Proceedings of the Czech–Japanese Seminar in Applied Mathematics 2006
Czech Technical University in Prague, September 14-17, 2006
pp. 25–36

PARALLEL ALGORITHM FOR NUMERICAL SOLUTION OF THE SHALLOW WATER EQUATION

STANISLAV BRAND[1]

Abstract. Parallel programming is the leading technique for accelerating numerical algorithm. This article deals with parallelization of the program for solving the shallow water equation in 2D, representing an example of a conservation law. The equation is solved by finite difference and finite volume methods within the Lax-Friedrichs, Lax-Wendroff and MacCormack scheme. The algorithm is parallelized using OpenMP and MPI. The efficiency measurement is made mainly on systems with shared memory and thanks to the sufficient number of processors there are some results of the mixed mode programming which used both OpenMP and MPI.

Key words. Conservation laws, parallel programming, OpenMP, MPI, mixed mode programming.

1. Introduction. This article summarizes the results of numerical solution of the shallow water equation. The first section shows the derivation of the one-dimensional shallow water equation and describes the two-dimensional shallow water system. The second section contains numerical schemes used, in the particular finite difference and finite volume variants of the well-known Lax-Friedrichs, Lax-Wendroff and MacCormack schemes. The third section describes parallelization, that was made for improving program performance. The fourth and fifth section contains several results of efficiency measurement and of the problem solved.

2. Solved problem. The Shallow water equation is a special case of the Euler equations describing wave motion in shallow water. The two-dimensional shallow water equation is often used for numerical simulation of natural river flows. For illustration, we derive the one-dimensional shallow water equation which is also often used. Consider a fluid in a channel and assume that the vertical velocity in the fluid is negligible and the horizontal velocity $v(x,t)$ is constant through any vertical cross section. For example, small aptitude wave in the fluid that is shallow relative to the wave length. We assume an incompressible fluid, where the density ρ is constant. Our variable is the height of water $h(x,t)$. The total mass in x_1, x_2 at time t can be written as $m_{x_1,x_2} = \int_{x_1}^{x_2} \rho h(x,t)\mathrm{d}x$. Vertical integration from 0 to $h(x,t)$ of the momentum $\rho v(x,t)$ gives us the mass flux to be $\rho v(x,t)h(x,t)$. By dropping out the constant ρ we get the conservation of mass equation in the form

$$h_t + (vh)_x = 0. \tag{2.1}$$

Now we get the conservation of momentum from the Euler equation

$$(\rho hv)_t + (\rho hv^2 + p)_x = 0. \tag{2.2}$$

In our case, the pressure p is determined from a hydrostatic law saying the pressure at depth y is ρgy, where g is the gravitation constant. Integrating this relation vertically from $y = 0$ to $y = h(x,t)$ we get the proper pressure term $p = \frac{1}{2}\rho gh^2$. Using this in (2.2), and pointing out ρ gives

[1]Department of Mathematics, Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University, Prague.

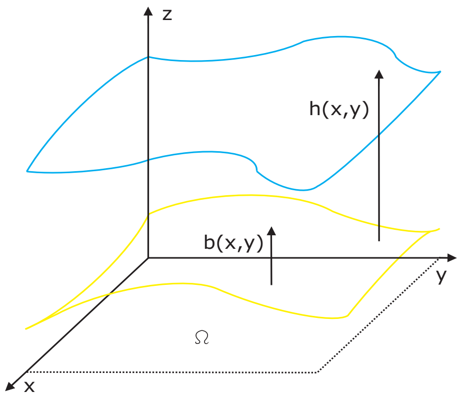


FIG. 2.1. *Meaning of variables for the shallow water equation.*

$$(hv)_t + (hv^2 + \frac{1}{2}gh^2)_x = 0. \tag{2.3}$$

One-dimensional shallow water system consists of the continuity equation (2.1) and the momentum equation (2.3). We use the two-dimensional shallow water equation whose derivation can be found in [3].

It can be written as

$$U_t + F(U)_x + G(U)_y = S, \tag{2.4}$$

where

$$U = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix},$$

$$F(U) = \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix} = \begin{bmatrix} u_2 \\ u_2^2/u_1 + \frac{1}{2}gu_1^2 \\ \frac{u_2u_3}{u_1} \end{bmatrix},$$

$$G(U) = \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix} = \begin{bmatrix} u_3 \\ \frac{u_2u_3}{u_1} \\ u_3^2/u_1 + \frac{1}{2}gu_1^2 \end{bmatrix},$$

$$S = \begin{bmatrix} 0 \\ -ghb_x \\ -ghb_y \end{bmatrix}.$$

Here $h = h(x, y)$ denotes the depth of water over the bottom topography $b = b(x, y)$, u,v are velocities in x and y directions, and g stands for the gravitational constant (see Figure 2.1).

3. Numerical solution. We used three finite difference methods for conservation laws, the Lax-Friedrichs(LF), the Lax-Wendroff(LW), and the MacCormack(MC) schemes. For details, we refer the reader to [4],[5]. All of these methods have some limitations. The LF scheme is first-order-accurate and diffusive. The LW and MC schemes are second-order-accurate, but oscillatory near shocks.

