University of Nizhni Novgorod
Faculty of Computational Mathematics & Cybernetics

Introduction to Parallel Programming

Section 11.

Parallel Methods for Graph Calculations

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

Contents

Overview of Graph Terminology…

- Graphs are widely used for modeling various phenomena, processes and systems
- Let ***G*** be a graph

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

where

- ***V*** is the set of vertices:

$$v_i, 1 \le i \le n,$$

- ***R*** is the set of arcs:

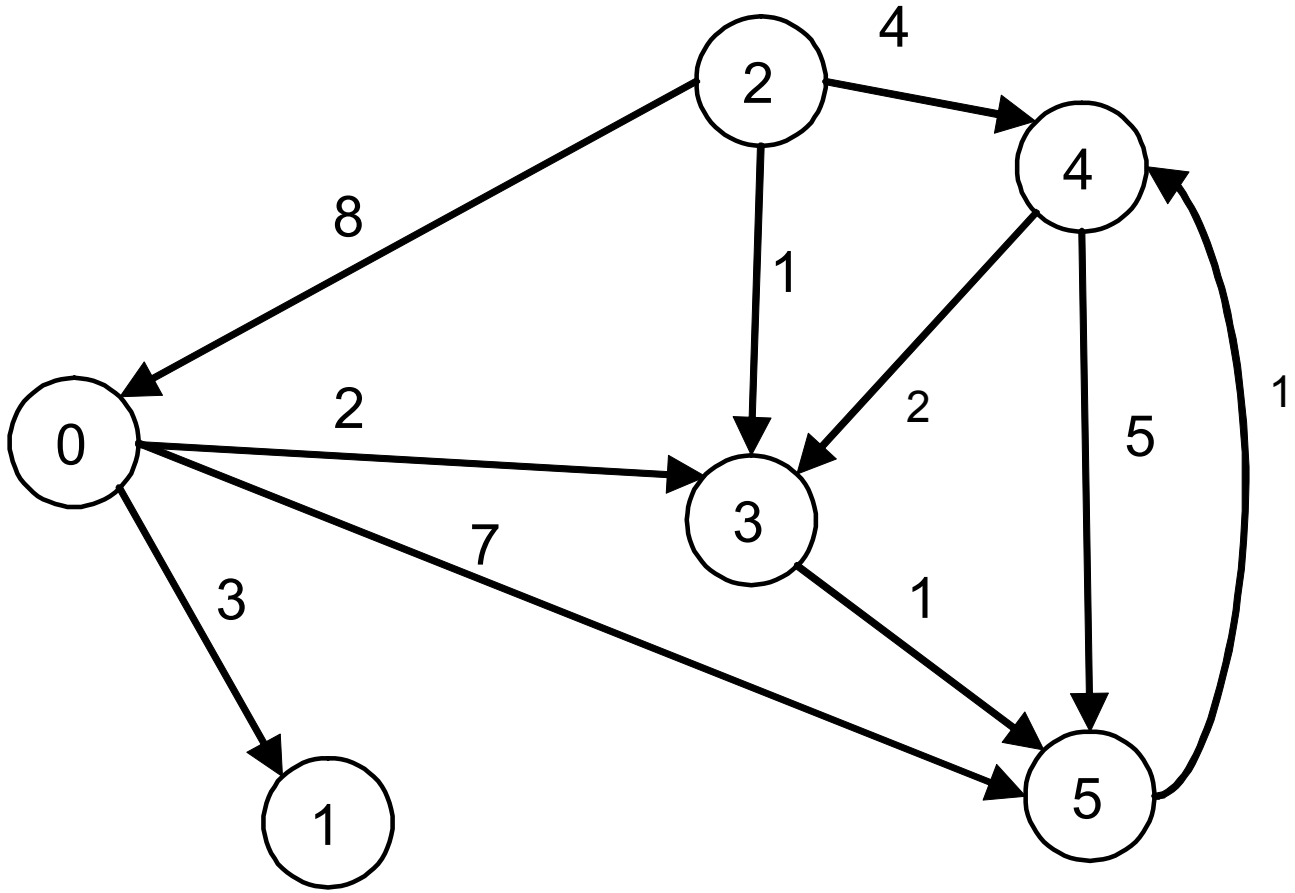
$$r_j = (v_{s_j}, v_{t_j}),\ 1 \le j \le m$$

- Generally, certain numerical characteristics (*weights*) may be assigned to the arcs:

$$w_j, 1 \le j \le m$$

Overview of Graph Terminology…

- **Example of the Weighted Directed Graph**



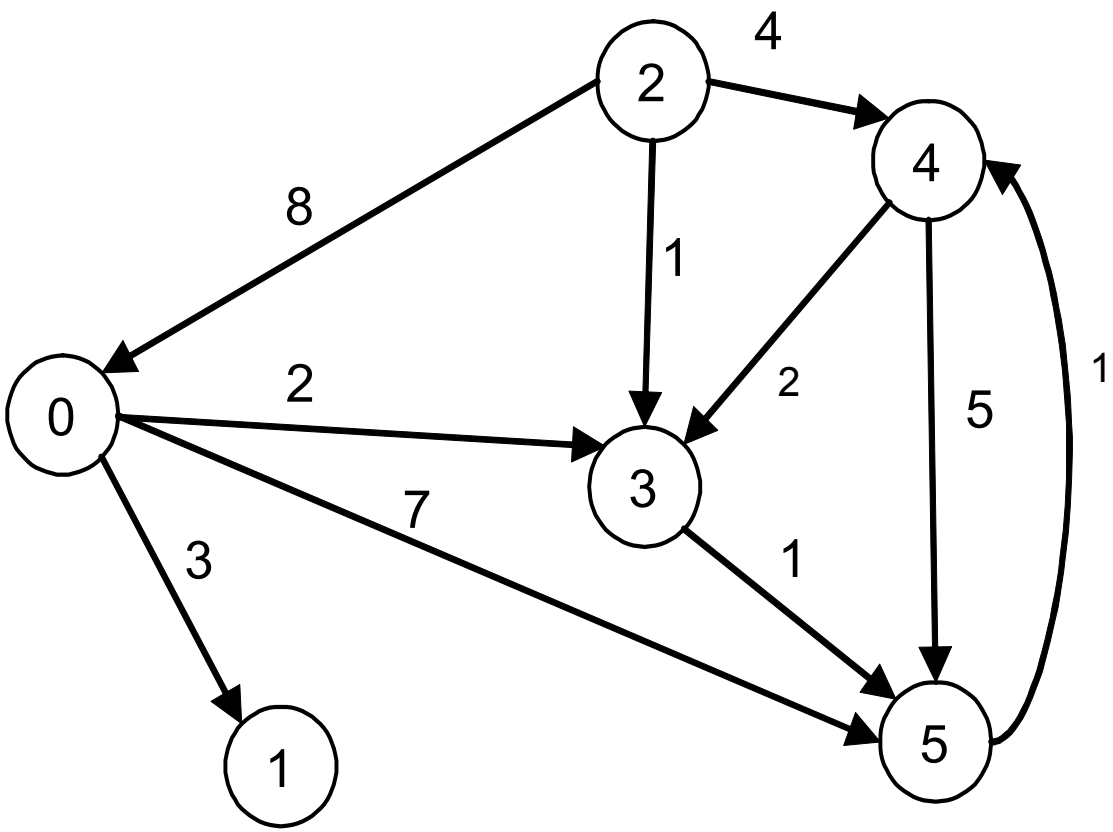
Overview of Graph Terminology…

- If the number of arcs in the graph is small then to set the graph it is possible to use the list enumerating the arcs of the graph
- To represent dense graphs, most if not all of vertices of which are linked by arcs, the *adjacency matrix* may efficiently be used:

$$A = (a_{ij}),\ 1 \le i,\ j \le n,\ \ a_{ij} = \begin{cases} w(v_i, v_j), & if \quad (v_i, v_j) \in R, \\ 0, & if \quad i = j, \\ \infty, & otherwise. \end{cases}$$

