Pavol Purcz

Parallel algorithm for spatially one-and two-dimensional initial-boundary-value problem for a parabolic equation

Kybernetika, Vol. 37 (2001), No. 2, [171]--181

Persistent URL: http://dml.cz/dmlcz/135399

KYBERNETIKA — VOLUME 37 (2001), NUMBER 2, PAGES 171 - 181

PARALLEL ALGORITHM FOR SPATIALLY ONE– AND TWO–DIMENSIONAL INITIAL–BOUNDARY–VALUE PROBLEM FOR A PARABOLIC EQUATION

PAVOL PURCZ

A generalization of the spatially one-dimensional parallel pipe-line algorithm for solution of the initial-boundary-value problem using explicit difference method to the two-dimensional case is presented. The suggested algorithm has been verified by implementation on a workstation-cluster running under PVM (Parallel Virtual Machine). Theoretical estimates of the speed-up are presented.

1. INTRODUCTION

The importance of the boundary-value problems (BVP) for engineering applications attracts and motivates continuous development of fast numerical algorithms for its solution. A number of various approaches have been suggested, such as finite elements methods, fast Fourier transform methods, multigrids methods and difference methods (e. g. [1, 2, 3, 5, 6, 7, 8, 10]), etc.

Some theoretical aspects of the use of an explicit difference method in the case of spatially one-dimensional initial-boundary-value problem (IBVP) were considered by P. M. Kogge [4]. Based on Kogge's ideas, E. E. Tyrtyshnikov [9] suggested a possible approach to development of a numerical algorithm for solving the spatially one-dimensional IBVP. In the present work such an algorithm is developed for spatially one- and two-dimensional IBVP, and verified on a workstation-cluster running under PVM (Parallel Virtual Machine). Moreover, some theoretical estimates of the speed-up of the algorithms are presented.

2. ONE–DIMENSIONAL INITIAL–BOUNDARY–VALUE PROBLEM

First, we describe the suggested algorithm for a spatially one-dimensional IBVP. Let us consider a spatially one-dimensional IBVP for a parabolic equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \quad t > 0, \quad 0 < x < l_x \tag{1}$$

$$u(0, x) = \varphi(x), \quad 0 < x < l_x \tag{2}$$

$$u(t,0) = \alpha(t), \quad t > 0 \tag{3}$$
$$u(t,l_x) = \beta(t), \quad t > 0 \tag{4}$$

where φ, α, β are given sufficiently smooth functions such that $\varphi(0) = \alpha(0)$, $\varphi(l_x) = \beta(0)$. We look for a sufficiently smooth function $u(t,x)$, which satisfies the equation (1) and conditions (2)–(4). For the numerical solution of the problem (1)–(4), let us consider the following explicit difference method of the form

$$\frac{u_i^k - u_i^{k-1}}{\tau} = \frac{u_{i-1}^{k-1} - 2u_i^{k-1} + u_{i+1}^{k-1}}{h_x^2}, \tag{5}$$

where $\tau > 0$ and $h_x > 0$ are the time step and the spatial step of discretization, satisfying the condition of convergence $\tau/h_x^2 \leq 0.5$ and u_i^k – is an approximate value of the solution at the point (k,i), $(i = 1,2,\ldots,(n_x - 1)$, $k = 1,2,\ldots$; $n_x \cdot h_x = l_x)$. Moreover, $u_0^k = \alpha(t_k)$, $u_{n_x}^k = \beta(t_k)$, and $u_i^0 = \varphi(x_i)$, where $x_i = ih_x$, $t_k = k\tau$.

Let us consider a natural number n_x such that $n_x = q.n$, where $q \geq 2$, $n \geq 2$. Then the explicit difference method (5) can be realized on a parallel computer with n processors $P_1, P_2, \ldots, P_n$. A geometrical interpretation of the pipe-line implementation for $n = 5$ is depicted in Figure 1.

