University of Nizhni Novgorod
Faculty of Computational Mathematics & Cybernetics

Introduction to Parallel Programming

Section 2.

Modeling and Analysis of Parallel Computations

Gergel V.P., Professor, D.Sc.,
Software Department

Contents

- "Operations-Operands" Graph as a Computation Model
- The Scheme of Parallel Algorithm Execution
- Evaluation of Parallel Execution Time
- Characteristics of Parallel Algorithm Efficiency
- *Example*: Partial Sums Computation
- Estimation of Maximum Possible Parallelism
- Analysis of Parallel Computation Scalability
- Examples
- Summary

Introduction

- The analysis of **parallelism efficiency** is a crucial point in the development of parallel algorithms:
  - The efficiency estimation of parallelizing a specific algorithm,
  - The estimation of maximum possible speedup for the solution of a certain problem type (the efficiency estimation of all parallel methods for solving a problem)

“Operations-Operands” Graph…

- The model “operations-operands” graph can be used for the description of the information dependencies in selected algorithms of solving problems
- To simplify the problem it can be assumed:
  - The execution time of any computational operations will be the same and will be equal to 1 (in some units of measurement),
  - The data transmission among computing devices is carried out instantaneously without any time consumption.

“Operations-Operands” Graph…

Let us represent the set of the operations, carried out by the algorithm of computational problem solution, and the information dependencies, which exist among the operations, as an *acyclic oriented graph*

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

where

$V = \{1,...,|V|\}$ is the set of graph vertexes, which represent the algorithm operations being executed,

R is a set of graph arcs; *r(i,j)* belongs to the graph only if the operation *j* makes use of the result obtained by execution of operation *i*

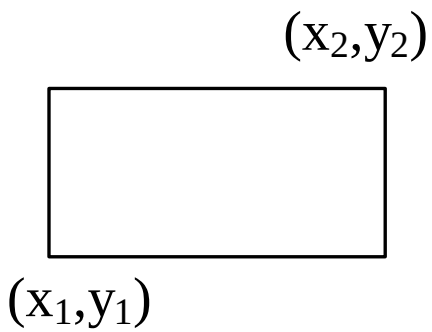
The vertexes without the incoming arcs may be used to assign the input operations, and the arcs without outgoing arcs may be used for output operations

$\overline{V}$ is the set of graph arcs without input arcs,

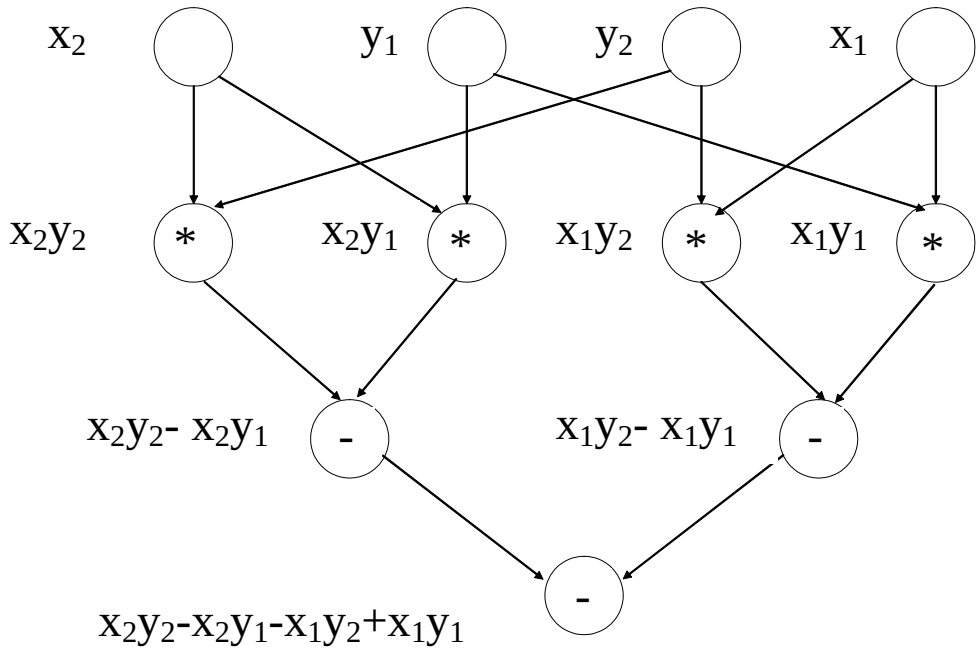
$d(G)$ is the diameter (length of maximum path).

