2. Parallel Computation Modeling and Analysis

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).

In this chapter we will describe the computation model as an “operations-operands” graph, which can be used for the description of the existing information dependencies in selected algorithms of problem solving. We will also give the maximum possible parallelism efficiency estimations, which may be obtained as a result of the analysis of the existing computation models. The practical uses of the theory described here are given in the third part of the teaching materials.

2.1. Computation Model as “Operations-Operands” Graph

The model “operations-operands” graph can be used for the description of the information dependencies in selected algorithms of solving problems (see, for example, Bertsekas and Tsitsiklis (1989)). To simplify the problem we will assume that in constructing a model the periods of execution of any computational operations will be the same and will be equal to 1 (in some units of measurement). Besides we will assume that the data transmission among computing processors is carried out instantaneously without any time consumption (which may be quite true, for instance, if there is a common shared memory in a parallel computing system). The analysis of the parallel algorithm communication complexity is carried out in the next chapter.

Let us depict the set of the operations, carried out in the computational problem solution algorithm to be studied, and the information dependencies, which exist among the operations as an *acyclic oriented graph*

$$G = (V, R),$$

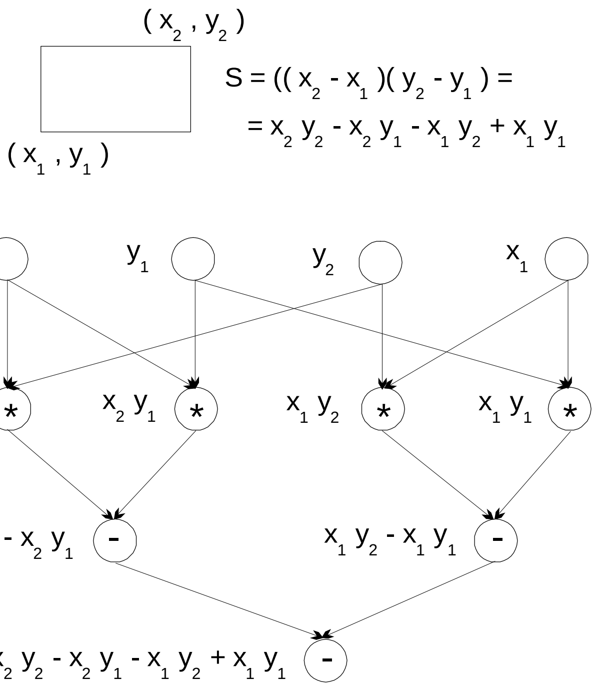

(x2 , y2)
(x1 , y1)
S = ((x2 - x1)(y2 - y1) =
= x2 y2 - x2 y1 - x1 y2 + x1 y1
x2
y1
y2
x1
x2 y2
x2 y1
x1 y2
x1 y1
x2 y2 - x2 y1
x1 y2 - x1 y1
x2 y2 - x2 y1 - x1 y2 + x1 y1

Figure 2.1. The Sample of computational model in the form of the "operations-operands" graph

where $V = \{1,...,|V|\}$ is the set of graph vertices, which represent the algorithm operations being executed, and R is a set of graph arcs (in this case $r = (i, j)$ belongs to the graph only if the operation *j* makes use of the result obtained by execution of operation *i*). To illustrate this Figure 2.1 shows the graph of the algorithm used to calculate the area of the rectangle specified by the coordinates of its two opposite angles. As the given example shows, various computation schemes may be used and various corresponding computational models can be constructed to carry out the selected problem solution algorithm. As it will be shown later different computation schemes possess different capabilities of parallelizing. Thus the task of selecting the most suitable for parallel execution of a computational scheme algorithm can be set in constructing a computation model.

In the computational model of the algorithm under consideration the vertices without the incoming arcs may be used to assign the input operations, and the vertex without outgoing arcs may be used for output operations. Let us denote the set of graph vertices without input vertices as $\overline{V}$, and the diameter (length of maximum path) of the graph as $d(G)$.

