Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 610398, 12 pages
http://dx.doi.org/10.1155/2014/610398

Research Article

A Computational Realization of a Semi-Lagrangian Method for Solving the Advection Equation

Alexander Efremov,[1] Eugeniya Karepova,[1,2] Vladimir Shaydurov,[1,3] and Alexander Vyatkin[1,2]

[1] *Institute of Computational Modeling SB RAS, Krasnoyarsk, Akademgorodok 660036, Russia*
[2] *Siberian Federal University, Svobodny Prospect, Krasnoyarsk 660041, Russia*
[3] *Beihang University, Haidian District, Beijing 100191, China*

Correspondence should be addressed to Eugeniya Karepova; e.d.karepova@icm.krasn.ru

Received 3 April 2014; Revised 16 August 2014; Accepted 19 August 2014; Published 27 October 2014

Academic Editor: Xiaohui Yuan

A parallel implementation of a method of the semi-Lagrangian type for the advection equation on a hybrid architecture computation system is discussed. The difference scheme with variable stencil is constructed on the base of an integral equality between the neighboring time levels. The proposed approach allows one to avoid the Courant-Friedrichs-Lewy restriction on the relation between time step and mesh size. The theoretical results are confirmed by numerical experiments. Performance of a sequential algorithm and several parallel implementations with the OpenMP and CUDA technologies in the C language has been studied.

1. Introduction

Many physical phenomena in transport processes are modeled by time-dependent hyperbolic conservation laws [1–4]. The finite volume method (FVM) is a standard conservative method to construct numerical approximations for solving hyperbolic conservation problems. Modern modifications of FVM [5–8] provide well-established conservative methods for solving the governing advection equations. Moreover, some of them were developed to treat high gradients and discontinuities of a solution [7, 8]. In spite of their advances for hyperbolic equations, these methods have the limitation consisting in a time step restriction, mainly for stability sake. On the other hand, during the last three decades the idea of applying the method of characteristics to advective quantities forward in time has rapidly developed and has gained popularity in many areas [9–13]. In contrast to traditional Eulerian schemes, semi-Lagrangian algorithms provide unconditional stability and allow using large time steps. Despite unconditional stability, these methods are explicit and therefore they look well suited for parallelization. Now semi-Lagrangian methods are intensively studied and their efficiency for convection-dominated problems is proved. For a more detailed discussion about the comparison of traditional Eulerian and semi-Lagrangian schemes for hyperbolic conservation laws, see [6, 14, 15].

Initially semi-Lagrangian algorithms, as methods of characteristics, were developed with application in climate prediction [16–21]. The simplest schemes use the approximation of a trajectory (or curvilinear characteristic) by a straight line and employ a low-order interpolation to compute a numerical solution. Nowadays, simplicity and efficiency of these schemes make them quite popular in different fields of numerical modeling like fluid dynamics applications [9, 12, 22], shallow water equations [10], fiber dynamics described by the Fokker-Planck equation [11], heat-conduction equation [23], and so forth. Now modern semi-Lagrangian algorithms involve a higher-order approximation of a curvilinear characteristic and employ a higher-order interpolation; see, for example, [22]. Recently, considerable efforts have been made to construct the conservative semi-Lagrangian methods [9, 20, 24–28]. For instance, Scroggs and Semazzi [9] presented a semi-Lagrangian finite volume

method that used a rectangular grid for a system of conservation laws and satisfied the discrete conservation relation, but the numerical results demonstrate some violation of full conservation. Early modifications of the semi-Lagrangian approach use a rectangular grid which is fixed throughout the simulation [9, 17, 24, 25]. Semi-Lagrangian schemes allow using spatial grids independently of one another. As a result, adaptive grids are widely used in modern versions of this approach [20, 28]. In spite of the progress in semi-Lagrangian methods, for most of them [9–12, 20, 24, 28] convergence has not been theoretically proved.

In this paper, we present a sketch of the theoretical proof and a numerical justification for the difference scheme of the semi-Lagrangian family. We start with the theorem about an exact equality that involves two spatial integrals over domains at neighboring time levels and the third integral over an inflow boundary. To prove convergence theoretically, we use a square grid, the bilinear interpolation, and the Runge-Kutta method for the fourth-order approximation of characteristics. This allows us to prove first-order convergence in a discrete analogue of the L_1-norm. The theoretical convergence estimates are confirmed by numerical results.

In the remaining part of the paper, some parallel implementations of this method are studied. We discuss the design subtleties of parallel implementations of the algorithm by the OpenMP-technology for shared memory computational systems and by the CUDA technology for general-purpose GPU programming. In addition, the influence of the Hyper-Threading technology on the performance of our OpenMP-code is studied. Moreover, the difficulties of the algorithm implementation and the performance for hybrid architecture computation systems are discussed for our CUDA codes.

2. Formulation of the Problem

Let $D = [0,1]\times[0,1]$. In the closed domain $[0,T]\times D$ consider the two-dimensional advection equation

$$\frac{\partial \rho}{\partial t} + \frac{\partial (u\rho)}{\partial x} + \frac{\partial (v\rho)}{\partial y} = 0, \tag{1}$$

where $u(t,x,y)$ and $v(t,x,y)$ are known and are sufficiently smooth in $[0,T]\times D$. We suppose for simplicity that $\forall t \in [0,T]$ the coefficients satisfy the no-slip conditions at the upper and lower sides of D

$$\begin{aligned} u(t,x,y)\big|_{y=0} &= u(t,x,y)\big|_{y=1} = 0, \\ v(t,x,y)\big|_{y=0} &= v(t,x,y)\big|_{y=1} = 0 \end{aligned} \tag{2}$$

and the flow conditions at the left and right sides of D

$$u(t,x,y)\big|_{x=0} \geq 0, \qquad u(t,x,y)\big|_{x=1} \geq 0. \tag{3}$$

For the unknown function $\rho(t,x,y)$ the following initial and boundary conditions are specified:

$$\forall (x,y) \in D \quad \rho(0,x,y) = \rho_{\text{init}}(x,y), \tag{4}$$

$$\forall (t,y) \in [0,T]\times[0,1] \quad \rho(t,0,y) = \rho_{\text{in}}(t,y). \tag{5}$$

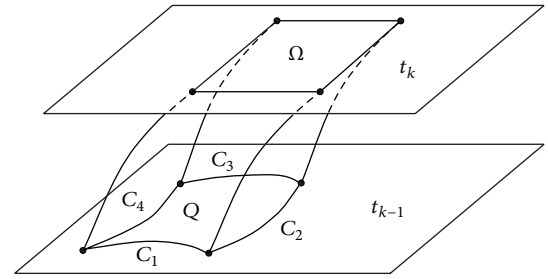

Figure 1: Curvilinear quadrangle Q.

3. Numerical Scheme

Subdivide the time segment $[0,T]$ into K time levels $t_k = k\tau$, $k = 0,\ldots,K$, with the time step $\tau = T/K$. Let Ω be a closed quadrangle at the time level t_k. For each of its points on the segment $t \in [t_{k-1}, t_k]$ we construct the characteristics defined by the system of ordinary differential equations

$$\begin{aligned} \overline{x'}(t) &= u\left(t, \overline{x}(t), \overline{y}(t)\right), \\ \overline{y'}(t) &= v\left(t, \overline{x}(t), \overline{y}(t)\right), \\ & t \in [t_{k-1}, t_k], \end{aligned} \tag{6}$$

with the initial value at level $t = t_k$ as a parameter:

$$\overline{x}(t_k) = x_0, \qquad \overline{y}(t_k) = y_0, \quad (x_0, y_0) \in \Omega. \tag{7}$$

With the help of these characteristics the edges of Ω generate four surfaces S_n, $n = 1,\ldots,4$, with the edges C_n at $t = t_{k-1}$ (Figure 1).

