Parallel LOD Scheme for 3D Parabolic Problem with Nonlocal Boundary Condition

Raimondas Čiegis

Vilnius Gediminas Technical University,
Saulėtekio av. 11, LT-10223 Vilnius, Lithuania
rc@fm.vtu.lt

Abstract. A parallel LOD algorithms for solving the 3D problem with nonlocal boundary condition is considered. The algorithm is implemented using the parallel array object tool *ParSol*, then a parallel algorithm follows semi-automatically from the serial one. Results of computational experiments are presented.

1 Problem Formulation

Boundary conditions are important part of any mathematical model. Recently new types of boundary conditions are proposed and investigated. Many physical and technological processes are described by mathematical models consisting of elliptic or parabolic problems with non-local boundary conditions. A review of such applications and mathematical results for analysis of one-dimensional problems is presented in the recent survey paper of Dehghan [9]. Numerical algorithms for solving linear and nonlinear parabolic problems with nonlocal boundary conditions are investigated in [5, 7, 8, 11, 12].

In this paper we consider parallel numerical algorithms for solving 3D parabolic problem with the additional integral boundary condition. Let $Q_T = \Omega \times [0,T]$, $\Omega = (0;1) \times (0;1) \times (0;1)$ be a domain with the boundary $\partial\Omega$. This boundary is split into two parts $\partial\Omega = \partial\Omega_1 \cup \partial\Omega_2$, $\partial\Omega_2 = \{X : (x_1, x_2, 0),\ 0 \le x_j \le 1,\ j = 1, 2\}$. In Q_T we consider a parabolic equation

$$\frac{\partial u}{\partial t} = \sum_{\alpha=1}^{3} \frac{\partial}{\partial x_\alpha} \left(k_\alpha(X,t) \frac{\partial u}{\partial x_\alpha} \right) - q(X,t)u + f(X,t), \tag{1}$$

subject to boundary conditions:

$$u(X,t) = \mu_1(X,t), \quad X \in \partial\Omega_1 \times (0,T],$$
$$u(X,t) = \mu_0(t)\mu_2(X), \quad X \in \partial\Omega_2 \times (0,T],$$

initial condition:

$$u(x_1, x_2, x_3, 0) = u_0(x_1, x_2, x_3), \quad X \in \Omega \cup \partial\Omega\,,$$

and the additional nonlocal condition:

$$\int_0^1 \int_0^1 \int_0^1 \rho(X,t)\, u(X,t)\, dx_3\, dx_2\, dx_1 = M(t). \tag{2}$$

Here $k_\alpha, q, d, \rho, f, u_0, M, \mu_j, j = 1, 2$ are given continuous functions, and the functions $u(X,t)$, $\mu_0(t)$ are unknown. Thus the initial-boundary problem (1)–(2) is over-specified, and the integral condition is used to identify the boundary condition function $\mu_0(t)$, i.e. we solve an inverse problem. When this boundary value is obtained, we can use any efficient method to solve a standard three-dimensional parabolic boundary value problem.

The existence and uniqueness of the solution of 2D problem is studied in [2]. The analysis of the forward Euler method and a modified Locally One Dimensional (LOD) scheme is presented in [10, 16]. At each splitting step of the LOD scheme one-dimensional problems were approximated by the forward Euler method, thus the obtained LOD method was only conditionally stable.

The analysis of new finite difference schemes (including the LOD method) is presented in [3, 4]. It is proved that integral (2) can be approximated by the trapezoidal rule, if the initial condition is approximated in consistent way.

High-performance computers with massive parallel processors are developing very fast and parallel numerical algorithms play an important role in large-scale scientific and engineering computations. Three groups of methods are widely used for solving multidimensional parabolic initial-boundary value problems: a) explicit algorithms, b) fully implicit approximations, c) splitting methods. In splitting methods the multidimensional problem is reduced to a sequence of one dimensional implicit difference systems with tridiagonal matrix. Special parallel versions of the serial factorization algorithm are used to implement LOD algorithms on multiprocessor computers. A reduction of communication costs is the second main problem in developing efficient parallel splitting algorithms for parallel computers with distributed memory.