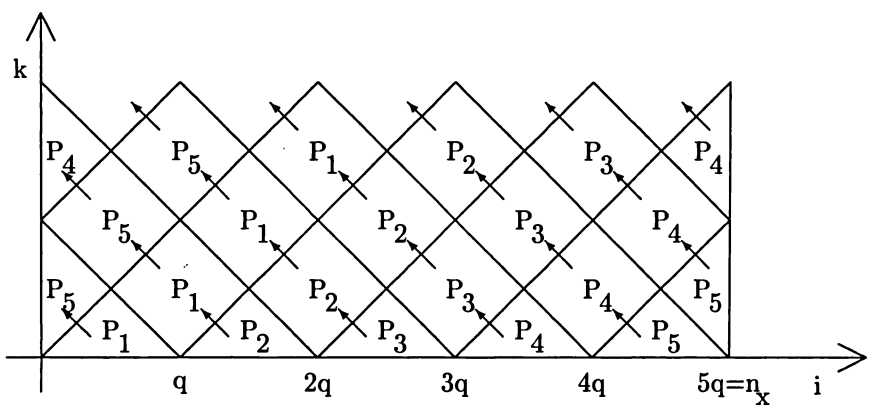

Fig. 1. Geometrical interpretation in one-dimensional case.

Each processor starts computations with the initial conditions, and computes the values of solution in a triangle block. Then the data exchange between processors is performed in the "north-west" direction. In the case of a pipe-line implementation with processors connected in series, each processor sends data to its "left" neighbour. After finishing the data exchange, a new step of computations is performed; each processor computes the values of solution in a corresponding turned square block. The "boundary" processors also use the boundary conditions. Then the data exchange and computation of the values of solution in turned square blocks repeats.

Generally for n processors we can write the whole data exchange process in the form:

$$P_{(i+1) \bmod(n)} \to P_i \to P_{n[(n-i+1) \operatorname{div}(n)]+(i-1) \bmod(n)}, \qquad (i = 1, 2, \ldots, n). \tag{6}$$

3. GENERALIZATION OF A PARALLEL ALGORITHM IN A GEOMETRICAL WAY

Now let us describe a generalization of the spatially one-dimensional algorithm for the two-dimensional case. There is a certain difficulty consisting in the following. In the one-dimensional case, the triangle and square blocks give us a complete covering of the considered domain (see Figure 1). However, in the two-dimensional case we have pyramids instead of triangles, and dipyramids instead of squares; these two types of blocks do not cover the considered domain completely, and there are also other types of blocks. The complete set of blocks which give the complete covering in the case of the spatially two-dimensional problem is shown in Figure 2.

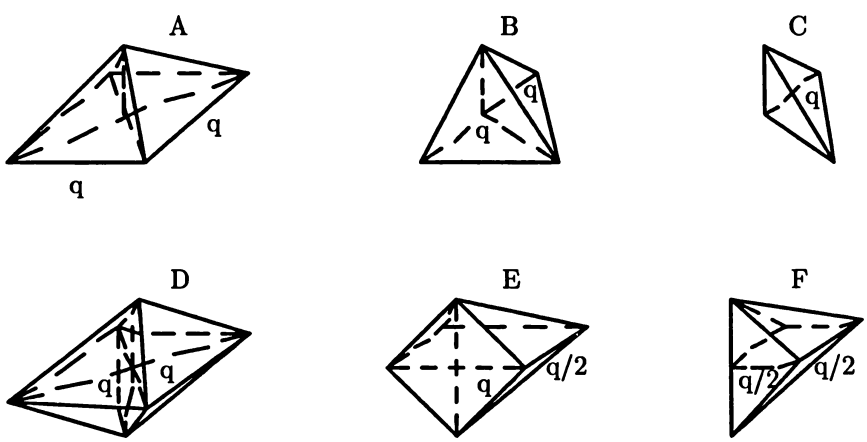

Fig. 2. A complete set of blocks in the two-dimensional case.