If Ω is located near the inflow boundary $x = 0$, surfaces S_n can cross the plane $x = 0$. In this case we get an additional curvilinear polygon I on the plane $x = 0$ (Figure 2).

Generally speaking, I and Q can be triangular, pentagonal, or empty domains. If one of them is empty, then the integral over an empty domain is supposed to be equal to zero. Since there is no fundamental difference, we consider only the most common case with quadrangular domains. For Ω, Q, and I the following statement is valid.

Theorem 1. *For a smooth solution of the problem* (1)–(5) *we have the equality*

$$\begin{aligned} &\int_\Omega \rho(t_k,x,y)\, d\Omega \\ &\quad = \int_Q \rho(t_{k-1},x,y)\, dQ + \int_I (\rho u)(t,0,y)\, dI. \end{aligned} \tag{8}$$

Proof. Denote the volume bounded by Ω, Q, I, and surfaces S_n by V and its boundary by Γ. Apply the Gauss-Ostrogradsky theorem to the left-hand side of the equality

$$\int_V \left(\frac{\partial \rho}{\partial t} + \frac{\partial (u\rho)}{\partial x} + \frac{\partial (v\rho)}{\partial y} \right) dV = 0. \tag{9}$$

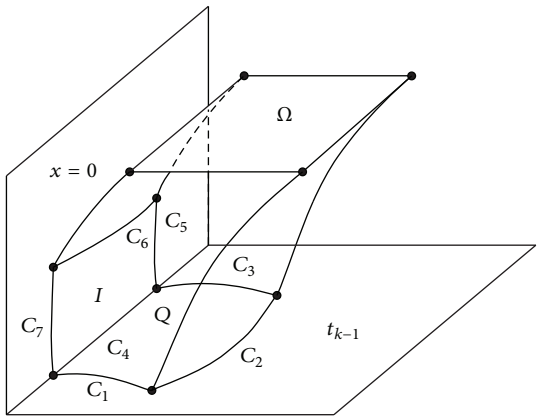

FIGURE 2: Appearance of the boundary quadrangle I.

Then

$$\int_V \left(\frac{\partial \rho}{\partial t} + \frac{\partial (u\rho)}{\partial x} + \frac{\partial (v\rho)}{\partial y} \right) dV = \int_\Gamma (\rho, \rho u, \rho v) \cdot \left(n_t, n_x, n_y\right)^T d\Gamma = \int_\Gamma \rho\,(1, u, v) \cdot \left(n_t, n_x, n_y\right)^T d\Gamma = 0. \tag{10}$$

Here (n_t, n_x, n_y) is the outer normal to Γ and sign "$\cdot$" means the scalar product. The normal (n_t, n_x, n_y) equals $(1, 0, 0)$ on Ω, $(-1, 0, 0)$ on Q, and $(0, -1, 0)$ on I. For any S_n the normal (n_t, n_x, n_y) is orthogonal to all tangent directions of S_n including the tangent of characteristics $(1 + u^2 + v^2)^{-1/2}(1, u, v)$. Therefore $(1, u, v) \cdot (n_t, n_x, n_y)^T = 0$. Taking into account this reasoning in (10), we get the equality

$$\int_\Omega \rho\,(t_k, x, y)\, d\Omega - \int_Q \rho\,(t_{k-1}, x, y)\, dQ - \int_I (\rho u)\,(t, 0, y)\, dI = 0 \tag{11}$$

that is equivalent to the statement of theorem. □

Now construct the uniform mesh D_h with mesh-size $h = 1/N$, $N \geq 2$:

$$D_h = \left\{ \left(x_i, y_j\right) : x_i = ih,\ y_j = jh;\ i, j = 0, \ldots, N \right\}. \tag{12}$$

We will find an approximate solution $\rho^h(t, x, y)$ at each time level $t = t_r$ $\forall r = 0, \ldots, K$ as a grid function with values

$$\widetilde{\rho}^r_{i,j} = \rho^h\left(t_r, x_i, y_j\right) \quad \forall i, j = 0, \ldots, N, \tag{13}$$

unlike the values of the exact solution

$$\rho^r_{i,j} = \rho\left(t_r, x_i, y_j\right). \tag{14}$$

To construct the difference scheme with a variable stencil, we suppose that the function ρ^h at time level t_{k-1} is already known and we need to find it at level t_k. To compute $\widetilde{\rho}^k_{i,j}$ for some $i, j = 1, 2, \ldots, N-1$, we take the square $\Omega_{i,j}$ with four vertices $(x_i \pm h/2, y_j \pm h/2)$ and apply Theorem 1. To determine $\widetilde{\rho}^k_{i,j}$ on the boundary of D we use the rectangles $\Omega_{i,j}$ which are adjoined to this boundary inside D (Figure 3).

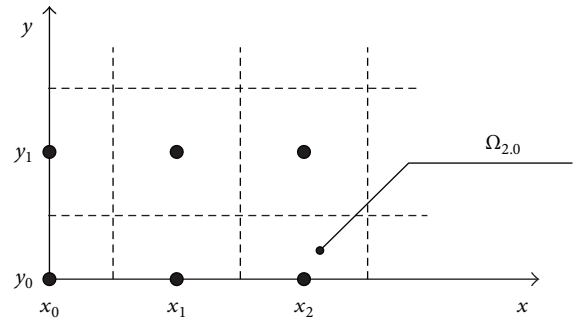

FIGURE 3: Boundary rectangles.

Note that $\widetilde{\rho}^k_{0,j} = \rho^k_{0,j}$ are known from the boundary condition (5).

Without loss of generality we describe the construction of the difference equations for inner nodes with $i, j = 1, 2, \ldots, N-1$ only. Thus, due to Theorem 1 we get

$$\int_{\Omega_{i,j}} \rho\,(t_k, x, y)\, d\Omega = \int_{Q^{k-1}_{i,j}} \rho\,(t_{k-1}, x, y)\, dQ + \int_{I^{k-1}_{i,j}} (\rho u)\,(t, 0, y)\, dI. \tag{15}$$

Here the curvilinear polygons $Q^{k-1}_{i,j}$ and $I^{k-1}_{i,j}$ are formed by the characteristics (6) that issue out of the edges of square $\Omega_{i,j}$. To compute the second integrals in (15) numerically, first we replace the exact function $\rho(t_{k-1}, x, y)$ by its bilinear interpolant

$$\rho^I_h\,(t_{k-1}, x, y) = \sum_{q=0}^{N} \sum_{p=0}^{N} \rho^{k-1}_{p,q} \varphi_{p,q}\,(x, y) \tag{16}$$

with the help of the basis functions

$$\varphi_{p,q}\,(x) = \begin{cases} \left(\dfrac{1 - \left|x_p - x\right|}{h} \right) \left(\dfrac{1 - \left|y_q - y\right|}{h} \right) \\ \qquad \forall\,(x, y) \in \left[x_{p-1}, x_{p+1}\right] \times \left[y_{q-1}, y_{q+1}\right], \\ 0 \quad \text{otherwise}, \end{cases} \qquad \forall p, q = 0, \ldots, N. \tag{17}$$

To compute the integral over the domain $I^{k-1}_{i,j}$, we also use the bilinear interpolant

$$(\rho u)^I_\tau\,(t, y) = \sum_{q=0}^{N} \sum_{r=0}^{K} \rho_{\text{in}}\left(t_r, y_q\right) u\left(t_r, 0, y_q\right) \psi_{r,q}\,(t, y) \tag{18}$$

with the basis functions

$$\psi_{r,q}(x) = \begin{cases} \left(\frac{1-|t_r - t|}{\tau}\right)\left(\frac{1-|y_q - y|}{h}\right) \\ \qquad \forall (t,y) \in [t_{r-1}, t_{r+1}] \times [y_{q-1}, y_{q+1}], \\ 0 \quad \text{otherwise}, \end{cases}$$
$$\forall r = 0, \ldots, K,\ q = 0, \ldots, N. \tag{19}$$