We therefore also tested the composite scheme as well which combines these methods in order to eliminate their problems. For more information about the composite schemes, see [6],[7]. All computations were made with the composite scheme that was a composition of three LW steps and one diffusive LF step needed for removing spurious oscillations.

The schemes in the following sections are based on the finite volume and finite difference discretization.

3.1. Lax-Friedrichs scheme. Finite volume method yields:

$$U_i^{n+1} = \frac{1}{N}\sum_{k=1}^{N} U_k^n - \frac{\Delta t}{\mu(D_i)}\sum_{k=1}^{N} F_{i,k}^n \Delta y_k - G_{i,k}^n \Delta x_k, \tag{3.1}$$

where

$$\begin{aligned} F_{i,k}^n &= \frac{1}{2}[F(U_i^n) + F(U_k^n)] \\ G_{i,k}^n &= \frac{1}{2}[G(U_i^n) + G(U_k^n)]. \end{aligned} \tag{3.2}$$

Finite difference method yields:

$$\begin{aligned} U_{i,j}^{n+1} &= \frac{1}{4}(U_{i,j-1}^n + U_{i,j+1}^n + U_{i-1,j}^n + U_{i+1,j}^n) \\ &- \frac{k}{2h}\left[F(U_{i+1,j}^n) - F(U_{i-1,j}^n)\right] - \frac{k}{2l}\left[G(U_{i,j+1}^n) - G(U_{i,j-1}^n)\right]. \end{aligned} \tag{3.3}$$

3.2. Lax-Wendroff scheme. Finite volume method yields:

$$\begin{aligned} U_i^{n+\frac{1}{2}} &= \frac{1}{N}\sum_{k=1}^{N} U_k^n - \frac{\Delta t}{2\mu(D_i)}\sum_{k=1}^{N} F_{i,k}^n \Delta y_k - G_{i,k}^n \Delta x_k, \\ U_i^{n+1} &= U_i^n - \frac{\Delta t}{\mu(D_i)}\sum_{k=1}^{N} F_{i,k}^{n+\frac{1}{2}} \Delta y_k - G_{i,k}^{n+\frac{1}{2}} \Delta x_k, \end{aligned} \tag{3.4}$$

where

$$\begin{aligned} F_{i,k}^n &= \frac{1}{2}[F(U_i^n) + F(U_k^n)], \quad F_{i,k}^{n+\frac{1}{2}} = \frac{1}{2}[F(U_i^{n+\frac{1}{2}}) + F(U_k^{n+\frac{1}{2}})], \\ G_{i,k}^n &= \frac{1}{2}[G(U_i^n) + G(U_k^n)], \quad G_{i,k}^{n+\frac{1}{2}} = \frac{1}{2}[G(U_i^{n+\frac{1}{2}}) + G(U_k^{n+\frac{1}{2}}). \end{aligned} \tag{3.5}$$

Finite-difference method yields:

$$\begin{aligned} U_{i,j}^{n+\frac{1}{2}} &= \frac{1}{4}(U_{i,j-1}^n + U_{i,j+1}^n + U_{i-1,j}^n + U_{i+1,j}^n) \\ &- \frac{k}{4h}[(F(U_{i+1,j}^n) - F(U_{i-1,j}^n)] - \frac{k}{4l}[(G(U_{i,j+1}^n) - G(U_{i,j-1}^n)] \\ U_{i,j}^{n+1} &= U_{i,j}^n \\ &- \frac{k}{2h}[(F(U_{i+1,j}^{n+\frac{1}{2}}) - F(U_{i-1,j}^{n+\frac{1}{2}})] - \frac{k}{2l}[(G(U_{i,j+1}^{n+\frac{1}{2}}) - G(U_{i,j-1}^{n+\frac{1}{2}})]. \end{aligned} \tag{3.6}$$

3.3. MacCormack schemes. Finite volume method yields:

$$
\begin{aligned}
U_i^{n+\frac{1}{2}} &= U_i^n - \frac{\Delta t}{\mu(D_i)} \sum_{k=1}^{N} F_{i,k}^n \Delta y_k - G_{i,k}^n \Delta x_k, \\
U_i^{n+1} &= \frac{1}{2}\left[U_i^n + U_i^{n+\frac{1}{2}} - \frac{\Delta t}{\mu(D_i)} \sum_{k=1}^{N} F_{i,k}^{n+\frac{1}{2}} \Delta y_k - G_{i,k}^{n+\frac{1}{2}} \Delta x_k \right],
\end{aligned}
\tag{3.7}
$$