Let us consider the spatially two-dimensional IBVP for the parabolic equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}, \quad t > 0, \quad 0 < x < l_x, \quad 0 < y < l_y \tag{7}$$

$$u(0, x, y) = \varphi(x, y), \quad 0 < x < l_x, \quad 0 < y < l_y \tag{8}$$

$$u(t, 0, y) = \alpha(t, y), \quad t > 0, \quad 0 < y < l_y \tag{9}$$

$$u(t, l_x, y) = \beta(t, y), \quad t > 0, \quad 0 < y < l_y \tag{10}$$

$$u(t, x, 0) = \gamma(t, x), \quad t > 0, \quad 0 < x < l_x \tag{11}$$

$$u(t, x, l_y) = \delta(t, x), \quad t > 0, \quad 0 < x < l_x \tag{12}$$

where $\varphi, \alpha, \beta, \gamma, \delta$ are given sufficiently smooth functions such that $\varphi(0,0) = \alpha(0,0) = \gamma(0,0)$, $\varphi(l_x,0) = \beta(0,0) = \gamma(0,l_x)$, $\varphi(l_x,l_y) = \beta(0,l_y) = \delta(0,l_x)$, $\varphi(0,l_y) = \alpha(0,l_y) = \delta(0,0)$. We look for the solution $u(t,x,y)$, that satisfies (8) – (12). For the numerical solution we use the difference scheme of the following form:

$$\frac{u_{ij}^k - u_{ij}^{k-1}}{\tau} = \frac{u_{i-1j}^{k-1} - 2u_{ij}^{k-1} + u_{i+1j}^{k-1}}{h_x^2} + \frac{u_{ij-1}^{k-1} - 2u_{ij}^{k-1} + u_{ij+1}^{k-1}}{h_y^2}, \tag{13}$$

where $\tau > 0$ is the time step and $h_x > 0$, $h_y > 0$ are spatial steps of a discretization, satisfying the condition of convergence $\tau / \max(h_x^2, h_y^2) \leq 0.5$ and u_{ij}^k is the approximate solution in the point (k,i,j), $i = 1,2,\ldots,(m_x - 1)$, $j = 1,2,\ldots,(n_y - 1)$, $k = 1,2,\ldots$; $m_x.h_x = l_x$, $n_y.h_y = l_y$. Moreover, $u_{0j}^k = \alpha(t_k, y_j)$, $u_{m_x j}^k = \beta(t_k, y_j)$, $u_{i0}^k = \gamma(t_k, x_i)$, $u_{in_y}^k = \delta(t_k, x_i)$ and $u_{ij}^0 = \varphi(x_i, y_j)$, where $x_i = i.h_x$, $y_j = j.h_y$, $t_k = k\tau$. The algorithm is based on the same idea as in Section 2 and presented using a geometrical interpretation for parallel processing of the considered problem.

Let us take natural numbers m,n such that $m_x = qm$, $n_y = qn$ where $q \geq 2$, $n_y \geq 2$, $n_x \geq 2$. For instance, let $m = 4$ and $n = 2$. Then the explicit difference scheme (13) can be realized on a parallel computer with $m \times n$ processors ordered as shown (see Figure 3).

P_{21}	P_{22}	P_{23}	P_{24}
P_{11}	P_{12}	P_{13}	P_{14}

Fig. 3. An example: 4 × 2 processors.

Let us introduce the following types of three dimensional blocks for the two-dimensional problem:

A — tetragonal pyramide,
B — rhombic sfenoide,
C — half a rombic sfenoide,
D — tetragonal dipyramide,
E — half a tetragonal dipyramide,
F — quarter of a tetragonal dipyramide.

At the beginning, all processors compute values lying in the blocks of the type A. At this step, computations in each processor are based on the formula (13) and initial conditions (8) are also used. After finishing these computations, the data exchange is performed. The values at the points lying on the west and south walls of the considered type A blocks, as well as the values at the nearest points lying inside the type A blocks, are transferred according to the following rule, depicted in Figure 4:

$$P_{(i+1) \bmod(m),j}, P_{i,(j+1) \bmod(n)} \to P_{i,j}, \qquad (i = 1, 2, \ldots, m; \quad j = 1, 2 \ldots n). \tag{14}$$

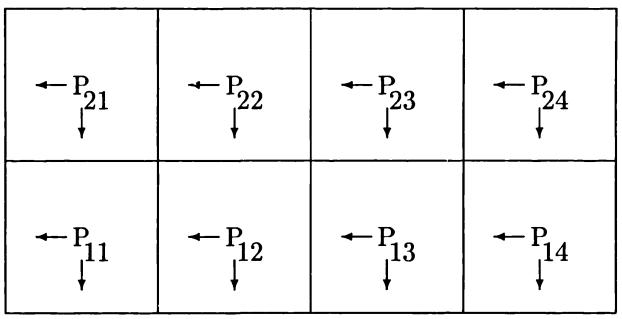

Fig. 4. Data exchange between processors: a layer of blocks A.