“Operations-Operands” Graph…

Example: the graph of the algorithm used to calculate the square of the rectangle specified by the coordinates of its two opposite angles



$$S = (x_2-x_1)(y_2-y_1) =$$
$$= x_2y_2-x_2y_1-x_1y_2+x_1y_1$$



“Operations-Operands” Graph

- Different computation schemes possess different capabilities of parallelizing. Selecting the computational scheme that is **the most suitable** for parallel execution has to be taken in constructing a computation model
- The algorithm operations, which do not have paths among them within the selected computation scheme, may be executed **in parallel**

The Scheme of Parallel Algorithm Execution

- 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)* :

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

  - ***i*** is the number of operation,
  - P_i is the number of processor at which the operation ***i*** has be executed,
  - t_i is the operation start time.
- It is necessary to meet the following requirements:
  - the same processor must not be assigned to different operations simultaneously:

$$\forall i, j \in V : t_i = t_j \Rightarrow P_i \neq P_j,$$

  - all the necessary data must have been calculated before operation execution starts:

$$\forall (i, j) \in R \Rightarrow t_j \geq t_i + 1$$

Evaluation of Parallel Execution Time…

- The model of the parallel algorithm:

$$A_p(G, H_p)$$

- The time of parallel algorithm execution is determined by the maximum time value used in the schedule:

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

- Time of parallel algorithm execution with the optimal schedule:

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

Evaluation of Parallel Execution Time…

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

$$T_p = \min_G T_p(G)$$

- The minimum possible time of the parallel algorithm execution if an unlimited number of processors are used:

$$T_\infty = \min_{p \geq 1} T_p$$

(the concept of the computer system with the infinite number of processors usually called a ***paracomputer***)

Evaluation of Parallel Execution Time…

- The algorithm execution time if one processor is used (for the specific computation scheme):

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

- Time of sequential algorithm execution:

$$T_1 = \min_G T_1(G)$$

- The time of the best sequential solution:

$$T_1^* = \min T_1$$

Constructing such estimates is an important aspect in analyzing parallel algorithms, as it is necessary to evaluate the effect of the parallelism use

Evaluation of Parallel Execution Time…

- **Theorem 1**

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

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

Evaluation of Parallel Execution Time…

- **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 arcs (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 edges in the algorithm scheme.

Evaluation of Parallel Execution Time…

- **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.

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

Evaluation of Parallel Execution Time…

- **Theorem 4**

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

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

Evaluation of Parallel Execution Time…

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,

$$p \geq T_1 / T_\infty \Rightarrow T_p \leq 2T_\infty$$

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.

$$p < T_1 / T_\infty \Rightarrow \frac{T_1}{p} \leq T_p \leq 2\frac{T_1}{p}$$

Evaluation of Parallel Execution Time

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

Characteristics of Parallel Algorithm Efficiency…

- ***Speedup***

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

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

  (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)

Characteristics of Parallel Algorithm Efficiency…

- ***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)

Characteristics of Parallel Algorithm Efficiency…

Remarks:

- *Superlinear speedup* $S_p(n)>p$ may appear under certain circumstances:
  - There is 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),
  - The non-linear type of the dependency of the problem solution complexity with respect to the amount of the data being processed,
  - The difference of parallel and sequential method computational schemes.
- The attempts to improve the parallel computation efficiency with respect to one of the criteria (speedup or efficiency) may lead to the worsening of the situation for the other criterion, as the criteria of parallel computation efficiency are conflicting.

Characteristics of Parallel Algorithm Efficiency

- ***Computation cost***

$$C_p = pT_p$$

- The ***cost-optimal*** parallel algorithm is the method, the cost of which is proportional to the time of the best sequential algorithm execution.