The left-hand side of (15) is approximated by the midpoint quadrature rule with second-order accuracy:

$$\int_{\Omega_{i,j}} \rho(t_k, x, y)\, d\Omega \approx \operatorname{mes}(\Omega_{i,j})\, \rho_{i,j}^k \qquad \forall i, j = 0, 1, \ldots, N. \tag{20}$$

So, instead of the exact equality (15) we get the approximate one:

$$\operatorname{mes}(\Omega_{i,j})\, \rho_{i,j}^k \approx \int_{Q_{i,j}^{k-1}} \rho_h^I(t_{k-1}, x, y)\, dQ + \int_{I_{i,j}^{k-1}} (\rho u)_\tau^I(t, 0, y)\, dI. \tag{21}$$

To simplify the numerical computation of the right-hand side in (21), we approximate the domains $Q_{i,j}^{k-1}$ and $I_{i,j}^{k-1}$ by simpler ones. Since in the more general case both domains are curvilinear quadrangles, we demonstrate the approximation only for quadrangular $Q_{i,j}^{k-1}$. Introduce four additional points $(x_i \pm h/2, y_j)$ and $(x_i, y_j \pm h/2)$ on the square $\Omega_{i,j}$ at time level t_k and denote each of the eight nodes by $A_n = (\widehat{x}_n, \widehat{y}_n)$, $n = 1, \ldots, 8$. Out of each A_n we pass the corresponding characteristic to the time level t_{k-1} which gives the point $B_n = (\overline{x}_n, \overline{y}_n)$ (Figure 4).

To compute the coordinates of the point B_n numerically, we solve the system of ordinary differential equations (6) with the initial condition

$$\widetilde{x}(t_k) = x_n, \qquad \widetilde{y}(t_k) = y_n, \tag{22}$$

by the fourth-order Runge-Kutta method [29]. Thus, we find the approximation $B_n^h = (\widetilde{x}_n(t_{k-1}), \widetilde{y}_n(t_{k-1}))$ of the point B_n. The nodes B_n^h, $n = 1, \ldots, 8$, define the polygon $P_{i,j}^{k-1}$ which is considered as a quadrangle with four parabolic edges (Figures 4-5). The constructed domain $P_{i,j}^{k-1}$ approximates $Q_{i,j}^{k-1}$. In the same way we construct the polygon $L_{i,j}^{k-1}$ which approximates $I_{i,j}^{k-1}$. For the above approximation the following statement is valid [30].

Lemma 2. *Let the coordinates of the nodes B_n, $n = 1, \ldots, 8$, be computed within the fourth-order accuracy $\overline{x}_n - \widetilde{x}_n(t_k) = O(\tau^4)$, $\overline{y}_n - \widetilde{y}_n(t) = O(\tau^4)$. Assume that the ratio between τ and h is fixed: $\tau = \widetilde{c}h$. Then for all $i, j = 0, \ldots, N$*

$$\operatorname{mes}\left(Q_{i,j}^{k-1} \setminus P_{i,j}^{k-1}\right) + \operatorname{mes}\left(P_{i,j}^{k-1} \setminus Q_{i,j}^{k-1}\right) = O\left(h^4\right),$$
$$\operatorname{mes}\left(I_{i,j}^{k-1} \setminus L_{i,j}^{k-1}\right) + \operatorname{mes}\left(L_{i,j}^{k-1} \setminus I_{i,j}^{k-1}\right) = O\left(h^4\right), \tag{23}$$

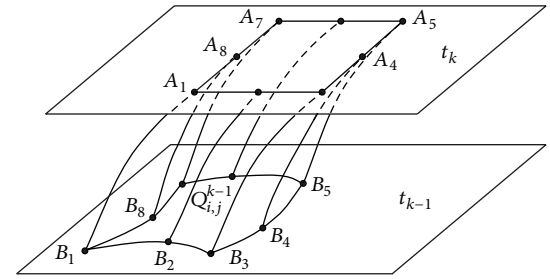

Figure 4: Approximation of a curvilinear quadrangle.

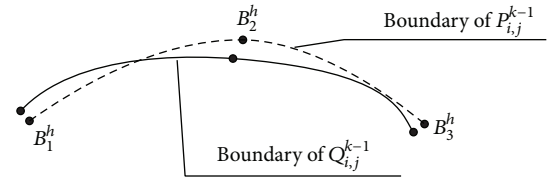

Figure 5: Approximation of nodes and edges.

where the notation $\operatorname{mes}(\Omega)$ *means the measure of the domain* Ω.

Thus, the replacement of $Q_{i,j}^{k-1}$ by $P_{i,j}^{k-1}$ and $I_{i,j}^{k-1}$ by $L_{i,j}^{k-1}$, $i = 1, \ldots, N$, $j = 0, \ldots, N$, reduces the approximate equality (21) to another one:

$$\operatorname{mes}(\Omega_{i,j})\, \rho_{i,j}^k \approx \int_{P_{i,j}^{k-1}} \rho_h^I(t_{k-1}, x, y)\, dP + \int_{L_{i,j}^{k-1}} (\rho u)_\tau^I(t, 0, y)\, dL. \tag{24}$$

Divide it by $\operatorname{mes}(\Omega_{i,j})$ and replace ρ_h^I by the interpolant $\widetilde{\rho}_h^I$ of the known grid function ρ^h at the level t_{k-1}:

$$\widetilde{\rho}_h^I(t_{k-1}, x, y) = \sum_{q=0}^{N} \sum_{p=0}^{N} \widetilde{\rho}_{p,q}^{k-1} \varphi_{p,q}(x, y). \tag{25}$$

As a result, we get the equation for finding $\widetilde{\rho}_{i,j}^k$ as an approximation of $\rho_{i,j}^k$:

$$\widetilde{\rho}_{i,j}^k = \frac{1}{\operatorname{mes}(\Omega_{i,j})} \int_{P_{i,j}^{k-1}} \widetilde{\rho}_h^I(t_{k-1}, x, y)\, dP + \frac{1}{\operatorname{mes}(\Omega_{i,j})} \int_{L_{i,j}^{k-1}} (\rho u)_\tau^I(t, 0, y)\, dL \qquad \forall i = 1, \ldots, N,\ j = 0, 1, \ldots, N, \tag{26}$$

$$\widetilde{\rho}_{0,j}^k = \rho_{\text{in}}(t_k, 0, y_j) \quad \forall j = 0, \ldots, N. \tag{27}$$

To compute the integrals numerically, we decompose the domain $P_{i,j}^{k-1}$ (or $L_{i,j}^{k-1}$) into several triangles which lie only in one cell $[x_{p-1}, x_p] \times [y_q, y_{q+1}]$, $p, q = 0, 1, \ldots, N-1$, and have only one parabolic edge and two straight-line edges parallel

to the coordinate axes. Then we replace the integral over the domain $P_{i,j}^{k-1}$ (or $L_{i,j}^{k-1}$) by a sum of integrals. Thus we compute integrals directly without any quadrature rule.

To evaluate the order of convergence, we use the discrete analogue of the $L_1(D)$-norm:

$$\left\|\rho^h\right\|_{L_1}^h = \sum_{i,j=0}^{N} \left|\rho_{i,j}^h\right| \operatorname{mes}\left(\Omega_{i,j}\right). \tag{28}$$

For the numerical solution computed by (26) the following theorem is valid.

Theorem 3. *Let the solution $\rho(t,x,y)$ of the problem* (1)–(5) *be sufficiently smooth and let the discrete solution ρ^h be computed by* (26). *Assume that $\tau = \tilde{c}h$. Then we have the following estimate: $\forall k = 0, 1, \ldots, K$*

$$\left\|\rho(t_k,\cdot) - \rho^h(t_k,\cdot)\right\|_{L_1}^h \le k\left(c_1h^2 + c_2\tilde{c}^2h^3\right), \tag{29}$$

with the constants c_1 and c_2 independent of k, h, τ, and $\tilde{c}$.

