# METHODS OF FORMING PARALLEL SCHEMES OF ALGORITHMS FOR SMP-ARCHITECTURE

**Sergiy Pogorilyy, Oleg Vereshchynsky**

Taras Shevchenko National University of Kiev, Kiev, Ukraine
*sdp@rpd.univ.kiev.ua*

## Introduction

Multiprocessor and multicore systems belong to a class of systems capable of proceeding large information scope. One of these types of architecture is Symmetric Multiprocessing Systems (SMP). The main idea is the presence of a special memory area that is shared between different processors, which is used for information exchange so that all processors perform the same addressing for all memory cells.

One of the biggest advantages of the SMP approach is its simplicity and universality during programming. Usually, a model of few parallel branches is used (however, it is also possible to organize interprocessor data exchange). Creating the processes (threads) and scheduling their work can provide great efficiencies in execution time. All the subsystem works only under the control of the operating system. In particular, in MS Windows SMP is supported, starting from version NT 4.0.

A typical example of SMP systems is modern multicore Intel processors (Core Duo, Core 2 Duo, Core 2 Quad, etc.). Using these processors for universal computing systems allows higher efficiency.

## Methods of forming parallel schemes of algorithms

The description of methodology is given, using Dantzig algorithm as an example. It is widely used for solving the routes problem in computer networks. The idea is to calculate the submatrices of the shortest paths $D_m$, in consecutive order with increasing dimension $m \times m$, by using recurrent procedure. Each intermediate matrix is, in fact, the shortest path's matrix for subgraph with nodes between $0$ and $m-1$.

The following notation is used:

$N$ – dimension of the graph, that reflects network topology; (1)

$\{i..k\}$ – range of integers from *i* to *k*;

$g_{ij}$ – length of an edge between *i* and *j* nodes;

$d_{ij}$ – length of the shortest path between *i* and *j* nodes;

$d_{ij}^{m}$ – path from the node *i* to *j* through *m*;

$isEdge(i, j)$ – predicate, obtains true if an edge between *i* and *j* nodes exists;

$isWay(i, j)$ – predicate, obtains true if a path between *i* and *j* nodes exists;

$edgeNum(i, j)$ – number of edges between *i* and *j* nodes;

$setWay(i, j, m)$ – setting of the path $d_{ij}^{m}$.

Each iteration of the algorithm can be divided in three consecutive steps:

1. Search of the shortest paths from node *m* to *j*<*m*.

Each path consists of two parts: from *m* to some intermediate node *i* ($g_{mi}$) and from *i* to *j* ($d_{ij}$).

Then:

$$d_{mj} = \min_{i \in \{0, m-1\}} \left(g_{mi} + d_{ij}\right). \tag{2}$$

2. Search of the shortest paths from node *i*<*m* to *m*.

Each path consists of two parts: from *i* to some intermediate node *j* ($d_{ij}$) and from *j* to *m* ($g_{jm}$).

Then:

$$d_{im} = \min_{j\in\{0,m-1\}}\left(d_{ij} + g_{jm}\right) \quad . \tag{3}$$

3. Search of the shortest paths from nodes *i<m* to *j<m*, taking into consideration existence of the new node *m*.

Each path consists of two parts: from *i* to the node *m* ( $d_{im}$ ) and from *m* to *j* ( $d_{mj}$ ).

Then:

$$d_{im} = \min\left(d_{im} + d_{mj}, d_{ij}\right). \tag{4}$$

The last step of the recurrent procedure gives us matrix $D_N$ , which, in fact, is the shortest paths matrix of the graph.

**Sequential algorithm scheme.** Formalizing of Dantzig algorithm is performed by using the mathematical apparatus of modified systems of Glushkov's algorithmic algebras (SAA). Using notation (1), we can form a few conjunctive conditions to shorten the formula of algorithm:

$$\alpha_1(m,i,j) = isEdge(m,i) \wedge isWay(i,j) \tag{5.a}$$

$$\alpha_2(m,i,j) = isEdge(j,m) \wedge isWay(i,j) \tag{5.b}$$

$$\alpha_3(m,i,j) = isWay(i,m) \wedge isWay(m,j) \tag{5.c}$$

Conditions $\alpha_1, \alpha_2,\ \alpha_3$ indicate the existing of paths $d^{i}_{mj}, d^{j}_{im},\ d^{m}_{ij}$ accordingly.

