https://doi.org/10.26552/com.C.2003.4.36-41

komunikácie
COMMUNICATIONS

Peter Hanuliak – Peter Varša *

GRID IMPLEMENTATION OF PARALLEL ALGORITHM FOR LAPLACEAN EQUATION COMPUTATION BY JACOBI ITERATION METHOD

The article is devoted to the problem of parallel algorithms and their practical implementations. On the basis of parallel computer analysis in the world the parallel systems are divided into two basic groups – synchronous and asynchronous systems – which are very different from the system point of view. This article describes the development of real parallel algorithms for Jacobi iteration. This individual practical example demonstrates the influence of decomposition strategies for performance evaluation of parallel Jacobi iteration and discusses the ways for their parallel implementations.

1. Introduction

For the contemporary technical and programmable level of the reachable computer means (personal computers, minicomputers, supercomputers, etc.) the use of various forms of basic principles of the parallel activity is dominant [8, 15, 16]. For example in section of technical equipment the continuous speeding up of the individual processor performance is achievable mainly through the parallel activity of the pipeline execution in combination with blowing up the capacity and number of various buffer memories (caches).

In the field of programming equipment the parallel support goes also in two levels[15, 17, 18]. The first level forms the district of the operation systems and in general the system supporting programming tools. The second level creates the user developing programming environments, which support the development of the modular application programs as the basic condition to their potential parallel activity. This parallel support goes in this time up to the level of the elementary program elements in the form of the objects (OOP – object oriented programming).

The architectures of the parallel systems

In system classification we can divide parallel systems to the two very different groups [1]:

- synchronous parallel architectures. To this group belong practically all known parallel architectures except the computer networks. The basic system properties are given through the existence of some kind of the common shared memory M by parallel processors P_i, which in substantial measure simplifies their application programming using. The principal model is illustrated in Fig. 1.
- asynchronous parallel architectures. This group covers the field of various forms of computer networks. Their basic property is the mutual interconnection both in the remote form of the distributed memory moduls M_k and the parallel processors P_i with using the existed telecommunication lines (WAN networks) and in the local form in reaching range of the used fixed lines (LAN networks), respectively. There is, in contrast to the first discussed group, no form of common shared memory in the connected system. The principal model is in Fig. 2.

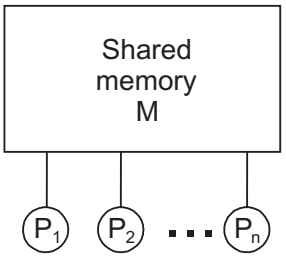

Fig. 1. Model of the system with shared memory.

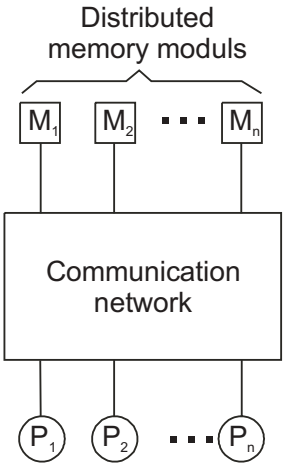

Fig. 2. Model of the parallel system with distributed memory moduls.

* **Peter Hanuliak, Peter Varša**
University of Žilina, Faculty of Management Science and Informatics, Slovakia
E-mail: hanuliak@frtk.fri.utc.sk, varsa@fri.utc.sk

The load balancing, inter-processor communication and transport protocol for such machines are being widely studied [6, 7, 13, 14, 15, 18]. With the availability of cheap personal computers, workstations and networking devises, the recent trend is to connect a number of such workstations to solve computation-intensive tasks in parallel on such clusters. To exploit the parallel processing capability of a NOW (Network of workstations), 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 [7, 16, 17].

The development of the parallel network algorithm includes the following activities:
- Decomposition – the division of the application into a set of parallel processes and data
- 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)

When designing a parallel program the description of the high-level algorithm must include also the method we intend to use to divide the application into processes and to distribute data to different nodes – the decomposition strategy. The chosen decomposition method drives the rest of program development. This is true in case of developing new application as porting serial code. The decomposition method says how to structure the code and data and defines the communication topology.

To choose the best decomposition method for these applications, it is necessary to understand the particular application problem, the data domain, the used algorithm and the flow of control in given application. Therefore, according to the character of given task the following decomposition models are used:
- perfectly parallel decomposition
- domain decomposition
- control decomposition
- object-oriented programming – OOP.

2. The role of performance

Quantitative evaluation and modelling of hardware and software components of the parallel systems are critical for the delivery of high performance. Performance studies apply to initial design phases as well as to procurement, tuning, and capacity planning analysis. As performance cannot be expressed by quantities independent of the system workload, the quantitative characterisation of application resource demands and their behaviour is an important part of any performance evaluation study. Among the goals of parallel systems performance analysis is to estimate the performance of a system or a system component or an application, to investigate the match between requirements and system architecture characteristics, to identify the features that have a significant impact on the application execution time, to predict the performance of a particular application on a given parallel system, to evaluate different structures of parallel applications. To the performance evaluation we briefly review the techniques most commonly adopted for the evaluation of parallel systems and its metrics.

2.1 Performance evaluation methods

To the performance evaluation we can use the following methods:
- analytical methods
  - application of queueing theory [7, 8, 9, 10, 11, 12]
  - Petri nets [7, 14]
- simulation methods [2, 5]
  - experimental measurement [7, 14]
  - benchmarks [14, 16]
- direct measuring of particular developed parallel application.

In order to extend the applicability of analytical techniques to the parallel processing domain, various enhancements have been introduced to model phenomena such as simultaneous resource possession, fork and join mechanism, blocking and synchronisation. Hybrid modelling techniques allow to model contention both at hardware and software levels by combining approximate solutions and analytical methods. However, the complexity of parallel systems and algorithms limits the applicability of these techniques. Therefore, in spite of its computation and time requirements, simulation is extensively used as it imposes no constraints on modelling.

Evaluating system performance via experimental measurements is a very useful alternative for parallel systems and algorithms. Measurements can be gathered on existing systems by means of benchmark applications that aim at stressing specific aspects of the parallel systems and algorithms. 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. Parallel benchmarks extend the traditional sequential ones by providing a wider set of suites that exercise each system component targeted workload. The Parkbench suite especially oriented to message passing architectures and the SPLASH suite for shared memory architectures are among the most commonly used benchmarks [14].

2.2. Performance evaluation metrics

For evaluating the parallel algorithms there have been developed several fundamental concepts [1, 6]. Tradeoffs among these performance factors are often encountered in real-life applications.

2.2.1. Performance concepts

Let $O(s, p)$ be the total number of unit operations performed by p-processor system for size s of the computational problem and

$T(s, p)$ be the execution time in unit time steps. In general, $T(s, p) < O(s, p)$ if more than one operation is performed by p processors per unit time, where $p \geq 2$. Assume $T(s, 1) = O(s, 1)$ in a single-processor system (sequential system). The *speedup factor* is defined as:

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

(Note that this speedup is a limit case and almost never obtained.) It is a measure of the speedup obtained by given algorithm when p processors are available for the given problem size s. Ideally, since $S(s, p) \leq p$, we would like to design algorithms that achieve $S(s, p) \approx p$.

The system *efficiency* for an p-processor system is defined by:

$$E(s, p) = \frac{S(s, p)}{p} = \frac{T(s, 1)}{p\ T(s, p)} \quad (2)$$

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

2.2.2. The isoefficiency concept

The workload w of an algorithm often grows in the order $O(s)$, where 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 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 is thus defined as

$$E(s, p) = \frac{w(s)}{w(s) + h(s, p)} \quad (3)$$

The workload $w(s)$ corresponds to useful computations while the overhead $h(s, n)$ is the useless time attributed 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 the 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 overhead 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 might need to grow polynomially 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 $k \geq 1$.

We can rewrite the 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:

$$w(s) = \frac{E}{1 - E} h(s, p) \quad (4)$$

The factor $C = E/(1 - E)$ is a constant for a fixed efficiency E. Thus we can define the isoefficiency function as follows: $f_E(p) = C \cdot h(s, p)$. If the workload grows as fast as $f_E(p)$, then a constant efficiency can be maintained for a given algorithm-architecture combination.

3. Theoretical part

Partial differential equations (PDE) are used to model a variety of different kinds of physical systems: weather, airflow over a wing, turbulence in fluids, and so on. Some simple PDE's can be solved directly, but in general it is necessary to approximate the solution at a finite number of points using iterative numerical methods. Here we show how to solve in parallel one specific PDE – Laplace's equation in two dimensions – by means of a grid computation method that employs a finite-difference method. Although we focus on this specific problem, the same programming techniques are used in grid computations for solving other PDE's and in other applications such as image processing etc.

3.1. Laplace's Equation

Laplace' equation is a practical example of Jacobi iteration application. The equation for two dimensions is as follows:

$$\frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} = 0 \quad (5)$$

Function $\Phi(x, y)$ represents some unknown potential, such as heat or stress.

Given a two-dimensional region and values for points of the region boundaries, the goal is to approximate the steady-state solution $\Phi(x, y)$ for points in the interior by the function $u(x,y)$. We can do this by covering the region with a grid of points and to obtain the values of $u(x_i, y_j) = u_{ij}$ as follows:

Let us consider square region $(a, b) \times (a, b)$, thus for coordinates of grid points is valid $x_i = i * h$, $y_j = j * h$, $h = (b-a)/N$ for $i,j = 0, 1, ..., N$. We replace partial derivations of $\Phi \sim u(x, y)$ by the differences of $u_{i,j}$:

$$\frac{\partial^2\Phi}{\partial x^2} \cong \frac{\Delta_x^{(2)} u_{i,j}}{h^2} = \frac{\Delta_x^{(1)} u_{i,j} - \Delta_x^{(1)} u_{i-1,j}}{h^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}$$

$$\frac{\partial^2\Phi}{\partial y^2} \cong \frac{\Delta_y^{(2)} u_{i,j}}{h^2} = \frac{\Delta_y^{(1)} u_{i,j} - \Delta_y^{(1)} u_{i,j-1}}{h^2} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}. \tag{6}$$

and after substituting (6) in (5) we obtain

$$u_{i,j} = (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1})/4 \quad i,j = 1, 2, \ldots N;$$

Each interior point is initialised to some value. The steady-state values of the interior points are then calculated by repeated iterations. On each iteration the new value of a point is set to a combination of the previous values of neighbouring points. The computation terminates either after a given number of iterations or when every new value is within some acceptable difference $\epsilon > 0$ of the previous value.

There are several iterative methods for solving Laplace's equation, including Jacobi iteration, Gauss – Seidel iteration, successive over-relaxation (SOR), and multigrid [1, 6, 19]. In this paper we show parallel implementation of Jacobi iteration using Message Passing Interface (MPI). The algorithms for other methods converge more rapidly but are somewhat more complex than Jacobi iteration. Their parallel implementations have similar communication and synchronisation patters and these aspects are the most important.

3.2. Jacobi iteration

We applied Jacobi point iterative method to the grid on given region $(0, 1) \times (0, 1)$ for which the boundary conditions are known. $(N-1)*(N-1)$ is the number of interior grid points. The Laplace' equation will be in form:

$$\begin{aligned} &u_{i,j} = (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1})/4 \;\; i,j = 1, 2, \ldots N-1; \\ &u_{0,j} = -y_j^2;\; u_{N,j} = 1 - y_j^2; \quad j = 1, 2, \ldots N-1; \\ &u_{i,0} = x_i^2;\; u_{i,N} = x_i^2 - 1; \quad i = 1, 2, \ldots N-1; \end{aligned} \tag{7}$$

where

$$x_i = i/N; \qquad y_j = j/N;$$

In this case the exact solution is known:

$$U_{i,j} = x_i^2 - y_j^2$$

3.3. Local communication

A local communication structure is obtained when an operation requires data from a small number of other tasks. It is then straightforward to define channels that link the tasks responsible for performing the operation (the consumers) with the tasks holding the required data (the producers) and to introduce appriopriate send and receive operations in the producer and consumer tasks, respectively.

For Jacobi finite difference method a two-dimensional grid is repeatedly updated by replacing the value at each point with some function of the values at a small fixed number of neightbouring points. The following expression uses a four-point stencil to update each element $X_{i,j}$ of a two -dimensional grid X:

$$X_{i,j}^{(t+1)} = (X_{i-1,j}^{(t)} + X_{i+1,j}^{(t)} + X_{i,j-1}^{(t)} + X_{i,j+1}^{(t)})/4 \tag{8}$$

This update is applied repeatedly to compute a sequence of values $X_{i,j}^{(1)}, X_{i,j}^{(2)}, \ldots$ and so on. The notation $X_{i,j}^{(t)}$ denotes the value of grid point $X_{i,j}$ at step t.

Let us assume that a partition has used domain decomposition techniques to create a distinct task for each point in two dimensional grid. Hence, a task allocated the grid point $X_{i,j}$ must compute the sequence

$$X_{i,j}^{(1)}, X_{i,j}^{(2)}, X_{i,j}^{(3)}, \ldots \tag{9}$$

This computation requires in turn the four corresponding sequences

$$\begin{aligned} &X_{i-1,j}^{(0)}, X_{i-1,j}^{(1)}, X_{i-1,j}^{(2)}, \ldots \\ &X_{i+1,j}^{(0)}, X_{i+1,j}^{(1)}, X_{i+1,j}^{(2)}, \ldots \\ &X_{i,j-1}^{(0)}, X_{i,j-1}^{(1)}, X_{i,j-1}^{(2)}, \ldots \\ &X_{i,j+1}^{(0)}, X_{i,j+1}^{(1)}, X_{i,j+1}^{(2)}, \ldots \end{aligned} \tag{10}$$

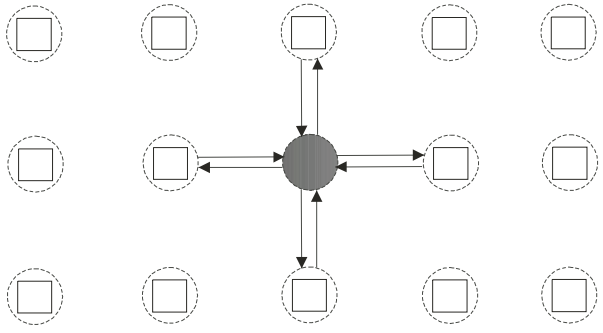

Fig. 3. Local communication.

which are produced by four tasks handling grid points $X_{i-1,j}$, $X_{i+1,j}$, $X_{i,j-1}$ and $X_{i,j+1}$, that is, by its four neighbours in the grid. For these values to be communicated, we define channels linking each task that requires a value with the task that generates that value. This yields the channel structure illustrated in Fig. 3. For first T-steps each task then executes the following logic:

```
for t = 0 to T - 1 do
  begin
    send X(t)i,j to each neighbour;
    receive X(t)i-1,j, X(t)i+1,j, X(t)i,j-1, X(t)i,j+1 from neighbours;
    compute X(t+1)i,j using Equation (8);
  endfor
```

4. Experimental part and results

We used two decomposition strategies in order to analyse their influence to the performance evaluation:

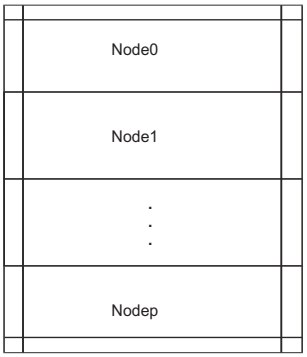

Fig. 4. Domain decomposition 1.

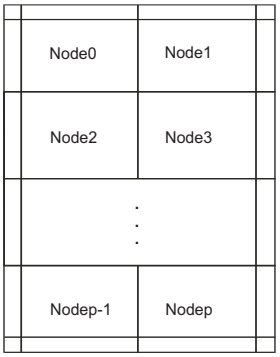

Fig. 5. Domain decomposition 2.

a) the decomposition of Jacobi iteration according to Fig. 4 (domain decomposition 1). In this strategy there was given each computation node a horizontal strip of U_{ij}. After each iteration „boundary conditions"of the strip have to be shared with neghbouring nodes (Fig. 6 and 7).

b) the decomposition according to Fig. 5 (domain decomposition 2). In this strategy we used twice much computation nodes as in the first case.

In Fig. 8. we illustrate the performance of both decomposition strategies. We can see that when using more computation processors we did not come to increased performance. The causes are in increasing the overheads (control, communication, synchronisation) more than the speed-up of higher computation nodes. The realised experiments were done on parallel system at EPCC Edinbourgh (parallel system Cray T3E).

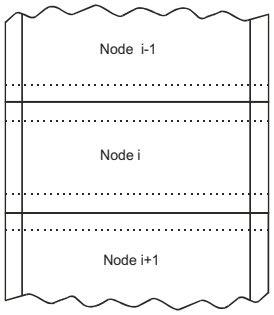

Fig. 6. The shared points.

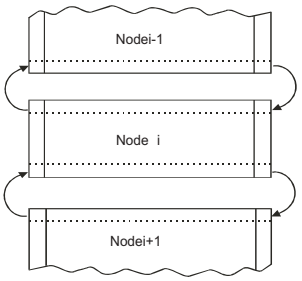

Fig. 7. Exchange of the values.