Example: Partial Sums Computation…

- Simple problem of finding partial sums of numerical value sequence (*prefix sum problem*):

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

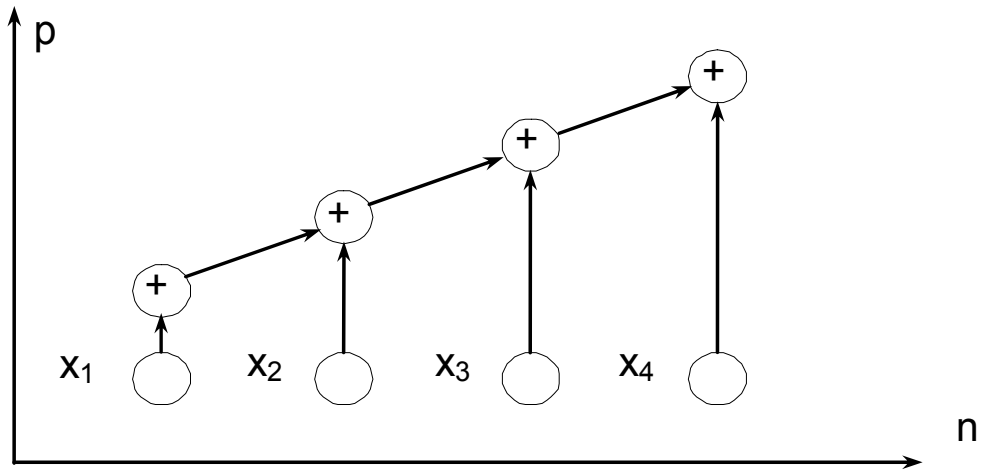
- The computation of the total sum of the available set of values (particular case of the general *reduction problem*):

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

Example: Partial Sums Computation…

- **Sequential summation** of the elements of a series of values

$$S = \sum_{i=1}^{n} x_i, \qquad G_1 = (V_1, R_1)$$

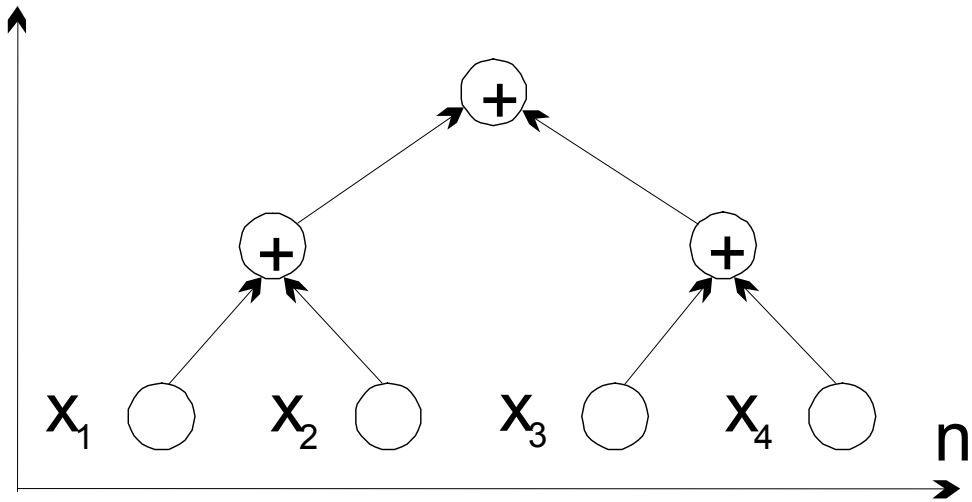


*This "standard" sequential summation algorithm allows **only strictly serial execution** and cannot be parallelized*

Example: Partial Sums Computation…

❑ Cascade Summation Scheme

$$G_2 = (V_2, R_2)$$



- $V_2 = \{ (v_{i1},\ldots,v_{ili}),\ 0 \le i \le k,\ 1 \le l_i \le 2^{-i}n \}$ are graph vertexes,
- $(v_{01},\ldots,\ v_{0n})$ - input operations,
- $(v_{11},\ldots,v_{1n/2})$ - the operations of the first iteration and etc,
- $R_2 = \{ (v_{i-1,2j-1}v_{ij}),(v_{i-1,2j}v_{ij}),\ 1 \le i \le k,\ 1 \le l_i \le 2^{-i}n \}$ - the set of the graph arcs.