Overview of Graph Terminology

❑ Example of the Adjacency Matrix



$$\begin{pmatrix} 0 & 3 & \infty & 2 & \infty & 7 \\ \infty & 0 & \infty & \infty & \infty & \infty \\ 8 & \infty & 0 & 1 & 4 & \infty \\ \infty & \infty & \infty & 0 & \infty & 1 \\ \infty & \infty & \infty & 2 & 0 & 5 \\ \infty & \infty & \infty & \infty & 1 & 0 \end{pmatrix}$$

The Problem of the Search for All the Shortest Paths…

- **Problem Statement**:
  - Let the weighted directed graph ***G*** with *N* vertices be given and all weights have non-negative values,
  - The problem to be studied is to find the minimum paths for each pair of the graph vertices of the given graph ***G***,
  - For solving *the All-Pairs Shortest Path Problem*, in most cases, *the Floyd's algorithm* is used

The Problem of the Search for All the Shortest Paths…

- **The Sequential Floyd's Algorithm**

```
// Algorithm 11.1
// The sequential Floyd's algorithm
for (k = 0; k < n; k++)
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      A[i,j] = min(A[i,j],A[i,k]+A[k,j]);
```

- The complexity of the Floyd's algorithm is $O(n^3)$

The Problem of the Search for All the Shortest Paths…

❑ Problem Decomposition into Subtasks…

– The effective way of parallel computations is to execute simultaneously all the operations at each iteration k, $0 \le k < n$,

– These operations can be computed in parallel as the elements a_{ik} and a_{kj} are not changed at iteration k for any pair of indices (i,j).

Let's prove it:

$$a_{ij} \leftarrow \min (a_{ij}, a_{ik} + a_{kj})$$

- for $i=k$ we will have:

$$a_{kj} \leftarrow \min (a_{kj}, a_{kk} + a_{kj}),$$

but then the value a_{kj} will not change as $a_{kk}=0$,

- for $j=k$ the argumentation is the same.

The Problem of the Search for All the Shortest Paths…

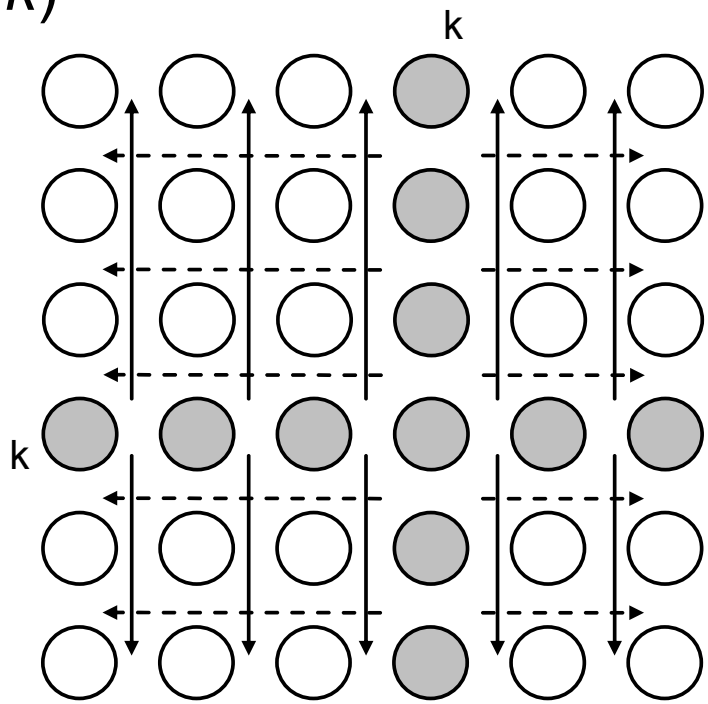
- **Problem Decomposition into Subtasks:**
  - ***The basic subtask*** can be set as calculations, that corresponds to renovating a single element of the matrix ***A*** (to indicate the subtasks the corresponding indices of the matrix elements can be taken)

The Problem of the Search for All the Shortest Paths…

- **Analysis of Information Dependencies…**
  - Computations in the subtasks become possible only if each subtask (*i,j*) contains elements a_{ij}, a_{ik}, a_{kj} of matrix *A,*
  - Each element a_{kj} of row *k* of matrix *A* must be transmitted to all the subtasks (*k,j*), $0 \le j < n$, and each element a_{ik} of row *k* of matrix *A* must be transmitted to all the subtasks (*i,k*), $0 \le i < n$.

The Problem of the Search for All the Shortest Paths…

- **Analysis of Information Dependencies:**
  - The Information Dependencies of the Basic Subtasks (the arrows show the direction of exchanging values at iteration *k*)



The Problem of the Search for All the Shortest Paths…

- **Scaling and Distributing the Subtasks among the Processors:**
  - The use of *the block-striped scheme* of matrix *A* decomposition is a possible way to aggregate the computations:
    - Renovating the elements of one or several rows (*horizontal block-striped decomposition*),
    - Renovating the elements of one or several columns (*vertical block-striped decomposition*)
  - Hereinafter the horizontal block-striped decomposition of the matrix *A* is considered

The Problem of the Search for All the Shortest Paths…

- **Efficiency Analysis…**
  - The speed up and efficiency of the parallel Floyd's algorithm looks as follows:

$$S_p = \frac{n^3}{(n^3/p)} = p \qquad E_p = \frac{n^3}{p \cdot (n^3/p)} = 1$$

Developed method of parallel computations allows to achieve ideal speed-up and efficiency characteristics

The Problem of the Search for All the Shortest Paths…

- **Efficiency Analysis** (detailed estimates):
  - Time of the calculation part of the parallel algorithm can be evaluated as follows:

$$T_p(calc) = n^2 \cdot \lceil n/p \rceil \cdot \tau,$$

  - The total duration of executing communication operations can be determined by the following expression:

$$T_p(comm) = n\lceil \log_2 p \rceil (\alpha + wn/\beta)$$

  (we suppose that all data transmissions can be done simultaneously at the same iteration)

The total execution time for the parallel Floyd's algorithm can be determined in the following way:

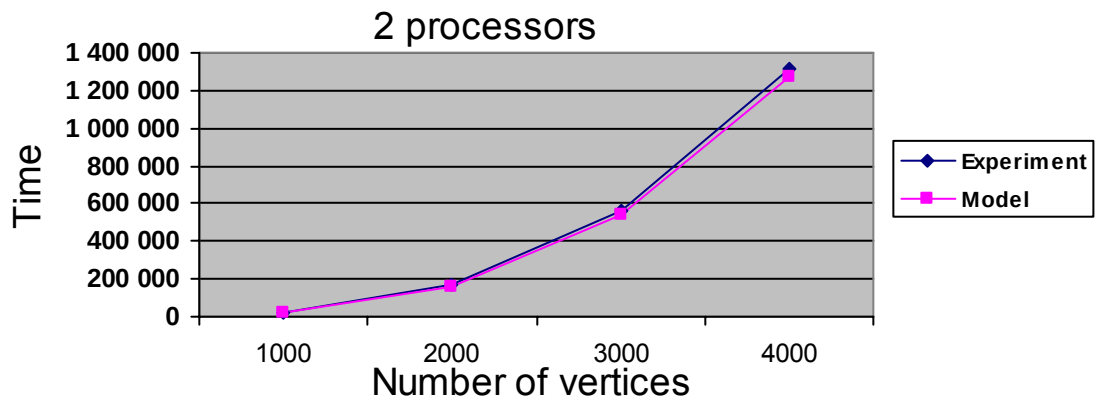
$$T_p = n^2 \cdot \lceil n/p \rceil \cdot \tau + n \cdot \lceil \log_2(p) \rceil (\alpha + w \cdot n/\beta)$$

The Problem of the Search for All the Shortest Paths…

❑ Results of Computational Experiments…

– Comparison of theoretical estimations and results of computational experiments

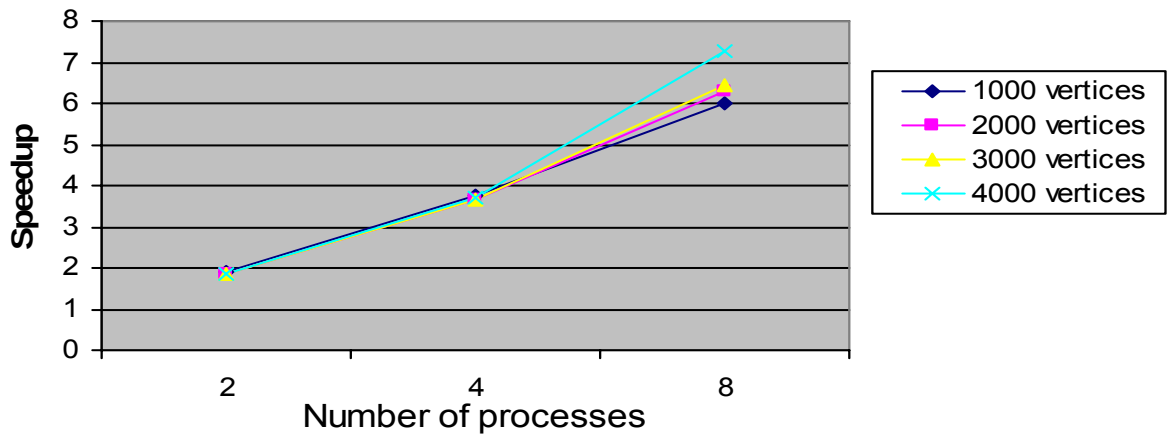
The number of vertices	Sequential algorithm		Parallel algorithm					
			2 processors		4 processors		8 processors	
	Model	Experiment	Model	Experiment	Model	Experiment	Model	Experiment
1000	41,600	39,686	21,800	20,996	11,900	10,530	6,910	6,590
2000	317,000	306,228	159,000	166,219	80,500	83,468	41,400	48,796
3000	1070,000	1027,611	537,000	557,887	270,000	279,830	137,000	160,013
4000	2540,000	2445,473	1270,000	1320,205	639,000	661,641	323,000	336,713



The Problem of the Search for All the Shortest Paths

- **Results of Computational Experiments:**
  - **Speedup**

The number of vertices	Sequential algorithm	Parallel algorithm					
		2 processors		4 processors		8 processors	
		Time	Speed up	Time	Speed up	Time	Speed up
1000	39,686	20,996	1,890	10,530	3,769	6,590	6,022
2000	306,228	166,219	1,842	83,468	3,669	48,796	6,276
3000	1027,611	557,887	1,842	279,830	3,672	160,013	6,422
4000	2445,473	1320,205	1,852	661,641	3,696	336,713	7,263



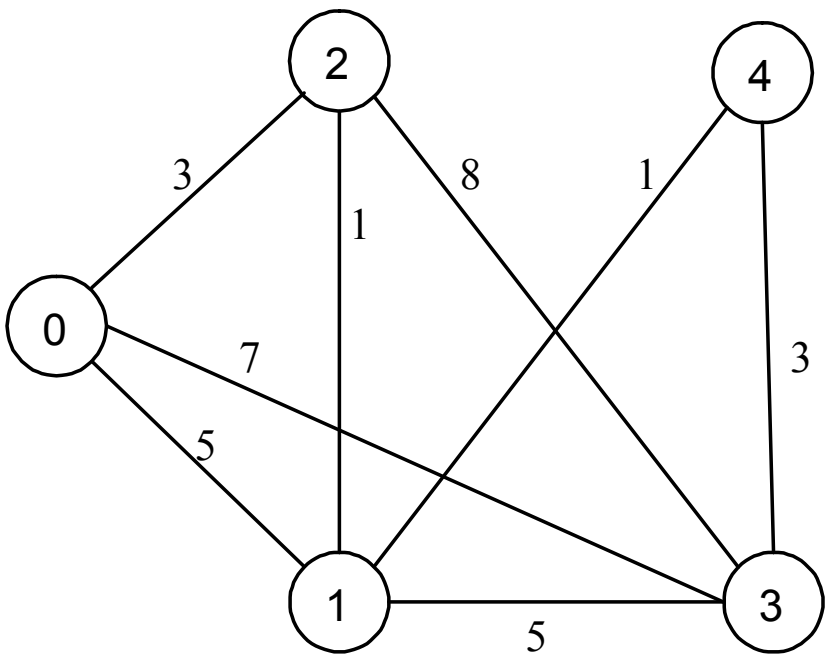
The Problem of Finding the Minimum Spanning Tree…