In this paper we consider the LOD parallel algorithm for solving three dimensional problem (1)–(2) with the nonlocal boundary condition. The rest of the paper is organized as follows. In Section 2, we formulate the LOD finite-difference scheme. In Section 3 the parallel LOD algorithm is proposed. The parallel array object tool *ParSol* is used for its implementation. Then a parallel algorithm follows semi-automatically from the serial one. The complexity and scalability analysis of the parallel LOD algorithm is done. In Section 4 results of computational experiments are presented to test the accuracy and the efficiency of the parallel algorithm.

2 Locally One Dimensional Method

In Q_T we define a uniform grid $Q_{h\tau} = \omega_h \times \omega_\tau$:

$$\omega_h = \big\{(x_{1i}, x_{2j}, x_{3k}) :\ x_{\alpha,i} = ih,\ h = \frac{1}{J},\ 0 < i < J\big\},$$
$$\omega_\tau = \big\{t^n :\ \ t^n = n\tau,\ n = 1, 2, \ldots, N,\ N\tau = T\ \big\}.$$

Let γ_h be a boundary of ω_h, we split it into two parts $\gamma_h = \gamma_{1h} \cup \gamma_{2h}$. Let $U^n_{ijk} = U(x_{1i}, x_{2j}, x_{3k}, t^n)$ be a discrete approximation to the exact solution of differential problem (1)–(2).

We propose unconditionally stable LOD scheme, which approximates 3D parabolic problem and the integral condition:

$$\begin{cases} \dfrac{U^{n+j/3} - U^{n+(j-1)/3}}{\tau} = A_j U^{n+1/3} + \delta_{1j} f^{n+1}, \;\; j = 1,2,3, \;\; X \in \omega_h, \\ U^{n+1/3}_{ijk} = (I - \tau A_2)(I - \tau A_3)\mu_1^{n+1}, \;\; X \in \gamma_{1h}(x_1 = 0) \cup \gamma_{1h}(x_1 = 1), \\ U^{n+2/3}_{ijk} = (I - \tau A_3)\mu_1^{n+1}, \;\; X \in \gamma_{1h}(x_2 = 0) \cup \gamma_{1h}(x_2 = 1), \\ U^{n+1}_{ijk} = \mu_1^{n+1}, \;\; X \in \gamma_{1h}, \\ U^{n+1}_{ijk} = \mu_0^{n+1}\mu_2, \;\; X \in \gamma_{2h}, \\ S_h U^{n+1} = M(t^{n+1})\,. \end{cases} \tag{3}$$

Boundary conditions are approximated consistently with the approximation of the differential equations [17]. Here we use the following difference operators:

$$A_\alpha U = \big(a_\alpha U_{\bar{x}_\alpha}\big)_{x_\alpha} - \frac{1}{3} q(X, t^n) U, \;\; \alpha = 1,2,3,$$

$$a^n_{\alpha,\,ijk} = k_\alpha\big(x_{1i} - \frac{h}{2}\delta_{1\alpha},\, x_{2j} - \frac{h}{2}\delta_{2\alpha},\, x_{3k} - \frac{h}{2}\delta_{3\alpha}\big),$$

$$U_{x_1} = \frac{U_{i+1,jk} - U_{ijk}}{h}, \quad U_{\bar{x}_2} = \frac{U_{ijk} - U_{i,j-1,k}}{h}.$$

Integral condition (2) is approximated by the trapezoidal rule

$$S_h U^{n+1} := \sum_{i,j,k=0}^{J} c_i c_j c_k\, \rho^{n+1}_{ijk}\, U_{ijk}\, h^3 = M(t^{n+1}), \tag{4}$$

$$c_0 = \frac{1}{2}, \;\; c_l = 1, \; l = 1, \ldots, J-1, \;\; c_J = \frac{1}{2}.$$