Using (5) it is possible to write down conditions of paths $d^{i}_{mj},\ d^{j}_{im},\ d^{m}_{ij}$ optimality for current step of iterative process:

$$\begin{aligned}\beta_1 = &\left(\overline{isWay(m,j)}\right) \vee \left(g_{mi} + d_{ij} < d_{mj}\right) \vee \\ &\vee \left((g_{mi} + d_{ij} = d_{mj}) \wedge (edgeNum(i,j) + 1 < edgeNum(m,j))\right)\end{aligned} \tag{6.a}$$

$$\begin{aligned}\beta_2 = &\left(\overline{isWay(i,m)}\right) \vee \left(g_{jm} + d_{ij} < d_{mi}\right) \vee \\ &\vee \left((g_{jm} + d_{ij} = d_{mi}) \wedge (edgeNum(i,j) + 1 < edgeNum(i,m))\right)\end{aligned} \tag{6.b}$$

$$\begin{aligned}\beta_3 = &\left(\overline{isWay(i,j)}\right) \vee \left(d_{im} + d_{mj} < d_{ij}\right) \vee \\ &\vee \left((d_{im} + d_{mj} = d_{ij}) \wedge (edgeNum(i,m) + edgeNum(m,j) < edgeNum(i,j))\right)\end{aligned} \tag{6.c}$$

Path is considered optimal, if it has minimal length, or, if there are paths with equal lengths, if it contains fewer edges in it.

Operating conditions (5) and (6) give us formula of each step of the algorithm:

1. $SubDanc1(m) = \underset{j\notin\{1..m\}}{\{}\ \underset{i\notin\{1..m\}}{\{}\ \underset{\alpha_1}{(}\ \underset{\beta_1}{(}\ setWay(m,i,j) \vee E) \vee E) \times (++i)\} \times (++j)\}$ (7.a)
2. $SubDanc2(m) = \underset{i\notin\{1..m\}}{\{}\ \underset{j\notin\{1..m\}}{\{}\ \underset{\alpha_2}{(}\ \underset{\beta_2}{(}\ setWay(i,m,j) \vee E) \vee E) \times (++j)\} \times (++i)\}$ (7.b)
3. $SubDanc3(m) = \underset{i\notin\{1..m\}}{\{}\ \underset{j\notin\{1..m\}}{\{}\ \underset{\alpha_3}{(}\ \underset{\beta_3}{(}\ setWay(i,j,m) \vee E) \vee E) \times (++j)\} \times (++i)\}$ (7.c)

where operation $(++)$ means value increment by one.

Using (7), sequential regular scheme of an algorithm can be formed:

$$Dancig = \underset{m>N}{\{}\ SubDanc1(m) \times SubDanc2(m) \times SubDanc3(m) \times (++m)\}. \tag{8}$$

**Data processing principle.** The paradigm of data parallelism is used, so every thread processes some part of informational set without intersection with other threads during this stage. This way, for further work it is necessary to formulate main principles of processing the data, placed in memory.

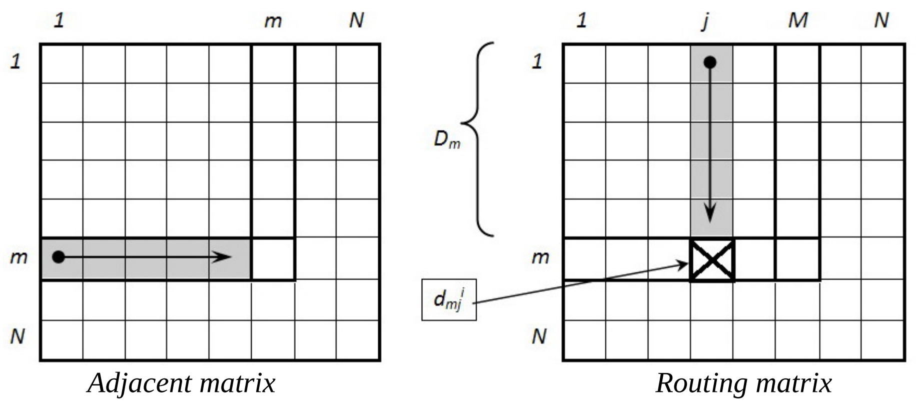
Information about graph is saved in two matrices. The first one is the adjacent matrix of the graph. It is actually input information for the algorithm and remains unchangeable during its work (read-only access). The second matrix contains routes. The cell with coordinates *(i, j)* contains number of the node, which is intermediate on the way between nodes *i* and *j*. Considering that, each path is divided into two parts: from *i* to some intermediate node *k* and from *k* to *j*. If *k=j*, the path is simply equal to the graph edge. Using this matrix allow dynamical calculating of the shortest path length between any two nodes. After the last iteration of an algorithm this matrix represents the result of its work.