During further steps, the domains processed by each processor move gradually in the "north-east" direction, as shown in the diagrams below.

At the next step, we have a layer formed by blocks of types B and C. There is no analogy for such a layer in the one-dimensional case. This layer is shown in Figure 5, as well as the directions of the data exchanges. For example, each of the processors P_{11}, P_{12} and P_{13} processes two blocks of the type B; each of the processors P_{14}, P_{21}, P_{22} and P_{23} processes one block of the type B and two blocks of the type C (meanwhile, two blocks of the type C create one block of the type B); processor P_{24} processes four blocks of the type C. Afterwards, the data (input and output) exchange is performed according to the following rules:

Processor P_{11} receives the data from processors P_{12} and P_{21} before starting its computations, and sends the computed values to processors P_{14}, P_{21} after finishing its computations; other processors work similarly. Since we consider the pipe-line implementation, then in the considered problem the following permanent data exchange rules hold for all further layers:

$$P_{i,j} \to P_{m[(m-i+1) \div (m)]+(i-1) \bmod(m),j}, P_{i,n[(n-j+1) \div (n)]+(j-1) \bmod(n)} \tag{15}$$

$$P_{(i+1) \bmod(m),j}, P_{i,(j+1) \bmod(n),j} \to P_{i,j} \quad (i = 1, 2, \ldots, m; \quad j = 1, 2 \ldots n). \tag{16}$$

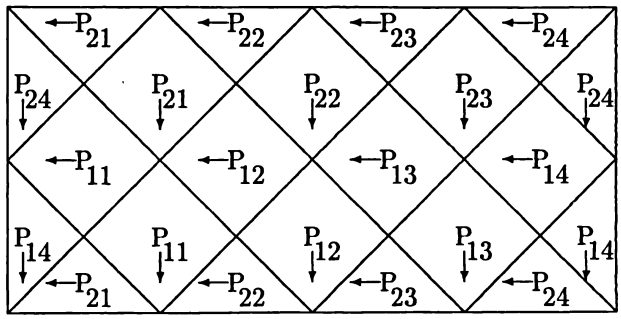

Fig. 5. Data exchange between processors: a layer of blocks B and C.

Objects D, E, and F create the layer of blocks which is analogous to the layer consisting of squares and triangles in the one-dimensional case. For this layer, processors work as shown in Figure 6.

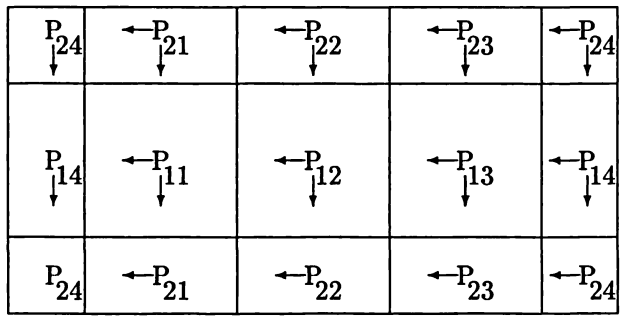

Fig. 6. Data exchange between processors: a layer of blocks D, E and F.

According to this diagram, processors P_{11}, P_{12}, and P_{13} process type D blocks; processors P_{14}, P_{21}, P_{22} and P_{23} process two blocks of the type E; processor P_{24} processes four blocks of the type F (four blocks of the type F create two blocks of the type E or one block of the type D). The data exchange is performed in accordance with the above permanent rules (15) – (16).

At the next step, we have again a layer consisting of blocks of the types B and C. This layer is similar to the previously considered layer of B and C blocks, but

the roles of processors are slightly different. In our geometrical interpretation of the algorithm, the data exchange diagram for this layer can be obtained by shifting the processors in Figure 5 in the "north-east" direction, which gives the data exchange rules depicted in Figure 7.

