average from all execution times.

A.2 Measurement Methodology for Parallel Algorithm

The basic prerequisite for the measurement of complex performance metrics of the parallel system is the development of a particular parallel algorithm for the architecture of the workstation with available software. GEM parallel algorithms have been implemented for a distributed memory architecture with the help of MPI and the shared memory architecture with the help of OpenMP standard.

A.3 Measurement Methodology for Parallel Algorithm on Shared Memory Architecture

For the measurement of execution time of parallel algorithms for architectures with shared memory analogue methods to measuring sequential algorithms were used. Time execution of algorithm was measured using the built-in functions

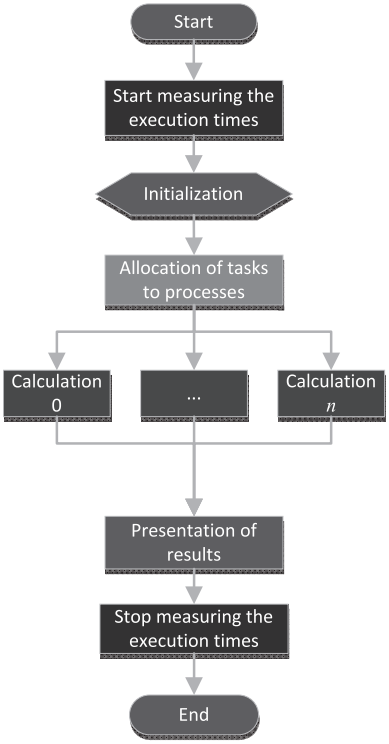

Fig. 15. Flowchart of measurement of execution time for architectures with shared memory.

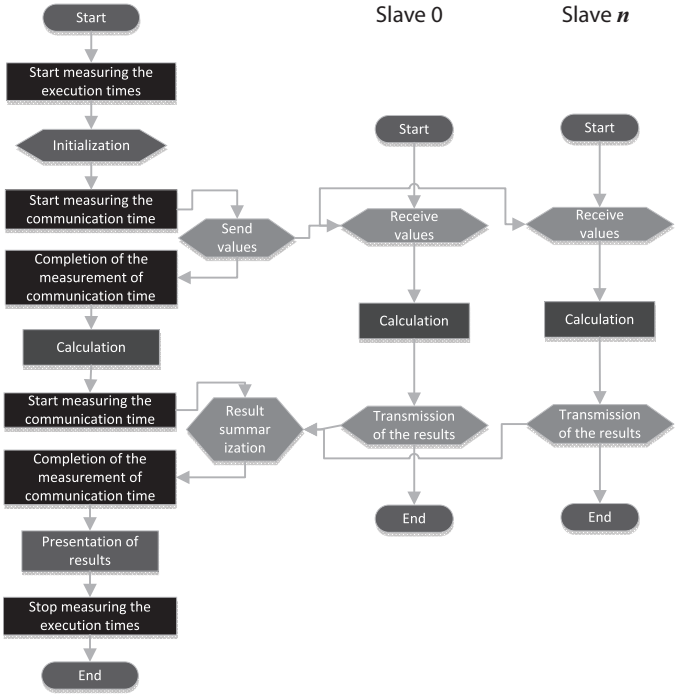

Fig. 16. The principle of measuring the execution time and communication time for architectures with distributed memory.

omp_get_wtime() in library OpenMP, which returns the number of milliseconds from the specified point. Fig. 15 shows the methodology of direct-measuring.

