INFORMATICA, 2006, Vol. 17, No. 3, 309–324

Parallel Numerical Algorithms for 3D Parabolic Problem with Nonlocal Boundary Condition

Raimondas ČIEGIS
Vilnius Gediminas Technical University
Saulėtekio al. 11, LT-10223 Vilnius, Lithuania
e-mail: rc@fm.vtu.lt

Received: January 2006

Abstract. Three parallel algorithms for solving the 3D problem with nonlocal boundary condition are considered. The forward and backward Euler finite-difference schemes, and LOD scheme are typical representatives of three general classes of parallel algorithms used to solve multidimensional parabolic initial-boundary value problems. All algorithms are modified to take into account additional nonlocal boundary condition. The algorithms are 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 and the accuracy and efficiency of the presented parallel algorithms are tested.

Key words: parallel algorithms, finite-difference schemes, parabolic problems, nonlocal conditions.

1. Problem Formulation

Many physical and technological processes are described by mathematical models consisting of elliptic or parabolic problems with non-local boundary conditions. Parabolic initial-boundary value problems in one dimension were considered firstly by Cannon (1963) and Kamynin (1964), a generalization of linear problems for two-phase Stefan problem is studied in Cannon and van der Hoek (1982) (see also papers of Ionkin (1977), Ionkin (1980), Makarov and Kulyev (1985)). A review of such applications and mathematical results for analysis of one-dimensional problems is presented in the recent survey paper of Dehghan (2005).

Numerical algorithms for solving linear and nonlinear parabolic problems with nonlocal boundary conditions are investigated in Čiegis (1991, 2004), Čiegis *et al.* (2001, 2002), Ekolin (1991), Fairweather and Lopez–Marcos (1996), see also references given in these papers.

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 $\partial\Omega = \partial\Omega_1 \cup \partial\Omega_2$:

$$\partial\Omega_2 = \big\{X\colon\ (x_1,x_2,0),\ 0 \leqslant x_j \leqslant 1,\ j=1,2\big\}.$$

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:

$$\int_0^1 \int_0^1 \int_0^1 \rho(X,t)u(X,t)\,\mathrm{d}x_3\,\mathrm{d}x_2\,\mathrm{d}x_1 = M(t). \tag{2}$$

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 the information about the boundary value is obtained, we can use any efficient method to solve a standard three-dimensional parabolic problem.

The existence and uniqueness of the solution of 2D problem is studied by Cannon, Lin and Matheson (1993). Noye and Dehghan (1999) have investigated the forward Euler method and a modification of Locally One Dimensional (LOD) scheme. At each splitting step of the LOD scheme one-dimensional problems were approximated by the forward Euler method, thus the obtained method was only conditionally stable. A similar LOD method was used to solve 3D problem in Dehghan (2002). We note that in all these papers integrals were approximated by high order numerical integration methods.

The analysis of new finite difference schemes is presented in Čiegis (2005a, 2005b). It is proved that the integrals can be approximated by the trapezoidal rule, if the initial condition is approximated in consistent way. The accuracy of the backward Euler scheme is investigated and an algorithm for implementation of this scheme is presented. Sufficient conditions for the existence of a discrete solution are given. The implicit LOD scheme is also proposed in Čiegis (2005a, 2005b) and efficient implementation algorithm is given.

With rapid development of high-performance computers with massive parallel processors, 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.

1. The explicit algorithms are well suited for developing efficient data-parallel algorithms. However explicit algorithms are only conditionally stable and the stability

condition $\tau \leqslant Ch^2$ should be imposed on the time-step size, thus integration of the problem is done with a very small time-step.