We propose to change the simplest approximation of the initial condition

$$U^0 = u_0(X), \;\; X \in \omega_h \cup \gamma_h$$

by the following one, which exactly satisfies the discrete nonlocal condition:

$$U^0 = \frac{M(t^0)\, u_0(X)}{S_h u_0}\,, \quad X \in \omega_h \cup \gamma_h\,. \tag{5}$$

Then the truncation error of the discrete initial condition is given by

$$|U^0 - u_0(X)| = \mathcal{O}(h^2)\,,$$

but this error is not propagating in time due to the stability of the LOD method with respect to the initial condition. This new discretization of the initial condition is mass conservative, therefore the accuracy of approximation of the boundary condition μ^n is increased to the second order.

The LOD scheme is implemented as follows. The first two subproblems for $j = 1, 2$ are standard: we solve $(J-1)^2$ systems of linear equations, the matrix of each system is tridiagonal. Total costs of these two steps are $\mathcal{O}(J^3)$ floating point operations.

The serial implementation algorithm of the third step was proposed in [3]. By using the Dirichlet boundary condition at $\gamma_{1h}(x_1 = 1)$ and discrete 1D equations with operator A_3 we obtain the factorization coefficients $\tilde{\alpha}^{n+1}$, $\tilde{\beta}^{n+1}$ such that:

$$U^{n+1}_{ijk} = \tilde{\alpha}^{n+1}_{ijk} U^{n+1}_{i,j,k-1} + \tilde{\beta}^{n+1}_{ijk}, \;\; 0 < i, j < J, \; k = J, \ldots, 1.$$

Then the solution is expressed in the following form:

$$\begin{aligned} U^{n+1}_{ijk} &= \alpha^{n+1}_{ijk} U^{n+1}_{ij0} + \beta^{n+1}_{ijk}, \;\; i, j = 0, \ldots, J, \\ \alpha^{n+1}_{ijk} &= \tilde{\alpha}^{n+1}_{ijk} \alpha^{n+1}_{i,j,k-1}, \quad k = 1, \ldots, J, \quad \alpha^{n+1}_{ij0} = 1, \\ \beta^{n+1}_{ijk} &= \tilde{\alpha}^{n+1}_{ijk} \beta^{n+1}_{i,j,k-1} + \tilde{\beta}^{n+1}_{ijk}, \quad \beta^{n+1}_{ij0} = 0. \end{aligned} \tag{6}$$

By using the discrete non-local condition we find the function:

$$\mu^{n+1}_0 = \frac{M(t^{n+1}) - S_h \beta^{n+1}}{S_h(C)}, \quad C_{ijk} = \alpha^{n+1}_{ijk} \mu_{2ij}.$$

After determination of μ^{n+1}_0 solution U^{n+1} is computed by using (6). The complexity of the third step of LOD scheme is equal to $\mathcal{O}(J^3)$.

3 Parallel Algorithm

Let us assume that we have p processors, which are connected by three dimensional mesh, i.e. $p = p_1 \times p_2 \times p_3$. The grid ω_h (a data set) is decomposed into a number of 3D subgrids by using a block distribution scheme. Then each subgrid ω_{hp} has

$$\frac{(J+1)}{p_1} \times \frac{(J+1)}{p_1} \times \frac{(J+1)}{p_1} = \frac{(J+1)^3}{p}$$

computational points of the grid ω_h and it is assigned to one processor, which is responsible for all computations of the local part of vector U.

Since the sub-domains are connected at their boundaries, processors dealing with neighbouring sub-domains have to exchange boundary information with each other at every time-step. More exactly, the update of vector U^{n+1} at grid points which lie beside cutting planes (i.e. boundary nodes of the local part of the vector U) needs a special attention, since information from the neighbouring

processors is required to compute new values of U^{n+1}. Such information is obtained by exchanging data with neighbour processors in the specified topology of processors. The amount of exchanged data depends also on the grid stencil, which is used to discretize the PDE model. A star-stencil of seven points is used in (3), therefore local subgrids are enlarged by two *ghost* points in each dimension of the subgrid.