A.4 Measurement Methodology for Parallel Algorithm on Distributed Memory Architecture

Direct measurements on a distributed memory architectures are based on measuring the total execution time of a parallel algorithm. When analyzing the parallel algorithm it is necessary to take into account overhead costs that are necessary to ensure communication between processes. Individual components can be measured separately

$$T(s,p)_{\text{total}} = T(s,p)_{\text{comm}} + T(s,p)_{\text{comp}}. \qquad (25)$$

Overhead costs consists of time required for communication between processes $T(s,p)_{\text{comm}}$ and computational cost consists of calculation time $T(s,p)_{\text{comp}}$. The algorithm is then divided by the carrying out the communication and of carrying out the calculation itself. The measurement is performed for different problem sizes and number of different processes. The Execution time was measured using the built-in functions *MPI_Wtime()*, which is contained in the MPI library. The measurement is repeated for different numbers of processes as well as for different size of the problem. The duration of the algorithm was measured by using the tool *MPI_Wtime()* which is a function of the standard MPI library. This function returns the number of milliseconds from a designated point. The overall methodology of measurement is shown in Fig. 16.

B. The Architecture

For architectures with shared memory OpenMP standard was used. All algorithms were implemented in the C language. GNU GCC compiler was used in version 4.6. Experiments on algorithm for environment with shared memory took place on multicore server with two Intel Xeon processors.

Measurements of algorithms for environments with distributed memory architecture have been carried out on the NOW parallel computer, which consisted of eight SMP computers that are connected to each other through a gigabit Ethernet switch (switch) 3Com. Computer specifications are listed in Table 2. On the workstations there was installed 64-bit Ubuntu Linux 12.10. MPICH2 in version 1.4.0 was used as MPI implementation.

C. Measurements on Architectures with Distributed Memory

C.1 Execution Time

The total execution time of parallel algorithms were measured at a constant input load with a variable number of processes as well as the variable input load with a constant number of processes. The following measurements are calculated for the input load $n = 1500$, i.e., input matrix size is 1500×1500. The calculation is performed in double precision decimal numbers (DOUBLE).

C.2 Effect of Growth of Number of Processes

Fig. 17 illustrates the dependence of execution time implementation of synchronous and asynchronous version of the GEM algorithm, the number of processes at a constant load. The graph shows that the execution time with an increasing number of processes is decreases. The asynchronous algorithm according to analyses conducted in previous sections should be the

Table 2. Parameters of SMP computers.

Number of nodes	Procesor	Number of cores	RAM	Network interface
8	Quad core Q6600 2.4 Ghz	4	4 GB	1 Gb/s

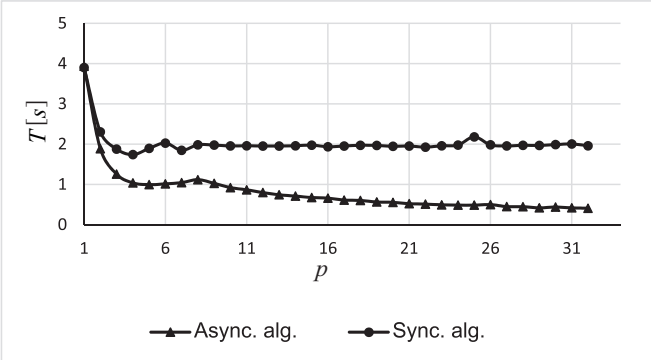

Fig. 17. Effect of change in the number of processes on the execution time.

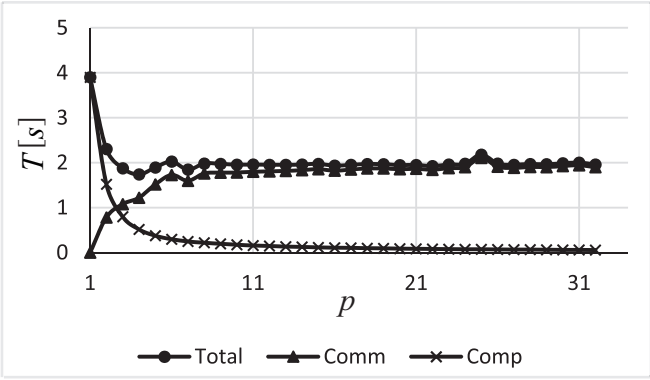

Fig. 18. Analysis of the impact of computing and communication delays for synchronous GEM alg. at the input load $n = 1500$.