Proof. Use the induction on k. For $k = 0$ inequality (29) is valid because of the exact initial condition (4). Suppose the estimate (29) is valid for some $k - 1 \ge 0$ and prove it for k.

From Theorem 1 $\forall i = 1, \ldots, N$, $j = 0, \ldots, N$ we get

$$\begin{aligned}\int_{\Omega_{i,j}} \rho\left(t_k,x,y\right) d\Omega = &\int_{Q_{i,j}^{k-1}} \rho\left(t_{k-1},x,y\right) dQ \\ &+ \int_{I_{i,j}^{k-1}} \rho\left(t,0,y\right) u\left(t,0,y\right) dI.\end{aligned} \tag{30}$$

For the first integral we have the equality

$$\begin{aligned}&\rho\left(t_k,x_i,y_j\right) \operatorname{mes}\left(\Omega_{i,j}\right) \\ &\quad = \int_{\Omega_{i,j}} \rho\left(t_k,x,y\right) d\Omega + \delta_{i,j}^k \operatorname{mes}\left(\Omega_{i,j}\right),\end{aligned} \tag{31}$$

where

$$\left|\delta_{i,j}^k\right| \le \begin{cases} \tilde{c}_1h^2 & \forall i = 1, \ldots, N-1, \\ \tilde{c}_2h & \text{when } i = N. \end{cases} \tag{32}$$

In the second and third integrals we replace the polygons $Q_{i,j}^{k-1}$ and $I_{i,j}^{k-1}$ by $P_{i,j}^{k-1}$ and $L_{i,j}^{k-1}$, respectively, according to the foregoing approximation. Then we replace $\rho(t_{k-1},x,y)$ and $\rho(t,0,y)u(t,0,y)$ by their piecewise bilinear interpolants. Due to Lemma 2 and boundedness of the functions $\rho(t,x,y)$ and $u(t,x,y)$, formulae (30)-(31) are represented in the following way:

$$\begin{aligned}\rho_{i,j}^k \operatorname{mes}\left(\Omega_{i,j}\right) = &\int_{P_{i,j}^{k-1}} \rho_h^I\left(t_{k-1},x,y\right) dP \\ &+ \int_{L_{i,j}^{k-1}} \left(\rho u\right)_\tau^I\left(t,0,y\right) dL + \delta_{i,j}^k \operatorname{mes}\left(\Omega_{i,j}\right) \\ &+ \gamma_{i,j}^{k-1} + \eta_{i,j}^{k-1} \operatorname{mes}\left(P_{i,j}^{k-1}\right) + \theta_{i,j}^{k-1} \operatorname{mes}\left(L_{i,j}^{k-1}\right),\end{aligned} \tag{33}$$

where $\left|\gamma_{i,j}^{k-1}\right| \le \tilde{c}_3h^4$, $\left|\eta_{i,j}^{k-1}\right| \le \tilde{c}_4h^2$, $\left|\theta_{i,j}^{k-1}\right| \le \tilde{c}_5\tau h$. Now multiply (26) by $\operatorname{mes}(\Omega_{i,j})$ and subtract it from (33). Then we have

$$\begin{aligned}&\left(\rho_{i,j}^k - \tilde{\rho}_{i,j}^k\right) \operatorname{mes}\left(\Omega_{i,j}\right) \\ &\quad = \int_{P_{i,j}^{k-1}} \sum_{p,q=0}^{N} \left(\rho_{p,q}^{k-1} - \tilde{\rho}_{p,q}^{k-1}\right) \psi_{p,q}\left(x,y\right) dP \\ &\qquad + \delta_{i,j}^k \operatorname{mes}\left(\Omega_{i,j}\right) + \gamma_{i,j}^{k-1} + \eta_{i,j}^{k-1} \operatorname{mes}\left(P_{i,j}^{k-1}\right) \\ &\qquad + \theta_{i,j}^{k-1} \operatorname{mes}\left(L_{i,j}^{k-1}\right).\end{aligned} \tag{34}$$

Now, let us sum up the modulus of the last equation for all $i = 1, \ldots, N$, $j = 0, \ldots, N$ and use the decomposition

$$\tilde{\rho}_{i,j}^{k-1} = \rho_{i,j}^{k-1} + \xi_{i,j}^{k-1} \tag{35}$$

at level t_{k-1} with a grid function $\xi^{k-1}(x,y)$ satisfying the estimate

$$\left\|\xi^{k-1}\right\|_{L_1}^h \le (k-1)\left(c_1h^2 + c_2\tilde{c}^2h^3\right), \tag{36}$$

due to the induction hypothesis. Then we get

$$\begin{aligned}\left\|\xi^k\right\|_{L_1}^h \le &\sum_{i,j=0}^{N} \left(\int_{P_{i,j}^{k-1}} \sum_{p,q=0}^{N} \left|\xi_{p,q}^{k-1}\right| \psi_{p,q}\left(x,y\right) dP\right) \\ &+ \sum_{i,j=0}^{N} \left|\delta_{i,j}^k\right| \operatorname{mes}\left(\Omega_{i,j}\right) + \left|\gamma_{i,j}^{k-1}\right| \\ &+ \sum_{i,j=0}^{N} \left(\left|\eta_{i,j}^{k-1}\right| \operatorname{mes}\left(P_{i,j}^{k-1}\right) + \left|\theta_{i,j}^{k-1}\right| \operatorname{mes}\left(L_{i,j}^{k-1}\right)\right).\end{aligned} \tag{37}$$

Since

$$\begin{aligned}&\sum_{i,j=0}^{N} \left|\delta_{i,j}^k\right| \operatorname{mes}\left(\Omega_{i,j}\right) \le \left(\tilde{c}_1 + 2\tilde{c}_2\right)h^2, \\ &\sum_{i,j=0}^{N} \operatorname{mes}\left(P_{i,j}^{k-1}\right) \le 1, \qquad \sum_{i,j=0}^{N} \operatorname{mes}\left(L_{i,j}^{k-1}\right) = \tau,\end{aligned} \tag{38}$$

we have

$$\begin{aligned}\left\|\xi^k\right\|_{L_1}^h \le &\sum_{i,j=0}^{N} \left(\int_{P_{i,j}^{k-1}} \sum_{p,q=0}^{N} \left|\xi_{p,q}^{k-1}\right| \psi_{p,q}\left(x,y\right) dP\right) \\ &+ \left(\tilde{c}_1 + 2\tilde{c}_2 + \tilde{c}_3 + \tilde{c}_4\right)h^2 + \tilde{c}_5h\tau^2.\end{aligned} \tag{39}$$

Finally consider the transformations

$$\sum_{i,j=0}^{N}\left(\int_{P_{i,j}^{k-1}}\sum_{p,q=0}^{N}\left|\xi_{p,q}^{k-1}\right|\psi_{p,q}(x,y)\,dP\right)$$
$$=\sum_{p,q=0}^{N}\left(\sum_{i,j=0}^{N}\int_{P_{i,j}^{k-1}}\left|\xi_{p,q}^{k-1}\right|\psi_{p,q}(x,y)\,dP\right) \quad (40)$$
$$\le\sum_{p,q=0}^{N}\left(\left|\xi_{p,q}^{k-1}\right|\int_{D}\psi_{p,q}(x,y)\,dD\right).$$

It leads to the inequality

$$\left\|\xi^{k}\right\|_{L_1}^{h}\le\sum_{p,q=0}^{N}\left|\xi_{p,q}^{k-1}\right|\operatorname{mes}\left(\Omega_{p,q}\right)+\left(\widetilde{c}_1+2\widetilde{c}_2+\widetilde{c}_3+\widetilde{c}_4\right)h^2+\widetilde{c}_5h\tau^2. \quad (41)$$