Example: Partial Sums Computation…

- The number of the cascade scheme iterations appears to be equal to the value:

$$k = \log_2 n$$

- The total number of summation operations is:

$$K_{\text{seq}} = n/2 + n/4 + \ldots + 1 = n - 1$$

- In parallel execution of separate iterations of the cascade scheme the total number of parallel summation operations is equal to:

$$\mathrm{K}_{par} = \log_2 n$$

Example: Partial Sums Computation…

- The speedup and efficiency of the cascade scheme of the summation algorithm may be estimated as:

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

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

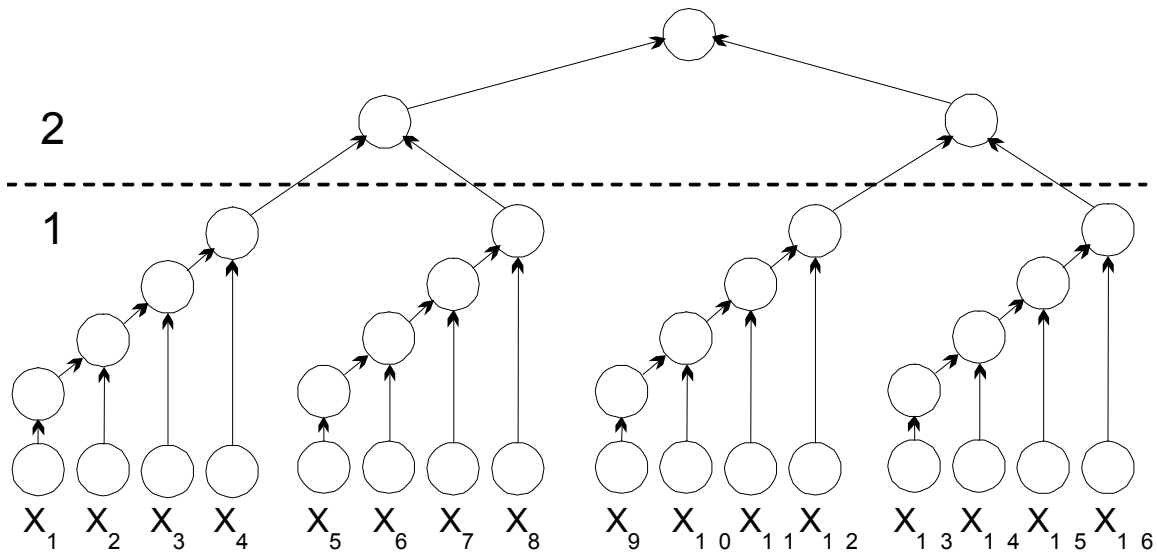
where *p=n/2* is the number of processors necessary for the cascade scheme execution:

  - the time of parallel cascade scheme execution coincides with the paracomputer estimate in Theorem 2,
  - the efficiency of processors decreases when the number of summable values increases:

$$\lim E_p = 0 \quad npu \quad n \to \infty$$

Example: Partial Sums Computation…

- **Modified Cascade Scheme:**
  - 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,
  - During the second phase a conventional cascade scheme is used for the obtained $(n/\log_2 n)$ sums of separate groups.



Example: Partial Sums Computation…

- 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
- This summation method is characterized by the following values:

$$T_P = 2\log_2 n$$

for $p = (n/\log_2 n)$ processors

Example: Partial Sums Computation…

- 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 shows that the speedup for the suggested parallel algorithm has decreased twice,
  - For the efficiency of the new summation method it is possible to obtain asymptotic nonzero estimate:

$$E_p = (n-1)/2n \geq 0.25,\ \lim E_p = 0.5\ when\ n \to \infty.$$

  - The modified cascade algorithm is cost-optimal as the calculation cost is proportional to the time of sequential algorithm execution:

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

Example: Partial Sums Computation…

- **Computation of All Partial Sums…**
  - 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