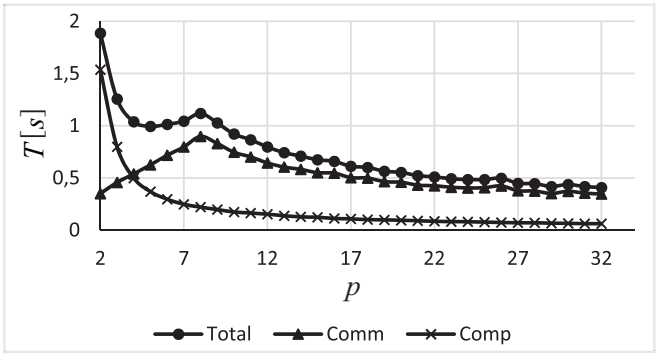

Fig. 19. Analysis of the impact of computing and communication delays for asynchronous GEM alg. at the input load $n = 1500$.

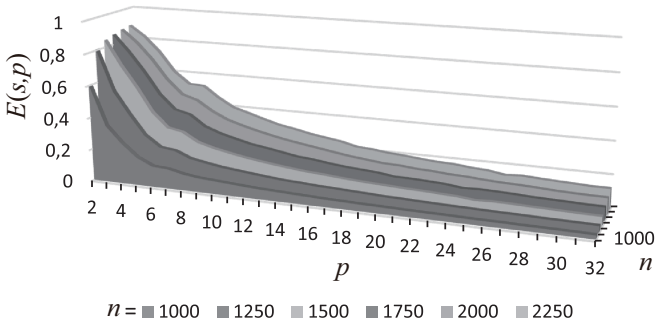

Fig. 20. Efficiency of synchronous algorithm.

limited by the time of transmission of row to the neighbouring process. Contrary synchronous algorithm has a breaking point, where communication time exceeds the computational time and it will affect the overall execution time. For a detailed analysis of this phenomenon it is necessary to divide the total execution time on partial components; time communication between processes (t_{comm}) and computation time (t_{comp}).

Fig. 18 shows the total execution time of synchronous GEM algorithm divided into individual components.

Gradual enlarging the number of processes involved in the computation reduces the calculation time, but from a certain tipping point, where communication achieved faster growth than the time of computation, execution time begins to be limited by time of communication. From this point begins the overall execution time to increase.

Fig. 19 shows the total execution time asynchronous GEM algorithm that is divided into individual components. The graph shows that communication for a small number of processes increases, this is due to unwanted synchronization process, which is caused by the fact that the computation is so demanding that there is only limited pipe-lining of calculations. Processes are forced to wait for valid data. With increasing number of processes starts to fulfil pipeline of calculations and communication begins overlap with calculations. When there are a sufficiently large number of processes, communication stabilizes at a constant value.

C.3 The Efficiency of the Use of Computing Resources

Efficiency of utilization of computing resources is an important indicator for assessing the suitability of a specific algorithm for parallel computer architectures. Efficiency is a measure of utilization of computational capacity during the calculation of the parallel program. Fig. 20 shows the dependence of the efficiency synchronous GEM algorithm from the number of processes and the size of the input problem. The graph shows that the efficiency of utilization of computing resources with increasing number of processes rapidly decreases for all range of input load. Synchronous algorithm use inefficiently allocated computing resources.

Fig. 21 shows the dependence of the efficiency of asynchronous GEM algorithm on the number of processes and the size of the input problem. The graph shows that the efficiency of utilization of computing resources with increasing number of processes decreases, but even at low input size asynchronous algorithm achieves better efficiency than the synchronous version.