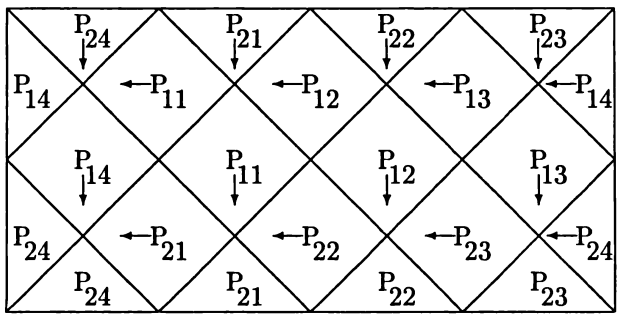

Fig. 7. Data exchange between processors: a layer of blocks B and C.

The last layer in our description is a layer of blocks of the type D. The corresponding data exchange diagram is shown in Figure 8.

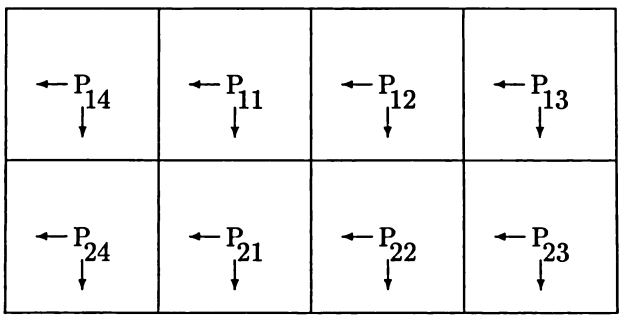

Fig. 8. Data exchange between processors: a layer of blocks D.

During the following steps, the sequence of the described layers (BC, DEF, BC, D) repeats, and we can use the above rules (15) – (16) for the data exchange in each sub-step.

4. SAMPLE IMPLEMENTATION

The suggested algorithm has been verified in both cases, one- and two-dimensional, by implementing it in FORTRAN with parallel extensions (Fortran-Lib version 3.3.11) on a workstation-cluster running under PVM (pvm version 3.3.11) using two sample examples:

a) for spatially one-dimensional IBVP:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \quad t > 0, \quad 0 < x < 4 \tag{17}$$

$$u(0,x) = x^2, \quad 0 < x < 4 \tag{18}$$

$$u(t,0) = 2t, \quad t > 0 \tag{19}$$

$$u(t,4) = 2t + 16, \quad t > 0. \tag{20}$$

Let us take the spatial step of discretization $h = 0.25$, the time step $\tau = 0.01$, the number of processors $nproc = 4$ and the maximum number of nodes processed by one processor at each time layer $q = 4$ (q may be only even). The number of steps taken is $pc = 10$.

The results of computations were checked by comparing them with the exact solution $u(t,x) = 2t + x^2$, while the accuracy 10^{-6} was reached.

b) for spatially two-dimensional IBVP:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}, \quad t > 0, \quad 0 < x < 4, \quad 0 < y < 2 \tag{21}$$

$$u(0,x,y) = x^2 + y^2, \quad 0 < x < 4, \quad 0 < y < 2 \tag{22}$$

$$u(t,0,y) = 4t + y^2, \quad t > 0, \quad 0 < y < 2 \tag{23}$$

$$u(t,4,y) = 4t + 16 + y^2, \quad t > 0, \quad 0 < y < 2 \tag{24}$$

$$u(t,x,0) = 4t + x^2, \quad t > 0, \quad 0 < x < 4 \tag{25}$$

$$u(t,x,2) = 4t + x^2 + 4, \quad t > 0, \quad 0 < x < 4. \tag{26}$$

Let us take the spatial steps of discretization $h_x = h_y = 0.25$, the time step $\tau = 0.01$, the number of processors $nproc = 8$ and the maximum number of nodes processed by one processor at each time layer $q = 4$ (q may be only even). The number of steps taken is $pc = 10$.

The results of computations were checked by comparing them with the exact solution $u(t,x) = 4t + x^2 + y^2$, while accuracy comparable to the previous example was reached.

5. SPEED-UP

Spatially one-dimensional IBVP: Let us suppose that values in the points u_i^0, $i = 0, 1, \ldots, n_x$, determined by initial condition $\varphi(x)$, are computed in advance and stored in memory. Let's consider a layer, which is $nq + 1$ points long (in the spatial axis) and q points wide (in the time axis).