Denote $c_1=(\widetilde{c}_1+2\widetilde{c}_2+\widetilde{c}_3+\widetilde{c}_4)$ and $c_2=\widetilde{c}_5$. Thus due to the relation $\tau=\widetilde{c}\,h$ we get inequality (29). □

Corollary 4. *Let the conditions of Theorem 3 be valid. Then for $t_k=T$ one has the estimate*

$$\left\|\rho(T,\cdot)-\rho^{h}(T,\cdot)\right\|_{L_1}^{h}\le T\left(\frac{c_1h}{\widetilde{c}}+c_2\widetilde{c}h^2\right). \quad (42)$$

Remark 5. Let the functions $\rho_{\text{init}}(x,y)$ and $\rho_{\text{in}}(t,y)$ be greater than zero in the initial and boundary conditions (4)-(5). Then the interpolants $\widetilde{\rho}_h^I(t_0,x,y)$ and $(\rho u)_\tau^I(t,0,y)$ due to (3) are nonnegative. The integration of them results in nonnegative values in (26). Thus, by induction we can prove the inequality

$$\rho^{h}\left(t_r,x_i,y_j\right)\ge 0\quad\forall r=1,\dots,K,\ i,j=0,\dots,N. \quad (43)$$

Remark 6. The strategy of the domain approximation with 8 nodes is not optimal, of course. In fact it is enough to use 4 nodes for rectangles. But a theoretical justification becomes much more complicated. We did not demonstrate it here since the main purpose of the paper is to verify parallel properties of the proposed algorithm. Of course, such an optimization reduces the number of arithmetical operations but has no influence on the parallel properties of the algorithm. The same applies to the difference schemes of higher order.

4. The Numerical Algorithm and Its Parallel Implementations

The constructed algorithm is implemented in the following way.

Algorithm 7 (sequential).

(1) Set $\rho^h(0,x_i,y_j)=\rho_{\text{init}}(x_i,y_j)$, $i,j=0,\dots,N$, as the initial data (4).

(2) *Time loop*: for each time step $k=1,\dots,K$ do:

Space loop: for each cell $\Omega_{i,j}$, $i=1,\dots,N$, $j=0,\dots,N$ do:

(2.1) For each node $A_n=(\widehat{x}_n,\widehat{y}_n)$, $n=1,\dots,4$, solve the system (18)–(20) and determine the corresponding vertex coordinates $B_n^h=(\widetilde{x}_n(t_{k-1}),\widetilde{y}_n(t_{k-1}))$ of a polygon $P_{i,j}^{k-1}$.

(2.2) If a certain characteristic $A_nB_n^h$ intersects the plane $x=0$ then the coordinates of this cross-point are determined.

(2.3) Compute

$$J_{i,j}^{k-1}=\int_{P_{i,j}^{k-1}}\rho_h^I(t_{k-1},x,y)\,dP+\int_{L_{i,j}^{k-1}}(\rho u)^I(t,0,y)\,dL, \quad (44)$$

according to (26) where the integrals are calculated over each nonempty intersection $P_{i,j}^{k-1}\cap\{[x_p,x_{p+1}]\times[y_q,y_{q+1}]\}$ and $L_{i,j}^{k-1}\cap\{[t_{k-1},t_k]\times[y_q,y_{q+1}]\}$ separately.

(2.4) Compute $\rho^h(t_k,x_i,y_j)=J_{i,j}^{k-1}/\text{mes}(\Omega_{i,j})$.

The end of the space loop.

(2.5) Put $\rho^h(t_k,0,y_j)=\rho_{\text{in}}(t_k,y_j)$ for all $j=0,\dots,N$.

(2.6) If necessary, calculate the norms of a solution, an error, and other statistic data with respect to an actual time step.

The end of the time loop.

Note that items (2.1)–(2.3) are compute-intensive, especially the procedure of determining the mutual arrangement of $P_{i,j}^{k-1}$ and the cells $\{[x_p,x_{p+1}]\times[y_q,y_{q+1}]\}_{p,q=0}^{N-1}$ at the previous time level in item (2.3).

The algorithm is explicit with respect to time, since to calculate $\rho^h(t_k,x,y)$ at each time level t_k the data are used only from the previous time level t_{k-1}.

Another advantage of Algorithm 7 is data independence in the general space loop; that is, the items (2.1)–(2.4) are carried out for any pair (i,j), $i=1,\dots,N$, $j=0,\dots,N$, independently. In this connection the data parallelism is used.

In the shared memory case for the OpenMP-technology it is sufficient to parallelize the general space loop at each time level using an OpenMP directive like the following one:

```
#pragma omp parallel for collapse (2)
...
```

In order for paralleling to be correct, the data-sharing attributes of all variables to intermediate outcomes of items (2.1)–(2.4) have to be private for each thread.

Another justified approach to paralleling the algorithm is the usage of the NVIDIA CUDA technology for GPU.

The main aspects of the parallel implementation related to various features of general-purpose GPU programming are briefly discussed below.

All functions used in the numerical calculations on a CPU must be recompiled for a GPU. If we use NVIDIA CUDA these functions must be declared with the special qualifier _host_ _device_ which indicates that the NVCC compiler creates two versions of its executable code for a CPU (host) and for a GPU (device) separately. GPU will call a device version of a function while CPU will call its host version.

The principles of efficient CUDA programming are as follows: (1) the maximal use of inherent parallelism of the problem and (2) the optimization of memory access.

The first version of our parallel CUDA-algorithm is based on the inherent parallelism of our numerical explicit approach. Every thread treats only one cell $\Omega_{i,j} \in D_h$; hence, the space loop body (items (2.1)–(2.4) of Algorithm 7) is the general computation kernel.

While programming for GPU, the correct defining of the kernel configuration is important. The kernel configuration includes two parameters, namely, the number of blocks (blockCount) in a grid and the number of threads (blockSize) per block. There is a limit in 1024 threads per block for our GPU NVIDIA hardware. In the first CUDA-algorithm no threads amount optimization is used.

Consequently, the simplest parallel version of Algorithm 7 for the CPU/GPU hybrid architecture is the following.

Algorithm 8 (CUDA parallel, version 1).

(1) Calculate the kernel configuration (blockSize, blockCount) using data about a computational domain.

(2) Allocate host (CPU) and device (GPU) memory; copy initial data from host to device.

(3) *Time loop*: for each time step $k = 1, \ldots, K$ do:

  (3.1) Call the first CUDA kernel (basic):

    (3.1.1) For each cell $\Omega_{i,j}$, $i = 1, \ldots, N$, $j = 0, \ldots, N$, execute items (2.1)–(2.4) of Algorithm 7 in parallel.

  (3.2) *Synch point*: wait for calculations to be completed.

  (3.3) Call the second CUDA kernel (assistive):

    (3.3.1) Copy data from an actual time level array to the previous one in parallel.

  (3.4) *Synch point*: wait for copying to be completed.

  (3.5) If necessary, copy results from device to host.

  (3.6) If necessary, calculate the norms of a solution, an error, and other statistic data with respect to an actual time step.

  The end of the time loop.

(4) Copy the results from device to host.

In order to decrease the execution time of Algorithm 8, items (3.5)-(3.6) must be performed as rare as possible, for instance, only at time levels where accuracy control, data for drawing, and so forth are needed.

Algorithm 8 has two general disadvantages: (1) small speedup in comparison to the sequential version (Figure 8) and (2) impossibility of execution with a fine computational mesh (Table 2). What is the matter with these problems?

First, the general loop has a lot of selection statements.

The main selection is between two different ways of catching in item (3.1.1) of Algorithm 8 (to be more exact, in the item (2.3) of sequential Algorithm 7). The cells whose trajectories intersect the boundary and the internal cells are processed in different ways. We can use two different kernels which execute in parallel.