Example: Partial Sums Computation…

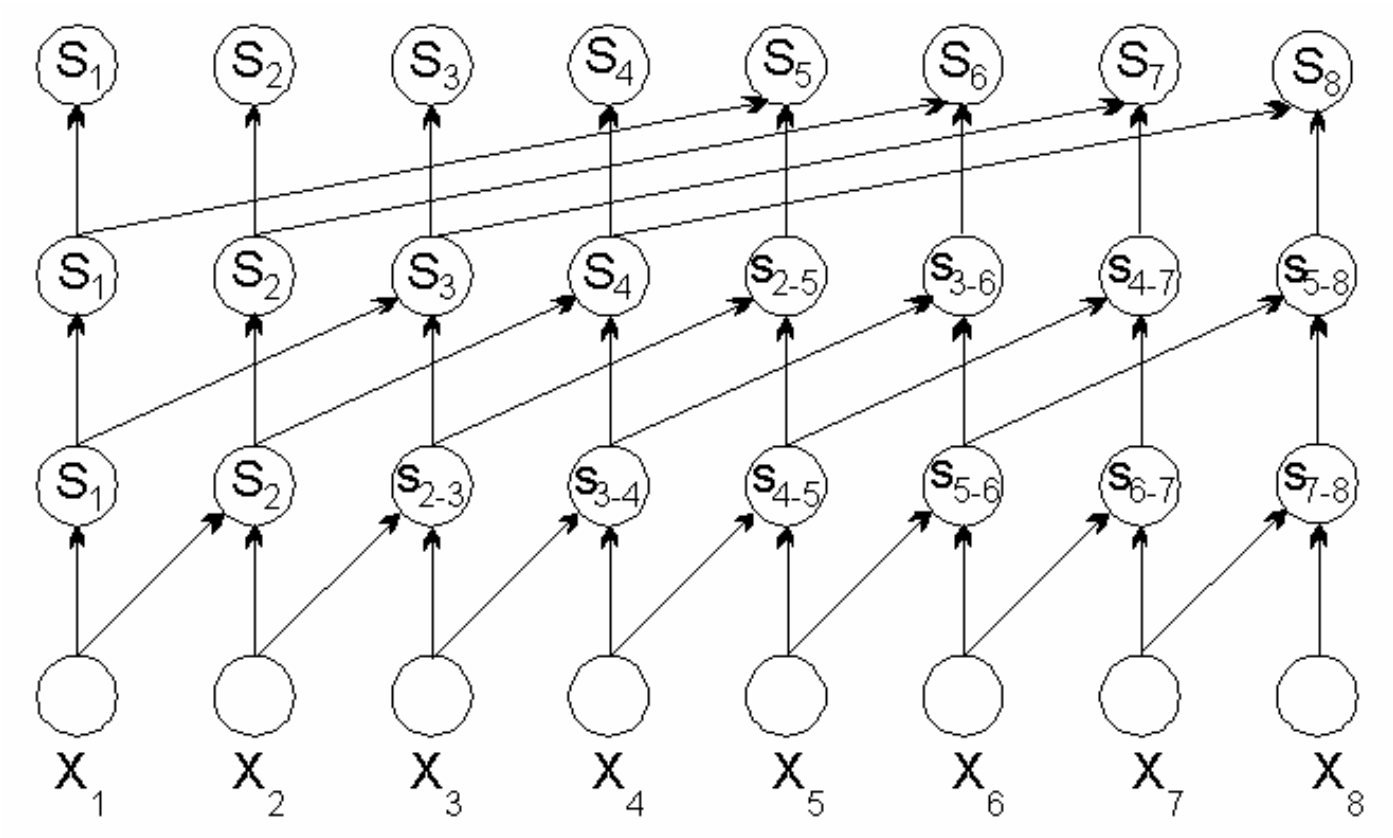
❑ Computation of All Partial Sums…

– The algorithm, which provides obtaining the results in $log_2 n$ parallel operations:

- 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.

Example: Partial Sums Computation…

- Computation of All Partial Sums…



Example: Partial Sums Computation

❑ Computation of All Partial Sums:

– The total number of the executed scalar operations is defined by the value:

$$\mathrm{K}_{seq} = n \log_2 n,$$

– The necessary number of processors is defined by the number of summarized values:

$$p=n,$$

– The speedup and efficiency of the parallel algorithm are estimated in the following way:

$$S_p = T_1 / T_p = n / \log_2 n$$

$$E_p = T_1 / pT_p = n / \left(p \log_2 n \right) = n / n \log_2 n = 1 / \log_2 n$$

Estimation of Maximum Possible Parallelism…

- Estimation of parallel computation efficiency requires the knowledge of the best (maximum possible) values of speedup and efficiency
- Attainment of the ideal values $S_p=p$ for speedup and $E_p=1$ for efficiency may not be provided for all time-consuming computational problems

Estimation of Maximum Possible Parallelism…

❑ Amdahl's Law

– Maximum possible speedup $T_p=p$ may not be achieved because of 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,

– The computation speedup, if p processors are used, is limited by the value:

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

Estimation of Maximum Possible Parallelism…

- **Amdahl's Law. Remarks:**
  - The part of sequential computations may be decreased considerably if we choose more appropriate methods for parallelizing
  - ***Amdahl's Effect***:
    - 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 as a result of increasing the computational complexity of the problem to be solved,
    - In that case the speedup $S_p = S_p(n)$ is the ascending function of the parameter ***n***.

Estimation of Maximum Possible Parallelism…

❑ Gustafson-Barsis's Law…

Let us estimate the maximum possible 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.