In general, the number of arithmetical operations for computing one value in the point u_i^k inside the considered layer and on its border is not the same. It depends on the computational complexity of the points specified by boundary conditions.

Each processor computes the values in $\frac{q^2}{2}$ points inside the considered area, which corresponds in geometrical interpretation to a rotated square block (see Figure 1).

We need five arithmetic operations for computing the values u_i^k, $(i = 1, 2, \ldots, n_x - 1$, $k > 0)$ according to the difference scheme (5).

Each processor, which uses for computing also the boundary conditions, computes values in $\frac{q^2}{2} - q$ points inside the considered layer and in $2 \times q$ points on its border. In the geometrical interpretation it corresponds to two triangles standing opposite to each other on both boundaries (left and right) along the time-axis (see Figure 1). Let us denote the number of arithmetic operations needed for computing the values in the points u_0^k and $u_{n_x}^k$ as $r(\alpha)$ and $r(\beta)$, respectively. The number of arithmetic operations needed in the sequential case is $5[(nq - 1)q] + [r(\alpha) + r(\beta)]q$, and in the parallel case it is $\max\{5q^2; 5\frac{q^2}{2} + 5(\frac{q^2}{2} - q) + [r(\alpha) + r(\beta)]q\}$.

Now we are able to define the speed-up function $f(n, q)$ as a rate of the number of the arithmetic operations for computing the considered layer of the values in $(nq + 1) \times q$ points in the sequential case to the number of arithmetic operations in parallel. This gives:

$$\begin{aligned} f(n,q) &= \frac{5(nq-1)q + [r(\alpha) + r(\beta)]q}{\max\{5q^3; 5(q^2-1)q + [r(\alpha) + r(\beta)]q\}} \\ &= \frac{5(nq-1) + r(\alpha) + r(\beta).}{\max\{5q^2; 5(q^2-1) + r(\alpha) + r(\beta)\}}. \end{aligned}$$

If the functions $\alpha(t)$, $\beta(t)$ are constant, then $r(\alpha) = r(\beta) = 0$ and we can write the speed-up function in the form:

$$f(n,q) = \frac{5[(nq-1)q]}{5q^2} = \frac{nq-1}{q} \sim n.$$

Spatially two-dimensional IBVP: Since all aforementioned considerations for one-dimensional case hold for two-dimensional case as well, we only briefly describe obtaining of the speed-up function for this case. As above, suppose that the values in the points u_{ij}^0, $i = 0, 1, \ldots, n_x$, $j = 0, 1, \ldots, n_y$ determined by initial condition $\varphi(x, y)$, are computed in advance and stored in memory.

Let's consider a three-dimensional layer of $(mq+1) \times (nq+1)$ points of plane (in the spatial axes) times q points of height in the time-axis. Each processor computes the values in $2 \times \frac{q(q^2-1)}{3}$ points at the main layer and the values in $4 \times \frac{q(q^2+2)}{12}$ points at

the secondary layer inside the considered area, which corresponds in the geometrical interpretation to the blocks of types D and B (see Figure 2). We need nine arithmetic operations for computing the values u_{ij}^k, $(i = 1, 2 \ldots, m_x - 1,\ j = 1, 2, \ldots, n_x - 1,\ k > 0)$ according to the difference scheme (13). Each processor, which is used for computing also the boundary conditions, computes the values at least in $\frac{q(q^2-1)}{3} - q^2 + q$ points at the main layer, and in $\frac{q(q^2+2)}{12} - \frac{q^2}{4}$ points at the secondary layer inside the considered area, and at most in $2 \times (q^2 - q)$ points at the main layer and $2 \times \frac{q^2}{4}$ points at the secondary area on its border. In the geometrical interpretation this corresponds to four blocks of type F standing in the corners (see Figure 6), and 2×two blocks of type C standing opposite to each other on both boundaries (left and right, or up and down) – see Figure 5 or Figure 7). Let us denote the number of arithmetic operations needed for computing the values in the points u_{0j}^k, $u_{n_x j}^k$, u_{i0}^k, $u_{in_y}^k$ as $r(\alpha)$, $r(\beta)$, $r(\gamma)$ and $r(\delta)$. The number of the arithmetic operations needed in the sequential case is $9(mq-1)(nq-1)q + nq[r(\alpha) + r(\beta)] + mq[r(\gamma) + r(\delta)]$, and in the parallel case