In the parallel algorithm the implementation of the third step of the LOD scheme is modified to the following one:

$$U^{n+1} = V^{n+1} + \gamma^{n+1} W^{n+1}\,,$$

where V^{n+1} is a solution of the discrete boundary value problem

$$\begin{cases} \dfrac{V^{n+1} - U^{n+2/3}}{\tau} = A_3 V^{n+1}, \;\; X \in \omega_h, \\ V^{n+1} = \mu_1(X, t^{n+1}), \;\; X \in \gamma_{1h}, \\ V^{n+1} = \mu_0^n \, \mu_2(X, t^{n+1}), \;\; X \in \gamma_{2h}\,. \end{cases} \tag{7}$$

Function W^{n+1} is a solution of the auxiliary problem

$$\begin{cases} W^{n+1}/\tau = A_3 W^{n+1}, \;\; X \in \omega_h, \\ W^{n+1} = 0, \;\; X \in \gamma_{1h}, \\ W^{n+1} = \mu_2(X, t^{n+1}), \;\; X \in \gamma_{2h}\,. \end{cases} \tag{8}$$

Then we find μ_0^{n+1} by using the discrete nonlocal condition:

$$\gamma^{n+1} = \frac{M(t^{n+1}) - S_h V^{n+1}}{S_h W^{n+1}}\,.$$

Thus during implementation of the parallel LOD algorithm we solve $4(J-1)^2$ systems of linear equations with tridiagonal matrix.

The complexity of solving one tridiagonal system of J equations by the serial factorization algorithm is equal to $8J$ arithmetical operations.

For two processors the Gaussian elimination process is started simultaneously at the first and last equations and it goes in opposite directions. Processors exchange two factorization coefficients at the end of the first stage of the factorization algorithm. The total complexity of this modified algorithm is equal to $8J$ arithmetical operations.

For the case when a system is distributed between $p_1 > 2$ processors, we use the Wang parallel factorization algorithm [14]. It solves the tridiagonal system by using $17J$ arithmetical operations. The main idea is to reduce the given system to a new tridiagonal system of p_1 equations, where each processor has only one equation. Such small system is solved by using the serial factorization algorithm. The total costs of the parallel Wang algorithm in the worst case when

the simplest algorithm is used to broadcast data to the master process can be estimated as

$$T_p(J) = \frac{17J}{p_1} + 8p_1 + p_1(\alpha + \beta).$$

4 Complexity and Scalability Analysis

We will estimate the complexity of the LOD algorithm by counting basic operations. At each time step the following amount of work is done:

1. Coefficients of the LOD scheme (3) are computed. The complexity of this step is J^3.
2. $3(J-1)^2$ systems with tridiagonal matrix are solved. The complexity of this step is aJ^3.
3. Discrete approximations of integrals $S_h(V^{n+1})$, $S_h(W^{n+1})$ are computed. The complexity of this step is bJ^3.
4. The values of the solution on boundary γ_{2h} are updated with a known function μ_0^{n+1}. The complexity of this step is cJ^2.

As a result, the total complexity of the serial LOD algorithm can be expressed as

$$W = (1 + a + b)J^3 + cJ^2 = (1 + a + b)J^3 + \mathcal{O}(J^2)\,. \tag{9}$$

The communication step is implemented before updating vectors $U^{n+j/3}$, $j = 1, 2, 3$ and only neighbouring processors are communicating with each other. Each processor exchanges with its six neighbours vector elements corresponding to boundary points of the local subdomain. A total amount of data, exchanged between two processors, is equal to $J^2/p^{2/3}$ elements. This can be done in

$$T_{1,p}(J) = \alpha + \beta\frac{J^2}{p^{2/3}}$$

time, by using the *odd–even* data exchange algorithm. Here α is the message startup time and β is the time required to send one element of data.