2. The fully implicit approximations are unconditionally stable, but they require solving systems of linear equations with very large sparse matrix. Such systems are solved by iterative methods and the efficiency of parallel algorithms depend on the efficiency of parallel versions of these iterative algorithms.
3. In splitting methods the multidimensional problem is reduced to a sequence of one dimensional implicit difference systems with three-diagonal 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 three parallel algorithms 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 forward Euler finite-difference scheme. We investigate the efficiency of the developed parallel algorithm and present results of the scalability analysis. The parallel forward Euler algorithm is implemented by using the parallel array object tool *ParSol*. Then a parallel algorithm follows semi-automatically from the serial one. Results of computational experiments are presented to test the accuracy and the efficiency of the parallel algorithm. In Section 3 the parallel backward Euler finite-difference scheme is proposed. The obtained systems of linear equations are solved by the preconditioned Conjugate Gradient method. *ParSol* tool is used to implement a parallel algorithm. The complexity analysis of the scalability analysis is presented. In Section 4 the parallel LOD scheme is proposed. The efficiency of the algorithm is investigated and results of numerical experiments are discussed.

2. The Forward Euler Method

In this section we investigate the efficiency of the parallel forward Euler method. In Q_T we define a uniform grid $Q_{h\tau} = \omega_h \times \omega_\tau$:

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

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).

2.1. *Formulation of the Algorithm*

The forward Euler approximation of equation (1) and boundary conditions is defined by

$$\begin{cases} \dfrac{U^{n+1}-U^n}{\tau} = \sum\limits_{\alpha=1}^{3} A_\alpha U^n + f^n, & X \in \omega_h, \\ U^{n+1} = \mu_1(X, t^{n+1}), & X \in \gamma_{1h}, \\ U^{n+1} = \mu_0^{n+1}\mu_2(X, t^{n+1}), & X \in \gamma_{2h}. \end{cases} \tag{3}$$

Here we use the following difference operators:

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

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

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

Integral condition (2) is approximated by the trapezoidal rule

$$S_h U^{n+1} = M(t^{n+1}), \tag{4}$$

where

$$S_h V = \sum_{i,j,k=0}^{J} c_i c_j c_k \rho^{n+1}_{ijk} V_{ijk} h^3,$$

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

Let us define functions $\tilde{U}$, B:

$$\tilde{U}_{ijk} = \begin{cases} U_{ijk}, & 0 < k \leqslant J, \\ 0, & k = 0, \end{cases} \qquad B_{ijk} = \begin{cases} 0, & 0 < k \leqslant J, \\ \mu_2(x_{1i}, x_{2j}), & k = 0. \end{cases}$$

Then we find μ_0^{n+1} by using the discrete nonlocal condition (4):

$$\mu_0^{n+1} = \frac{M(t^{n+1}) - S_h\tilde{U}^{n+1}}{S_h B}. \tag{5}$$

The unique solution of discrete problem (3)–(5) exists under the condition

$$S_h B \neq 0.$$

An important part of the formulation of the discrete method deals with the approximation of the initial condition. We propose to change the simplest approximation of the initial condition

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

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

$$U^0 = \frac{M(t^0)u_0(X)}{S_h u_0}, \quad X \in \omega_h \cup \gamma_h. \tag{6}$$

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 propagated in time due to stability of the forward Euler method with respect to initial condition. It is important to note that this new discretization of the initial condition is mass conservative. Therefore the accuracy of the approximation of the boundary condition μ^n is increased to the second order.

It is easy to prove that the forward Euler scheme is stable only if $\tau \leqslant Ch^2$, thus integration is done with very small time step for small h. Results of computational experiments are presented in Čiegis (2005a, 2005b).

2.2. *Parallel Algorithm*

The power of modern personal computers is increasing constantly, but not enough to fulfill all scientific and engineering computational demands. This situation is typical, when we solve three dimensional problems. In such cases, parallel computing can be the best solution.

The serial forward Euler algorithm can be modified to the parallel algorithm by using data parallel decomposition (see Kumar *et al.* (1994)). 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

$$\frac{(J+1)}{p_1} \times \frac{(J+1)}{p_2} \times \frac{(J+1)}{p_3} = \frac{(J+1)^3}{p}$$

computational points of the grid ω_h and it is assigned to one processor. All processors simultaneously perform the same code but with different data sets. Each processor 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}. Note, that these nodes are inner nodes in the global grid ω_h. Such information is obtained by exchanging data with neighbour processors in the specified topology of processors and the amount of 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.