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

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:

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

Estimation of Maximum Possible Parallelism

❑ Gustafson-Barsis's Law

Simplification of the last estimation may be reduced to ***Gustafson-Barsis's law***

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

With regard to these circumstances the speedup estimate obtained in accordance with the Gustafson-Barsis's law is also referred to as *scaled speedup*. The fact is, this characteristic may show how efficiently parallel computation may be organized if the complexity of problems increases

Analysis of Parallel Computation Scalability…

*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 utilization*

Analysis of Parallel Computation Scalability…

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.

$$T_0 = pT_p - T_1 \ .$$

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 utilization may be expressed as

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

Analysis of Parallel Computation Scalability…

- For the fixed problem complexity (T_1=*const*), the efficiency will usually decrease with the increasing number of processors due to the growth of the total overheads T_0,
- If the number of processors is fixed, the efficiency of processor utilization may be improved by the increase of the complexity T_1 of the problem being solved,
- 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.

Analysis of Parallel Computation Scalability…

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

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

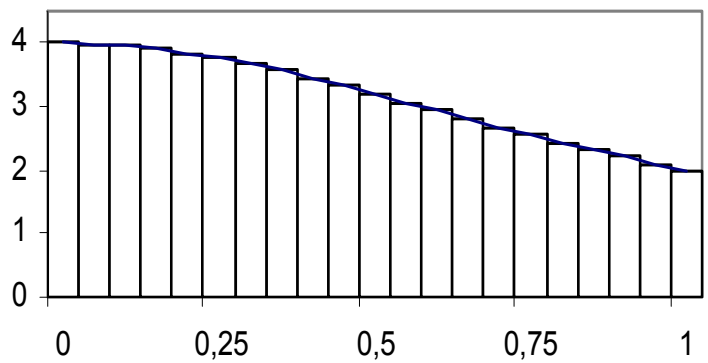
– The dependency *n=F(p)* between the problem complexity and the number of processors generated by the latter relation is referred to as the ***isoefficiency function***.

Example*: *Computation of the constant π...

- The value of constant π can be computed by means of the integral

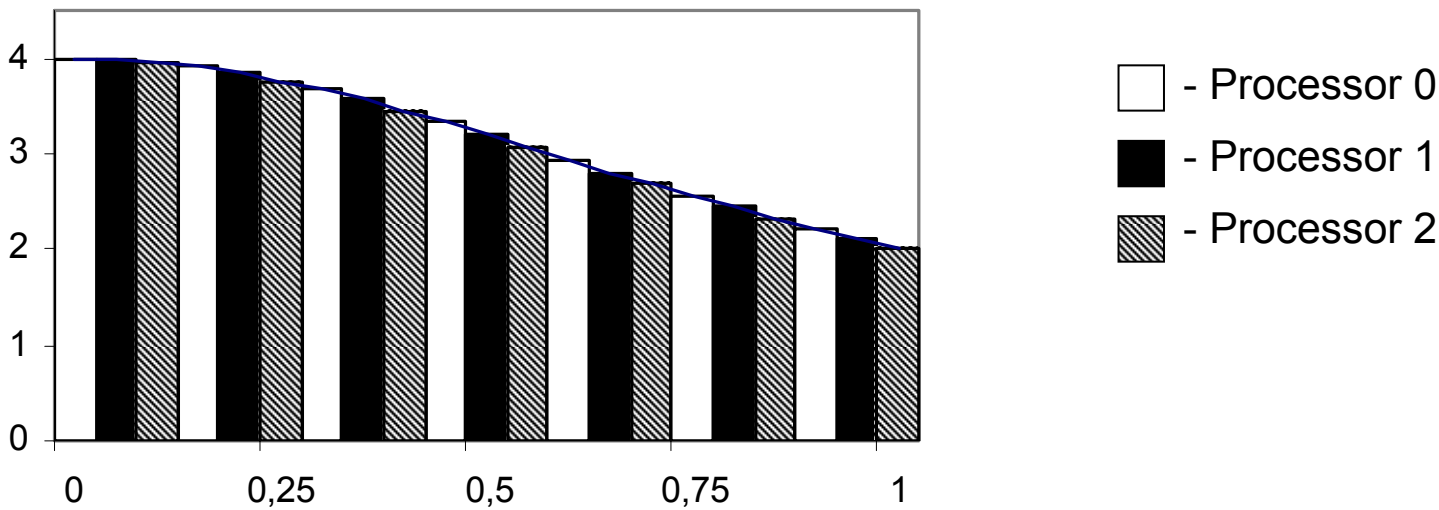
$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx$$

- To compute this integral the method of rectangles can be used for numerical integration



Example*: *Computation of the constant π...

- Cyclic scheme can be used to distribute of calculations among the processors
- Partial sums, that were calculated on different processors, have to be summed



Example: *Computation of the constant π…*

Efficiency Analysis…

- ***n*** – number of intervals of segment [0,1],
- Computational complexity of the algorithm:

$$W = T_1 = 6n,$$

- Number of grid nodes for each processor

$$m = \lceil n/p \rceil \le n/p + 1,$$

- Computational complexity for each processor

$$W_p = 6m = 6n/p + 6.$$

Example: *Computation of the constant* π

Efficiency Analysis

- **Time** of parallel algorithm execution

$$T_p = 6n/p + 6 + \log_2 p,$$

- **Speedup**

$$S_p = T_1/ T_p = 6n / (6n/p + 6 + \log_2 p),$$

- **Efficiency**

$$E_p = 6n / (6n + 6p + p\log_2 p),$$

- **Isoefficiency function**

$$W = K(pTp - W) = K(6p + p\log_2 p)$$

$$\Rightarrow n = [K(6p + p\log_2 p)]/6, \quad (K=E/(1–E))$$

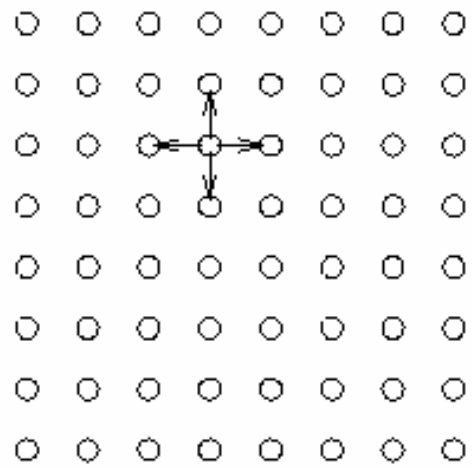
Ex. E=0.5 when p = 8 $\rightarrow n$ = 12

when p = 64 $\rightarrow$ n = 128

Example: *Finite Difference Method…*

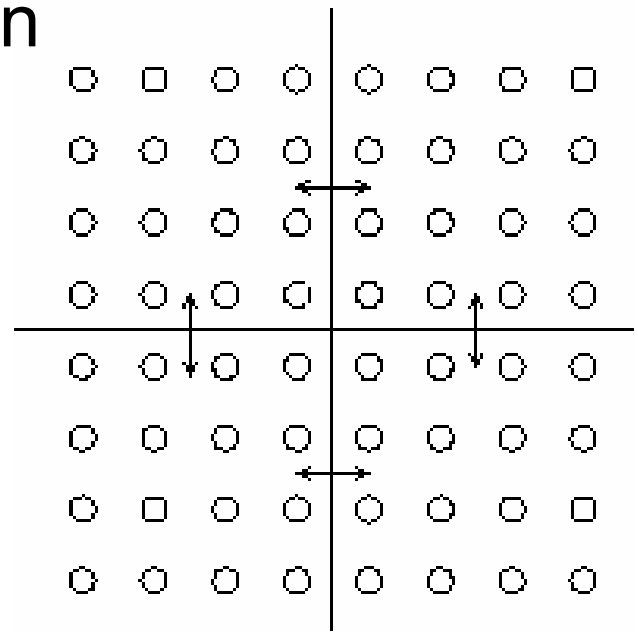
- The finite difference method is widely used for numerical solving of partial differential equations,
- Lets consider the scheme ($N = 2$)

$$X_{i,j}^{t+1} = w(X_{i,j-1}^{t} + X_{i,j+1}^{t} + X_{i-1,j}^{t} + X_{i+1,j}^{t}) + (1-w)\, X_{i,j}^{t}$$



Example*: *Finite Difference Method…

- Each processor performs calculations on the rectangle subregion with the $(n/\sqrt{p})*(n/\sqrt{p})$ nodes of the grid,
- After each iteration it is necessary to perform the synchronization



Example: *Finite Difference Method*

Efficiency Analysis

- $T_1 = W = 6n^2M$ *(M – number of iterations),*
- $T_p = 6M + M \log_2 p$,
- **Efficiency**

$$S_p = T_1 / T_p = 6n^2 / (6n^2/p + \log_2 p),$$

- **Isoefficiency function**

$$W = K(pT_p - W) = K(p\log_2 p),$$

$$\Rightarrow n^2 = [K(p\log_2 p)]/6, \quad (K=E/(1-E))$$

The finite difference method is more scalable than the method of rectangles

Summary…

- The section describes the computational model as the “operations-operands” graph, which can be used for the description of the existing information dependencies in the selected algorithms of problem solving
- 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 such widely used criteria as *speedup*, *efficiency*, *cost* and *isoefficiency* are discussed

Summary

- To demonstrate the applications of the models and the methods of parallel algorithm analysis it has been considered the problem of finding the partial sums, the problem of numerical integration and the finite difference method
- To obtain the estimates of maximum possible values of efficiency characteristics *Amdahl's law* and *Gustafson-Barsis's law* are discussed
- Finally the conception of the function isoefficiency is given

Discussions…

- How can the time of parallel algorithm execution be defined?
- How can the minimum possible time of problem solving be defined?
- What estimates should be used as the characteristics of the sequential problem solving time?
- How to define the minimum possible time of parallel problem solving according to the “operands-operations” graph?
- What number of processors corresponds to the parallel algorithm execution time comparable in the order with the estimates of minimum possible time of problem solving?
- Is it possible to attain superlinear speedup?

Discussions

- What is the contradictoriness of the speedup and efficiency characteristics?
- What is the concept of the cost-optimal algorithm?
- How is the Amdahl's law formulated? Which aspect of parallel computation does it allow to take into account?
- What suppositions are used to ground the Gustafson-Barsis's law?
- Which algorithm is scalable? Give examples of methods with the different level of scalability.

Exercises…

- Develop a model and evaluate speedup and efficiency of the parallel computations for:
  - The problem of the scalar product of two vectors,
  - The problem of choosing the maximum and minimum values for the given set of values,
  - The problem of finding the mean value for the given set of values.
- Evaluate according the Amdahl's law the maximum possible speedup for these problems.

Exercises

- Evaluate the scalability speedup for these problems.
- Construct the isoefficiency function for these problems.
- Work out a model and make a complete analysis of parallel computation efficiency (speedup, efficiency, maximum possible 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.
- **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.

References…

- **Bertsekas, D.P., Tsitsiklis, J.N.** (1989). Parallel and distributed Computation. Numerical Methods. - Prentice Hall, Englewood Cliffs, New Jersey.
- **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.
- **Zomaya, A.Y.** (Ed.) (1996). Parallel and Distributed Computing Handbook. - McGraw-Hill.

Next Section

- **Communication Complexity Analysis of Parallel Algorithms**

Author's Team

Gergel V.P., Professor, Doctor of Science in Engineering, Course Author

Grishagin V.A., Associate Professor, Candidate of Science in Mathematics

Abrosimova O.N., Assistant Professor (chapter 10)

Kurylev A.L., Assistant Professor (learning labs 4,5)

Labutin D.Y., Assistant Professor (ParaLab system)

Sysoev A.V., Assistant Professor (chapter 1)

Gergel A.V., Post-Graduate Student (chapter 12, learning lab 6)

Labutina A.A., Post-Graduate Student (chapters 7,8,9, learning labs 1,2,3, ParaLab system)

Senin A.V., Post-Graduate Student (chapter 11, learning labs on Microsoft Compute Cluster)

Liverko S.V., Student (ParaLab system)

About the project

The purpose of the project is to develop the set of educational materials for the teaching course “Multiprocessor computational systems and parallel programming”. This course is designed for the consideration of the parallel computation problems, which are stipulated in the recommendations of IEEE-CS and ACM Computing Curricula 2001. The educational materials can be used for teaching/training specialists in the fields of informatics, computer engineering and information technologies. The curriculum consists of **the training course “Introduction to the methods of parallel programming”** and **the computer laboratory training “The methods and technologies of parallel program development”**. Such educational materials makes possible to seamlessly combine both the fundamental education in computer science and the practical training in the methods of developing the software for solving complicated time-consuming computational problems using the high performance computational systems.

The project was carried out in Nizhny Novgorod State University, the Software Department of the Computing Mathematics and Cybernetics Faculty (http://www.software.unn.ac.ru). The project was implemented with the support of Microsoft Corporation.