2.2. The Scheme of Parallel Algorithm Execution

The algorithm operations, which do not have paths among them within the selected computation scheme, may be executed in parallel (for the computation scheme shown in Figure 2.1, for instance, first all the multiplication operations may be executed in parallel, and then the first two subtraction operations may be realized in parallel). A possible way to describe the parallel algorithm execution is given below (see, for instance, Bertsekas and Tsitsiklis (1989)).

Let p be the number of processors to execute an algorithm. Then to execute computations in parallel it is necessary to specify the set *(schedule)*

$$H_p = \{(i, P_i, t_i) : i \in V\},$$

where for each operation $i \in V$ the number of processor P_i used to execute the operation and the operation start time t_i are given. To make the schedule realizable it is necessary to meet the following requirements in specifying the set H_p:

1) $\forall i, j \in V : t_i = t_j \Rightarrow P_i \neq P_j$, i.e. the same processor must not be assigned to different operations simultaneously,

2) $\forall(i,j)\in R \Rightarrow t_j \ge t_i + 1$, i.e. all the necessary data must have been calculated before operation execution starts.

2.3. Evaluation of Parallel Algorithm Execution Time

The computation scheme of the algorithm G in combination with the schedule H_p may be considered as the model of the parallel algorithm $A_p(G,H_p)$, executed with the use of p processors. The time of parallel algorithm execution is determined by the maximum time value used in the schedule

$$T_p(G,H_p) = \max_{i\in V}(t_i + 1) .$$

For the selected computation scheme it is desirable to use the schedule which provides the minimum algorithm execution time

$$T_p(G) = \min_{H_p} T_p(G,H_p) .$$

The decrease of execution time may be provided by fitting the best computation scheme

$$T_p = \min_{G} T_p(G) .$$

Estimates $T_p(G,H_p)$, $T_p(G)$ and T_p may be used as the time criteria in parallel algorithm execution. Besides to analyze the maximum possible parallelism it is possible to specify the estimate of the fastest algorithm execution

$$T_\infty = \min_{p\ge 1} T_p .$$

Estimate T_∞ may be considered as the minimum possible time of the parallel algorithm execution if an unlimited number of processors are used (the concept of the computer system with the infinite number of processors usually called a *paracompute*r is widely used in the theoretical analysis of parallel computations).

Estimate T_1 defines the algorithm execution time if one processor is used and thus represents the execution time of the sequential version of problem solution algorithm. Constructing such an estimate is an important task in analyzing parallel algorithms, as it is necessary to evaluate the effect of the parallelism use (of speedup while solving the problem). It is evident that

$$T_1(G) = \left|\overline{V}\right| ,$$

where $\left|\overline{V}\right|$, as it has already been defined , is the number of vertices of the computational scheme G without the input vertices. It is important to note that if in determining the estimate T_1 we are limited to the consideration of only one selected problem solution algorithm and use the value

$$T_1 = \min_{G} T_1(G) ,$$

then the speedup coefficients obtained in accordance with the given estimate will characterize the efficiency of parallelizing the selected algorithm. To evaluate the efficiency of the parallel solution of the computational problems under consideration the time of the sequential solution must be evaluated with regard to various sequential algorithms, that is to use the value

$$T_1^* = \min T_1 ,$$

where the operation of minimum is taken over the set of all the possible sequential algorithms for a given problem.

We will consider the theoretical statements, which characterizes the properties of parallel algorithm execution time estimates (see Bertsekas and Tsitsiklis (1989)).

Theorem 1. The maximum path length of the algorithm computation scheme determines the minimum possible time of parallel algorithm execution, i.e.

$$T_\infty(G) = d(G) .$$

Theorem 2. Let there be a path from each input vertex for a certain output vertex in the algorithm computation scheme. Besides let the input power of the scheme vertices (the number of incoming arcs) not exceed 2. Then the minimum possible time of parallel algorithm execution is limited from below by the value.

$$T_\infty(G) = \log_2 n \,,$$

where n is the number of input verticesin the algorithm scheme.