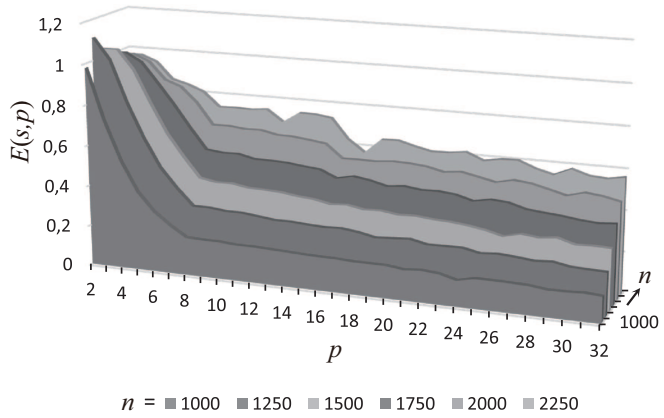

Fig. 21. Efficiency of asynchronous algorithm.

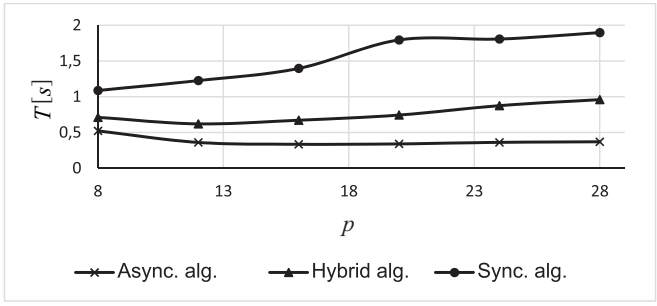

Fig. 22. Effect of change in the number of processes on computation time, $n = 1500$.

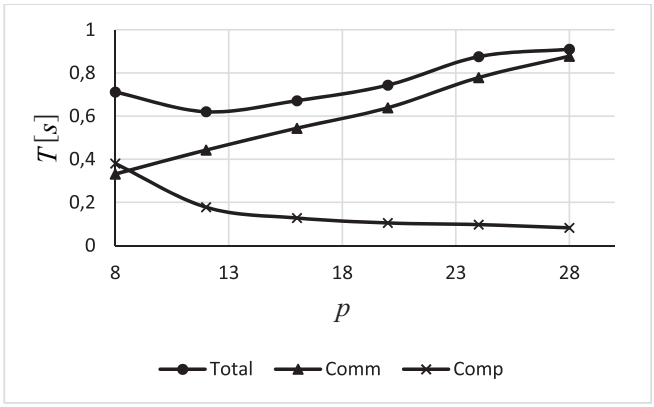

Fig. 23. The impact of computation and communications delays for a total execution time of hybrid algorithm, $n = 1500$.

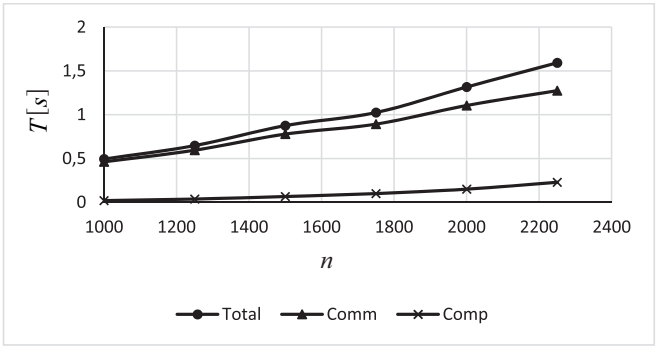

Fig. 24. The impact of computing and communication delays on the overall execution time of hybrid algorithm with increasing input load, $p = 24$.