When the required information is exchanged, processors compute in parallel coefficients of local part of the matrix. The complexity of this step is given by

$$T_{2,p}(J) = \frac{J^3}{p}\,.$$

Parallel computation of integrals $S_h(V^{n+1})$ and $S_h(W^{n+1})$ requires global communication among all processors during summation of local parts of integrals. The complexity of reduce operation depends strongly on the architecture of the parallel computer (see [13]). We will estimate the time required to reduce local values of integrals between p processors by $B(p) = R(p)(\alpha_b + \beta_b)$, where $R(p)$ depends on the algorithm used to implement the `MPI_ALLREDUCE` operation

and the architecture of the computer. For the simplest reduce algorithm, when every processor sends its result to the master processor, who finishes computation of the integral and broadcasts the global sum to all processors, $R(p) = p$. Thus the complexity of parallel computation of both integrals and updating boundary values on γ_{2h} is given by

$$T_{3,p}(J) = 2R(p)(\alpha_b + \beta_b) + b\frac{J^3}{p} + c\frac{J^2}{p^{2/3}}\,.$$

The time required to solve all $4(J-1)^2$ systems of linear equations is estimated as

$$T_{4,p}(J) = \frac{4}{3}\,\frac{17aJ^3}{8p} + 4J^2\big(8p^{1/3} + p^{1/3}(\alpha+\beta)\big).$$

Summing up all obtained estimates we compute the complexity of the parallel LOD algorithm

$$T_p(J) = \Big(1 + \frac{17a}{6} + b\Big)\frac{J^3}{p} + 6\Big(\alpha + \beta\frac{J^2}{p^{2/3}}\Big) + c\frac{J^2}{p^{2/3}} + 2R(p)(\alpha_b + \beta_b). \quad (10)$$

According to the definition of the isoefficiency function, we must find the rate at which the problem size W needs to grow with p for a fixed efficiency of the algorithm. Let $H(p,W) = pT_p - W$ be the total overhead of a parallel algorithm. Then the *isoefficiency* function $W = g(p,E)$ is defined by the implicit equation (see [14]):

$$W = \frac{E}{1-E}\,H(p,W)\,.$$

The total overhead of the parallel LOD algorithm is given by

$$\begin{aligned} H(p,W) &= \frac{11}{6}aJ^3 + 6\alpha p + (6\beta + c)\,p^{1/3}J^2 + 2pR(p)(\alpha_b + \beta_b) \\ &= \frac{11a}{6(1+a+b)}W + 6\alpha p + \frac{(6\beta+c)p^{1/3}}{(1+a+b)^{2/3}}W^{2/3} + 2pR(p)(\alpha_b+\beta_b). \end{aligned}$$

The first term defines a range of possible values of E. This term in $H(p,W)$ arises due to the fact the parallel algorithm does not coincide with the serial LOD algorithm. For simplicity of notation we take E such, that

$$\frac{11aE}{6(1+a+b)(1-E)} = \frac{1}{2}\,.$$

Since it is impossible to get the isoefficiency function in a closed form as a function of p, we will analyze the influence of each individual term. The component that requires the problem size to grow at the fastest rate determines the overall asymptotic isoefficiency function. After simple computations we get the following three isoefficiency functions

$$W = \mathcal{O}(p), \quad W = \mathcal{O}(p), \quad W = \mathcal{O}\big(pR(p)\big)\,.$$

Thus the the overall asymptotic isoefficiency function is defined by the overheads of the global reduction operation. Let us assume that processors are connected by three–dimensional mesh $p^{1/3} \times p^{1/3} \times p^{1/3}$. Then the global *reduce* and *broadcast* operations can be implemented with $R(p) = p^{1/3}$. Thus the problem size W has to grow as $\mathcal{O}(p^{4/3})$ to maintain a certain efficiency. For a hypercube mesh we have smaller costs of the global reduction operation $R(p) = \log p$, then isoefficiency function is close to linear $W = \mathcal{O}(p \log p)$.

We note, that in the case of a moderate number of processors $p = \mathcal{O}(J)$, the costs of global reduction operation can be ignored and the isoefficiency function $W = \mathcal{O}(p)$ depends linearly on p.

