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 PARALLEL PROGRAMMING

Lecture 2. Basic Notions and Definitions

Nizhni Novgorod

2014

The key matter of development of parallel algorithms for solving complex sci-tech problems is the parallelism efficiency analysis usually consisting in estimation of computation speedup (solution time reduction). Such speedup estimate may be applicable to the selected computation algorithm (estimation of parallelization efficiency for a specific algorithm).

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 (this case is considered in more detail in chapter 10). 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:

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

2.2.1. Sequential Summation Algorithm

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

$S = 0,$

$S = S + x_1,...$

Computational scheme of the algorithm may be presented the following way (see Figure 2.1):

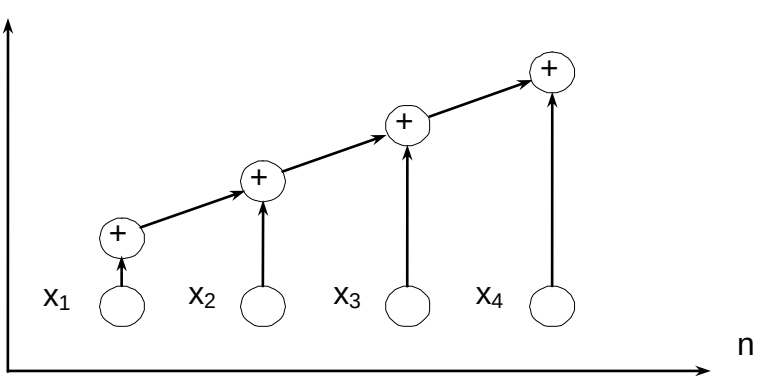

Figure 2.1. The sequential computing scheme of the summation algorithm

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.

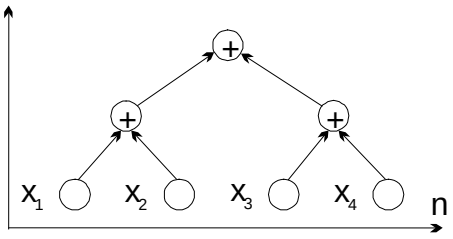

Figure 2.2. Cascade scheme of the summation algorithm

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

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

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 $T_1 = \mathrm{K}_{seq}$, $T_p = \mathrm{K}_{par}$, 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:

$$\lim E_p \to 0 \quad if \quad n \to \infty$$

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.

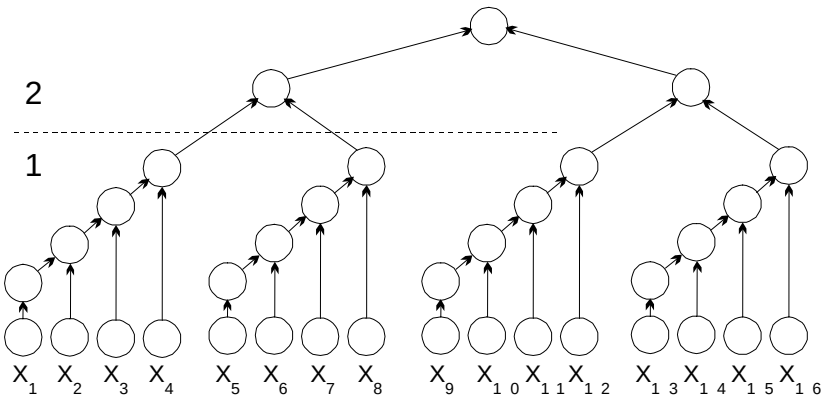

Figure 2.3. Modified cascade summation scheme

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

$$\log_2 (n / \log_2 n) \le \log_2 n$$

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

$$T_P = 2 \log_2 n, \; p = (n / \log_2 n).$$

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

$$E_P = (n-1) / 2n \ge 0.25, \; \lim E_p \to 0.5 \quad \textit{where} \quad n \to \infty .$$

Test questions

1. How do you define speedup and efficiency?
2. Is it possible to ensure superlinear speedup?
3. Why are speedup values contradictory to those of efficiency?

4. What problem is related to parallelization of a sequential algorithm for integer summation?
5. What is the essence of the cascade summation scheme? Why is its modified version described here?
6. What is the difference between the speedup and efficiency values of the described cascade summation schemes?

Practice

Develop a model; estimate the parallel computation speedup and efficiency values

- For finding the scalar product of two vectors

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

- For finding the maximum and minimum values for a given numerical data set

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

- For finding the average value for a given numerical data set

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

References

1. **Amdahl, G.** (1967). Validity of the single processor approach to achieving large scale computing capabilities. In AFIPS Conference Proceedings, Vol. 30, pp. 483-485, Washington, D.C.: Thompson Books.
2. **Bertsekas, D.P., Tsitsiklis, J.N.** (1989). Parallel and distributed Computation. Numerical Methods. - Prentice Hall, Englewood Cliffs, New Jersey.
3. **Grama, A.Y., Gupta, A. and Kumar, V.** (1993). Isoefficiency: Measuring the scalability of parallel algorithms and architectures. IEEE Parallel and Distributed technology. 1 (3). pp. 12-21.
4. **Gustavson, J.L.** (1988) Reevaluating Amdahl's law. Communications of the ACM. 31 (5). pp.532-533.
5. **Kumar V., Grama, A., Gupta, A., Karypis, G.** (1994). Introduction to Parallel Computing. - The Benjamin/Cummings Publishing Company, Inc. (2nd edn., 2003)

6. **Quinn, M. J.** (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.