$$\max\Big\{9q^3; 9\Big[\frac{q(q^2-1)}{3} + \frac{q(q^2-1)}{3} - q^2 + q + 4\Big(\frac{q(q^2+2)}{12} - \frac{q^2}{4}\Big)\Big] + \Big(\frac{q^2}{2} - \frac{q}{2} + 2\frac{q^2}{4}\Big)[r(\alpha) + r(\beta) + r(\gamma) + r(\delta)]\Big\}.$$

Now we can define the function of the speed-up $f(m, n, q)$ similarly to the one-dimensional case:

$$f(m,n,q) = \frac{9(mq-1)(nq-1) + m[r(\alpha) + r(\beta)] + n[r(\gamma) + r(\delta)]}{\max\{9q^2; 9(q^2 - 2q + 1) + (q - \frac{1}{2})[r(\alpha) + r(\beta) + r(\gamma) + r(\delta)]\}}.$$

If the functions $\alpha(t, y)$, $\beta(t, y)$, $\gamma(t, x)$ and $\delta(t, x)$ are constant, then $r(\alpha) = r(\beta) = r(\gamma) = r(\delta) = 0$ and we can write the speed-up function in the form:

$$f(m,n,q) = \frac{(mq-1)(nq-1)}{q^2} \sim m \times n.$$

6. CONCLUDING REMARKS

The theoretical estimates showed the significant speed-up of order n in one-dimensional case and of order $m \times n$ in two-dimensional case, in comparison with the serial implementation of the difference methods (5), (13). Further theoretical estimates and computational experiments are the subject of the continuing study of the suggested algorithm.

ACKNOWLEDGEMENT

The author is grateful to Österreichischer Ost- und Südosteuropa Institut for sponsoring the visit of the author to the University of Salzburg, Austria, where the computational

experiments were done on a workstation cluster running under PVM, and especially to prof. P. Zinterhof, the Director of the Research Institute of Software Technology, University of Salzburg, for fruitful discussions and general support. The author also thanks P. Schmitzberger for technical help with implementation of the algorithms on a workstation cluster running under PVM.

(Received June 4, 1999.)

REFERENCES

[1] K. Burrage: Parallel methods for initial value problems. Appl. Numer. Math. *11* (1993), 5–25.

[2] J. Crank and P. Nicolson: A practical method for numerical evaluation of solutions of PDEs of the heat-conduction type. Proc. Camb. Phil. Soc. *43* (1947), 60–67.

[3] T. L. Freeman and C. Phillips: Parallel Numerical Algorithms. Prentice Hall, Englewood Cliffs, N.J. 1992.

[4] P. M. Kogge: Parallel solution of recurrence problems. IBM J. Res. Develop. *2* (1974), 18, 138–148.

[5] J. M. Ortega and R. G. Voigt: Solution of PDE on Vector and Parallel Computers. SIAM, Philadelphia, 1985.

[6] M. Pavluš: Schwarz algorithm for solution of a quasiparabolic equation. Vestnik Moskov. Univ. *4* (1992), 15, 27–35.

[7] D. W. Peaceman and H. H. Rachford: The numerical solution of parabolic and elliptic differential equations. J. Soc. Indust. Appl. Math. *3* (1955), 28–41.

[8] G. D. Smith: Numerical Solution of PDE. Finite Difference Methods. Second edition. Clarendon Press, Oxford 1978.

[9] E. E. Tyrtyshnikov: Parallelization of some numerical methods. In: Numerical Solution of Partial Differential Equation, Košice 1992.

[10] M. Vajteršic: Algorithms for Elliptic Problems. Efficient Sequential and Parallel Solvers. VEDA, Bratislava 1988.

RNDr. Pavol Purcz, Department of Mathematics, Faculty of Civil Engineering, Technical University, Vysokoškolská 4, 042 00 Košice. Slovak Republic.
e-mail: purczp@ccsun.tuke.sk