We use data parallelism only in Algorithm 8. However, there is yet another class of parallelism to be exploited on NVIDIA GPU. This parallelism is similar to the task parallelism that is found in multithreaded CPU applications. NVIDIA CUDA task parallelism is based on CUDA streams. A CUDA stream represents a queue of GPU operations such as kernel launches, memory copies, and event starts and stops. The order in which operations are added to the stream specifies the order in which they will be executed. Each stream may be considered as a certain task on the GPU, and there are opportunities for these tasks to execute in parallel [31]. Thus we apply CUDA streams to our two kernels to improve parallelism and total GPU utilization.

There are many selections in item (2.3) for determining mutual arrangement of $P_{i,j}^{k-1}$ and cells of the mesh of the previous time level. Unfortunately, we cannot avoid these selections.

Secondly, the CUDA kernel in (3.3) of Algorithm 8 idles in the context of computation. We apply loop unrolling in order to eliminate this kernel.

Thirdly, the basic CUDA kernel of Algorithm 8 is not optimal for memory access. To optimize concurrent read access global memory of simultaneously running threads, constant memory is preferable to use. Applying this approach in our program we allocate all invariable values in constant memory GPU.

Moreover, for the optimization of parameters of kernels launch we use the CUDA Occupancy Calculator. It calculates optimal streaming multiprocessor (SM) utilization taking into account GPU compute capability, CUDA device properties, the number of blocks in a grid, the number of threads per block, the size of the shared-memory space, and the number of registers per thread.

Consequently, the second CUDA-version of Algorithm 7 for the CPU/GPU hybrid architecture is the following.

Algorithm 9 (CUDA parallel, version 2).

(1) Calculate kernel configuration (blockSize, blockCount) using data about a computational domain.

(2) Allocate host (CPU) and device (GPU) memory, copy initial data from host to device, and copy constant data from host to constant memory of device.

(3) *Time loop*: for each time step $k = 1, 3, 5, \ldots, K-1$ do:

(3.1) Call the first CUDA kernel to boundary cells access by the first CUDA stream:

(3.1.1) Execute items (2.1)–(2.4) of Algorithm 7 in parallel for each cell $\Omega_{i,j}$ whose characteristics intersect the boundary.

(3.2) Call the second CUDA kernel to inner cells access by the second CUDA stream:

(3.2.1) Execute items (2.1), (2.3)-(2.4) of Algorithm 7 in parallel for each internal cell $\Omega_{i,j}$.

(3.3) *Synch point*: wait for calculations of both kernels to be completed.

(3.4) Call the first CUDA kernel to boundary cells access by the first CUDA stream:

(3.4.1) Execute items (2.1)–(2.4) of Algorithm 7 in parallel for each cell $\Omega_{i,j}$ whose characteristics intersect the boundary.

(3.5) Call the second CUDA kernel to inner cells access by the second CUDA stream:

(3.5.1) Execute items (2.1), (2.3)-(2.4) of Algorithm 7 in parallel for each internal cell $\Omega_{i,j}$.

(3.6) *Synch point*: wait for calculations of both kernels to be completed.

(3.7) If necessary, copy results from device to host.

(3.8) If necessary, calculate the norms of a solution, an error, and other statistic data with respect to an actual time step.

The end of the time loop.

(4) If K is odd then repeat items (3.1)-(3.2).

(5) Copy results from device to host.

5. Numerical Experiments

Specify the velocities

$$u(t,x,y) = 100y(1-y)\left[\frac{\pi}{2} - \operatorname{arctg}(x)\right],$$
$$v(t,x,y) = \operatorname{arctg}\left(\frac{x(1-x)\,y(1-y)(1+t)}{10}\right) \tag{45}$$

and take the initial and boundary conditions in the following form:

$$\forall (x,y) \in D \quad \rho(0,x,y) = \rho_{\text{init}}(x,y) = 1,$$
$$\forall (t,y) \in [0,T] \times [0,1] \quad \rho(t,0,y) = \rho_{\text{in}}(t,y) = 1. \tag{46}$$

Numerical experiments were performed with the ICM SB RAS FLAGMAN computation system of the following configuration.

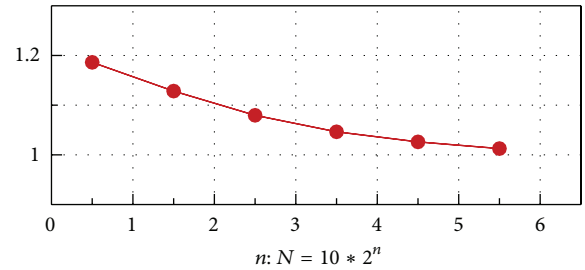

Figure 6: The order of convergence.

Hardware. CPU: INTEL XEON, 2×6 cores, HyperThreading; 40 Gb DDR3; GPU: NVIDIA TESLA C2050 (CC 2.0).

Software. OS: UBUNTU 11.04; C/C++: GCC 4.5.2, INTEL C++ Compiler 13.1.0; CUDA C/C++: NVCC 5.0; NVIDIA CUDA 5.0; BOOST 1.53; NVIDIA CUDA-GDB 5.0.

One of the purposes for the numerical experiments was to check the convergence order in τ and h. Therefore the computations were performed on the sequence of $N \times N$ regular square grids, $N = 10 \cdot 2^n$, $n = 0, \ldots, 6$. The number of time steps is defined by $\tau = h/5$.

Assume that $\{\rho_h^n\}_{n=0}^6$ is the set of solutions found on the sequence of square grids. The expression $\log_2(\|\rho - \rho_h^n\|_{L_1}^h / \|\rho - \rho_h^{n+1}\|_{L_1}^h)$ as a function in n can be considered as the order of convergence (Figure 6). The corresponding exact values of $\rho_{i,j}^K$ were computed by the characteristics method directly. In Figure 6 we can see the first order of convergence in h.

In our sequential and OpenMP computational experiments we compare the GCC and Intel C++ compilers. The execution time for the better Intel compiling code is on average by 15% less than the better GCC one. All presented numerical results were compiled with −O2 optimization level. Let us remark that we try to use other compiler options (−O3, −parallel, −AVX), but the performance increases only slightly or sometimes even decreases. For the CUDA-version we used the NVCC compiler.

The results of computation speedup of the OpenMP-version are presented in Table 1 and in Figure 8. The first line of the table shows speedup (or rather slowing down) of one thread that executed the OpenMP-code as compared with the sequential code. Generally, the compiler optimization under the OpenMP library and overhead related to the synchronization of OpenMP-threads can be estimated for these data. As we can see in our case, overhead is small.

One of the purposes of the studies is to assess influence of the HyperThreading (HT) technology on the parallelism. As long as only 12 physical cores are available on the CPU, we have access to 24 logical cores when HT is enabled and to 12 logical cores when HT is disabled, respectively. Experiments show that when HT is enabled the execution time with 24 threads is about 14% less than that with 12 threads when HT is disabled (Figure 7). As our code is compute-intensive, probably, the advantage of HT may be related to optimization of memory access.

Table 1: Speedup of OpenMP-code (HyperThreading is switched Off/On).

Number of threads	Number of points in one space dimension (number of the time steps)				
	80 ∗ 80 (400)	160 ∗ 160 (800)	320 ∗ 320 (1600)	640 ∗ 640 (3200)	1280 ∗ 1280 (6400)
1	0.94/1.00	0.99/1.00	0.99/1.00	0.99/1.00	0.99/1.00
4	3.86/3.89	3.90/3.93	3.92/3.93	3.93/3.95	3.93/3.96
8	7.40/7.35	7.48/7.37	7.53/7.57	7.55/7.59	7.56/7.60
12	10.86/10.86	11.08/11.04	11.17/11.12	11.21/11.14	11.20/10.86
16	4.15/8.76	4.77/8.94	6.32/8.96	7.54/8.98	7.60/8.88
20	4.69/10.63	5.82/10.85	6.55/10.94	7.26/10.96	7.89/10.73
24	5.71/10.98	6.17/12.80	7.30/12.55	7.97/12.83	8.45/12.85

Table 2: Execution time of all versions of program.

