



The Ministry of Education and Science of the Russian Federation

Lobachevsky State University of Nizhni Novgorod

Computing Mathematics and Cybernetics faculty

The competitiveness enhancement program

of the Lobachevsky State University of Nizhni Novgorod

among the world's research and education centers

Strategic initiative

“Achieving leading positions in the field of supercomputer technology
and high-performance computing”

Introduction to MPI

Lecture 2. Basic Notions and Definitions

Nizhni Novgorod

2014

Lecture_2_. Basic Notions and Definitions

The analysis of parallelism efficiency is a crucial point in the development of parallel algorithms for solving complicated research and engineering problems. Parallelism efficiency analysis is, as a rule, the evaluation of the computation process speedup (reducing the time needed for solving a problem). Forming the speedup estimation may be carried out for selected computational algorithm (the efficiency estimation of parallelizing a specific algorithm). Another important approach may be the construction of the maximum possible speedup estimation for the solution of a certain problem type (the efficiency estimation of the best parallel approach for solving a problem). This lecture is dedicated to basic efficiency characteristics. It also gives a visible example of efficiency characteristics estimation.

2.1. Parallel Algorithm Efficiency Characteristics

Speedup. This is a speedup obtained if a parallel algorithm is used for p processors in comparison to the sequential computations. It is determined by the value

$$S_p(n) = T_1(n) / T_p(n),$$

i.e. as the ratio of the problem solution time on a scalar computer to the time of parallel algorithm execution (value n is used for parameterization of computation complexity of the problem being solved and can be understood as, for instance, the amount of input problem data).

Efficiency. The efficiency of the processor utilization by the parallel algorithm in solving a problem is determined by the formula

$$E_p(n) = T_1(n) / (pT_p(n)) = S_p(n) / p$$

(the efficiency value determines the mean fraction of algorithm execution time, during which the processors are actually used for solving the problem).

The expressions given above demonstrate that at best $S_p(n) = p$ and $E_p(n) = 1$. The following two issues should be taken into account in practical application of these criteria for parallel computation efficiency estimation.

Under certain circumstances the speedup may appear to be greater than the number of the processors being used, i.e. $S_p(n) > p$. In this case the speedup is considered to be superlinear. Despite the fact that these situations are paradoxical (the speedup is greater than the number of processors), in practice superlinear speedup takes place. One of the reasons of this phenomenon may

be the disparity of sequential and parallel programs execution. For instance, when a problem is solved on one processor RAM appears to be insufficient for storing of all the data being processed, and as a result, it is necessary to use a slower external memory (if several processor are used, RAM may be sufficient because the data are being shared among processors). One more reason for superlinear speedup may be the non-linear character of the dependency of the problem solution complexity with respect to the amount of the data being processed. Thus, for instance, the well-known bubble sorting algorithm is characterized by as square dependency of the necessary operation amount with respect to the number of data being ordered. As a result, as the data file is being distributed among the processors, the speedup, which is greater than the number of processors, may be obtained. The source of superlinear speedup may be also the difference of parallel and sequential method computational schemes;

Studying the case more carefully, one may pay attention to the fact that the attempts to improve the parallel computation quality with respect to one of the characteristics (speedup or efficiency) may lead to the worsening of the situation for the other criterion , as the characteristics of parallel computation quality are conflicting. Thus, for instance, speedup increase may be provided by the larger number of processors, which leads, as a rule, to an efficiency drop. And vice versa, efficiency increase is in many cases achieved if the number of processors is decreased (in the limiting case the ideal efficiency $E_p(n) = 1$ is easily provided if only one processor is used). As a result, the development of parallel computation method often involves selection of some compromise variant with respect to the desirable efficiency and speedup criteria.

2.2. Partial Sums Computations

To demonstrate the problems, which may arise when parallel computation methods are developed, we will consider a rather simple problem of finding partial sums of numerical value sequence:

$$S_k = \sum_{i=1}^k x_i, \quad 1 \leq k \leq n,$$

where n is the number of summable (the number of data being summarized) values (this problem is also known as *prefix sum problem*).

We will start the study of possible parallel solution method for the problem with the even simpler variant of its formulation: the computation of the total sum of the available set of values (in this form the summation problem is a particular case of the general *reduction problem*)

$$S = \sum_{i=1}^n x_i .$$

2.2.1. Sequential Summation Algorithm

The traditional algorithm for solving the problem is sequential summation of the elements of a series of numbers

$$S = 0,$$

$$S = S + x_1, \dots$$

Computational scheme of the algorithm may be presented the following way (see Figure 2.2):

$$G_1 = (V_1, R_1) ,$$

where $V_1 = \{v_{01}, \dots, v_{0n}, v_{11}, \dots, v_{1n}\}$ is the set of operation (vertices v_{01}, \dots, v_{0n} designate the input operations, each vertex $v_{1i}, 1 \leq i \leq n$, corresponds to addition of value x_i to the accruing amount S), and

$$R_1 = \{(v_{0i}, v_{1i}), (v_{1i}, v_{1i+1}), 1 \leq i \leq n-1\}$$

is the set of arcs defining the information dependencies of the operations.