Parallel numerical objects. Special tools are developed to simplify parallelization of sequential algorithms, e.g. *Diffpack* tool [15] and *PETSc* toolkit [1]. We have developed new tool *ParSol* of parallel numerical arrays, which can be used for semi–automatic parallelization of data parallel algorithms, that are implemented in `C++`. Such algorithms are usually constructed for solving PDEs and systems of PDEs on logically regular rectangular grids. *ParSol* is a library of parallel array objects, a functionality of which is similar to *Distributed Arrays* in *PETSc*. We list the following main features of *ParSol* (see [6]): a) created for `C++` programming language, b) based on HPF ideology, c) the library heavily uses such `C++` features as OOP and template, d) MPI 1.1 standard is used to implement parallelization.

ParSol arrays have a number of advantages for programming mathematical algorithms, such as virtual indexing, built-in array operations, automated management of dynamically allocated memory, periodic boundary conditions. *ParSol* arrays simulate numerical objects of linear algebra and many useful basic vector operations are supported within the *ParSol* library, e.g. parallel computation of vector norms, the inner product of two vectors, scaling of vectors.

The LOD algorithm can not be described as a simple data parallel algorithm, but *ParSol* library is used to implement the algorithm (3) and only the Wang algorithm requires a special treatment.

5 Results of Computational Experiments

In this section we present some results of computational experiments. Computations were performed on IBM SP5 computer at CINECA, Bologna. We have solved problem (1)–(2) with the following coefficients and the exact solution:

$$k_\alpha(X,t) = 1 + (x_1^2 + x_2^2 + x_3^2)t, \quad q(X,t) = (x_1 + x_2 + x_3)t^2,$$
$$M(t) = e^t(A^3 + B^3), \quad A = 2(e^{0.5} - 1), \quad B = 2(2 - e^{0.5}),$$
$$\rho(X,t) = 1 + x_1x_2x_3, \quad u(X,t) = \exp\big(0.5(x_1 + x_2 + x_3) + t\big).$$

In order to scale the computation time for different space steps $h = 1/(J-1)$, a solution was computed in time intervals $[0, T(J)]$, where

$$T(40) = 0.4, \quad T(80) = 0.04, \quad T(120) = 0.005, \quad T(160) = 0.001.$$

In Table 1 we present the values of experimental speedup $S_p(J) = \dfrac{T_1(J)}{T_p(J)}$ and efficiency $E_p(J) = \dfrac{S_p(J)}{p}$ coefficients for different sizes of the discrete problem.

Table 1. The speedup and efficiency coefficients for the LOD method. CPU time of the sequential algorithm (in s): $T_1(40) = 64.8$, $T_1(80) = 105.2$, $T_1(120) = 94.98$, $T_1(160) = 131.0$

p	$S_{p,40}$	$E_{p,40}$	$S_{p,80}$	$E_{p,80}$	$S_{p,120}$	$E_{p,120}$	$S_{p,160}$	$E_{p,160}$
2	1.979	0.990	2.001	1.000	2.115	1.058	1.955	0.978
4	3.880	0.970	4.062	1.016	4.236	1.059	4.662	1.166
8	7.043	0.880	7.684	0.961	8.284	1.036	9.388	1.173
16	11.54	0.721	14.30	0.894	15.23	0.952	18.10	1.131
32	18.72	0.585	26.73	0.835	29.67	0.927	34.77	1.087

It follows from results, presented in Table 1, that the parallel LOD algorithm scales well.

Remark 1. We see that a superlinear speedup of the parallel algorithm is obtained, when more processors are used. This effect is due to special properties of cash memory usage in SP5 processors. We implemented a simple test, where matrix operations $A := A + B$, $C := C - D$ were performed many times. The dimension of matrix is taken to be $160 \times 160 \times 160$. The following results were obtained:

$$T_1 = 35.3, \;\; T_2 = 15.3, \;\; T_4 = 7.18, \;\; T_8 = 2.83, \;\; T_{16} = 1.29, \;\; T_{32} = 0.65.$$

Acknowledgment

The work has been performed under the Project HPC–EUROPA (RII3–CT-2003-506079), with the support of the European Community – Research Infrastructure Action under the FP6 "Structuring the European Research Area" Programme. I gratefully acknowledge the hospitality and excellent working conditions in CINECA, Bologna. In particular I thank Dr. Giovanni Erbacci for his help.