As it was written, Dantzig algorithm can be divided into three independent steps:

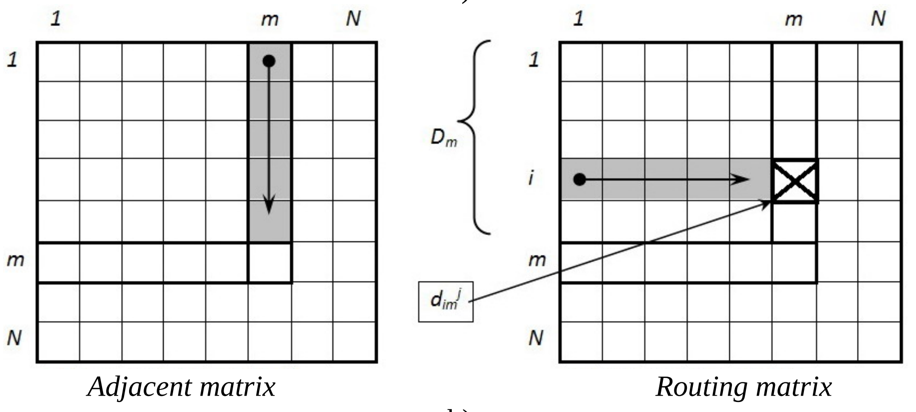
1. Search of the shortest path $d_{mj}^{i}$
2. Search of the shortest path $d_{im}^{j}$
3. Search of the shortest path $d_{ij}^{m}$

It is important to notice that step 3 can be performed only when the two previous steps totally finished their work.

The first two stages are schematically represented in Fig.1 a) and b). As we can see, calculating of any new element $d_{mj}^{i}$ ($d_{im}^{j}$) requires processing the entire column (row) of submatrix $D_m$ (arrows mark data, that have to be ***read*** for calculating a corresponding element). That's why input data should be divided in such a way to be optimal, because the scheduling of the threads can afford avoiding of excessive data exchange and decrease synchronization expenditures in the aforementioned cases.



a)



b)

Fig.1

A further scheme of work is proposed.

The proposed algorithm works with 2N threads. During the first stage they divide matrix into equal rows, during the second one – into columns. There is no intersection of data areas, which are processed by different threads, even during reading that also can slow down the algorithm.

**Forming the parallel scheme of the Dantzig algorithm**

**Dividing the input matrix.** Some new notations are used to implement parallelizing. It is necessary to divide $m \times m$ matrix into *N* approximately equal rectangular parts.

In the following notation $b_i$ means the first row of *i*–th part, and $e_i$ - the last one. They can be calculated by using the following formulas:

$$b_i^{m,N} = (i-1) \times ]m/N[ + 1 \qquad (9)$$

$$e_i^{m,N} = i \times ]m/N[ \qquad (10)$$

It lets us calculate the limits of the memory area that is processed by only one thread (in a similar way a matrix can be divided into columns).

Thread synchronization. Let's describe generalized thread synchronization object. In fact, it is a barrier that can delay the execution of the threads, until there are enough of them. Such an object can be represented with the formula:

$$B_N(i) = T(\lambda_i) \times S(\lambda_1) \times \ldots \times S(\lambda_{i-1}) \times S(\lambda_{i+1}) \times \ldots \times S(\lambda_N) = T(\lambda_i) \times \left( \prod_{\substack{j=1 \\ j \neq i}}^{N} S(\lambda_j) \right) \qquad (11)$$

where $T(\lambda)$ – check point;

$S(\lambda)$ – synchronization point (it can be interpreted as an empty loop);

$N$ – total number of the threads;

$i$ – number of the thread, that is synchronized.

Each thread has to get through (*N-1*) synchronization point (it must unblock its own point before this) in order to pass the barrier.

**Equivalent transformation of the algorithm scheme.** The apparatus of identical transforms of SAA can be applied to (8) [1].

$$Dancig = \overset{2N}{\underset{l=1}{\dot{\bigvee}}} \underset{m>N}{\{} \left( \left( \underset{\omega}{(} PSub1(m,N,l) \vee PSub2(m,N,l)) \times B_{2N}(l) \right) \right) \times \\ \times \left( (PSub3(m,2N,l) \times B_{2N}(l)) \right) \times (++m) \} \qquad (12)$$

The main advantage of (12) is a lower, comparing with other schemes, number of synchronizations, which can mean an increase in efficiency.

**Conclusion**

The approach to solving the network routing problem, based on the Dantzig algorithm, has been analyzed. Formalizing the algorithm, using mathematical apparatus SAA-M, has been performed and the sequential regular scheme (8) obtained.

The concept of algorithm parallelizing has been formed for the systems with shared memory. Two approaches were analyzed: in the first case the load of each thread decreased, and in the second, due to the load increasing, synchronization expenditures decreased. Transformation of the Parallel Regular Scheme (PRS) has been performed using SAA-M.

Using the mathematical apparatus of Gluskov's SAA-M allows us to concentrate on the concepts and methods of algorithm schemes transformation. This, in turn, allows abstraction from the concrete details of programming implementation.

## References

[1] S.D. Pogorilyy, Yu.V. Boyko, A.D. Gusarov, S.I. Lozytskyi. An approach to the parallel solution of a high-dimensional basic flow problem. Cybernetics and Systems Analysis, Volume 45, Issue 2 (March 2009) pages: 291 – 296. Springer Science and Business Media Inc. ISSN:1060-0396.