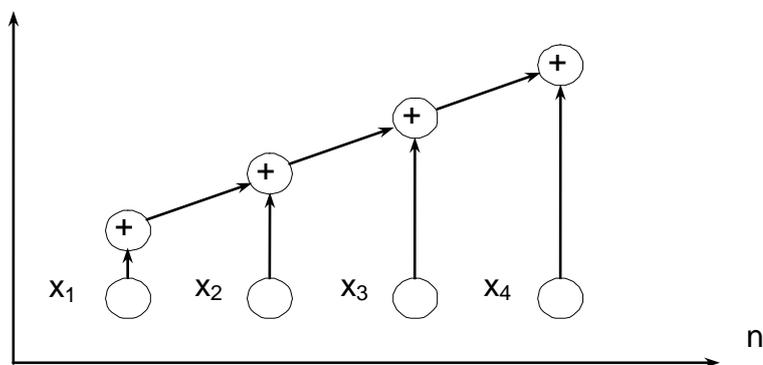


Figure 2.1. The sequential computing scheme of the summation algorithm

As it may be noted, this “standard” summation algorithm allows only strictly sequential execution and cannot be parallelized.

2.2.2. Cascade Summation Scheme

Summation algorithm parallelism becomes possible only if we apply another method of computation process construction, based on the use of the associative property of summation. The new summation variant obtained as result (which is known as a *cascade scheme*) consists of the following (see Figure 2.2):

- At the first operation of the cascade scheme all the input data is partitioned to pairs, and for each pair the sum of their values is computed,
- Later all the sums are also partitioned to pairs, and again the summation of the pair values is executed and etc.

This computing scheme may be presented as a graph (let $n = 2^k$)

$$G_2 = (V_2, R_2),$$

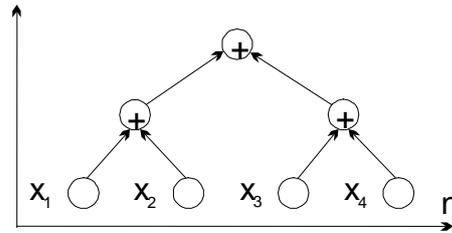


Figure 2.2. Cascade scheme of the summation algorithm

where $V_2 = \{(v_{i1}, \dots, v_{il_i}), 0 \leq i \leq k, 1 \leq l_i \leq 2^{-i} n\}$ are graph vertices ((v_{01}, \dots, v_{0n}) - input operations, $(v_{11}, \dots, v_{1n/2})$ - the first iteration operations and etc.), and the set of the graph arcs is defined as

$$R_2 = \{(v_{i-1,2j-1}, v_{ij}), (v_{i-1,2j}, v_{ij}), 1 \leq i \leq k, 1 \leq j \leq 2^{-i} n\}.$$

It is easily estimated that the number of the cascade scheme operations appears to be equal to the value

$$k = \log_2 n,$$

and the total number of summation operations

$$K_{sequ} = n/2 + n/4 + \dots + 1 = n - 1$$

coincides with the number of operations in sequential variant of the summation algorithm. In parallel execution of the cascade scheme the total number of parallel summation operations is equal to

$$K_{par} = \log_2 n.$$

As the execution time for any computational operation is considered to be identical and equal to 1 so $T_1 = K_{seq}$, $T_p = K_{par}$, thus the speedup and efficiency characteristics of the summation algorithm cascade scheme may be estimated as

$$S_p = T_1 / T_p = (n - 1) / \log_2 n,$$

$$E_p = T_1 / pT_p = (n - 1) / (p \log_2 n) = (n - 1) / ((n / 2) \log_2 n),$$

where $p = n / 2$ is the number of processors necessary for the cascade scheme execution.

The analysis of the obtained characteristics shows that the time of parallel cascade scheme execution coincides with the paracomputer estimate in theorem 2. However, in this case the efficiency of processors decreases when the number of summable values increases:

$$\lim E_p \rightarrow 0 \quad \text{if } n \rightarrow \infty$$

2.2.3. Modified Cascade Scheme

Asymptomatic nonzero efficiency may be provided if, for instance, a modified cascade scheme is used (see Bertsekas and Tsitsiklis (1989)). To simplify the estimate creation it is possible to assume that $n = 2^k$, $k = 2^s$. In this case all the calculation in the new variant of the cascade scheme are subdivided into two sequentially executed summation phases (see figure. 2.3):

- During the first phase of computations all the summarized values are subdivided into $(n / \log_2 n)$ groups. There are $\log_2 n$ elements in each group. Then the sum of the values is calculated for each group by the sequential summation algorithm. The calculations in each group may be carried out independently (that is in parallel that requires not fewer that $(n / \log_2 n)$ processors);
- During the second phase a conventional cascade scheme is used for the obtained $(n / \log_2 n)$ sums of separate groups.

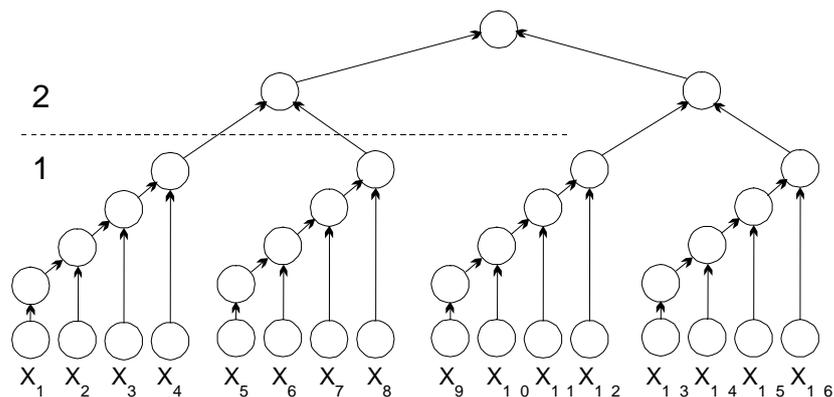


Figure 2.3. Modified cascade summation scheme

Thus the execution of the first phase requires $\log_2 n$ parallel operations if $p_1 = (n / \log_2 n)$ processors are used. The execution of the second phase requires

$$\log_2 (n / \log_2 n) \leq \log_2 n$$

parallel operations for $p_2 = (n / \log_2 n) / 2$ processors. As a result, this summation method is characterized by the following values:

$$T_p = 2 \log_2 n, \quad p = (n / \log_2 n).$$

With respect to the estimates obtained the speedup and efficiency of the modified cascade scheme are defined by the relations:

$$S_p = T_1 / T_p = (n - 1) / 2 \log_2 n,$$

$$E_p = T_1 / pT_p = (n - 1) / (2(n / \log_2 n) \log_2 n) = (n - 1) / 2n.$$

The comparison of the given estimates to the conventional cascade scheme characteristics shows that the speedup for the suggested parallel algorithm has decreased twice. However, for the efficiency of the new summation method it is possible to obtain asymptotic nonzero estimate from below

$$E_p = (n - 1) / 2n \geq 0.25, \quad \lim_{n \rightarrow \infty} E_p \rightarrow 0.5 \quad \text{where} \quad n \rightarrow \infty.$$

2.3. References

Additional information on parallel computation modeling and analysis may be found in, for instance, Bertsekas and Tsitsiklis (1989). Useful information is also contained in Kumar et al. (1994), Quinn (2004).

The consideration of the academic problem of the numeric value sequence summation was carried out in Bertsekas and Tsitsiklis (1989).

A systematic discussion (for the time when the book was published) of the parallel computation modeling and analysis issues is given in Zomaya (1996).

Test questions

1. How are the concepts “speedup” and “efficiency” defined?
2. Is it possible to attain superlinear speedup?
3. What is the contradictoriness of the speedup and efficiency characteristics?
4. What does the problem of parallelizing a sequential algorithm of the numeric values summation lie in?
5. What is the essence of the summation cascade scheme? What is the aim of considering the modified version of the scheme?
6. What is the difference between the speedup and efficiency characteristics for the discussed versions of the summation cascade scheme?
7. What is the parallel algorithm of all the partial sums computation of a numeric value sequence?

Practice

1. Develop a model and evaluate speedup and efficiency of the parallel computations:

- For the problem of the scalar product of two vectors

$$y = \sum_{i=1}^N a_i b_i ,$$

- For the problem of choosing the maximum and minimum values for the given set of numeric values

$$y_{\min} = \min_{i \leq i \leq N} a_i , y_{\max} = \max_{i \leq i \leq N} a_i ,$$

- For the problem of finding the mean value for the given set of numeric values

$$y = \frac{1}{N} \sum_{i=1}^N a_i .$$

References

1. Pacheco, P. (1996). Parallel Programming with MPI. - Morgan Kaufmann.
2. Gropp, W., Lusk, E., Skjellum, A. (1999a). Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation). - MIT Press.
3. Gropp, W., Lusk, E., Thakur, R. (1999b). Using MPI-2: Advanced Features of the Message Passing Interface (Scientific and Engineering Computation). - MIT Press.
4. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J. (1996). MPI: The Complete Reference. - MIT Press, Boston, 1996.