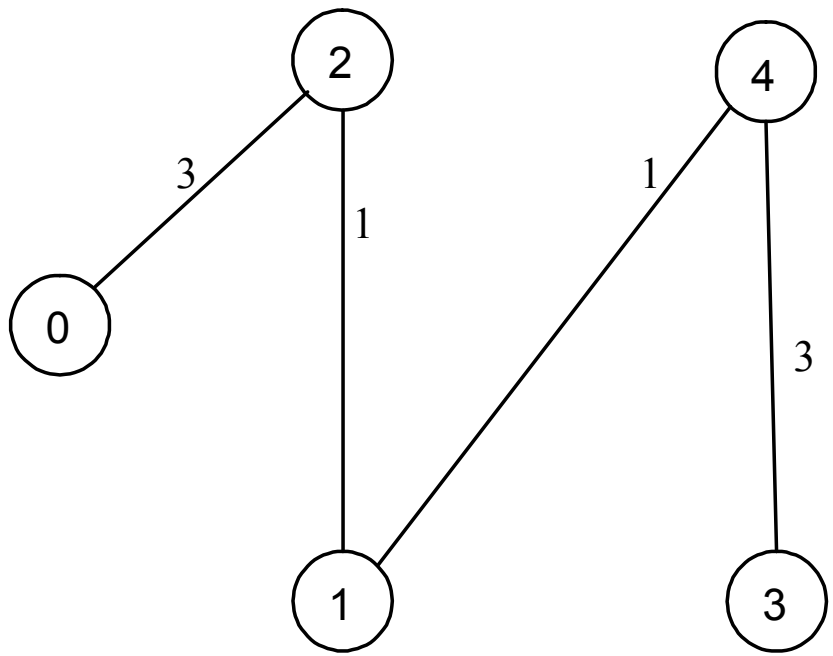
❑ Problem Statement:

- *The spanning tree* of the undirected graph *G* is the subgraph *T* of the graph *G*, which is a tree and contains all the vertices of graph *G*,
- The subgraph weight for the weighted graph is equal to the sum of all the weights of the subgraph arcs. Thus, *the minimum spanning tree* (*MST*) *T* may be defined as the spanning tree of the minimum weight,
- Thus, the problem to be studied is to find the Minimum Spanning Tree for a given graph *G*

The Problem of Finding the Minimum Spanning Tree…

- **The Example of Weighted Undirected Graph and its Minimum Spanning Tree**





The Problem of Finding the Minimum Spanning Tree…

The Sequential Prim's Algorithm…

– Let V_T be the set of the vertices, which are already included into MST by the algorithm. Let the values d_i, $1 \le i \le n$, characterize the arcs of minimum weight from the vertices, which are not included into MST yet, to the set V_T, i.e.

$$\forall i \notin V_T \Rightarrow d_i = \min\{w(i,u) : u \in V_T, (i,u) \in R\}$$

(if for some vertex i there is no arcs in V_T, the value d_i is set by ∞),

– At the start of the algorithm execution the root vertex MST s is selected. It is assumed that $V_T=\{s\}$, $d_s=0$

The Problem of Finding the Minimum Spanning Tree…

- **The Sequential Prim's Algorithm:**
  - The operations, carried out at each iteration, consist in the following:
    - Calculating the values d_i for all the vertices not included into MST,
    - Finding the vertex t of graph G, which has the arc of the minimum weight, to the set V_T

$$t : d_t = \min(d_i),\ i \notin V_T$$

    - Including the vertex t into V_T.
  - The time complexity of calculating MST is estimated by the square dependence subject to the number of graph vertices $O(n^2)$

The Problem of Finding the Minimum Spanning Tree…

Problem Decomposition into Subtasks:

- The operations performed at each iteration, are independent and may be executed simultaneously,
- To provide uniform load balancing each processor should contain:
  - The subset of vertices

$$V_j = \{v_{i_j+1}, v_{i_j+2}, \ldots, v_{i_j+k}\}, \quad i_j = k \cdot (j-1), k = \lceil n/p \rceil,$$

  - The block of the vector *d*

$$\Delta_j = \{d_{i_j+1}, d_{i_j+2}, \ldots, d_{i_j+k}\},$$

  - The vertical stripe of the adjacency matrix of graph *G*

$$A_j = \{\alpha_{i_j+1}, \alpha_{i_j+2}, \ldots, \alpha_{i_j+k}\}$$

(α_s is the *s-th* column of the matrix *A*),

  - The common part of set V_j and the set of vertices V_T

The Problem of Finding the Minimum Spanning Tree…

- **Analysis of Information Dependencies:**
  - The parallel Prim's algorithm consist in the following:
    - Finding the vertex t of the graph G, which has the minimum weight arc to the set V_T; to get this vertex it is necessary to carry out the search of minimum values in the blocks Δ_i, $0 \le i < p$, at each processor and accumulate the obtained values at the root processor,
    - Transmitting the number of the selected vertex to all the processors for including it into the spanning tree,
    - Renovating the set of values d_i with regard to the new vertex t, which has been added,
  - Thus, two types of information dependencies exist between processors in the course of parallel computations. The root processor accumulates the data from all the processors and the root processor transmits messages to all the processors

The Problem of Finding the Minimum Spanning Tree…

- **Scaling and Distributing the Subtasks among the Processors:**
  - Distributing the subtasks among the processors must be done with regards to the type of the communication operations performed by the Prim's algorithm. For efficient implementation of the required communications among the basic subtasks the network topology has to be a hypercube or a complete graph

The Problem of Finding the Minimum Spanning Tree…

- **Efficiency Analysis…**
  - **Speed-up and Efficiency generalized estimates**

$$S_p = \frac{n^3}{(n^3/p)} = p \qquad E_p = \frac{n^3}{p \cdot (n^3/p)} = 1$$

Developed method of parallel computations allows to achieve ideal speed-up and efficiency characteristics

The Problem of Finding the Minimum Spanning Tree…

- **Efficiency Analysis** (detailed estimates):

  - Time of the calculation part of the parallel algorithm:

$$T_p(calc) = n\lceil n/p \rceil \cdot \tau,$$

  - The total duration of executing communication operations:

$$T_p^1(comm) = n(\alpha \log_2 p + 3w(p-1)/\beta)$$

$$T_p^2(comm) = n \log_2 p(\alpha + w/\beta)$$

The total execution time for Prim's parallel algorithm:

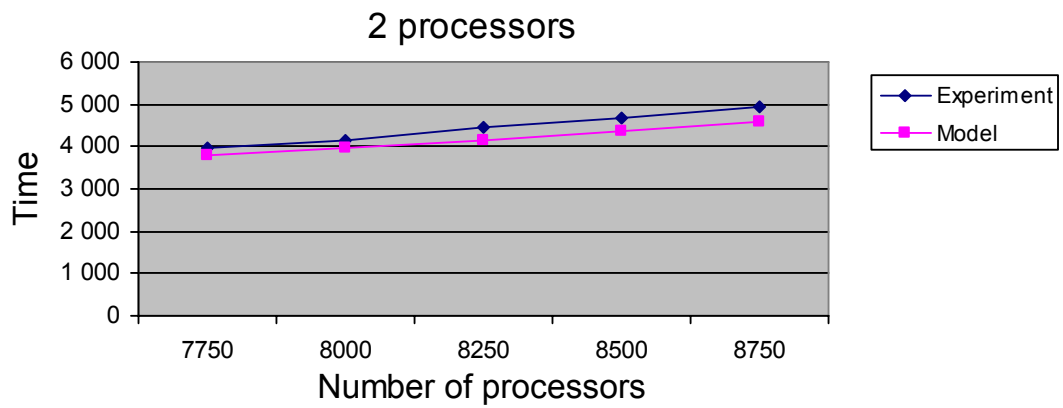
$$T_p = n\lceil n/p \rceil \cdot \tau + n(\alpha \cdot \log_2 p + 3w(p-1)/\beta + \log_2 p(\alpha + w/\beta))$$