Version	Number of mesh points in one space dimension (number of the time steps)				
	80 ∗ 80 (400)	160 ∗ 160 (800)	320 ∗ 320 (1600)	640 ∗ 640 (3200)	1280 ∗ 1280 (6400)
Sequential, −O0	20.16	159.40	1268.46	10107.80	∗
Sequential, −O2	9.99	78.97	626.72	4980.61	39598.90
Sequential, −O3	9.87	78.08	619.80	4936.25	39202.91
OpenMP(12), −O0, HT Off	1.98	12.52	103.45	819.06	6519.87
OpenMP(12), −O2, HT Off	0.92	7.13	56.10	444.15	3535.53
OpenMP(24), −O2, HT On	0.91	6.17	49.93	388.20	3080.91
CUDA version 1	2.55	13.06	74.02	∗∗	∗∗
CUDA version 2	3.20	8.27	42.60	308.82	∗∗

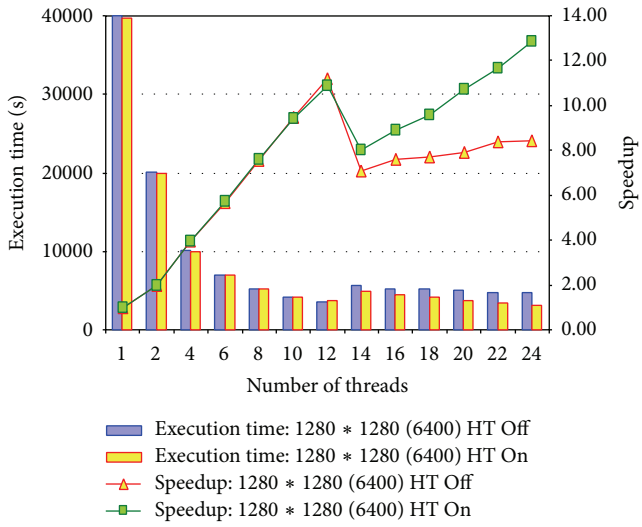

Figure 7: Execution time of OpenMP-code (the main vertical axis) and speedup in comparison with the sequential code (additional vertical axis). The comparison of results when HyperThreading is switched On or Off.

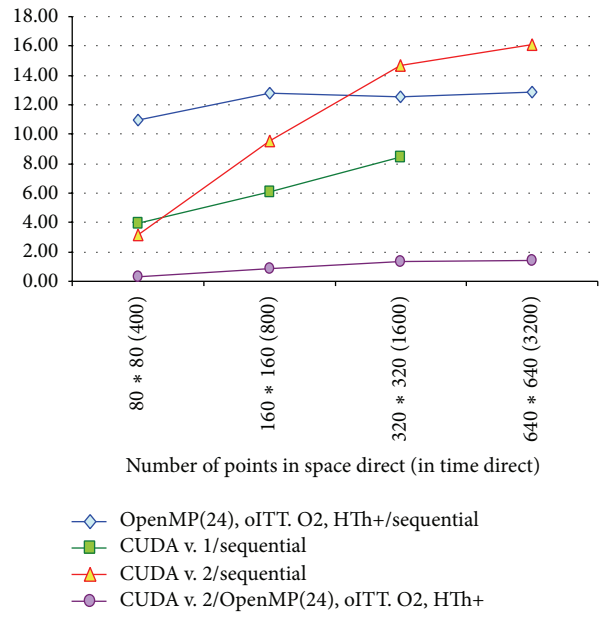

Figure 8: Speedup of parallel versions.

The execution time of the sequential, OpenMP, and two versions of CUDA codes is given in Table 2. The ≪∗≫ symbol signifies that the result has not been reached in reasonable time; for the CUDA-versions the ≪ ∗∗ ≫ symbol means that the kernel launch failed due to registers bottleneck. For OpenMP the results on 12 threads when HT is disabled and on 24 threads when HT is enabled are demonstrated.

Comparative information on a possibility of code optimization by the GCC compiler is also presented in Table 2. The execution time of the code compiled without optimization and with −O2 and −O3 optimization levels, respectively, was measured. It is clear from Table 2 that compiler optimization considerably (more than two times) reduces the execution time of the sequential code. As the compiler

Table 3: Speedup of parallel versions.

	Number of mesh points in one space dimension (number of the time steps)			
	80 ∗ 80 (400)	160 ∗ 160 (800)	320 ∗ 320 (1600)	640 ∗ 640 (3200)
OpenMP(24), -O2, HT ON/sequential	10.98	12.80	12.55	12.83
CUDA version 1/sequential	3.92	6.05	8.47	∗∗
CUDA version 2/sequential	3.12	9.55	14.71	16.13
CUDA version 2/OpenMP(12), -O2, HT Off	0.29	0.86	1.32	1.44
CUDA version 2/OpenMP(24), -O2, HT On	0.28	0.75	1.17	1.26

```
_global_ void unit_foo (float* inA, float out, int size) {
    int i=blockIdx.x*blockDim.x+threadIdx.x;
    if (i<size)
        out[i] = do_smb_with(inA[i]);
}
```

Pseudocode 1

does not optimize the CUDA-code, the execution time of the CUDA-versions does not depend on compiling options.

Table 3 and Figure 8 present the data on computation speedup of the best OpenMP-version and two CUDA-versions in comparison with the sequential program compiled with −O2. Speedup of the second CUDA-version in comparison with the OpenMP-version is given as well. Numerical experiments show that on fine grids in comparison with the sequential program the best Open-MP program with 24 threads gives more than 12 times speedup and the second CUDA-version shows about 16 times speedup.

6. Discussion

Nowadays there are a lot of algorithms of the family of semi-Lagrangian methods. As we mentioned above, the presented method is based on square grid only and uses accessory algorithms that makes computation more resource-intensive. Nevertheless, such a complication allows us to prove first-order convergence. Furthermore, the theorem which allows one to take into account a volume of substance passed through a boundary is presented. This enables us to prove the balance equation. Numerical experiments completely confirm the theoretical convergence results.

As for parallel implementation of our approach, we can note the following.

Though the algorithm is explicit we are not satisfied with the results of the CUDA-versions.

First of all, there is a problem with feasibility of CUDA-code on fine grids (more than 640 × 640 nodes). Profile-feedback analysis shows at least two causes: (1) assumed computational domain decomposition and (2) the problem of hardware constraint on the amount of registers on streaming multiprocessors.

Concerning the first item, it should be noted that in our approach a 2D computational domain is mapped to a 1D array of cells, and every thread treats one cell (e.g., see Pseudocode 1).

For the parameters of kernels launch to be readily adaptable, we can apply "thread reuse"; namely, every thread treats some uncoupled cells (see Pseudocode 2).

Besides, we can use several video adapters. In this case we should resolve the problem of the distribution of computational cells between devices under the following restriction: we do not know in advance how many cells of the previous time level are required to calculate an actual value (e.g., varying-width shadow line).

The problem with registers in our case is related to deep nesting level of functions calls in the item (2.3) for determining the mutual arrangement of $P_{i,j}^{k-1}$ and cells of a mesh of the previous time level. Indeed, this is a bottleneck of our sequential algorithm. To resolve this issue, we should modify the initial sequential algorithm, unfortunately.

In addition, the item (2.3) has many flow control instructions ("if" statements, mainly), but we cannot say that this branching creates some significant performance penalty in terms of SIMT architecture. Indeed, in a CUDA kernel, any flow control instruction (if, switch, do, for, while) can significantly affect the instruction throughput by causing threads of the same warp to diverge [32]. If this happens, the different execution paths must be serialized, since all of the threads of a warp share a program counter; this increases the total number of instructions executed for this warp. When all the different execution paths have completed, the threads converge back to the same execution path.