The communication step is implemented before updating vector U^{n+1} and only neighbouring processors need to communicate with each other. Data exchange steps are synchronization points in the code and they divide the computation into phases. When all required data is saved in the local memory, the new values of U^{n+1} are computed locally and in parallel on each processor.

The computation of both integrals S_h and $S_h B$ requires global communication of all processors: first processors compute discrete sums of local parts of vectors and then these local results are summed up. Different algorithms can be used to implement the global reduction step. We note that in MPI there exists a special function `MPI_ALLREDUCE`, which computes a sum and distributes it to all processors. It is assumed that MPI library is optimized for each type of super-computer, taking into account specific details of the computer network.

Thus the parallel implementation of the forward Euler method is identical to its serial implementation. We will show that implementation of the parallel LOD algorithm is different from the serial one.

Parallel numerical objects. The major difficulty in using parallel computers deals with the fact, that writing a parallel program (or parallelizing existing sequential codes), requires the knowledge of special methods and tools, which is not trivial to be mastered. Special tools are developed to simplify the parallelization of algorithms, e.g. *Diffpack* tool (see Langtangen and Tveito (2003)) and *PETSc* toolkit (see Balay *et al.* (2005)). We have developed a 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 Čiegis *et al.* (2005)):

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,

e) *ParSol* is an open source library.

Comparing to native `C++` arrays, *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 various

norms, the inner product of two vectors, scaling of vectors. They are implemented as methods of parallel arrays.

Performance analysis. We will estimate the complexity of the forward Euler algorithm by counting basic operations (updates of U_{ijk}^{n+1} at one grid point by formula (3)). It follows from the forward Euler algorithm that at each time step we should compute:

1. New values of vector U^{n+1}, which are updated by formula (3). The complexity of this step is J^3.
2. One three-dimensional discrete approximation of integral (4). The complexity of this step is cJ^3.
3. One two-dimensional integral $S_h(B)$, and the updated boundary condition with a known function μ_0^{n+1}. The complexity of this step is bJ^2.

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

$$W = (1+c)J^3 + bJ^2 = (1+c)J^3 + \mathcal{O}(J^2). \tag{7}$$

The complexity of one parallel update of a local part of vector U^{n+1} is given by

$$T_{1,p}(J) = \frac{J^3}{p}.$$

At the beginning of new time step 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

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

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.

Parallel computation of integrals $S_h(\tilde{U}^{n+1})$, $S_h(B)$ requires global communication among all processors during summation of local parts of integrals. The complexity of such reduce operation depends strongly on the architecture of the parallel computer (see Hockney (1991)). We will estimate the time required to reduce local values of integrals between p processors by

$$B(p) = R(p)(\alpha_b + \beta_b), \tag{8}$$

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 one parallel computation of both integrals and update of the vector at boundary points $z = 0$ is given by

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

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

$$T_p(J) = (1+c)\frac{J^3}{p} + 6\Big(\alpha + \Big(\beta + \frac{b}{6}\Big)\frac{J^2}{p^{2/3}}\Big) + 2R(p)(\alpha_b + \beta_b). \tag{9}$$

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 Kumar *at al.* (1994)):

$$W = \frac{E}{1-E}H(p, W). \tag{10}$$

For simplicity of notation we take $E = 0.5$.

The total overhead of the parallel forward Euler algorithm is given by

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

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 $W = \mathcal{O}(p)$ linearly depends on p.

2.3. *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

$$T(40) = 0.2, \quad T(80) = 0.02, \quad T(120) = 0.004, \quad T(160) = 0.001.$$

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

It follows from results, presented in Table 1, that the parallel forward Euler algorithm scales well, and experimental results confirm the theoretical prediction that isoefficiency function of the parallel algorithm is close to linear one. We note that a parallel implementation of the forward Euler algorithm was obtained by using the parallelization tool *ParSol*.