The Problem of Finding the Minimum Spanning Tree…

❑ Results of Computational Experiments…

– Comparison of theoretical estimations and results of computational experiments

The number of vertices	Sequential algorithm		Parallel algorithm					
			2 processors		4 processors		6 processors	
	Model	Experiment	Model	Experiment	Model	Experiment	Model	Experiment
7750	4,234	4,137	3,778	3,971	4,383	3,462	5,005	3,837
8000	4,512	4,498	3,971	4,148	4,560	3,621	5,190	3,976
8250	4,798	4,800	4,168	4,442	4,739	3,763	5,376	4,120
8500	5,094	5,206	4,369	4,680	4,920	3,992	5,564	4,241
8750	5,398	5,689	4,574	4,950	5,103	4,623	5,754	4,424

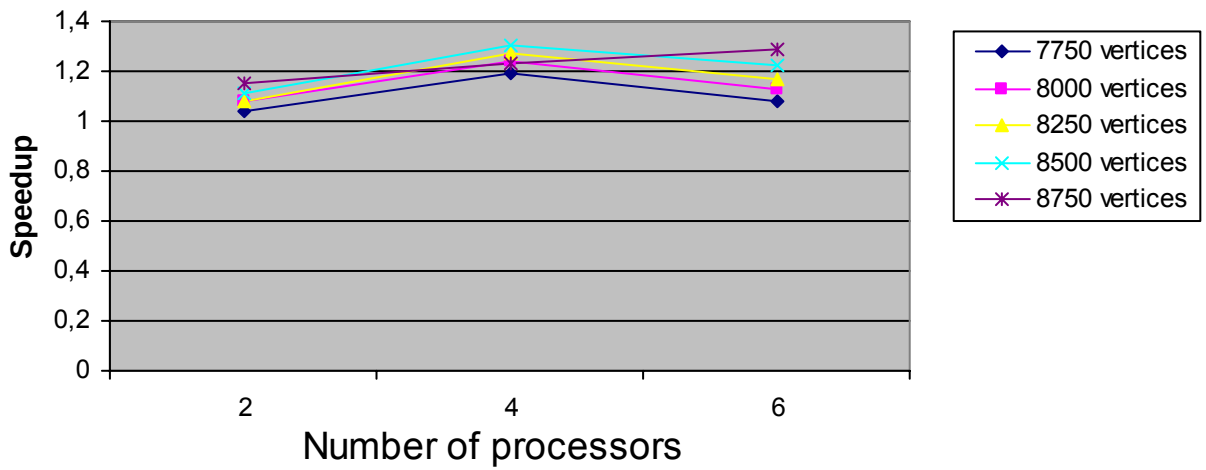


The Problem of Finding the Minimum Spanning Tree

❑ Results of Computational Experiments:

– Speed up

The number of vertices	Sequential algorithm	Parallel algorithm					
		2 processors		4 processors		6 processors	
	Time	Time	Speed up	Time	Speed up	Time	Speed up
7750	4,137	3,971	1,042	3,462	1,195	3,837	1,078
8000	4,498	4,148	1,084	3,621	1,242	3,976	1,131
8250	4,800	4,442	1,081	3,763	1,276	4,120	1,165
8500	5,206	4,680	1,112	3,992	1,304	4,241	1,228
8750	5,689	4,950	1,149	4,623	1,231	4,424	1,286

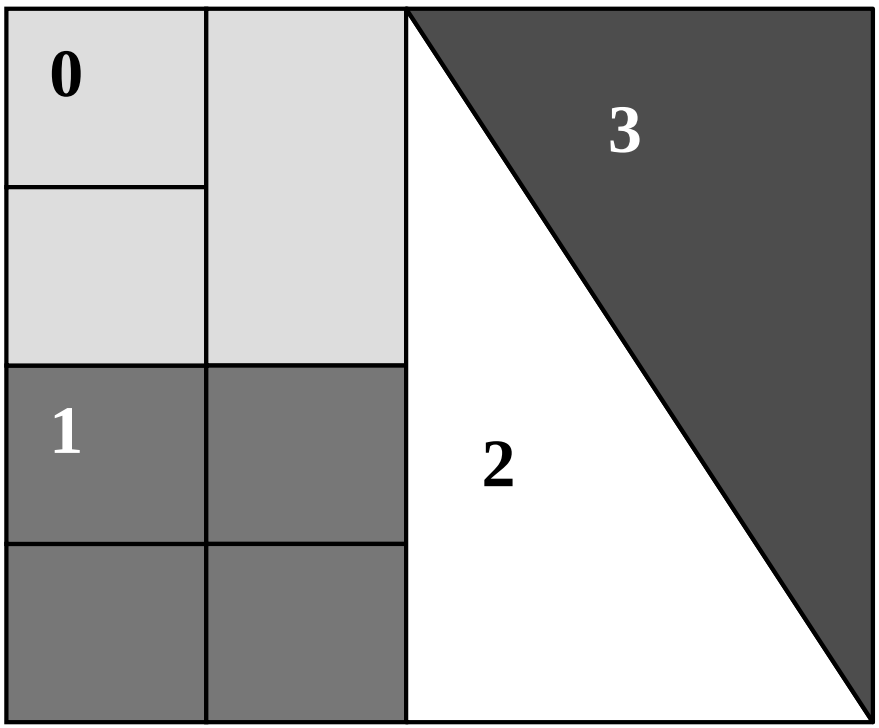


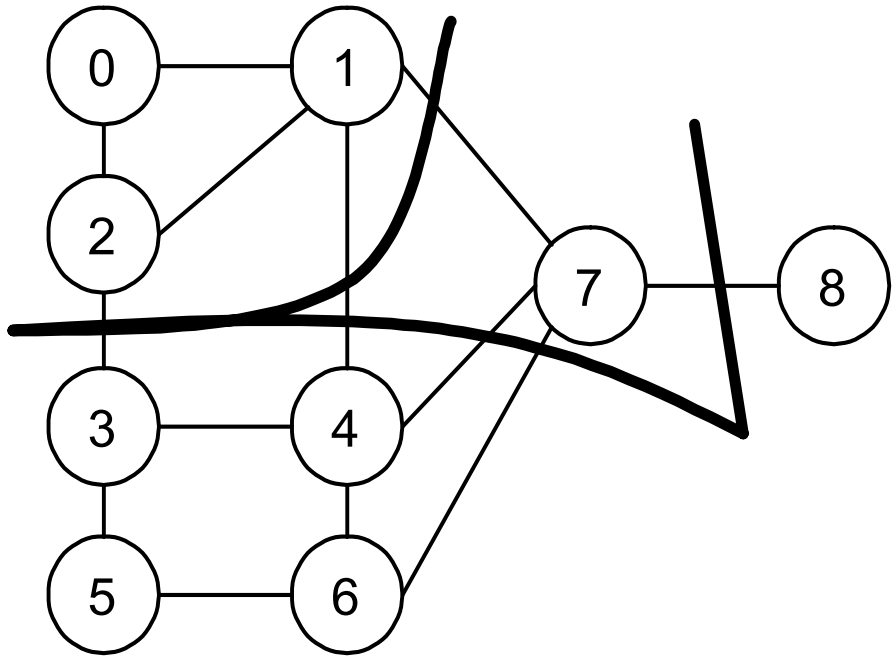
The Problem of the Optimum Graph Partition…