Theorem 3. If the number of the used processors decreases, the algorithm execution time increases in proportion to the decrease of the number of processors, i.e.

$$\forall q = cp, \quad 0 < c < 1 \Rightarrow T_p \le cT_q \,.$$

Theorem 4. For any number of the processors used the following upper estimate for parallel algorithm execution time is true:

$$\forall p \Rightarrow T_p < T_\infty + T_1 / p \,.$$

Theorem 5. The algorithm execution time comparable with the minimum possible time T_∞ can be achieved if the number of processors is in the order of $p \sim T_1 / T_\infty$, to be precise,

$$p \ge T_1 / T_\infty \Rightarrow T_p \le 2T_\infty \,.$$

If there are fewer processors, the time of algorithm execution cannot exceed the best computation time with the given number of processors more than twice, i.e.

$$p < T_1 / T_\infty \Rightarrow \frac{T_1}{p} \le T_p \le 2\frac{T_1}{p} \,.$$

These theorems allow to form the basis for the following recommendations concerning the rules of parallel algorithm creation:

1) The graph with the minimum possible diameter must be used while choosing the algorithm computation scheme (see Theorem 1);
2) The efficient number of processors for parallel execution is determined by the value $p \sim T_1 / T_\infty$ (see Theorem 5);
3) The parallel algorithm execution time is limited from above by the values given in Theorems 4 and 5.

In order to specify the recommendations on the creating the schedule of parallel algorithm execution we will consider the proof of theorem 4.

The proof of the theorem 4. Let H_∞ be the schedule for achieving the minimum possible execution time T_∞. For each iteration τ, $0 \le \tau \le T_\infty$, of the H_∞ schedule execution the number of operations carried out during the iteration τ will be written as n_τ. The schedule of the algorithm execution with the use of p processors may be constructed in the following way. We will divide the algorithm execution into T_∞ steps; at each step τ all n_τ operations, which were carried out during the iteration τ of the H_∞ schedule, must be carried out. The execution of these operations must be accomplished not more than in $\lceil n_\tau / p \rceil$ iterations with the use of p processors. As a result, the time of algorithm T_p execution may be evaluated the following way:

$$T_p = \sum_{\tau=1}^{T_\infty} \left\lceil \frac{n_\tau}{p} \right\rceil < \sum_{\tau=1}^{T_\infty} \left(\frac{n_\tau}{p} + 1 \right) = \frac{T_1}{p} + T_\infty \,.$$

The proof of the theorem offers a practical method of constructing the parallel algorithm schedule. First the schedule with no regard for the limitations of the number of used processors may be created (a paracomputer schedule). Then according to the scheme of the theorem derivation, the schedule for a finite number of processors can be constructed.

2.4. 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.

Selecting the necessary parallel method of problem solving, it is very useful to estimate the computation cost, which is defined as the product of the parallel problem execution time and the number of the processor being used.

$$C_p = pT_p.$$

In this connection it is possible to define the concept of the ***cost-optimal*** parallel algorithm, which is defined as the method, the cost of which is proportional to the time of the best sequential algorithm execution.

To illustrate the introduced concepts in the next chapter we will consider a case of solving the problem of calculation of the partial sum for sequence of numerical values. Besides, in part 3 of the teaching materials these characteristics are used to estimate the efficiency of the considered parallel algorithm for solving the typical problems of computational mathematics.