Fortunately, this rule works in the cases where the control flow depends on the thread ID only. However, in our case we have many branches which do not depend on thread ID. Inside our computational kernel, the main part of branches looks like Pseudocode 3.

As we can see, our "IF" statement does not depend on a thread ID. Thus, this type of branch does not affect the performance of the CUDA kernel.

```
__global__ void vec_foo (float* inA, float out, int size) {
    for (int i=blockIdx.x*blockDim.x+threadIdx.x;
                    i<size; i+=blockDim.x*gridDim.x)
    out[i] = do_smb_with(inA[i]);
}
```

Pseudocode 2

```
if ((indCurSqOx[1] >= 0) && (indCurSqOy[1] >= 0)) {
     do_smth();
} else {
     do_smth_else();
}
```

Pseudocode 3

7. Conclusion

We present an unconditionally stable semi-Lagrangian scheme of the first-order accuracy. The numerical experiments confirm the theoretical studies. Performance of sequential and several parallel algorithms implemented with the OpenMP and CUDA technologies in the C language was studied. The optimization potential of the CUDA-version remains unexhausted yet.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work was supported by Project 14-11-00147 of the Russian Scientific Foundation.

References

[1] S. K. Godunov, A. V. Zabrodin, M. Y. Ivanov, A. N. Kraiko, and G. P. Prokopov, *Numerical Solving of Multidimensional Gas Dynamics Problems*, Science Publisher, Moscow, Russia, 1976, (In Russian).

[2] S. Godunov, *Mathematical Physics Equations*, Science Publisher, Moscow, Russia, 1979, (In Russian).

[3] J. D. Anderson, *Computational Fluid Dynamics: The Basics with Applications*, McGraw-Hill, New York, NY, USA, 1995.

[4] M. Lentine, J. T. Grétarsson, and R. Fedkiw, "An unconditionally stable fully conservative semi-Lagrangian method," *Journal of Computational Physics*, vol. 230, no. 8, pp. 2857–2879, 2011.

[5] D. Levy, G. Puppo, and G. Russo, "Central WENO schemes for hyperbolic systems of conservation laws," *Mathematical Modelling and Numerical Analysis*, vol. 33, no. 3, pp. 547–571, 1999.

[6] R. J. LeVeque, *Finite Volume Methods for Hyperbolic Problems*, Cambridge Texts in Applied Mathematics, Cambridge University Press, 2002.

[7] S. Clain, S. Diot, and R. Loubère, "A high-order finite volume method for systems of conservation laws with Multi-dimensional Optimal Order Detection (MOOD)," *Journal of Computational Physics*, vol. 230, no. 10, pp. 4028–4050, 2011.

[8] M. Kaser and A. Iske, "ADER schemes on adaptive triangular meshes for scalar conservation laws," *Journal of Computational Physics*, vol. 205, no. 2, pp. 486–508, 2005.

[9] J. S. Scroggs and F. H. M. Semazzi, "A conservative semi-Lagrangian method for multidimensional fluid dynamics applications," *Numerical Methods for Partial Differential Equations*, vol. 11, no. 5, pp. 445–452, 1995.

[10] J. Behrens, "A parallel adaptive finite-element semi- lagrangian advection scheme for the shallow water equations," in *Modeling and Computation in Environmental Sciences*, vol. 59 of *Notes on Numerical Fluid Mechanics (NNFM)*, pp. 49–60, 1997.

[11] A. Klar, P. Reuterswärd, and M. Seaïd, "A semi-Lagrangian method for a Fokker-Planck equation describing fiber dynamics," *Journal of Scientific Computing*, vol. 38, no. 3, pp. 349–367, 2009.

[12] O. Pironneau, "On the transport-diffusion algorithm and its applications to the Navier-Stokes equations," *Numerische Mathematik*, vol. 38, no. 3, pp. 309–332, 1982.

[13] E. Carlini, M. Falcone, and R. Ferretti, "A time-adaptive semi-Lagrangian approximation to mean curvature motion," in *Numerical Mathematics and Advanced Applications*, pp. 732–739, Springer, Berlin, Germany, 2006.

[14] D. R. Durran, *Numerical Methods for Wave Equations in Geophysical Fluid Dynamics*, Springer, New York, NY, USA, 1999.

[15] K. W. Morton, *Numerical Solution of Convection-Diffusion Problems*, Chapman & Hall, London, UK, 1996.

[16] A. Staniforth and J. Cote, "Semi-Lagrangian integration schemes for atmospheric models—a review," *Monthly Weather Review*, vol. 119, no. 9, pp. 2206–2223, 1991.

[17] A. Priestley, "A quasi-conservative version of the semi-Lagrangian advection scheme," *Monthly Weather Review*, vol. 121, no. 2, pp. 621–629, 1993.

[18] H. Ritchie, "Semi-Lagrangian advection on a Gaussian grid," *Monthly Weather Review*, vol. 116, no. 2, pp. 608–619, 1987.

[19] A. Robert, T. L. Yee, and H. Ritchie, "A semi-Lagrangian and semi-implicit numerical integration scheme for multilevel atmospheric models," *Monthly Weather Review*, vol. 113, no. 3, pp. 388–394, 1985.

[20] J. Behrens and L. Mentrup, "A conservative scheme for 2D and 3D adaptive semi-Lagrangian advection," Tech. Rep. TUM M0411, Technische Universität München, Fakultät für Mathematik, 2004.

[21] A. Wiin-Nielson, "On the application of trajectory methods in numerical forecasting," *Tellus*, vol. 11, pp. 180–186, 1959.

[22] V. V. Shaidurov, G. I. Shchepanovskaya, and V. Yakubovich, "Numerical simulation of supersonic flows in a channel," *Russian Journal of Numerical Analysis and Mathematical Modelling*, vol. 27, no. 6, pp. 585–601, 2012.

[23] H. Chen, Q. Lin, V. V. Shaidurov, and J. Zhou, "Error estimates for triangular and tetrahedral finite elements in combination with a trajectory approximation of the first derivatives for advection-diffusion equations," *Numerical Analysis and Applications*, vol. 4, no. 4, pp. 345–362, 2011.

[24] T. N. Phillips and A. J. Williams, "Conservative semi-Lagrangian finite volume schemes," *Numerical Methods for Partial Differential Equations*, vol. 17, no. 4, pp. 403–425, 2001.

[25] J. P. R. Laprise and A. Plante, "A class of semi-Lagrangian integrated-mass (SLIM) numerical transport algorithms," *Monthly Weather Review*, vol. 123, pp. 553–565, 1995.

[26] T. Arbogast and W. Wang, "Convergence of a fully conservative volume corrected characteristic method for transport problems," *SIAM Journal on Numerical Analysis*, vol. 48, no. 3, pp. 797–823, 2010.

[27] K. Takizawa, T. Yabe, and T. Nakamura, "Multi-dimensional semi-Lagrangian scheme that guarantees exact conservation," *Computer Physics Communications*, vol. 148, no. 2, pp. 137–159, 2002.

[28] A. Iske and M. Kaser, "Conservative semi-Lagrangian advection on adaptive unstructured meshes," *Numerical Methods for Partial Differential Equations*, vol. 20, no. 3, pp. 388–411, 2004.

[29] E. A. Novikov, *Explicit Methods for Stiff Problems*, Science Publisher, Novosibirsk, Russia, 1997, (In Russian).

[30] V. I. Krylov, *Approximate Computation of Integrals*, Science Publisher, Moscow, Russia, 1967, (In Russian).

[31] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley, 2010.

[32] *CUDA C Best Practices Guide V.6.0*, NVIDIA Corporation, 2014, http://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf.