# TO PREDICTION OF PERFORMANCE IN PARALLEL ALGORITHMS

[a]FILIP JANOVIČ, [b]PETER HANULIAK

[a]*University Žilina, Slovakia, email: filip@janovic.sk*
[b]*Polytechnic institute, Dubnica nad Vahom, Slovakia, email: phanuliak@googlemail.com*

Abstract: With the availability of powerful PC and networking devices, the recent trend in parallel computing is to connect a number of individual workstations to solve computation intensive tasks in parallel way on connected class of workstations. Current trends in high performance computing are to use networks of workstations as a cheaper alternative to traditionally used massively parallel multiprocessors or supercomputers. The individual workstations as parallel computers based on modern symmetric multicore implemented within workstation. To exploit the parallel processing capability of such cluster, the application program must be paralleled. On application ex. we demonstrate the various influences in process of performance prediction modelling and the consequences for their parallel implementations.

Keywords: network of workstations, parallel algorithm, performance modelling, performance prediction, isoefficiency, communication overheads.

## 1. Introduction

There has been an increasing interest in the use of networks of workstations (NOW) connected together by high speed networks for solving large computation intensive problems [9, 12]. Their typical architectures are at Fig.1. and its alternative modification at Fig 2.

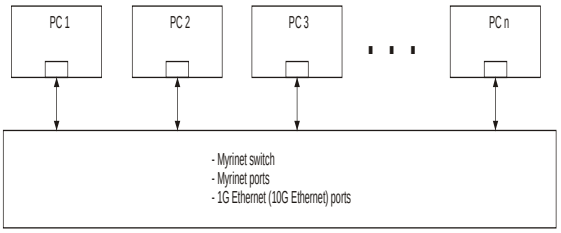


*Fig.1. Typical architecture of NOW.*

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 connected together by high speed networks have given rise to a range of hardware and network related issues on any given platform.

Effective use of these parallel computers requires a detailed understanding of the new complexities of parallel programming. Performance bottlenecks will be many times larger than was the case for the smaller parallel machines that have been in use last decade. The increased complexity of more complex NOW (single PC workstations, symmetrical multiprocessor workstation – SMP) makes it difficult to accurately predict execution time for a particular application.

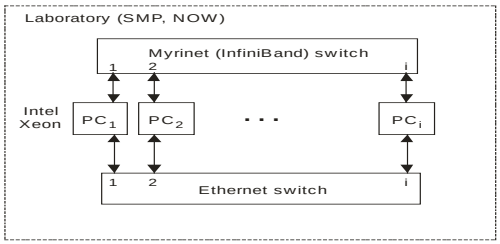


*Fig.2. Alternate architecture of NOW.*

It has been therefore difficult to develop simple formulations to predict the execution time of parallel programs due to the complexity of characterising parallel hardware and software. In an attempt to clarify these characterisations, we introduce a methodology for applying a simple prediction performance model based on isoefficiency concept law. Our formulation results in predictions of execution time not only on available systems but also for theoretical parallel systems. This allows programmers to select the optimal number of processors to apply to a particular problem or to select an appropriate problem size for the number of processors available. In short, we are able to quantify the scalability of a specific algorithm when it is run on a specific or on theoretical parallel computer. Our results illustrate key performance limitations of parallel systems, showing the impact of overhead and the scaling of problem size.

## 2. Parallel algorithms

In principal we can divide parallel algorithms into two following classes

- parallel algorithm using shared memory. These algorithms are developed for parallel computers with dominated shared memory as actual symmetrical multiprocessors or multicore systems (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. Generally we can say that IPC communication in parallel system with shared memory can use more 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 how to do it for a particular application problem (Decomposition strategy) belongs to the most important step in developing a 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
- inter-process 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 the concrete application problem, the data domain, the used algorithm and the 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 nodes (mapping).

## 3. Performance modelling

To performance evaluation of parallel algorithms we can use analytical approach to get under given constraints analytical laws or some other derived analytical relations. The most known analytical relations have been derived without considering architecture and communication complexity. That means a performance $P \approx f$ (computation). Such assumptions could be real in many cases in existed massively multiprocessor systems in the world but not in NOW and Grid.

In NOW we have to take into account all aspects that are important for complex performance evaluation according the relation $P \approx f$ (architecture, computation, communication, synchronisation etc.). Theoretically we can use following solution methods to get a function of complex performance

Quantitative evaluation and modelling of hardware and software components of any parallel systems are critical for the delivery of complexity and high performance of used parallel

algorithms [2]. To evaluate parallel algorithms there have been developed several following fundamental concepts

- analytical
  - application of queuing theory [4, 6, 7]
  - asymptotic analysis [3, 8]
  - Petri nets [7]
- simulation [10]
- experimental measurement [3].

### 3. 1. Performance metrics

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

#### 3.1.1. Speed up

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

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

#### 3.1.2. Efficiency

The system efficiency for a p processor system is defined by

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

#### 3.1.3. Scalability metrics

Scalability metrics describe the application characteristics in terms of relative gain or loss in performance as a function of the number of the allocated resources. A popular measure of how effectively an application uses a given parallel system is speedup.

#### 3.1.4. Isoefficiency concept

The workload of any parallel algorithm often grows in the order O(s), where s is the size of concrete problem. 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 E when implementing a parallel algorithm on a parallel system. Let h be the total communication overhead involved in the algorithm implementation. This overhead is usually a function of both machine size and problem size, thus denoted h = h(s, p). The efficiency of a parallel algorithm implemented on a given parallel computer we defined using workload and overhead functions as

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

The workload w(s) corresponds to useful computations while the overhead h(s, p) are useless times related to synchronisation and data communication delays. In general, the overhead increases with respect to both 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).

With a fixed problem size (fixed workload), 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 h grows slower than the workload w. 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 w is allowed to grow properly with increasing machine size.

For a given algorithm, the workload w(s) might need to grow polynomial or exponentially with respect to p in order to maintain a fixed efficiency. Different parallel 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 ≥1. The smaller a power of p in the isoefficiency function is, the more scalable the parallel system. Here, the system includes the algorithm and architecture combination.

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-E is a constant for a fixed efficiency E. thus we can define the isoefficiency function as $f_E(p)$=C. h(s, p). If the workload grows as fast as $f_E(p)$ then a constant efficiency can be maintained for a given algorithm-architecture combination.

### 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.

In this sense analysis and modelling of complexity in parallel algorithms (PA) is rationalised to the analysis of complexity of own calculations, that mean that the function of control and communication overheads are not a part of derived relations for execution time T (s, p). In this sense the function in the relation for isoefficiency suppose, that dominate 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 [2]

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

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 [1, 11

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

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

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

, where $T_{comp}$, $T_{par}$, $T_{comm}$, $T_{syn}$ denote the individual overheads for calculations, parallelisation overheads, communication and synchronisation overheads. 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. In general nonlinear influence of h (s, p) could be at performance

parallel algorithm modelling dominant (Fig. 3.). Then for asymptotic isoefficiency analysis is true

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

, 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).

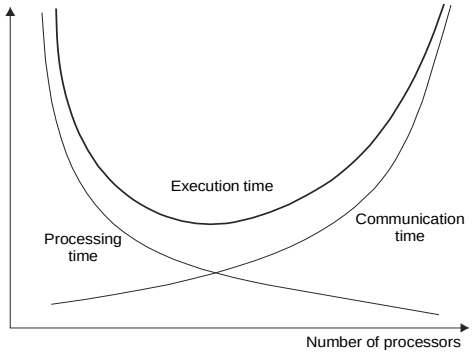


*Fig. 3. Relations among times in parallel algorithms.*

Processing 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 a average value of performed calculation 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

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

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.

## 5. Decomposition of matrix models

In general we are considering the typical possible decomposition strategies in following matrix

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

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

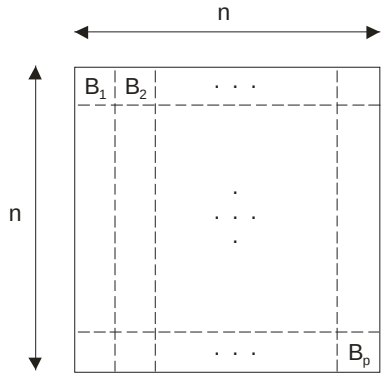


*Fig. 4. Decomposition strategy to blocks.*

Depending of used decomposition methods there are derived needed communication activities. In general square matrix in two dimensions in halts $n^2$ elements (complexity for sequential algorithm, which are equally divided to p build parallel processes, that means every parallel process gets $n^2 / p$ 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 square blocks to every parallel process as illustrated at Fig. 4.
- mapping of p columns or rows.

Depending of used decomposition methods there are derived needed communication activities. In general square matrix in two dimensions in halts $n^2$ elements (complexity for sequential algorithm, which are equally divided to p build parallel processes, that means every parallel process gets $n^2 / p$ elements.

### 5.1. Matrix decomposition to blocks

For mapping matrix elements in blocks a inter process communication is performed on the four neighbouring edges of blocks (Fig. 5.), 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).

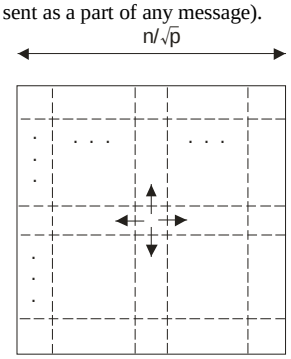


*Fig. 5. Communication consequences for decomposition to blocks.*

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

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

where

- $t_s$ - is a start up time (time to initialise a communication)
- $t_w$ - characterise needed time to transmit one word of message.

Graphical explanation of used communication parameters illustrates Fig. 5.

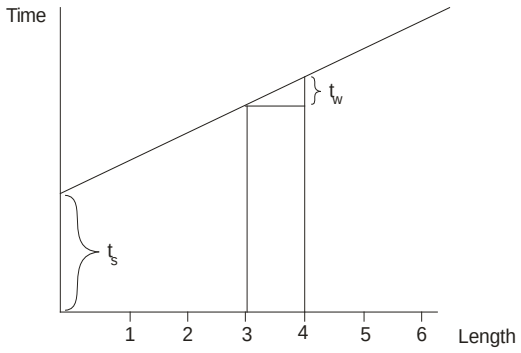


*Fig. 5. Illustration of communication parameters.*

This equation is correct for $p \geq 9$, 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

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

### *5.2. 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

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

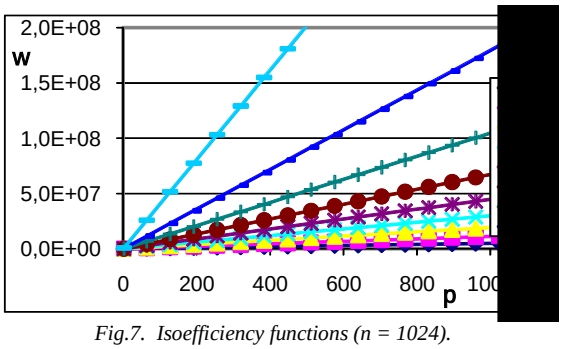


*Fig.7. Isoefficiency functions (n = 1024).*

Fig. 7. illustrate isoefficiency functions for individual constant values of effectivity (E = 0,1 až 0,9) for n = 1024 using the communication constants of parallel computer Cray T3E ($t_s$ = 3 µs, $t_w$ = 0,063 µs).

## 6. 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. Queueing 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 of 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. 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.

**Literature:**

1. Fortier P., Howard M., Computer system performance evaluation and prediction, 544 p., 2003, Digital Press
2. Goldreich O., *Computational complexity*, Cambridge University Press, 632 pages, 2010
3. Hanuliak I., Hanuliak P., *Performance evaluation of iterative parallel algorithms*, *Kybernetes*, Volume 39, No. 1, pp. 107 – 126, 2010, United Kingdom
4. Hanuliak M., Hanuliak I., *To the correction of analytical models for computer based communication systems*, Kybernetes, The International Journal of Systems & Cybernetics, West Yorkshire, Volume 35, No. 9, 1492-1504, 2006, United Kingdom
5. Hanuliak J., Hanuliak I., *To performance evaluation of distributed parallel algorithms*, Kybernetes, The International Journal of Systems & Cybernetics, West Yorkshire, Volume 34, No. 9/10, pp. 1633-1650, 2005,United Kingdom
6. Hanuliak J., Hanuliak M., *Analytical modelling of distributed computer systems*, NOW, In Proc.: TRANSCOM 2005, Zilina, (2005), 103-110
7. Hillston J., *A Compositional Approach to Performance Modelling*, University of Edinburg, 172 pages, 2005, Cambridge University Press, United Kingdom

8. Hudik M., *Performance optimization of broadcast collective operation on multi-core cluster*, ICSC Leden 2012, Kunovice, Czech republic (in print)
9. Kirk D. B., Hwu W. W., *Programming massively parallel processors*, Morgam Kaufmann, 280 pages, 2010
10. Kostin A., Ilushechkina L., *Modelling and simulation of distributed systems*, 440 pages, Jun 2010, Imperial College Press.
11. Kumar A., Manjunath D., Kuri J., *Communication Networking* , 750 pp., 2004, Morgan Kaufmann

**Primary Paper Section:** I

**Secondary Paper Section:** IN, JC, JD