D. Measurement of the Hybrid Algorithm

The hybrid algorithm uses a combination of environment with distributed memory along with the environment with shared memory. The measurements are focused on the comparison of the hybrid algorithm with asynchronous GEM algorithm considering that the hybrid algorithm is based precisely on this algorithm. Measurements were performed on a network consisting of 7 NOW SMP computers, each containing Intel Q6600 quad-core processor. Each process in the hybrid algorithm was divided into four threads. The graph shows the number of processes multiplied by the number of cores in the computer so that we can compare the algorithms to each other. Fig. 22 shows the growth in the number processes to the overall execution time with constant input load $n = 1500$. The hybrid algorithm shows worse results than asynchronous algorithm. This is due to the necessity initialization of the threads at each matrix transition and also their implicit synchronization at the end of each calculation cycle. The graph shows that the hybrid algorithm is more efficient than synchronous algorithm. Since the hybrid algorithm is based on asynchronous algorithm that it is more efficient than synchronous, and so the hybrid algorithm is also more efficient than synchronous algorithm.

D.1 Influence of Growth Processes

For a detailed analysis of the hybrid algorithm, it is necessary to divide the total execution time of the algorithm on partial components - t_{comm}, t_{comp}, and time initialization threads. The component of initialization time of hybrid algorithm could not be done due to the fact that this component is highly dependent on the operating system, which provides initialization and schedule them to run on the processor. Its values show great variability in comparison to the communication time. Fig. 23 shows the total time of implementation of hybrid algorithm divided into individual components.

D.2 Effect of Growth of Load

Fig. 24 illustrates the effect of increasing input load on a total execution time of the hybrid algorithm, as well as its individual components at constant number of processes $p = 28$. The increasing input load increases computation time and the amount of communication between processes. The result is that the overall execution time increases.

D.3 The Efficiency of the Use of Computing Resources

Fig. 25 shows the dependence of the efficiency of hybrid algorithm to the number of processes and the size of the input problem. The graph shows that the efficiency of computing resources utilization with an increasing number of processes decreases, but with an increased input load efficiency declines less. Compared with asynchronous algorithm the hybrid algorithm is less scalable. This is due to the fact that after computing is done the threads are merged back to the single process and this results

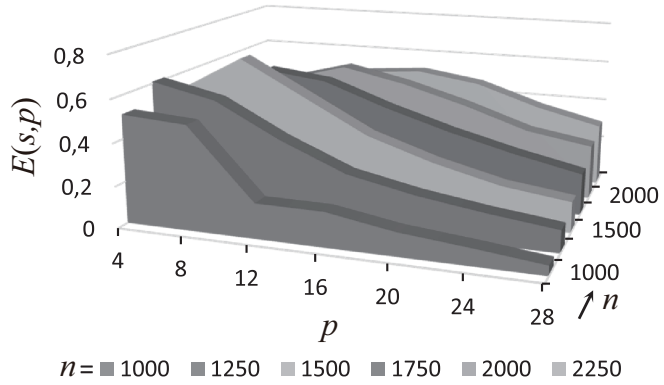

Fig. 25. Efficiency of the hybrid algorithm.

in their synchronization, which results in lower concatenation of calculations.

X. CONCLUSION

Based on the conducted experiments and by analysis of the obtained results, we can conclude that GEM synchronous algorithm from theoretical analysis of the costs of the parallel algorithm is an inefficient algorithms and is not suited for massive parallel systems. The experimental results confirmed that an increase in the number of processes, communication rapidly increases and it quickly begins to dominate among the components of the total execution time of the algorithm. Communication load becomes the dominant factor that limits the effectiveness of the algorithm. The measured efficiency also shows that this algorithm inefficient uses the allocated computing resources. Similar algorithms will benefit more from reducing synchronization, so that there was a concatenation calculations, rather than from more powerful communication networks.

GEM asynchronous algorithm by theoretical analysis of cost of parallel algorithm is one of the efficient algorithms that are suitable for massive parallel systems. The experimental results confirmed the effectiveness of the algorithm. With the increasing of input load efficiency increases significantly. From experiments the communication also appears to be the limiting factor, but the assumption is that the massive parallel system would be fully pipelined with term of calculation and so the communications will have less impact.

Computational complexity of hybrid algorithm, which is derived from the asynchronous algorithm, increases the execution time by initialization of the threads and implicit synchronization. Therefore, a hybrid algorithm is less suitable for this type of application.

The algorithm with shared memory works effectively only to a point, while the input load does not cause the need to access to global memory (cache miss).

Conclusion from the experimental results and also from the theoretical analysis of the finite methods for calculating the system of linear equations is that the best algorithm suitable for massively parallel deployment is the asynchronous implementation with use of MPI. The hybrid implementation as it was described in this article is not suitable for massive parallel deployment, because of the overhead created by starting of new threads and by implicit synchronization which increase the overall amount of synchronicity in algorithm.

Experimental results confirmed the need for more efficient communication and effective communication algorithms, which will be adapted to new architectures of parallel computers.

REFERENCES

[1] A. B. Abderazek, *Multicore Systems On-Chip Practical Software/ Hardware Design*, 2nd ed. Imperial College Press, 2013.
[2] M. Anthony and M. Harvey, *Linear Algebra: Concepts and Methods*. Cambridge University Press, 2012.
[3] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems Concepts and Design*, 5th ed. Addison Wesley, 2011.
[4] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proc. IEEE GCE*, Chicago, IL, USA, Austin, TX, 2008, pp. 1–10.
[5] F. Gebali, *Algorithms and Parallel Computing*. Wiley, 2011.
[6] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Johns Hopkins University Press, 2012.
[7] P. Hanuliak and I. Hanuliak, "Performance evaluation of iterative parallel algorithms," *Kybernetes*, 36, 2007.
[8] P. Hanuliak and I. Hanuliak, "Performance evaluation of iterative parallel algorithms," *Kybernetes*, 39:107126, 2010.
[9] A. Holubek, "Performance prediction and optimization of parallel algorithm," *InterTech*, Poznan, 2010.
[10] J. Jeffers and J. Reinders, "Intel Zeon Phi coprocessor high performance programming," *Elsevier Sci. & Technol. Books*, 2013.
[11] J. Joseph and C. Fellenstein, *Grid computing*. Prentice Hall PTR, 2004.
[12] J. Micek, M. Hyben, M. Fratrik, and J. Puchyova, "Voice command recognition in multirobot systems: Information fusion," *Int. J. Advv Robotic Syst.*, vol. 9, pp. 1–9, 2012.
[13] P. Pacheco, *An Introduction to Parallel Computing*. Morgan Kaufmann, 2011.
[14] Y. K. Penya, "Last-generation applied artificial intelligence for energy management in building automation," in *Proc. IFAC Int. Conf. Fieldbus Systems, Appl.*, 2003, pp. 79–83.
[15] P. Sevck and O. Kovar, "Very efficient exploitation of FPGA block RAM memories in the complex digital system design," *J. Inform. Control. Management Syst.*, vol. 8, pp. 403–414, 2010.
[16] D. Soudris and A. Jantsch, *Scalable Multi-Core Architectures: Design Methodologies and Tools*. Springer: New York, 2012.

Martin Húdik received his Ph.D. and Master degrees in Computer Science from the University of Žilina in 2010 and 2013, respectively. He is currently a Lecturer/Researcher at the University of Žilina. His research interests include the analysis and modeling of distributed systems, high-performance computing, intelligence in embedded systems, and sensor wireless networks. He has published in the areas of distributed computer systems modeling and optimization of performance of distributed systems, and sensor wireless networks. He is currently leading lectures at the Department of Technical Cybernetics from the area of application development for Android, distributed systems, and networking.

Michal Hodoň works as a Research Assistant at the Department of Technical Cybernetics, Faculty of Management Science and Informatics, University of ilina, Slovakia. He received Master degree at the University of Žilina in 2009 in the field of Computer engineering, followed by Ph.D. title in 2013 in the field of Applied Informatics with the dissertation thesis "Onboard localization technologies for vehicle positioning". His work is oriented on the development of different prototypes of embedded systems where different control algorithms upon various application scenarios could be investigated practically.