Send Orders of Reprints at bspsaif@emirates.net.ae

38 *The Open Cybernetics & Systemics Journal,* 2012, *6,* 38-47

Open Access

Analytical Method of Performance Prediction in Parallel Algorithms

Peter Hanuliak*

Polytechnic Institute, Dubnica nad Vahom, Dukelskastvrt 1404/61, SK - 018 41 Dubnica nad Vahom, Slovakia

Abstract: With the availability of powerful personal computers, workstations and networking devices, the recent trend in parallel computing is to connect a number of individual workstations (PC, PC SMP) to solve computation-intensive tasks in parallel way on such typical clusters as NOW, SMP, Grid). In this sense it is not more true to consider traditionally evolved parallel computing and distributed computing as two separate research disciplines. Current trends in high performance computing (HPC) are to use networks of workstations (NOW, SMP) as a cheaper alternative to traditionally used massively parallel multiprocessors or supercomputers and to profit from unifying of both mentioned disciplines. The individual workstations could be so single PC (Personal computer) as parallel computers based on modern symmetric multiprocessor systems (SMP) implemented within workstation.

To exploit the parallel processing capability of such cluster, the application program must be paralleled. The effective way to do it for (parallelisation strategy) belongs to a most important step in developing effective parallel algorithm (optimisation). For behaviour analysis we have to take into account all overheads that have the influence to performance of parallel algorithms (architecture, computation, communication etc.). In this article we discuss such complex performance evaluation of abstract empty matrix for potential used decomposition strategies. For these decomposition strategies we derived analytical isoefficiency functions, which allow us to predict performance although for hypothetical parallel computer.

Keywords: Network of workstations, decomposition strategy, inter process communication, message passing interface, performance prediction, issoeficiency function.

1. INTRODUCTION

Distributed computing using cluster of powerful workstations (NOW, SMP, Grid) was reborn as a kind of "lazy parallelism". A cluster of computers could team up to solve many problems at once, rather than one problem at higher speed. To get the most out of a distributed parallel system, designers and software developers must understand the interaction between hardware and software parts of the system. It is obvious that use of a computer network based on personal computers would be principal less effective than the used typical massively parallel architectures in the world, because of higher communication overheads, but a network of more and more powerful workstations consisting on powerful personal computers (PC, PC - SMP), belongs for the future to very cheap, flexible and perspective parallel computers. Such a trend we can see in dynamic growth just in the parallel architectures based on the networks of workstations as a cheaper and flexible architecture in comparison to conventional multiprocessors and supercomputers. Principles of these conventional parallel computers are in this time effective implemented in modern symmetric multiprocessor systems (SMP) based on the same processors [1] (multiprocessor, multicores). Unifying of both approaches (NOW, SMP) open for the future the new possibilities in massive HPC computing.

There has been an increasing interest in the use of networks of powerful workstations (NOW) connected together by high-speed networks for solving large computation-intensive problems. This trend is mainly driven by the cost effectiveness of such systems as compared to parallel computer with massive number of tightly coupled processors and memories. Parallel computing on a cluster of powerful workstations (NOW, SMP, Grid) connected together by high-speed networks have given rise to a range of hardware and network related issues on any given platform.

Network of workstations (NOW) [2-4] has become a widely accepted form of high-performance parallel computing. Basic architecture of NOW is illustrated in Fig. (**1**). As in conventional multiprocessors, parallel programs running on such a platform are often written in an SPMD form (Single program – Multiple data) to exploit data parallelism or in an improved SPMD form to take into account also the potential of functional parallelism of a given application. Each workstation in a NOW is treated similarly to a processing element in a multiprocessor system. However, workstations are far more powerful and flexible than processing elements in conventional multiprocessors.

2. PARALLEL ALGORITHMS

In principal we can divide parallel algorithms into two following classes:

*Address correspondence to this author at the Polytechnic Institute, Dubnica nad Vahom, Dukelskastvrt 1404/61, SK - 018 41 Dubnica nad Vahom, Slovakia; Tel: +421 42 4424 123; Fax: +421 42 4428436;
E-mail: phanuliak@gmail.com

1874-110X/12

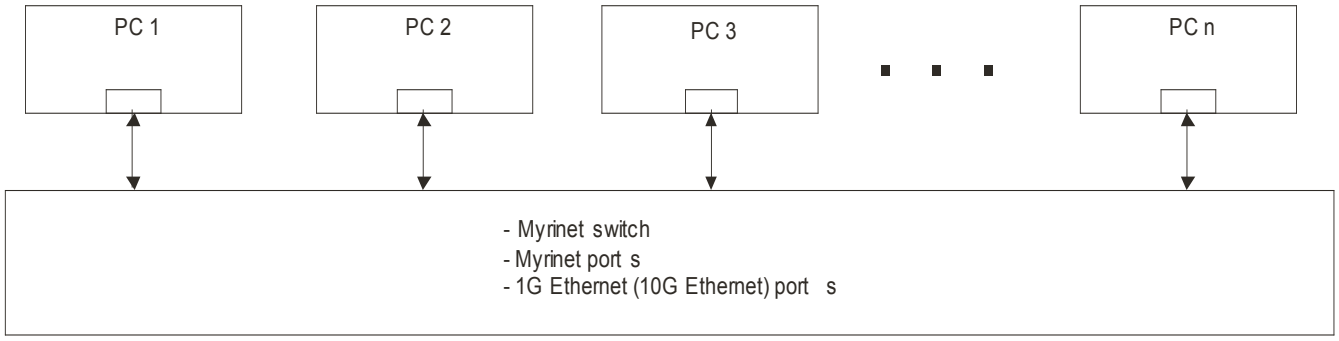

Fig. (1). Typical architecture of NOW.

- parallel algorithm using shared memory (PA). These algorithms are developed for parallel computers with dominated shared memory as actual symmetrical multiprocessors or multicore systems on motherboard (SMP)
- parallel algorithm using distributed memory (DPA). These algorithms are developed for parallel computers with distributed memory as actual NOW system and their higher integration forms named as Grid systems.

The main difference is in form of inter - process communication (IPC) among individual parallel processes [5]. Generally we can say that IPC communication in parallel system with shared memory can use more communication possibilities than in distributed systems.

2.1. Developing Parallel Algorithm

To exploit the parallel processing capability the application program must be parallelised. The effective way to do it for a particular application problem (decomposition strategy) belongs to the most important step in developing an effective parallel algorithm [5]. The development of the parallel network algorithm includes the following activities

- decomposition - the division of the application into a set of parallel processes
- mapping - the way how processes and data are distributed among the nodes
- interprocess communication - the way of corresponding and synchronisation among individual processes
- tuning - alternation of the working application to improve performance (performance optimisation).

The most important step is to choose the best decomposition method for given application problem. To do this it is necessary to understand concrete application problem, data domain, used algorithm and flow of control in given application. When designing a parallel program the description of the high-level algorithm must include, in addition to design a sequential program, the method you intend to use to break the application into processes (decomposition strategy) and distribute data to different computing nodes (mapping). The chosen decomposition method drives the rest of program development. This is true is in case of developing new application as in porting serial code. The decomposition method tells us how to structure the code and data and defines the communication topology [2, 6].

2.2. Methods of Performance Evaluation

To performance evaluation of parallel algorithms we can use analytical approach to get under given constraints analytical laws or some other derived analytical relations. Theoretically we can use following solution methods to get a function of complex performance

- analytical
  - application of queuing theory results [7]
  - order (asymptotic) analyse [8-10]
  - Petri nets [11]
- simulation methods [12]
- experimental
  - benchmarks [13]
  - direct measuring [14].

Analytical method is a very well developed set of techniques which can provide exact solutions very quickly, but only for a very restricted class of models. For more general models it is often possible to obtain approximate results significantly more quickly than when using simulation, although the accuracy of these results may be difficult to determine.

Simulation is the most general and versatile means of modelling systems for performance estimation. It has many uses, but its results are usually only approximations to the exact answer and the price of increased accuracy is much longer execution times. They are still only applicable to a restricted class of models (though not as restricted as analytic approaches.) Many approaches increase rapidly their memory and time requirements as the size of the model increases.

Evaluating system performance *via* experimental measurements is a very useful alternative for computer systems. Measurements can be gathered on existing systems by means of benchmark applications that aim at stressing specific aspects of computers systems. Even though benchmarks can be used in all types of performance studies, their main field of application is competitive procurement

and performance assessment of existing systems and algorithms.

3. PERFORMANCE EVALUATION METRICS

To evaluate parallel algorithms there have been developed several fundamental concepts. Tradeoffs among these performance factors are often encountered in real-life applications.

3.1. Speed Up

Let T(s, p) be the total number of unit operations performed by p processor system, s defines size of the computational problem. Then T(s, 1) defines execution time units for one processor system. Then speedup factor is defined as

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

It is a measure of the speedup factor obtained by given algorithm when p processors are available for the given problem size s. Since S (s, p) ≤ p, we would like to design algorithms that achieve S(s, p) ≈ p.

3.2. Efficiency Concept

The system efficiency for processor system with p computing nodes is defined by

$$E(s,p) = \frac{S(s,p)}{p} = \frac{T(s,\ 1)}{p\ T(s,p)}$$

A value of E(s, p) approximately equal to 1 for some p, indicates that such a parallel algorithm, using p processors, runs approximately p times faster than it does with one processor (Sequential algorithm).

3.3. The Isoefficiency Concept

The workload w of an algorithm often grows in the order O (s), where symbol O means the used upper limit in complexity theory and its parameter s is the problem size. Thus, we denote the workload w = w(s) as a function of s. In parallel computing is very useful to define an isoefficiency function relating workload to machine size p needed to obtain a fixed efficiency when implementing a parallel algorithm on a parallel system.

Let h(s, p) be the total overhead involved in the algorithm implementation. This overhead is usually a function of both machine size and problem size. The workload w(s) corresponds to useful computations while the overhead h (s, p) are useless times attributed to architecture, parallelisation, synchronisation and communication delays. In general, the overheads increase with respect to increasing values of s and p. Thus the efficiency is always less than 1. The question is hinged on relative growth rates between w(s) and h(s, p). The efficiency of a parallel algorithm is thus defined as

$$E(s,p) = \frac{w(s)}{w(s) + h(s,p)}$$

With a fixed problem size the efficiency decreases as p increase. The reason is that the overhead h(s, p) increases with p. With a fixed machine size, the overload grows slower than the workload. Thus the efficiency increases with increasing problem size for a fixed-size machine. Therefore, one can expect to maintain a constant efficiency if the workload is allowed to grow properly with increasing machine size. For a given algorithm, the workload might need to grow polynomial or exponentially with respect to p in order to maintain a fixed efficiency. Different algorithms may require different workload growth rates to keep the efficiency from dropping, as p is increased. The isoefficiency functions of common parallel algorithms are polynomial functions of p, i. e. they are $O(p^k)$ for some $k \geq 1$. The smaller the power of p in the isoefficiency function the more scalable the parallel system. We can rewrite equation for efficiency E(s, p) as

$$E(s, p) = 1 / (1+ (h (s, p) / w(s))$$

In order to maintain a constant E, the workload w(s) should grow in proportion to the overhead h(s,p). This leads to the following relation

$$w(s) = \frac{E}{1-E} h(s,p)$$

The factor

$$C = E / 1\text{-}E$$

is a constant for a fixed efficiency E. Thus we can define the isoefficiency function as follows

$$w (s, p) = C. h (s, p)$$

If the workload grows as fast as w (s, p) then a constant efficiency can be maintained for a given parallel algorithm.

4. MODELLING OF COMPLEXITY IN PARALLEL ALGORITHMS

To this time known results in complexity modelling on the in the world used classical parallel computers with shared memory (supercomputers, SMP and SIMD systems) or distributed memory (Cluster, NOW, Grid) mostly did not consider the influences of the parallel computer architecture and communication overheads supposing that they are lower in comparison to the latency of executed massive calculations [1, 9].

In this sense, analysis and modelling of complexity in parallel algorithms (PA) are rationalised to the analysis of complexity of own computations, that mean that the function of control and communication overheads are not a part of derived relations for computation time T (s, p). In general computation time of sequential and parallel algorithms given through multiplicity product of algorithm complexity Z_a(dimensionless number of performed computations steps) and a constant t_c as an average value of performed computation operations.

In this sense the function in the relation for isoefficiency suppose, that dominates influence to the overall complexity of the parallel algorithms has complexity of performed massive calculations. Such assumption has proved to be true in using classical parallel computers in the world (Supercomputers, massively multiprocessors – shared

memory, SIMD architectures etc.). To map mentioned assumption to the relation for asymptotic isoefficiency w(s) means that

$$w\ (s) = \max\ \left[T_{comp}\ , h\ (s,p) < T_{comp}\ \right] = \max\ \left[T_{comp}\right]$$

In opposite at parallel algorithms for the actually dominant parallel computers on the basis NOW (including SMP systems) and Grid is for complexity modelling necessary to analyse at least most important overheads from all existed overheads which are [15, 16]

- architecture of parallel computer (T_{arch})
- own calculations (T_{comp})
- communication latency (T_{comm})
  - start - up time (t_s)
  - data unit transmission (t_w)
  - routing
- parallelisation latency (T_{par})
- synchronisation latency (T_{syn}).

Taking into account all this kinds of overheads the total parallel execution time is

$$T\ (s,p)_{complex} = \sum \left(T(s,p)_{comp}\ , T(s,p)_{arch}\ , T(s,p)_{par}\ , T(s,p)_{comm}\ , T(s,p)_{syn}\right)$$

where $T(s, p)_{arch}$, $T(s, p)_{comp}$, $T(s, p)_{par}$, $T(s, p)_{comm}$, $T(s, p)_{syn}$ denote the individual overheads caused by architecture, computations, parallelisation, communication and synchronisation. The more important overhead parts build in the relation for isoefficiency the used the overhead function h (s, p), which influence in general is necessary to take into account in performance modelling of parallel algorithms.

$$h\ (s,p) = \sum \left(T(s,p)_{arch}\ , T(s,p)_{par}\ , T(s,p)_{comm}\ T(s,p)_{syn}\right)$$

The first part of h(s, p) function T_{parch} (s, p) (architecture influence of used parallel computer) is projected into used technical parameters t_c, t_s, t_w, which are constant for given parallel computer.

The second part of h(s, p) function T_{par} (s, p) (parallelisation of solved problem) depend from chosen decomposition strategy and their consequences are projected so to computation part T_{comp} (s,p) (increased computation complexity) as to communication part T_{comm} (s,p) (increased number of performed communications)

The third part of h(s, p) function T_{syn} (s, p) we can eliminate through optimisation of load balancing among individual computing nodes of used parallel system. For this purpose we would measure performance of individual computing nodes for given developed parallel algorithm and then based on measured results better redistribute input load. This activity we can repeatuntil we have optimal redistributed input load (optimal load redistribution based on real performance results).

In general nonlinear influence of h (s, p) could be in performance modelling of parallel algorithms (Fig. **2**). Then for asymptotic isoefficiency analysis is true

$$w\ (s) = \max\ \left[T\ (s,p)_{comp}\ , h\ (s,p)\right]$$

where the most important parts for dominant parallel computers (NOW, Grid) in overhead function h (s, p) is the influence of T_{comm} (Communication overheads).

Calculation time $T\ (s, p)_{comp}$ of parallel algorithm is given through quotient of sequential running time (Complexity product of sequential algorithm Z_{sa} and a constant t_c as an average value of performed computation operations) through number of used calculation nodes of the given parallel computer. Parallel calculation complexity of T(s,p) as a limit of a theoretical unlimited number of calculation nodes is given as

$$T(s,\ p)_{comp} = \lim_{p\to\infty} \frac{Z_{sa}.t_c}{p} = 0$$

Finally the assumed relation between $T\ (s, p)_{comp}$ and h (s, p) illustrates Fig. (**2**). For effective parallel algorithms we are seeking for the bottom part of whole execution time according Fig. (**2**).

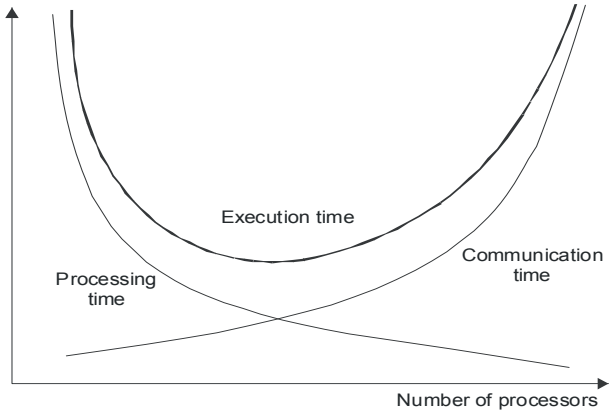

Fig. (2). Relations among parts of parallel execution time.

In relation to previous steps the kernel of asymptotic analyse of h(s, p) is analysis of communication part T_{comm} (s,p) including projected consequences of used decomposition methods. This analysis comes out from application isoefficiency concept to derived communication complexity in an analytical way for used decomposition strategies. In general derived isoefficiency function w(s) could have non-linear character at gradually increasing number of computing nodes. Analytical deriving of isoefficiency function w(s) allow us performance prediction of given parallel algorithm so for existed real as for hypothetical parallel system. So we have possibility to consider potential efficiency of given algorithm. Communication time $T\ (s, p)_{comm}$ is given through number of performed communication for considered decomposition strategy. Every communication within NOW is characterised

through two following communication parameters, which are illustrated at Fig. (**3**).

- t_s is defined parameter for communication initialisation
- t_w is defined parameter for data unit latency.

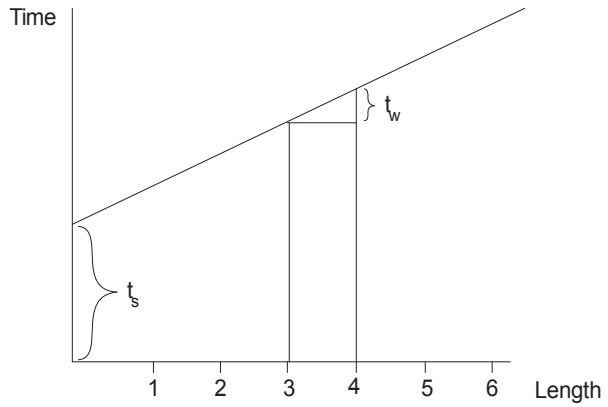

Fig. (3). Illustration of basic communication parameters.

These communication parameters are constant for concrete parallel system. At Table **1** we have illustrated some typical parallel computer examples with concrete values for their basic communication parameters.

Table 1. Technical Parameters of Parallel Computers (PC)

PC	t_s [µs]	t_w [µs]
IBM SP2	35	0,23
Intel DELTA	77	0,54
Intel Paragon	64	0,07
Meiko CS-2	87	0,08
NCUBE-2	154	2,40
ThinkingMachine CM-5	82	0,44
NOW on FDDI	1 150	1,10
NOW on Ethernet	1 500	5,00
Cray T3E	3	0,063

Communication overheads are given through two basic following components

- $f_1(t_s)$ function as the whole number of initialisation of performed communication
- $f_2(t_w)$ as function of whole performed data unit transmission usually time of word transmission for given parallel computer.

These two components limited performance of used parallel system based on NOW. Illustration of these communication parameters is at Fig. (**3**). These parameters are when using superposition we can write

$$T_{comm}(s,p) = f_1(t_s) + f_2(t_w)$$

Communication time $T(s, p)_{comm}$ is given through the number of performed communication operations in concrete parallel algorithm and depends from used decomposition model. To the practical illustration of communication overheads we used the possible matrix decomposition models.

For more complex parallel system named as Grid (network of NOW) accesses third component ($f_3(t_h)$, whichdetermine potential multiple crossing used NOW networks. This component is characterised through multiplying $hops l_h$among NOW networks and average latency time jumped NOW networks (NOW networks with the same communication speed) or a sum of individual latencies for jumped NOW networks (NOW networks with different same communication speed). Then for whole latency in Grid is valid

$$T_{comm}(s,p) = f_1(t_s) + f_2(t_w) + f_3(t_s, l_h).$$

In general latency $f_3(t_s, l_h)$, that is time to send message with m words among NOW networks, which have $l_h$hopsis given $as t_s + l_h t_h\, m\, t_w$, where the new parameters are

- l_h is the number of network hops
- m is the number of transmitted data units (usually words)
- $t_h$is average communication time for one hop (we suppose the same communication speed).

The new parameters t_h, l_h depend from a concrete architecture of Grid communication network and used routing algorithm in Grid.

For the analyse purposes it is necessary to derive for given parallel algorithm or a group of similar algorithms (in our case matrix algorithms) derived necessary communication functions and that always for given decomposition strategy) isoefficiency functions and basic constants for used parallel computer (NOW, Grid).

5. MATRIX MODELS

For analysing isoefficiency function of parallel algorithms, which are using matrix, we would analyse communication models in abstract form that means for empty matrix. Then we are considering the typical following n x m matrix

$$A = \begin{pmatrix} a_{11}, & a_{12}, & \ldots & , a_{1n} \\ a_{21}, & a_{22}, & \ldots & , a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1}, & a_{m2}, & \ldots & , a_{mn} \end{pmatrix}$$

To reduce number of variables in deriving process of issoefficiency function we will consider matrix with m = n (square matrix). For this purpose there are also following causes.

- we can transform any matrix n x m to n x n matrix through expanding rows (if m < n) or columns (if m > n)
- derivation process will be the same only when considering the workload instead of n^2 (square matrix) we should consider n x m (oblong matrix).

5.1. Decomposition Matrix Models

In general square matrix n x n in halts n^2 elements. In order to achieve effective parallel algorithm it is necessary to map every parallel process more than one matrix element. Then for mapping a cluster of matrix elements there are in principal two ways

- mapping of p columns or p rows as illustrated at Fig. (**4**).
- mapping of square blocks to every parallel process as illustrated at Fig. (**5**).

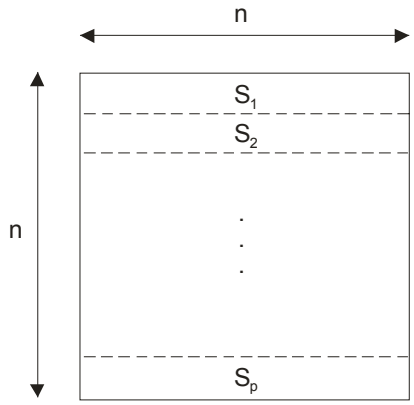

Fig. (4). Decomposition strategy to strips (rows).

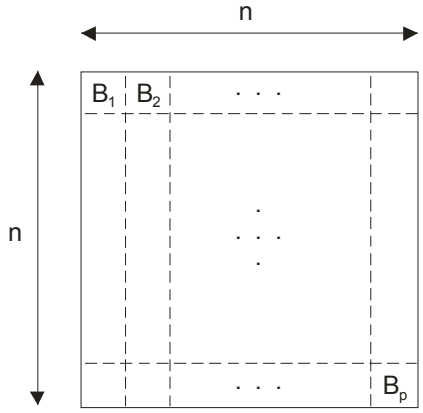

Fig. (5). Decomposition strategy to blocks.

5.1.1. Matrix Decomposition to Strips

Depending of used decomposition methods there are derived needed communication activities. In general square matrix in two dimensions n x n in halts n^2 elements. To decompose such matrix we map one or more matrix strips (parallel process) to computing node of the used parallel system. So in this way square matrix is equally divided to p build parallel processes, that mean every parallel process get n/ p rows.

For mapping matrix elements in strips (rows) inter process communication is performed on two neighbouring strips (Fig. **6**). So it is necessary in computation flow to exchange values of all board rows. Every parallel process therefore sends message with n matrix elements (row) and in the same way they receive message with n matrix elements (row) from its neighbouring row supposing that all data in row (n matrix elements) are sent as a part of any sending or receiving message.

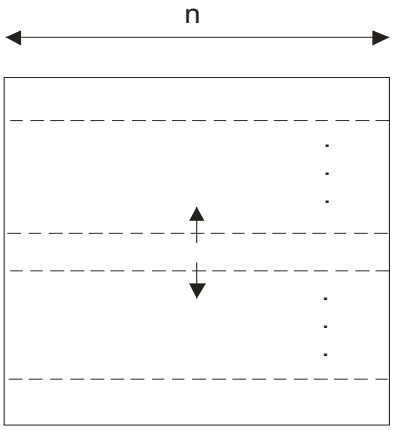

Fig. (6). Communication consequences for decomposition to strips (rows).

The requested communication time for decomposition method to strips (rows) is given according illustration at Fig. (**6**) as

$$T(s,\ \mathrm{p})_{comm} = T_{coms} = h(s,p) = 4\ (\mathrm{t_s} + n\ \mathrm{t_w})$$

And the derived final relations for performance evaluation as following

- execution time of sequential square matrix algorithm T (s, 1)

$$T(s,\ 1) = n^2\ t_c$$

- for computation execution time of parallel algorithm T (s, p) including overhead function

h (s, p)

$$T(s,p) = T(s,p)_{comp} + h(s,p) = \frac{n^2 t_c}{p} + 4\left(t_s + n\ t_w\right)$$

- parallel speed up S (s, p)

$$S(s,p) = \frac{T(s,1)}{T(s,p)} = \frac{n^2\ p\ t_c}{n^2 t_c + \ 4\ p\ (\mathrm{t_s} + n\ \mathrm{t_w})}$$

- efficiency E (s, p)

$$E(s,p) = \frac{S(s,p)}{p} = \frac{n^2\ \ t_c}{n^2 t_c + \ 4\ p\ (\mathrm{t_s} + n\ \mathrm{t_w})}$$

- constant C (this constant we need in process of deriving an isoefficiency function w (s))

$$\frac{E}{1-E} = C = \frac{n^2 t_c}{4\ \mathrm{p}\ (t_s + n\ t_w)}$$

5.1.1.1. Deriving Issoeficiency Functions (Scalability)

To deriving issoeficiencyfunction w (s) we used the general defined criterion. For their asymptotic magnitude is valid

$$w\ (s) = \max\left[\ T(s,p)_{comp},\ \text{h (s,p)}\right]$$

Input load complexity of sequential analysed matrix is given as w (s) = $n^2 t_c$. The defined criterion of efficiency E (s, p) allow us to derive final dependency for constant efficiency as

$$\frac{E}{1-E} = C = \frac{n^2 t_c}{4\ \text{p}\ (t_s + n\ t_w)}$$

To win a closed form of issoeficiency function we have used an approach in which we performed at first the analysis of increasing input load influenced the analysed expression contained $t_s$in relation to p so to keep this growth constant (we supposed thatt_w = 0). Then we get

$$\frac{E}{1-E} = C = \frac{n^2 t_c}{4\ p\ t_s}$$

From this expression for the wished function w (s) = n^2 t_c we get

$$w(s) = n^2 = 4\ C\ p\ \frac{t_s}{t_c}$$

With a similar approach we have been analysed the influence growth of input load caused the another part of expression from$t_w$in relation to p so to keep this growth constant (we supposed that t_w = 0). Then we get

$$\frac{E}{1-E} = C = \frac{n^2 t_c}{4\ \text{n}\ p\ t_w}$$

From this expression for the wished function w (s) = n^2 t_c we get

$$w(s) = n^2 = 4\ C\ n\ p \frac{t_w}{t_c}$$

The whole asymptotic isoefficiency function w (s)for assumed decomposition to matrix strips is given as

$$w\ (s)_{strips} = \max\left[\frac{n^2 t_c}{p},\ 4\ C\ p\ \frac{t_s}{t_c}, 4\ C\ n\ p\ \frac{t_w}{t_c}\right]$$

Based on analysis of computer technical parameters t_s, t_w, $t_c$for some parallel computers in the world they are valid following inequalitiest_s>>t_w>t_c. Alike is valid that p ≤ n. Using these inequalities it is necessary to analyse dominancy influence of the all derived expressions. From this analysis comes out that

- a value of first expression depends of both variables n, p. For given values of n (p≤n) is the value of first expression a decreasing function of p (parallel computation)
- a second expression is a growing function of variables n, p
- a third expression is a growing function of variables n, p where a size of input load is given as s = n^2 and from a size of parallel system p.

Then the asymptotic issoefficiency function is limited through dominancy conditions of second and third expressions. From their comparison comes out

- based on real conditiont$_w$ ≥ t_sa third expression is bigger or equal than a second expression and a issoefficiency function is limited through the third expression (for assumed technical parameters t_s = 35 μs, t_w= 0,23 μs) is the settled condition valid for n ≥ 152
- for n < 152 and for technical constants t_s = 35 μs, t_w= 0,23 μsissoefficiency function is limited through a second expression
- a third expression depends on the variable n (p ≤ n), that is for n ≥ 152 issoefficiency function is given for individual given value of efficiency E (s, p) for a needed value of parameter n (p ≤ n).

5.1.2. Matrix Decomposition to Blocks

For mapping matrix elements in blocks a inter process communication is performed on the four neighbouring edges of blocks (Fig. 7), which it is necessary in computation flow to exchange. Every parallel process therefore sends four messages and in the same way they receive four messages at the end of every calculation step supposing that all needed data at every edge are sent as a part of any message).

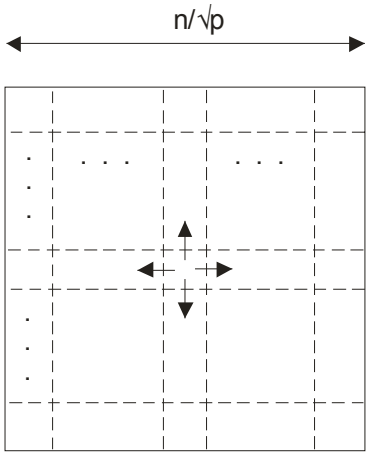

Fig. (7). Communication consequences for decomposition to blocks.

Then the requested communication time for this decomposition method is given as

$$T_{comb} = 8\ (t_s + \frac{n}{\sqrt{p}}\ t_w)$$

This equation is correct for p ≥ 9, because only under this assumption it is possible to build at least one square because only then is possible to build one square block with for communication edges. Using these variables for the communication overheads in decomposition method to blocks is correct

$$T(s,\ \mathrm{p})_{comm} = T_{comb} = h(s,p) = 8\,(t_s + \frac{n}{\sqrt{p}}\,t_w)$$

- for computation execution time of parallel algorithm T (s, p) including overhead function h (s, p)

$$T(s,\ \mathrm{p}) = T(s,p)_{comp} + h(s,p) = \frac{n^2.t_c}{p} + 8\,(t_s + \frac{n}{\sqrt{p}}\,t_w)$$

- parallel speed up S (s, p)

$$S(s,p) = \frac{T(s,1)}{T(s,p)} = \frac{n^2\ p\ t_c}{n^2 t_c + \ \ 8\ \ (\mathrm{p}\ t_s + \sqrt{p}\ n\ t_w)}$$

- efficiency E (s, p)

$$E(s,p) = \frac{S(s,p)}{p} = \frac{n^2\ t_c}{n^2 t_c + \ \ 8\ \ (\mathrm{p}\ t_s + \sqrt{p}\ n\ t_w)}$$

- constant C (this constant we need in process of deriving an isoefficiency function w (s))

$$\frac{E}{1-E} = C = \frac{n^2 t_c}{8\ \ (\mathrm{p}\ t_s + \sqrt{p}\ n\ t_w)}$$

5.1.2.1. Isoeffectivity Function

In the process of deriving needed isoeffectivity function we come out from derived function h(s, p) for analysed decomposition method as h (s,p) = T_{comb} according the relation

$$T_{comb} = 8\,(t_s + \frac{n}{\sqrt{p}}\,t_w)$$

After appropriate modifications we get for isoefficiency following final relations

$$w\ (s)_{blocks} = \max\left[\frac{n^2 t_c}{p},\ \ 8\,C\ p\,\frac{t_s}{t_c}, 8\,C\ n\,\sqrt{p}\,\frac{t_w}{t_c}\right]$$

5.1.3. Comparison and Evaluation of Isoefficiency

The whole asymptotic isoefficiency function w (s) for assumed decomposition to matrix strips is given as

$$w\ (s)_{strips} = \max\left[\frac{n^2 t_c}{p},\ \ 4\,C\ p\,\frac{t_s}{t_c}, 4\,C\ n\ p\,\frac{t_w}{t_c}\right]$$

The whole asymptotic isoefficiency function w (s) for assumed decomposition to matrix strips is given as

$$w\ (s)_{blocks} = \max\left[\frac{n^2 t_c}{p},\ \ 8\,C\ p\,\frac{t_s}{t_c}, 8\,C\ n\,\sqrt{p}\,\frac{t_w}{t_c}\right]$$

For p≥ 9 (condition to build a block)the third expression we substitute with the expression, which is greater and is a part of derived asymptotic isoefficiency function w (s) decomposition to matrix strips. After this substitution we get

$$w\ (s)_{blocks} = \max\left[\frac{n^2 t_c}{p},\ \ 8\,C\ p\,\frac{t_s}{t_c}, 4\,C\ n\ p\,\frac{t_w}{t_c}\right]$$

As we compare the second and the third expressions we get the condition under which is the third expression greater than the second one

$$n \geq 2\ t_s / t_w$$

that is for technical parameters t_s = 35 μs, t_w= 0,23 μs is this condition valid for n ≥ 304. Generally at validity of this condition for given values t_c, t_s, t_w, is issoeficiency function so for strips as blocks given as

$$w\ (s)_{blocks,strips} = 4\,C\ n\ p\,\frac{t_w}{t_c}$$

From the comparisons of both result expression for strips and blocks result following results

- the first expressions in both derived issoeficiency function are logically the same, because the respond to parallel computation part T_{comp} (s, p), which could be dominant only for small values of p (the derived limit for this expression for unlimited growth of p is zero
- from the comparison of both second expressions comes out that the second expression of isoefficiency to blocks is greater than corresponding expression for strips
- from the comparison of both third expressions comes out that the third expression of isoefficiency to strips is greater than corresponding expression for blocks.

The derived relations for issoeficiency function of matrix algorithms mean that for given value of efficiency E (s, p) is their quotient E / 1 - E in the relation for isoefficiency constant (named as C) and the isoefficiency function is the function of two variables n and p.

The parameters t_c, t_s, $t_w$are for given parallel computer architecture constants, which are known for typical parallel computers architectures or we can determine them through experimental measurements (parallel computer technical parameters).

Then for the given concrete value of E (s, p) and for given values of parameters p, n we can in analytical way the thresholds, for which growth of isoefficiency function means decreasing of efficiency of given parallel algorithm with assumed typical decomposition strategies. This means the minor scalability of the assumed algorithm. In case of decomposition strategy the approach is similar to analysed practical used decomposition matrix strategies.

6. RESULTS

From achieved results we illustrate at Fig. (**8**) isoefficiency functions for individual constant values of efficiency (E = 0,1 až 0,9) for n < 152 using the published technical parameters t_c, t_s, $t_w$communication constants of used NOW (t_c= 0,021 μs, t_s = 35 μs, t_w= 0,23 μs,).

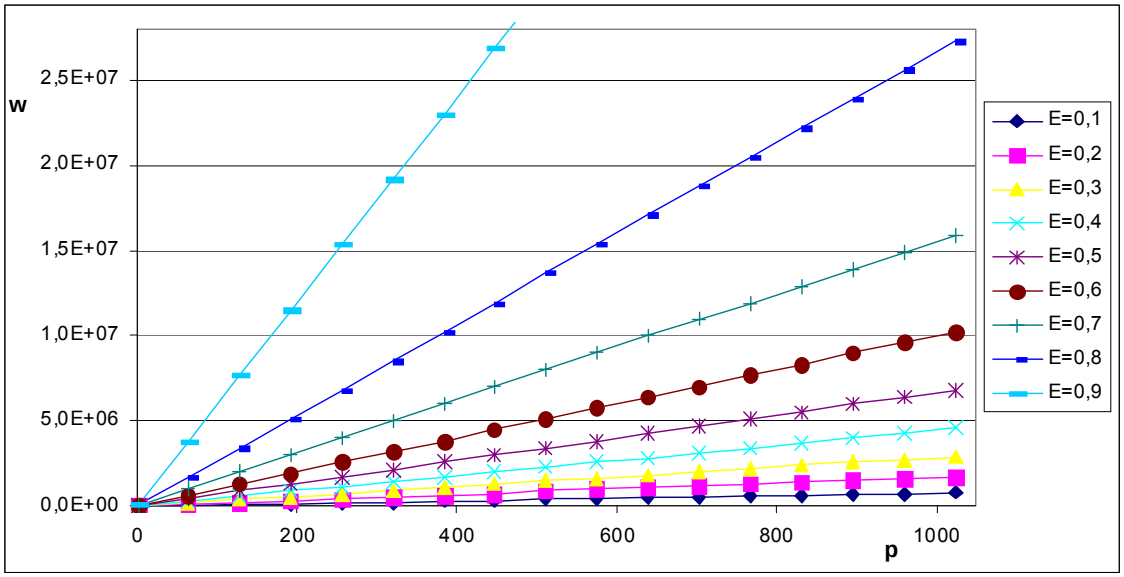

Fig. (8). Isoefficiency functions for n < 152.

At Fig. (**9**), we have illustrated isoefficiency functions for individual constant values of efficiency (E = 0,1 to 0,9) for n = 1024 and for communication parameters of parallel computer Cray T3E (t_c= 0,011µs, t_s = 3 µs, t_w= 0,063 µs,).

From both pictures (Figs. **8**, **9**) we can see that to keep a given value of efficiency we need step by step increasing number of computing processors and higher value of workload (useful computation) to balance higher communication overheads.

7. CONCLUSIONS

Performance evaluation as a discipline has repeatedly proved to be critical for design and successful use of operating systems. At the early stage of design, performance models can be used to project the system scalability and evaluate design alternatives. At the production stage, performance evaluation methodologies can be used to detect bottlenecks and subsequently suggests ways to alleviate them. Queuing networks and Petri nets models, simulation, experimental measurements, and hybrid modelling have been successfully used for the evaluation of system components. *Via* the extended form of isoefficiency concept for parallel algorithms we illustrated its concrete using to predicate the performance in typical matrix parallel algorithms. Based on derived issoeficiency function for matrix model the paper deals with the actual role of performance prediction in parallel algorithms.