- **Introduction:**
  - This problem is one of those, which very often arise in the research, which involves parallel computations. As an example it can be considered the problems with the area of calculations presented as two- or three-dimensional grids,
  - It is obvious that such problem of distributing the grid among processors can be reduced to the problem of optimum graph partition,
  - If a grid should be represented as a graph, each grid element may be associated with a graph vertex. The graph arcs should be used for reflecting the grid element closeness (for instance, the arcs between the vertices of the graph can be set only if the corresponding elements of the initial grid are neighbors)

The Problem of the Optimum Graph Partition…

- **The Example of Irregular Grid Partitioning and the Corresponding Graph:**





The Problem of the Optimum Graph Partition…

Problem Statement:

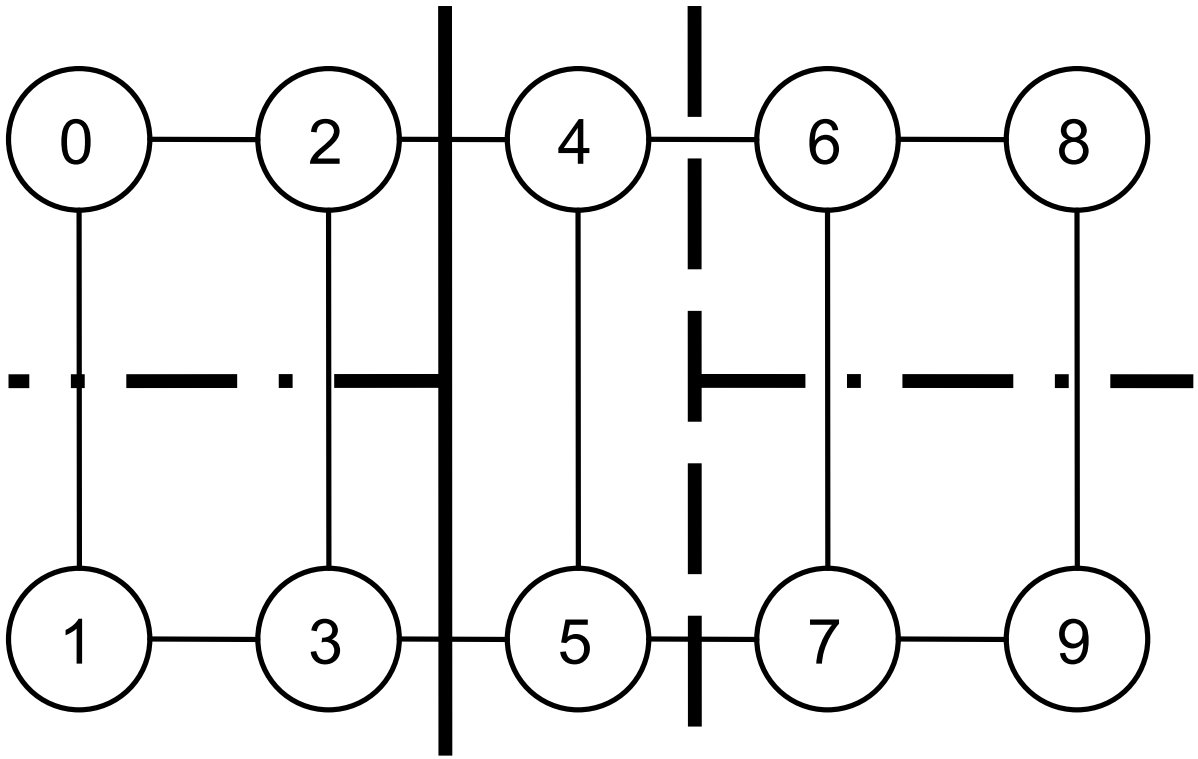
- Let there be given the weighted undirected graph *G=(V,E)*, each vertex *v* of which and each arc *e* of which are assigned a weight,
- *The problem of optimum graph partition* consists in partitioning its vertices into nonintersecting subsets with maximum close total vertex weights and the minimum total weight of the arcs, connecting the obtained vertex subsets,
- For the sake of simplicity the vertex weights and the arcs weights will further be assumed to be equal to 1

The Problem of the Optimum Graph Partition…

- **The Method of Bisection Recursive Division:**
  - The bisection division method may be recursively used for solving the problem of graph partition:
    - At the first iteration the graph is divided into two equal parts,
    - At the second step each of the parts is also divided into halves and so on
  - This method requires $log_2 k$ recursion levels for dividing the graph into *k* parts; also it requires *k-1* operations of bisection division. In case when the required number of bisection operations *k* is not a power of 2, each bisection division should be done in the corresponding relation

The Problem of the Optimum Graph Partition…

- **The Example of Graph Partitioning into Five Parts by the Method of the Bisection Recursive Division**



The Problem of the Optimum Graph Partition…

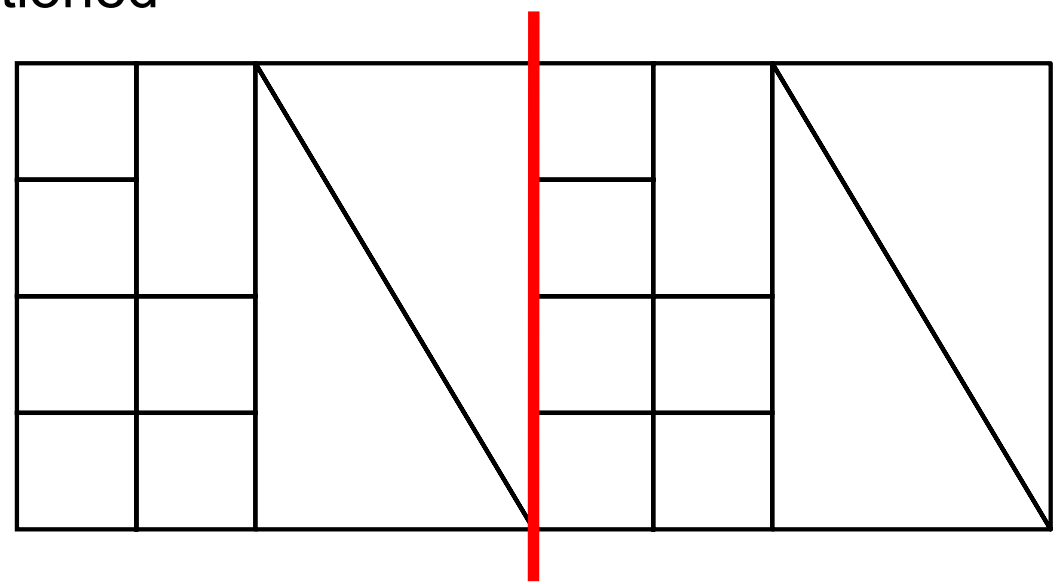
- **The Geometrical Methods…**
  - The geometric methods to dissect grids use only the coordinate data of the grid nodes,
  - As these methods do not take into account the information concerning the grid elements connectivity, they cannot explicitly provide minimizing the total weight of the interconnecting arcs of subgraphs (in terms of the graph, which corresponds to the grid). To minimize the interprocessor communications the geometric methods may optimize some additionally formulated criteria (for instance, the length of the border between the partitioned parts of the grid),
  - The geometric methods are usually fast and do not require any time-consuming computations