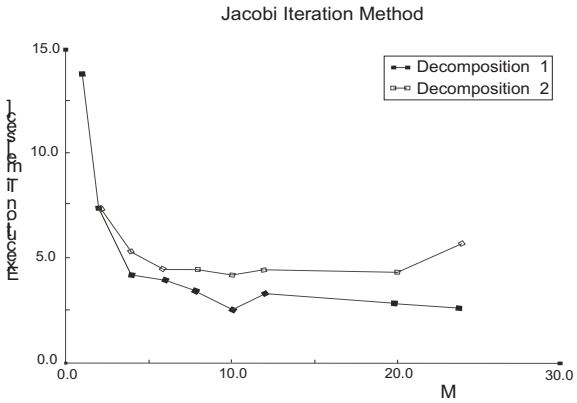

Fig. 8. The results (M = number of used processors)

References

[1] ANDREWS G. R., *Foundations of Multithreaded, Parallel, and Distributed Programming*, Addison Wesley Longman, Inc., 664 pp., 2000, USA

[2] BANKS J., DAI J. G.: *Simulation studies of multiclass queueing networks,* IEEE Transactions, Volume 29, 1997, pp. 213-219

[3] BASOGLU CH., LEE W., KIM Y.: *An efficient FFT algorithm for Super-scalar and VLIW Processor Architectures,* Real Time Imaging 3, pp. 441-453, 1997, USA

[4] ČERNÁ M., MACHALICKÝ M., VOGEL J., ZLATNÍK Č.: *A first course in numerical mathematics and programming (in czech),* Alfa/SNTL, Praha, 1987

[5] FODOR G., BLAABJERG S., ANDERSEN A.: *Modelling and simulation of mixed queueing and loss systems,* Wireless Personal Communication, N. 8, 1998, pp. 253-276

[6] GREENBERG D. S., PARK J. K., SCHVABE E. J.: *The cost of complex communication on simple networks*, Journal of Parallel and Distributed Computing 35, pp. 133-141, 1996

[7] HANULIAK I.: *Parallel architecture - multiprocessors, computer networks (in slovak),* 187 strán, 127 obr.,17 tab., Júl 1997, vyd.: Knižné centrum, Žilina

[8] HANULIAK I.: P*arallel computers and algorithms (in Slovak)*, Košice (Slovakia), ELFA Press 1999, 327 pp.
[9] HANULIAK J.: *To a complexity of parallel algorithms,* In Proceedings: TRANSCOM 2001 (4-th European Conference in Transport and Telecommunications), 25-27 June 2001, pp. 51-54, Žilina, Slovak Republic
[10] HANULIAK J., HANULIAK I., MATIASKO K.: *To parallel implementation of Discrete Fast Fourier Transform,* Journal of the Applied Sciences Mittweida, No. 15, 2000, pp. 3-10, Mittweida, Germany
[11] HANULIAK M.: *To the behaviour analysis of mobile data networks,* in Proceedings of 7th Scientific conference, November 9-10, pp. 170-175, 2001, T_RGU - JIU, Romania
[12] HARRISON P. G., PATEL N.: *Performance modelling of communication networks and computer architectures,* Addison – Wesley Publishers 1993, 480 pp.
[13] HSU W. T., PEN-CHUNG Y.: *Performance Evaluation of Wire-Limited Hierarchical Networks,* Parallel and Distributed Computing 41, 1997, pp. 156-172
[14] HESHAM EL-REWINI, TED. G. LEWIS: *Distributed and parallel computing,* 467 pp., Manning Publications Co., 1997, USA
[15] HWANG K., XU Z.: *Scalable Parallel Computing: Technology, Architecture, Programming,* Mc Graw-Hill Companies, 802 pp., 1998, USA
[16] KUMAR V., GRAMA A., GUPTA A., KARYPIS G.: *Introduction to Parallel Computing (Second Edition)*, Addison Wesley, 856 pp., 2001
[17] MARINESCU D. C., RICE J. R.: *On the scalability of asynchronous parallel computations,* Parallel and Distributed Computing 31, pp. 88-97, 1995
[18] NANCY A. L.: *Distributed Algorithms,* 872 pp., 1996, Morgan Kaufmann Publishers, Inc., USA
[19] VAJTERŠIC M.: *Modern algorithms for solving some elliptic pariial differential equations (in slovak),* Veda, Bratislava, 1988
[20] VARŠA P.: *Contribution to complexity of distributed parallel algorithms (in Slovak)*, Dissertation theses, March 2003, 94 pp., University of Zilina, Žilina, Slovakia
[21] WILLIAMS R.: *Computer Systems Architecture – A networking approach,* Addison Wesley, 660 pp., 2001, England.