To derive isoefficiency function in analytical way it is necessary to derive al typical used criterion for performance evaluation of parallel algorithms including their overhead

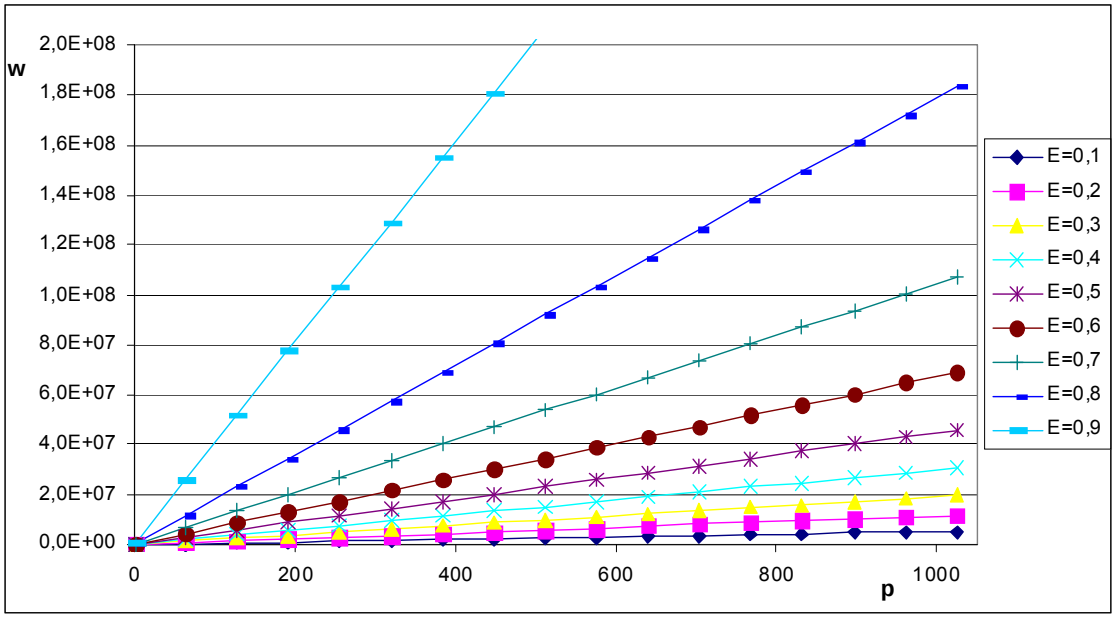

Fig. (9). Isoefficiency functions (n = 1024).

function (parallel execution time, speed up, efficiency). Based on this knowledge we are able to derive issoefficiency function as real criterion to evaluate and predict performance of parallel algorithms also oh the hypothetical parallel computers. So in this way we can say that this process includes complex performance evaluation including performance prediction.

Due to the dominant using of parallel computers based on the standard PC in form of NOW and their massively integration named as Grid (integration of many NOW), there has been great interest in performance prediction of parallel algorithms in order to achieve optimised parallel algorithms (effective parallel algorithms). Therefore this paper summarises the used methods for complexity analysis which can be applicable to all types of parallel computers (supercomputer, NOW, Grid). Although the use of NOW and Grid parallel computers should be in some parallel algorithms less effective than the in the world used massively parallel architectures (supercomputers) the parallel computers based on NOW and Grid belong nowadays to dominant parallel computers.

CONFLICT OF INTEREST

The author confirms that this article content has no conflicts of interest.

ACKNOWLEDGEMENTS

This work was done within the project Modelling, optimisation and prediction of parallel computers and algorithms at University of Zilina. The author gratefully acknowledges the crucial help of Prof. Ing. Ivan Hanuliak, PhD. as the supervisor of this project.

REFERENCES

[1] D. B. Kirk and W. W Hwu, "Programming massively parallel processors, Morgan Kaufmann," CUDA C Programming Guide 3.2, pp. 273-280, 2010.

[2] A. J. Barria, *Communication network and computer systems*. Imperial College Press: London, p. 276, 2006.

[3] J. Hanuliak and M. Hanuliak, "*Analytical modelling of distributed computer systems*," In: *TRANSCOM.* Žilina, Slovak Republic, 2005, pp. 103-110,

[4] D. A. Paterson and J. L. Hennessy, "Computer organisation and design," Morgan Kaufmann: USA, 2009, p. 12,

[5] P. Hanuliak and I. Hanuliak, "*Performance evaluation of iterative parallel algorithms*," Kybernetes: United Kingdom. vol. 39, no. 1, 2010, pp. 107-126.

[6] A. Kumar, D. Manjunath and J. Kuri, "*Communication Networking,*" Morgan Kaufmann: USA, 2004, p.750.

[7] M. Hanuliak, "To modelling of parallel computer systems," In : TRANSCOM 2007 – Section 3 (*7-th Europ. Conf. in Transport and Telecommun.*), Žilina, Slovak Republic, 2007, pp. 67-70.

[8] S. Arora, *Computational Complexity*, Cambridge University Press: United Kingdom, p. 594, 2009.

[9] J. Hanuliak, "*To performance evaluation of parallel algorithms in NOW*," Communications, The University of Zilina: Slovak Republic vol. 4, pp. 83-88, 2003.

[10] M. Hudik, *Performance optimization of broadcast collective operation on multi-core cluster*. ICSC Leden Kunovice: Czech Republic, pp. 51-55, 2012.

[11] J. Hillston, *A compositional approach to performance modelling*. University of Edinburgh, Cambridge University Press: United Kingdom, 2005, p. 172.

[12] A. Kostin and L. Ilushechkina, *Modelling and simulation of distributed systems*, Imperial College Press: London, p. 440, 2010.

[13] P. Fortier and M. Howard, *Computer system performance evaluation and prediction*. Digital Press: London, p. 544, 2003. E. Gelenbe, *Analysis and synthesis of computer systems*. Imperial College Press, p. 324, 2010.

[14] D. J. Lilja, *Measuring Computer Performance*. University of Minnesota, Cambridge University Press: United Kingdom, p. 280, 2005.

[15] T. A. Davis, *Direct methods for sparse Linear Systems*. Cambridge University Press: United Kingdom, pp.184, 2006.

[16] L. Wang, W. Jie and J. Chen, "Grid Computing: infrastructure, service, and application." CRC Press: USA, 2009.

Received: July 9, 2012 Revised: September 25, 2012 Accepted: October 1, 2012