The Problem of the Optimum Graph Partition…

- **The Geometrical Methods:**
  - Hereinafter the following geometrical methods are discussed:
    - Coordinate Nested Partition,
    - The Recursive Inertial Bisection Method,
    - Grid Partition by Means of Space-Filling Curve Technique

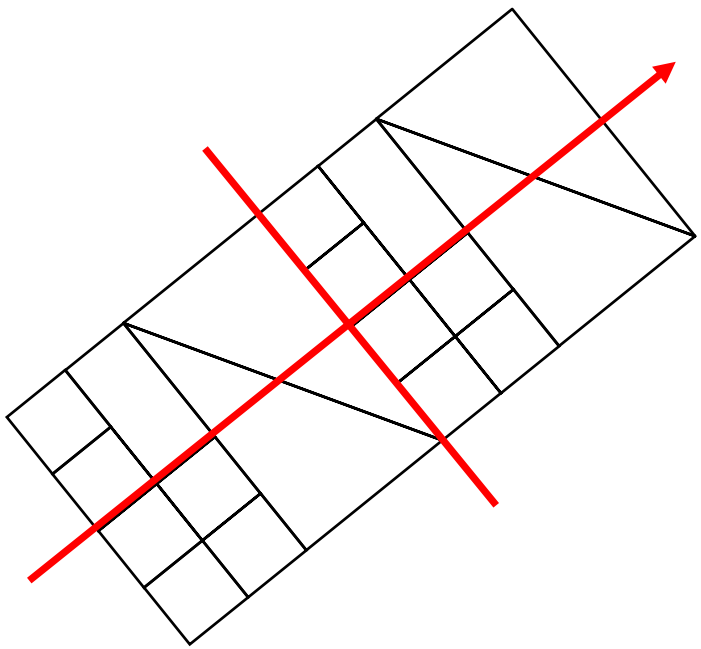
The Problem of the Optimum Graph Partition…

- **Coordinate Nested Partition:**
  - The general scheme of the method execution:
    - First the centers of mass of the grid elements are computed,
    - The points obtained are projected on the axis, which corresponds to the longest side of the grid, which is being partitioned



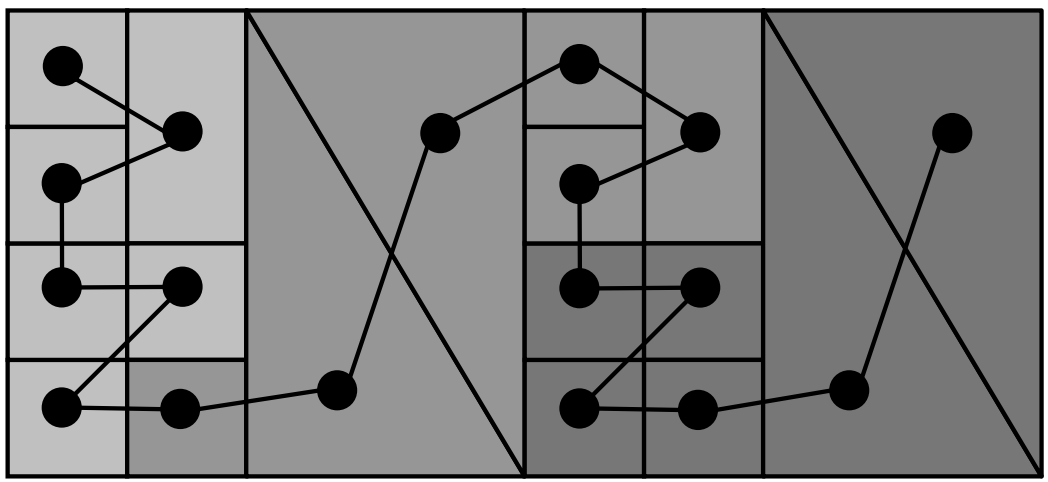
The Problem of the Optimum Graph Partition…

- **The Recursive Inertial Bisection Method:**
  - The Recursive Inertial Bisection Method constructs the main inertial axis assumed that all grid elements are mass points,
  - The bisection line, which is orthogonal to the obtained axis, produces, as a rule, the shortest border



The Problem of the Optimum Graph Partition…

- **Network Partition by Means of Space-Filling Curve Technique:**
  - The Peano curves are such curves, that fully fill multidimensional domains (for instance, a square or a cube)
  - After the sequence of all the grid elements put in order according to their position on the Peano curve, it is sufficient to divide the sequence into the necessary number of parts according to the established order



The Problem of the Optimum Graph Partition…

- **The Combinatorial Methods…**
  - Unlike the geometrical methods the combinatorial algorithms operate, as a rule, not with the grid but with the graph, constructed for the grid,
  - These methods make no regard to the information of the closeness of the grid element locations - they are guided by the adjacency of graph vertices,
  - The combinatorial methods provide more balanced partitions and lower information interactions of the obtained subgrids than the previously described methods,
  - However, such methods tend to operate much longer than the geometrical ones

The Problem of the Optimum Graph Partition…

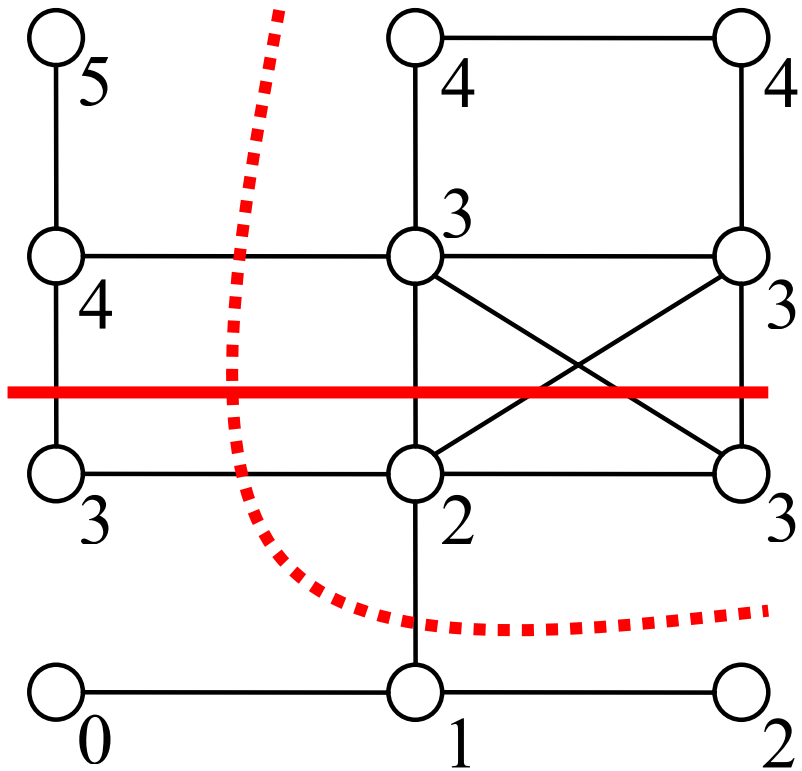
- **The Combinatorial Methods:**
  - Hereinafter the following combinatorial methods are discussed:
    - Graph Partition with Regards to Adjacency,
    - The Kernighan-Lin Algorithm