Table 1

The speedup and efficiency coefficients for the forward Euler method. CPU time of the sequential algorithm (in *s*): $T_1(40) = 76.9$, $T_1(80) = 173.4$, $T_1(120) = 258$, $T_1(160) = 274.9$

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	1.954	0.977	1.979	0.989	1.985	0.993	2.000	1.000
4	3.873	0.968	3.944	0.986	3.944	0.986	4.013	1.003
8	7.135	0.892	7.744	0.968	7.875	0.984	7.922	0.990
16	12.76	0.797	15.07	0.942	15.53	0.971	15.76	0.985
32	22.34	0.698	29.04	0.907	30.51	0.953	31.10	0.972

3. The Backward Euler Method

The backward Euler approximation of (1)–(2) is defined by

$$\begin{cases} \frac{U^{n+1}-U^n}{\tau} = \sum\limits_{\alpha=1}^{3} A_\alpha U^{n+1} + f^{n+1}, & X \in \omega_h, \\ U^{n+1} = \mu_1(X, t^{n+1}), & X \in \gamma_{1h}, \\ U^{n+1} = \mu_0^{n+1}\mu_2(X, t^{n+1}), & X \in \gamma_{2h}, \\ S_h U^{n+1} = M(t^{n+1}). \end{cases} \tag{11}$$

The solution U^{n+1} is expressed in the following form

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

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

$$\begin{cases} \frac{V^{n+1}-U^n}{\tau} = \sum\limits_{\alpha=1}^{3} A_\alpha V^{n+1} + f^{n+1}, & X \in \omega_h, \\ V^{n+1} = \mu_1(X, t^{n+1}), & X \in \gamma_{1h}, \\ V^{n+1} = \mu_0^{n}\mu_2(X, t^{n+1}), & X \in \gamma_{2h}. \end{cases} \tag{12}$$

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

$$\begin{cases} W^{n+1}/\tau = \sum\limits_{\alpha=1}^{3} A_\alpha W^{n+1}, & X \in \omega_h, \\ W^{n+1} = 0, & X \in \gamma_{1h}, \\ W^{n+1} = \mu_2(X, t^{n+1}), & X \in \gamma_{2h}. \end{cases} \tag{13}$$

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

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

A solution of the backward Euler scheme exists if the following condition

$$S_h W^{n+1} \neq 0$$

is satisfied. This requirement is much weaker than the one obtained for the forward Euler method.

The systems of linear equations (12) and (13) are solved by the CG iterative algorithm. We note, that if operators A_α and functions $\mu_2(X), \rho(X)$ do not depend on t, then it is sufficient to solve problem (13) only once.

The serial explicit CG algorithm for solving a system $\mathbf{A}X = F$ can be formulated as follows:

procedure `The serial CG algorithm`
begin
(1) $X^0, \quad n = 0, \quad R^0 = AX^0 - F, \quad P^0 = R^0$
(2) **while** $\left((R^n, R^n) > \varepsilon(R^0, R^0)\right)$
(3) $G^n = AP^n$,
(4) $\tau_{n+1} = \frac{(R^n, R^n)}{(G^n, P^n)}$,
(5) $X^{n+1} = X^n - \tau_{n+1} P^n$,
(6) $R^{n+1} = R^n - \tau_{n+1} G^n$,
(7) $\beta_n = \frac{(R^{n+1}, R^{n+1})}{(R^n, R^n)}$,
(8) $P^{n+1} = R^{n+1} + \beta_n P^n$,
(9) $n := n + 1$.
(10) **end while**
end

Now we will estimate the complexity of the backward Euler algorithm. At each time step we compute:

1. Coefficients of matrix $\mathbf{A}$ for solving systems of linear equations (12) and (13). The complexity of this step is J^3.
2. Solutions of two systems of linear equations $\mathbf{A}X = F$, defined by discrete problems (12) and (13). It is well-known that for large time steps τ the number of iterations of the CG method is bounded by $\mathcal{O}(J)$. For $\tau = \mathcal{O}(J^{-1})$ the number of iterations is close to constant. In our analysis we assume that this number is equal to $\sqrt{J}$. The complexity of one iteration is gJ^3.
3. Two three-dimensional discrete approximations of integrals $S_h(V^{n+1})$ and $S_h(W^{n+1})$. The complexity of this step is $2cJ^3$.
4. The updated boundary condition with a known function μ_0^{n+1}. The complexity of this step is bJ^2.

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

$$W = gJ^{3.5} + (1 + c)J^3 + bJ^2.$$

3.1. *Parallel Algorithm*

In addition to the analysis presented for the parallel forward Euler method we must consider the complexity of parallel implementation of the CG method.

1. At steps (5), (6) and (8) of the CG algorithm *saxpy* type operation is computed. Only local data is used by each processor and no communication among processors is required. The complexity of this part of computations is given by

$$T_{1p} = g_1 \frac{J^3}{p}.$$

2. During matrix-vector multiplication at step (3) processors exchange ghost elements of vector P^n, corresponding to its local boundary grid points. The complexity of this step is given by

$$T_{2p} = g_2 \frac{J^3}{p} + 6\Big(\alpha + \beta \frac{J^2}{p^{2/3}}\Big).$$

3. Parallel computation of the inner product of two vectors at steps (4) and (7) of the CG method requires global communication among all processors during summation of local parts of the product. The complexity of this step is given by

$$T_{3p} = g_3 \frac{J^3}{p} + 2R(p)(\alpha_b + \beta_b).$$

For more results on implementation and analysis of parallel CG methods, see Čiegis (2005c), Duff and van der Vorst (1999) and references given in these papers.

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

$$\begin{aligned} T_p(J) \;=\; & J^{1/2}\Big[g\frac{J^3}{p} + 2R(p)(\alpha_b + \beta_b) + 6\Big(\alpha + \beta\frac{J^2}{p^{2/3}}\Big)\Big] \\ & + (1+c)\frac{J^3}{p} + 6\Big(\alpha + \Big(\beta + \frac{b}{6}\Big)\frac{J^2}{p^{2/3}}\Big) + 2R(p)(\alpha_b + \beta_b). \end{aligned}$$

Comparing $T_p(J)$ with the complexity function of the forward Euler algorithm we get that asymptotic isoefficiency function of the backward Euler algorithm is the same as for the forward Euler algorithm. But since $g < 1$, the relative costs of local and global communications are larger for the backward Euler method. This conclusion is especially important for clusters of workstations, since their processors are connected with much slower network.

3.2. *Results of Computational Experiments*

In this section some results of computational experiments are presented. We compute a solution of the test problem from Section 2. In Table 2 the values of the speedup and efficiency coefficients are presented, when the problem is solved in time intervals $[0, T(J)]$:

$$T(40) = 0.5, \quad T(80) = 0.05, \quad T(120) = 0.01, \quad T(160) = 0.005.$$

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 had run 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:

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

Table 2

The speedup and efficiency coefficients for the backward Euler method. CPU time of the sequential algorithm (in *s*): $T_1(40) = 70.6$, $T_1(80) = 133.1$, $T_1(120) = 133.2$, $T_1(160) = 249.2$

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	1.982	0.991	2.290	1.145	1.984	0.992	2.077	1.038
4	3.831	0.958	4.653	1.163	4.278	1.057	4.155	1.039
8	6.143	0.768	8.461	1.058	9.109	1.139	8.523	1.065
16	10.28	0.642	16.21	1.013	17.94	1.121	19.03	1.189
32	15.70	0.491	30.51	0.953	34.75	1.086	38.16	1.193

4. Locally One Dimensional Method

In this section we propose unconditionally stable LOD scheme, which approximates 3D parabolic problem and the integral condition:

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

Boundary conditions are approximated consistently with the approximation of the differential equations as described in Samarskii (2001).

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 Čiegis (2005a, 2005b), see also Čiegis (2006), where this algorithm is investigated in detail. The complexity of LOD algorithm is equal to $\mathcal{O}(J^3)$.

4.1. *Parallel Algorithm*

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

$$\begin{cases} \frac{V^{n+1}-U^{n+2/3}}{\tau} = A_3 V^{n+1}, \ X \in \omega_h, \\ V^{n+1} = \mu_1(X, t^{n+1}), \qquad X \in \gamma_{1h}, \\ V^{n+1} = \mu_0^n \mu_2(X, t^{n+1}), \quad X \in \gamma_{2h}. \end{cases} \tag{14}$$

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

$$\begin{cases} W^{n+1}/\tau = A_3 W^{n+1}, \ X \in \omega_h, \\ W^{n+1} = 0, \qquad\qquad X \in \gamma_{1h}, \\ W^{n+1} = \mu_2(X, t^{n+1}), \ X \in \gamma_{2h}. \end{cases} \tag{15}$$

Then we find γ^{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 the 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 splitted between $p > 2$ processors, we use the Wang parallel factorization algorithm given in Kumar *et al.* (1994). 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 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 can be estimated as

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

It follows from the scalability analysis that the optimal number of processors is $p = \mathcal{O}(\sqrt{J})$ (here p is the number of processors used in one dimension of the 3D cube).

4.2. *Results of Computational Experiments*

We computed a solution of the test problem from Section 2 in time intervals $[0, T(J)]$:

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

The results are presented in Table 3.

Table 3

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 [-2pt]

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

References

Balay, S., K. Buschelman, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L. Curfman McInnes, B.F. Smith, H. Zhang (2005). *PETSc Users Manual*. ANL-95/11 – Revision 2.3.0, Argonne National Laboratory.

Cannon, J.R. (1963). The solution of the heat equation subject to the specification of energy. *Quart. Appl. Math.*, **21**, 155–163.

Cannon, J.R., Y. Lin, A. L. Matheson (1993). Locally explicit schemes for three-dimensional diffusion with non-local boundary specification. *Appl. Anal.*, **50**, 1–19.

Cannon, J.R. J. van der Hoek (1982). The classical solution of the one-dimensional two-phase Stefan problem with energy specification. *Ann. Math. Pura Appl.*, **130**(4), 385–398.

Čiegis, R. (2006). Parallel LOD scheme for 3D parabolic problem with nonlocal boundary condition. In W.E. Nagel *et al.* (Eds.) *Lecture Notes in Computer Science, International Conference EurPar2006*, Vol. 4128. Springer. pp. 679–688.

Čiegis, R. (2005a). Economical difference schemes for the solution of a two dimensional parabolic problem with an integral condition, *Differential Equations*, **41**(7), 1025–1029.

Čiegis, R. (2005b). 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.

Čiegis, R. (2004). 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*. Birkhauser, Boston. pp. 47–52.

Čiegis, R. (2005c) Analysis of parallel preconditioned conjugate gradient algorithms. *Informatica*, **16**(3), 317–332.

Čiegis, R. (1991). On the difference schemes for problems with non-local boundary conditions. *Informatica*, **2**(2), 155–170.

Čiegis, R., A. Jakušev, A. Krylovas and O. Suboč (2005). Parallel algorithms for solution of nonlinear diffusion problems in image smoothing. *Math. Modelling and Analysis*, **10**(2), 155–172.

Čiegis, R., A. Štikonas, O. Štikonienė, O. Suboč (2002). Monotone finite-difference scheme for parabolical problem with nonlocal boundary conditions. *Differential Equations*, **38**(7), 1027–1037.

Čiegis, R., A. Štikonas, O. Štikonienė, O. Suboč (2001). Stationary problems with nonlocal boundary conditions. *Mathematical Modelling and Analysis*, **6**, 178–191.

Dehghan, M. (2005). Efficient techniques for the second-order parabolic equation subject to nonlocal specifications. *Applied Numer. Math.*, **52**, 39–62.
Dehghan, M. (2002). Locally explicit schemes for three-dimensional diffusion with non-local boundary specification. *Applied Mathematics and Computation*, **135**, 399–412.
Duff, I.S., H.A. van der Vorst (1999). Developments and trends in the parallel solution of linear systems. *Parallel Computing*, **25**, 1931–1970.
Ekolin, G. (1991). Finite difference methods for a nonlocal boundary value problem for the heat equation. *BIT*, **31**(2), 245–255.
Fairweather, G., J.C. Lopez–Marcos (1996). Galerkin methods for a semilinear parabolic problem with nonlocal boundary conditions. *Adv. Comput. Math.*, **6**, 243–262.
Hockney, R. (1991). Performance parameters and benchmarking on supercomputers. *Parallel Computing*, **17**, 1111–1130.
Ionkin, N.I. (1977). Solution of a boundary value problem in heat conduction with a non-classical boundary condition. *Differential Equations*, **13**, 204–211.
Ionkin, N.I. (1980). Stability of a problem in heat transfer problem with a non-classical boundary condition. *Differential Equations*, **16**, 911–914.
Kamynin, L.I. (1964). A boundary value problem in the theory of heat conduction with non-classical boundary condition. *USSR Comput. Math. Math. Phys.*, **4**, 33–59.
Kumar, V., A. Grama, A. Gupta, G. Karypis (1994). *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings, Redwood City.
Langtangen, H.P., and A. Tveito (2003). *Advanced Topics in Computational Partial Differential Equations. Numerical Methods and Diffpack Programming*. Springer, Berlin.
Makarov, V.L., D.T. Kulyev (1985). Solution of a boundary value problem for a quasi-linear parabolic equation with nonclassical boundary conditions. *Differential Equations*, **21**, 296–305.
Noye, B.J., M. Dehghan (1999). New explicit finite difference schemes for two-dimensional diffusion subject specification of mass. *Numer. Meth. for PDE*, **15**, 521–534.
Samarskii, A.A. (2001). *The Theory of Difference Schemes*. Marcel Dekker, Inc., New York–Basel.

R. Čiegis has graduated from Vilnius University Faculty of Mathematics in 1982. He received the PhD degree from the Institute of Mathematics of Byelorussian Academy of Science in 1985 and the degree of habil. doctor of mathematics from the Institute of Mathematics and Informatics, Vilnius in 1993. He is a professor and the head of Mathematical Modeling department of Vilnius Gediminas Technical University. His research interests include numerical methods for solving nonlinear PDE, parallel numerical methods and mathematical modeling in nonlinear optics, porous media flows, technology, image processing, biotechnology.

Trimačio parabolinio uždavinio su nelokaliąja kraštine sąlyga lygiagretieji skaitiniai sprendimo algoritmai

Raimondas ČIEGIS

Papildoma nelokali sąlyga yra naudojama kraštinės sąlygos koeficiento radimui, todėl uždavinys priklauso atvirkštinių matematikos uždavinių klasei. Nagrinėjame tris duotojo uždavinio sprendimo algoritmus – išreikštinį ir neišreikštinį Eulerio, bei lokaliai vienmatį algoritmus. Visi algoritmai pritaikyti uždavinio su nelokalia kraštine sąlyga sprendimui. Lygiagrečiuosius algoritmus realizuojame naudodami *ParSol* lygiagrečiųjų masyvų biblioteką, leidžiančią beveik automatiškai gauti lygiagrečią programos versiją, kai nuoseklusis algoritmas realizuotas remiantis šiais masyvais. Pateikta algoritmų efektyvumo analizė, teoriniai rezultatai iliustruojami skaičiavimo eksperimento rezultatais.