2.5. 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:

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

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*)

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

2.5.1. 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 Figure2.2):

$G_1 = (V_1, R_1)$,

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

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

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

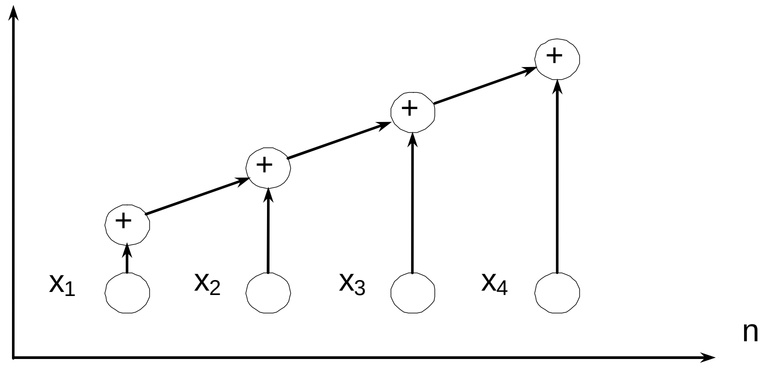


Figure 2.2. The sequential computing scheme of the summation algorithm

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

2.5.2. 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 2.3):

- 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 $n = 2^k$)

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

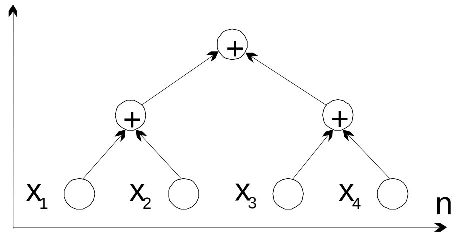


Figure 2.3. Cascade scheme of the summation algorithm

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

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

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

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

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:

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

2.5.3. 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. 2.4):

– 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.

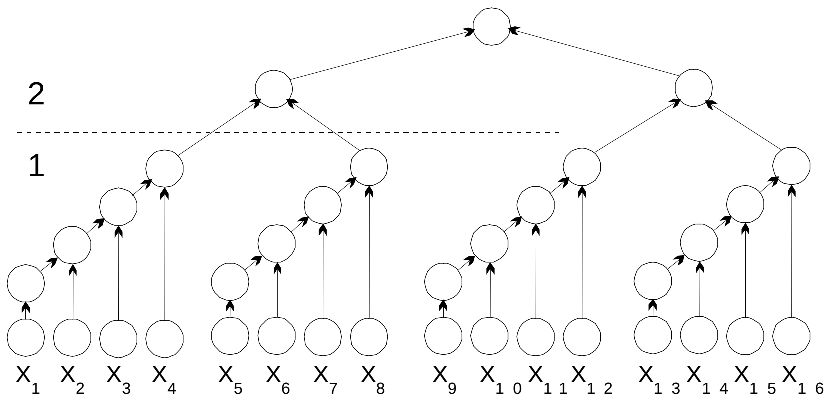


Figure 2. 4. Modified cascade summation scheme

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 efficiencyof 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

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

It may be also noted that the given values are achieved when the number of processors is equal to that defined in theorem 5. Besides it should be emphasized that, unlike the conventional cascade scheme, the modified cascade algorithm is cost-optimal as the calculation cost in this case

$$C_p = pT_p = (n/\log_2 n)(2\log_2 n)$$

is proportional to the time of sequential algorithm execution.

2.5.4. Computation of All Partial Sums

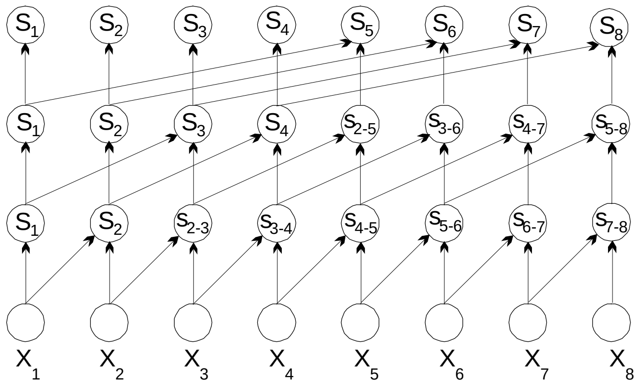
Now we will get back to original problem of computation of all partial sums for a numerical sequence and analyze the possible methods of parallel and sequential computations. The computation of all partial sums on a scalar computer may be done by means of the conventional sequential summation algorithm with the same number of operations (!)

$$T_1 = n.$$

In parallel execution the explicit use of the cascade scheme does not bring the desirable results. Efficient parallelizing requires new approaches (may be even those which do not have analogs in sequential programming) to the development of new parallel-oriented algorithms for solving problems. Thus, for the problem under consideration the algorithm, which provides obtaining the results in $\log_2 n$ parallel operations (like the case of the total sum computation) may be the following (see Figure 2.5) (see Bertsekas and Tsitsiklis (1989)):

– Before the beginning of the computations a copy of vector S of the summarized values is created ($S = x$);

– Later at each summation iteration i, $1 \le i \le \log_2 n$, an auxiliary vector Q is formed by shifting vector S to 2^{i-1} positions to the right (the positions to the left that become released due to the shift are set to zero values). The algorithm iteration is completed by the parallel operation of vector S and vector Q summation.



S1 S2 S3 S4 S5 S6 S7 S8
S1 S2 S3 S4 S2-5 S3-6 S4-7 S5-8
S1 S2 S2-3 S3-4 S4-5 S5-6 S6-7 S7-8
X1 X2 X3 X4 X5 X6 X7 X8

Figure 2.5. The scheme of parallel algorithm computation of all partial sums (values S_{i-j} denote the sums of values from i to j elements of the numerical sequence)

Altogether the parallel algorithm is executed in $\log_2 n$ parallel addition operations. At each algorithm iteration n scalar addition operations are executed in parallel, so the total number of the executed scalar operations is defined by the value

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

(the parallel algorithm contains a greater number of operations in comparison to the sequential summation method). The necessary number of processors is defined by the number of summarized values ($p = n$).

With respect to the obtained relations the speedup and efficiency of the parallel computation algorithm for computations of all partial sums are estimated in the following way:

$$S_P = T_1 / T_P = n / \log_2 n,$$

$$E_P = T_1 / pT_P = n / (p \log_2 n) = n / (n \log_2 n) = 1 / \log_2 n.$$

As it is clear from the given estimates, the algorithm efficiency also decreases when the number of summarized values increases. When it is necessary to increase this characteristic an algorithm modification may appear to be useful as it was in case of the conventional cascade scheme.

2.6. Estimation of Maximum Attainable Parallelism

Estimation of parallel computation quality requires the knowledge of the best (maximum attainable) values of speedup and efficiency . However, attainment of the ideal values $S_p = p$ for speedup and $E_p = 1$ for efficiency may not be provided for all computationally complicated problems. Thus, for the problem considered in 2.5 the minimum attainable time of the parallel computation of a sum of numeric values is $\log_2 n$. The theory described at the beginning of the section may contribute to a certain extend in the solution of the problem. In addition we will consider several patterns of relationship, which may be highly useful in constructing estimates of the maximum attainable parallelism.[1)].

1. Amdahl's law. Maximum speedup obtainment may be hindered by the presence of sequential calculations in the computations being carried out, as the former cannot be parallelized. Let f be the part of the sequential calculations in the applied data processing algorithm, then, in accordance with Amdahl's law, the computation process speedup, if p processors are used, is limited by the value

$$S_p \le \frac{1}{f+(1-f)/p} \le S^* = \frac{1}{f}\ .$$

Thus, for instance, if there are only 10% sequential instructions in the executed computations, the impact of parallelism use cannot exceed the tenfold data processing speedup. For the problem under consideration the computation of the sum of values for the cascade scheme the part of the sequential computations is $f = \log_2 n / n$. As a result, the value of the possible speedup is limited by the estimate $S^* = n / \log_2 n$.

Amdahl's law characterizes one of the most serious problems in the area of parallel programming (there are practically no algorithms without a certain part of sequential instructions). However, the part of sequential actions characterizes very often the sequential feature of the applied algorithms and does not characterize the possibility of parallel problem solution. As a result, the part of sequential computations may be decreased considerably if we choose methods that more appropriate for parallelizing.

It should be also mentioned that Amdahl's law is considered under the assumption that the part of sequential computation f is a constant value and does not depend on the parameter n, which defines the computational problem complexity. However, for a great number of problems the part $f=f(n)$ is a descending function of n. In this case the speedup for a fixed number of processors may be increased at the expense of increasing the computational complexity of the problem to be solved. This remark may be formulated as the statement that the speedup $S_p = S_p(n)$ is the ascending function of the parameter n (this statement is often referred to as the *Amdahl's effect*). Thus, for example, for a problem under consideration – the computation of the sum of values – when a fixed number of processors p are used, the summarized data set may be subdivided into blocks of n/p size. Partial sums may be computed in parallel for the blocks first. Then these sums may be summarized with the help of the cascade scheme. The duration of the sequential part of the executed operations (minimum possible parallel execution time) is in this case

$$T_p = (n/p) + \log_2 p\ ,$$

that leads to the estimation of the sequential computation part as the value

$$f = (1/p) + \log_2 p / n\ .$$

This expression shows that the sequential computation fraction f decreases with the increase of n. And in the limiting case we will obtain the ideal estimate of the maximum possible speedup $S^*=p$.

2. Gustafson-Barsis's law. Let us estimate the maximum attainable speedup proceeding from the existing fraction of sequential calculations in the executed parallel computations:

$$g = \frac{\tau(n)}{\tau(n)+\pi(n)/p}\ ,$$

where $\tau(n)$ is the time of sequential part and $\pi(n)$ is the time of parallel part of computation, i.e.

$$T_1 = \tau(n)+\pi(n)\ ,\ T_p = \tau(n)+\pi(n)/p\ .$$

With regard for the introduced value g we can obtain

$$\tau(n) = g\cdot(\tau(n)+\pi(n)/p),\ \ \pi(n)=(1-g)p\cdot(\tau(n)+\pi(n)/p),$$

that allows us to construct the speedup estimate

$$S_p = \frac{T_1}{T_p} = \frac{\tau(n)+\pi(n)}{\tau(n)+\pi(n)/p} = \frac{(\tau(n)+\pi(n)/p)(g+(1-g)p)}{\tau(n)+\pi(n)/p}\ ,$$

[1)] As earlier we will deduce the regularities with no account for the costs connected with data transmission arrangement. Communication complexity of parallel algorithms will be analyzed in Section 3.

which after simplification may be reduced to *Gustafson-Barsis's law* (see Quinn (2004))

$$S_p = g + (1-g)p = p + (1-p)g\ .$$

In connection with the problem under consideration – summation of values when p processors are used – the time of parallel execution, as it has been mentioned above, is

$$T_p = (n/p) + \log_2 p\ ,$$

that corresponds to the sequential fraction

$$g = \frac{\log_2 p}{(n/p) + \log_2 p}\ .$$

If we increase the number of the summarized values, the value g may be negligibly small. It will provide attaining the ideal possible speedup $S_p = p$.

One more important issue should be taken into account while considering the Gustafson-Barsis's law. If the number of processors used increases, the speed of decreasing the time of parallel problem solving may slacken (after it exceed a certain threshold). However, the complexity of the problems being solved may be increased at the expense of the computation time decrease (thus, for instance, for the problem under consideration we may increase the set of summarized values). The estimate of the obtained speedup may be defined with the help of the patterns, which have been formulated above. Such an analytic estimate is particularly useful as the problem solution of such more complicated cases on one processor may appear rather time-consuming and even impossible because of the RAM shortage, for instance. With regard to these circumstances the speedup estimate obtained in accordance with the Gustafson-Barsis's law is also referred to as the *scaled speedup*. The fact is, this characteristic may show how efficiently parallel computation may be organized if the complexity of problems increases.

2.7. Analysis of Parallel Computation Scalability

The aim of parallel computation application is in many cases not only to decrease the computation execution time, but also to provide the possibility of solving more complicated variants of problem (such statements of the problem which cannot be solved if only uniprocessor computing systems are used). The parallel algorithm capability to efficiently use processors when the computation complexity increases is an important characteristic of the executed calculations. In this connection, the parallel algorithm is referred to as a scalable algorithm if with the increase of the number of processors it provides the speedup increase maintaining constant level of efficiency in processor use . A possible method to characterize the scalability properties is described below.

Let us assess the *total overhead* expenses, which take place in parallel algorithm execution

$$T_0 = pT_p - T_1\ .$$

The total overhead expenses arise, as it is necessary to organize the interaction of processors. It is also necessary to fulfill some additional actions, synchronization of parallel computation and etc.

Making use of the previously introduced notation we can get new expressions for the time of solving the parallel problem solution and the speedup corresponding to it:

$$T_p = \frac{T_1 + T_0}{p}\ ,\quad S_p = \frac{T_1}{T_p} = \frac{pT_1}{T_1 + T_0}\ .$$

With the use of the obtained relation the efficiency of the processor use may be expressed as

$$E_p = \frac{S_p}{p} = \frac{T_1}{T_1 + T_0} = \frac{1}{1 + T_0 / T_1}\ .$$

The latter expression shows that if the problem complexity is fixed ($T_1 = const$), then the efficiency will decrease if the number of processors increases at the expense of the total overhead costs T_0. If the number of processors is fixed, the efficiency of processor used may be improved by the increase of the complexity T_1 of the problem being solved (it is assumed that with the increase of the complexity parameter n the total overhead expenses T_0 increase more slowly than the amount of computations T_1). As a result, if the number of processors increases, the necessary level of efficiency may be provided in the majority of cases by means of the corresponding problem complexity increase. In this connection the proportion of the necessary rates of calculation complexity increase and the number of processors being used becomes an important feature of parallel computations.

Let $E=const$ be the desirable efficiency level of the executed computations. Using the equation for the efficiency we may obtain

$$\frac{T_0}{T_1} = \frac{1-E}{E} \quad \text{or} \quad T_1 = KT_0,\ K = E/(1-E)\ .$$

The dependency $n=F(p)$ between the problem complexity and the number of processors generated by the latter relation is referred to as *isoefficiency function* (see Kumar et al. (1994)).

To illustrate this we will show the derivation of the isoefficiency function for the problem of summarizing numeric values. In this case

$$T_0 = pT_p - T_1 = p((n/p) + \log_2 p) - n = p\log_2 p$$

and the isoefficiency function looks as

$$n = Kp\log_2 p\ .$$

As a result, for instance, to provide the efficiency level $E=0.5$ (i.e. $K=1$) when the number of processors is $p=16$, the number of summarized values must not be smaller than $n=64$. If the number of processors is increased from p to q $(q>p)$ it is necessary to increase the number n of the summarized values $(q\log_2 q)/(p\log_2 p)$ times to provide the proportional speedup increase $(S_q/S_p)=(q/p)$.

2.8. Summary

The chapter describes the computational model as “operations-operands” graph, which may be used for the description of the existing information dependencies in the selected algorithms of problem solving. The model is based on an acyclic-oriented graph, the vertices of which represent operations, and the arcs correspond to the data dependencies of operations . If such a graph is available, it is enough to set the schedule, according to which the distribution of the executed operations among processors is fixed, to define a parallel algorithm.

Representation of calculations with the help of such models allows us to analytically obtain a number of characteristics of the parallel algorithms being developed. Among those characteristics there is the execution time, the optimal schedule scheme, the estimates of maximum possible processing speed of the problem solving methods. The concept of paracomputer as a parallel system with an unlimited number of processors is considered for simpler constructing theoretical estimates.

In order to estimate the efficiency of the parallel computation methods we have discussed such widely used in theory and practice of parallel programming basic quality indicators as speedup and efficiency. Speedup shows how many times faster solving the problem is carried out, if several processors are used. Efficiency characterizes the fraction of time when the processors of a computing system are actually used. The cost of computations is also an important characteristic of the developed algorithm. It is defined as the product of parallel problem solving time and the number of processors used.

To demonstrate the applications of the models and the methods of parallel algorithm analysis we have considered the problem of finding the partial sums of a numeric value sequence. The example helps us to illustrate the problem of sequential algorithm parallelizing complexity. The complexity arises, as these algorithms are not initially oriented at the possibility of parallel computation arrangements. To demonstrate the “hidden” parallelism we have shown the possibility of converting the initial sequential computation scheme and described the cascade scheme, which is obtained as a result of the conversion. Considering the same problem we have shown the possibility to introduce redundant computations for achieving greater parallelism in the executed computations.

In conclusion we have considered the problem of creating the estimates of maximum attainable values of efficiency criteria. Amdahl’s law may be used for the creation of such estimates, which allows us to take into account the existing sequential (non-parallelilzed) computations in the problem solving methods. Gustafson-Barsis's law provides the construction of scaled speedup estimates used to characterize how efficiently parallel computations may be organized with the increase of problem complexity. To define the dependence between the problem complexity and the number of processors, the observation of which provides the necessary efficiency level of parallel computations, we have introduces the concept of the *isoefficiency function*.

2.9. 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).

For the first time Amdahl’s law was stated in Amdahl (1967). Gustafson-Barsis's law was published in Gustavson (1988). The concept of isoefficiency was proposed in Grama et al. (1993).

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

2.10. Discussions

1. How is the “operations-operands” model defined?
2. How is the schedule for the distribution of computations among processors defined?
3. How is the time of parallel algorithm execution defined?
4. What schedule is optimal?
5. How can the minimum possible time of problem solving be defined?
6. What is a paracomputer? What can this concept be useful for?
7. What estimates should be used as the characteristics of the sequential problem solving time?
8. How to define the minimum possible time of parallel problem solving according to “operands-operations” graph?
9. What dependences may be obtained for parallel problem solving time if the number of processor being used is increased or decreased?
10. What number of processors corresponds to the parallel algorithm execution time (periods) comparable in the order with the estimates of minimum possible time of problem solving?
11. How are the concepts “speedup” and “efficiency” defined?
12. Is it possible to attain superlinear speedup?
13. What is the contradictoriness of the speedup and efficiency characteristics?
14. How is the concept of computation cost defined?
15. What is the concept of the cost-optimal algorithm ?
16. What does the problem of parallelizing a sequential algorithm of the numeric values summation lie in?
17. What is the essence of the summation cascade scheme? What is the aim of considering the modified version of the scheme?
18. What is the difference between the speedup and efficiency characteristics for the discussed versions of the summation cascade scheme?
19. What is the parallel algorithm of all the partial sums computation of a numeric value sequence?
20. How is Amdahl’s law formulated? Which aspect of parallel computation does it allow to take into account?
21. What suppositions are used to ground the Gustafson-Barsis's law?
22. How is the isoefficiency function defined?
23. Which algorithm is scalable? Give examples of methods with different level of scalability.

2.11. Exercises

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

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

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

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

2. Evaluate according the Amdahl's law the maximum attainable speedup for the problems given in 11.1
3. Evaluate the scalability speedup for the problems in 11.1
4. Construct the isoefficiency function for the problems given in 11.1

5. Work out a model and make a complete analysis of parallel computation efficiency (speedup, efficiency, maximum attainable efficiency, scalability speedup, isoefficiency function) for the problem of matrix – vector multiplication.

References

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.

Bertsekas, D.P., Tsitsiklis, J.N. (1989). Parallel and distributed Computation. Numerical Methods. - Prentice Hall, Englewood Cliffs, New Jersey.

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.

Gustavson, J.L. (1988) Reevaluating Amdahl's law. Communications of the ACM. 31 (5). pp.532-533.

Kumar V., Grama, A., Gupta, A., Karypis, G. (1994). Introduction to Parallel Computing. - The Benjamin/Cummings Publishing Company, Inc. (2nd edn., 2003)

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