The Problem of the Optimum Graph Partition…

- **Graph Partition with Regards to Adjacency:**
  - The graph is bisected at each algorithm iteration. Thus, partitioning the graph into the required number of parts is achieved by means of the recursive use of the algorithms,
  - The general scheme of the algorithm may be described by the following set of instructions:
    1. Set Iteration = 0
    2. Assigning the Iteration number to the arbitrary graph vertex
    3. Assigning the number *Iteration* + *1* to the unnumbered neighbors of the vertices with the Iteration number
    4. Set Iteration = Iteration + 1
    5. If there are any unnumbered vertices left, then proceed to step 3
    6. Make the graph bisection according to the enumeration

The Problem of the Optimum Graph Partition…

- **The Example of the Graph Partition Algorithm with Regard to Adjacency**



The Problem of the Optimum Graph Partition…

- **The Kernighan-Lin Algorithm…**
  - *The Kernighan-Lin algorithm* uses another approach in order to solve the problem of optimum graph partition. It is assumed from the very beginning, that some initial graph partition already exists. Then the approximation is improved in the course of a number of iterations

The Problem of the Optimum Graph Partition…

- **The Kernighan-Lin Algorithm** (*the general scheme*)…
  - Forming a set of pairs of vertices for permutation

    The vertices, which have not yet been rearranged at the given iteration, are used to form all the possible pairs (the pairs should contain vertices from each part of the given graph partition)
  - Constructing new variants of graph partition

    Each pair, prepared at step 1, is sequentially used for exchanging vertices among the parts of the available graph partition. It is done for obtaining the set of new partition variants
  - Choosing the optimum variant of graph partition

    The optimum variant is chosen for the set of new parts obtained in the course of graph partition. This variant is fixed as the new current graph partition. The pair of vertices, which corresponds to the selected variant, is marked as being used at the current iteration

The Problem of the Optimum Graph Partition…

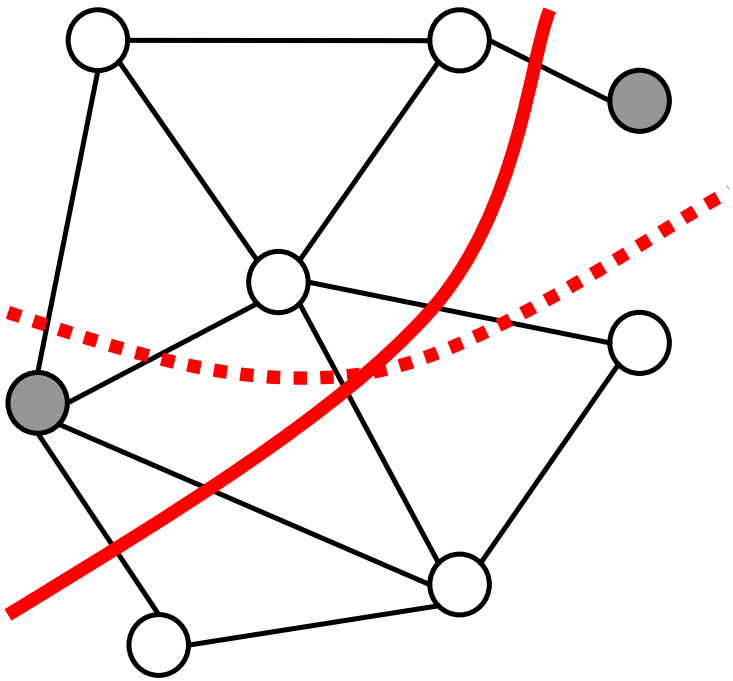
- **The Kernighan-Lin Algorithm** (*the general scheme*)
  - Finding the unused graph vertices

    If there are any graph vertices, which have not been used in permutation, the algorithm iteration is repeated beginning with step 1. If the graph vertex enumeration is completed, then the next step follows
  - Choosing the optimum graph partition

    The optimum variant of graph partition is chosen among all the graph partitions, obtained at the current iterations

The Problem of the Optimum Graph Partition…

- The Example of Permuting two Vertices (marked in grey) in the Kernighan-Lin Method



The Problem of the Optimum Graph Partition

❑ The Comparison of the Graph Partition Algorithms

		The Necessity of Coordinate Data	Accuracy	Execution Time	Suitability for Parallelization
Coordinate Nested Partition		Yes	●	●	●●●
Recursive Inertial Bisection Method		Yes	●●	●	●●●
Partition with Regards to Adjacency		No	●●	●●	●●
The Kernighan-Lin algorithm	1 iteration	No	●●	●●	●
	10 iterations	No	●●●◐	●●●	●●
	50 iterations	No	●●●●	●●●◐	●●

Summary

- Two well known algorithms of graph calculations are discussed:
  - The Floyd's algorithm for Solving The Problem of the Search for All the Shortest Paths,
  - The Prim's algorithm The Problem of Finding the Minimum Spanning Tree
- An overview of the methods of optimum graph partition is given:
  - Geometrical methods:
    - Coordinate Nested Partition,
    - The Recursive Inertial Bisection Method,
    - Graph Partition by Means of Space-Filling Curve Technique
  - Combinatorial Methods:
    - Graph Partition with Regards to Adjacency,
    - The Kernighan-Lin Algorithm

Discussions…

- Give the definition of the graph. What are the main methods of graph representation?
- What does the problem of searching all the shortest paths consist in?
- Give the general scheme of the Floyd's algorithm. What is the complexity of the algorithm?
- What does the approach of the Floyd's algorithm parallelization consist in?
- What is the problem of searching the minimum spanning tree? Give an example illustrating how the problem can be used in practice.

Discussions

- Give the general scheme of the Prim's algorithm. What is the complexity of the algorithm?
- What does the of the Prim's algorithm parallelization consist in?
- What is the difference between the geometrical and the combinatorial methods of graph partition? Which of them are preferable and why?
- Describe the method of coordinate nested partition and the Partition with Regards to Adjacency. Which of them is easier to implement?

Exercises

- Develop the parallel Floyd's algorithm. Carry out the computational experiments. Formulate the theoretical estimations with regard to the parameters of the computer system being used. Compare the theoretical estimations with the experimental data
- Develop the parallel Prim's algorithm. Cary out the computational experiments. Formulate the theoretical estimations with regard to the parameters of the computer system being used. Compare the theoretical estimations with the experimental data
- Develop the parallel program for the Kernighan-Lin algorithm. Estimate the suitability of the algorithm parallelization

References

- **Cormen, T.H., Leiserson, C. E. , Rivest, R. L. , Stein C.** (2001). Introduction to Algorithms, 2nd Edition. - The MIT Press.
- **Schloegel, K., Karypis, G., Kumar, V.** (2000). Graph Partitioning for High Performance Scientific Simulations.

Next Section

- Parallel Methods for Partial Differential Equations

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.