This work was also supported by the Lithuanian State Science and Studies Foundation within the framework of the Eureka Project EUREKA E!3691 OPTCABLES .

References

1. S. Balay, K. Buschelman, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L. Curfman McInnes, B.F. Smith, H. Zhang. PETSc Users Manual. ANL-95/11 - Revision 2.3.0, Argonne National Laboratory, 2005.
2. J.R. Cannon, Y. Lin, A. L. Matheson. Locally explicit schemes for three–dimensional diffusion with non–local boundary specification. *Appl. Anal.*, **50**, 1–19, 1993.
3. R. Čiegis. Economical difference schemes for the solution of a two dimensional parabolic problem with an integral condition, *Differential Equations*, **41**(7), 1025–1029, 2005.
4. R. Čiegis. Numerical Schemes for 3D Parabolic Problem with Non–Local Boundary Condition, In: *Proceedings of 17th IMACS World congress, Scientific Computation, Applied Mathematics and Simulation*, June 11-15, 2005, Paris, France, T2-T-00-0365, 2005.
5. R. Čiegis. Finite-Difference Schemes for Nonlinear Parabolic Problem with Nonlocal Boundary Conditions. In: M. Ahues, C. Constanda, A. Largillier (Eds.) *Integral Methods in Science and Engineering: Analytic and Numerical Techniques*, ISBN 0-8176-3228-X, Birkhauser, Boston, 47–52, 2004.
6. R. Čiegis, A. Jakušev, A. Krylovas and O. Suboč. Parallel algorithms for solution of nonlinear diffusion problems in image smoothing. *Math. Modelling and Analysis*, **10**(2), 155–172, 2005.
7. R. Čiegis, A. Štikonas, O. Štikonienė, O. Suboč. Monotone finite-difference scheme for parabolical problem with nonlocal boundary conditions. *Differential Equations*, **38**(7), 1027–1037, 2002.
8. R. Čiegis, A. Štikonas, O. Štikonienė, O. Suboč. Stationary problems with nonlocal boundary conditions. *Mathematical Modelling and Analysis*, **6**, 178–191, 2001.
9. M. Dehghan Efficient techniques for the second-order parabolic equation subject to nonlocal specifications *Applied Numer. Math.*, **52**, 39–62, 2005.
10. M. Dehghan Locally explicit schemes for three–dimensional diffusion with non–local boundary specification. *Applied Mathematics and Computation*, **135**, 399–412, 2002.
11. G. Ekolin. Finite difference methods for a nonlocal boundary value problem for the heat equation. *BIT*, **31**(2), 245–255, 1991.
12. G. Fairweather, J.C. Lopez-Marcos. Galerkin methods for a semilinear parabolic problem with nonlocal boundary conditions. *Adv. Comput. Math.*, **6**, 243–262, 1996.
13. R. Hockney. Performance parameters and benchmarking on supercomputers. *Parallel Computing*, **17**, 1111–1130, 1991.
14. V. Kumar, A. Grama, A. Gupta, G. Karypis. *Introduction to parallel computing: design and analysis of algorithms*. Benjamin/Cummings, Redwood City, 1994.
15. H.P. Langtangen and A. Tveito. *Advanced Topics in Computational Partial Differential Equations. Numerical Methods and Diffpack Programming.* Springer, Berlin, 2003.
16. B.J. Noye, M. Dehghan. New explicit finite difference schemes for two–dimensional diffusion subject specification of mass. *Numer. Meth. for PDE*, **15**, 521–534, 1999.
17. A.A. Samarskii. *The theory of difference schemes*. Marcel Dekker, Inc., New York–Basel, 2001.