where

$$
\begin{aligned}
F_{i,1}^n &= F(U_{i+1,j}^n), \quad F_{i,2}^n = F(U_{i,j+1}^n), \quad F_{i,3}^n = F_{i,4}^n = F(U_{i,j}^n) \\
F_{i,3}^{n+\frac{1}{2}} &= F(U_{i-1,j}^{n+\frac{1}{2}}), \quad F_{i,4}^{n+\frac{1}{2}} = F(U_{i,j-1}^{n+\frac{1}{2}}), \quad F_{i,1}^{n+\frac{1}{2}} = F_{i,2}^{n+\frac{1}{2}} = F(U_{i,j}^{n+\frac{1}{2}}) \\
G_{i,1}^n &= G(U_{i+1,j}^n), \quad G_{i,2}^n = G(U_{i,j+1}^n), \quad G_{i,3}^n = G_{i,4}^n = G(U_{i,j}^n) \\
G_{i,3}^{n+\frac{1}{2}} &= G(U_{i-1,j}^{n+\frac{1}{2}}), \quad G_{i,4}^{n+\frac{1}{2}} = G(U_{i,j-1}^{n+\frac{1}{2}}), \quad G_{i,1}^{n+\frac{1}{2}} = G_{i,2}^{n+\frac{1}{2}} = G(U_{i,j}^{n+\frac{1}{2}}).
\end{aligned}
\tag{3.8}
$$

Finite difference method yields:

$$
\begin{aligned}
U_{i,j}^{n+\frac{1}{2}} &= U_{i,j}^n \\
&\quad - \frac{k}{h}[(F(U_{i+1,j}^n) - f(U_{i,j}^n)] - \frac{k}{h}[(G(U_{i,j+1}^n) - G(U_{i,j}^n)] \\
U_{i,j}^{n+1} &= \frac{1}{2}(U_{i,j}^n + U_{i,j}^{n+\frac{1}{2}}) \\
&\quad - \frac{k}{2h}[(F(U_{i,j}^{n+\frac{1}{2}}) - F(U_{i-1,j}^{n+\frac{1}{2}})] - \frac{k}{2h}[(G(U_{i,j}^{n+\frac{1}{2}}) - G(U_{i,j-1}^{n+\frac{1}{2}})].
\end{aligned}
\tag{3.9}
$$

4. Stability. Each numerical scheme must satisfy the necessary stability condition in the form

$$
\Delta t \le \frac{1}{\frac{\rho_A}{\Delta x} + \frac{\rho_B}{\Delta y}}, \tag{4.1}
$$

where ρ_A and ρ_A are spectral radii of jacobian matrix A and B,

$$
A = \frac{\partial F(U)}{\partial U} = \begin{bmatrix} 0 & 1 & 0 \\ gh - u^2 & 2u & 0 \\ -uv & v & u \end{bmatrix}, \tag{4.2}
$$

$$
B = \frac{\partial G(U)}{\partial U} = \begin{bmatrix} 0 & 0 & 1 \\ -uv & v & u \\ gh - v^2 & 0 & 2v \end{bmatrix}. \tag{4.3}
$$

Unfortunately, this condition is not sufficient for nonlinear partial differential equations. To demonstrate stability and consistency, we have to use numerical results computed on a refined grid. We linearly interpolate the solution on the finest grid and compare it with the remaining solutions (see Tables 4.1 - 4.4).

5. Numerical results. This section contains results for the shallow water equation with the flat bottom. Results are represented by graphs displaying depth of water h at each point of the space domain (see Figures 5.1 - 5.3). The initial condition for the depth of water

Mesh h	$\mathcal{L}_\infty(0,T;\mathcal{L}_2)$ error of u	$\mathcal{L}_\infty(0,T;\mathcal{L}_\infty)$ error of u
0.0800000	0.0154219	0.0149480
0.0400000	0.0078344	0.0071600
0.0200000	0.0036292	0.0033470
0.0100000	0.0015629	0.0016555
0.0050000	0.0005506	0.0006410

TABLE 4.1
Table of convergence errors for the MacCormack scheme.

Mesh h	EOC u L_2	EOC u L_∞
0.0800000	0.0000000	0.0000000
0.0400000	0.9584786	1.2418251
0.0200000	1.3806293	1.0897255
0.0100000	0.9930047	1.1842024
0.0050000	1.0094179	1.2039963

TABLE 4.2
Table of EOC coefficients for the MacCormack scheme.

Mesh h	$\mathcal{L}_\infty(0,T;\mathcal{L}_2)$ error of u	$\mathcal{L}_\infty(0,T;\mathcal{L}_\infty)$ error of u
0.0800000	0.0154219	0.0149480
0.0400000	0.0078344	0.0071600
0.0200000	0.0036292	0.0033470
0.0100000	0.0015629	0.0016555
0.0050000	0.0005506	0.0006410

TABLE 4.3
Table of convergence errors for the composite scheme.

Mesh h	EOC u L_2	EOC u L_∞
0.0800000	0.0000000	0.0000000
0.0400000	0.9770784	1.0619210
0.0200000	1.1101687	1.0970910
0.0100000	1.2154725	1.0156016
0.0050000	1.5051878	1.3688708

TABLE 4.4
Table of EOC coefficients for the composite scheme.

h is presented in figures at time $t = 0$. The initial velocities u,v were set to zero. The solution was computed at the time interval $[0, 0.14]$ using the space domain $[-2, 2] \times [-2, 2]$. We use transmissive and reflective boundary conditions at points U_b on the boundary. The transmissive boundary condition are defined as $U_b = [h, uh, vh]^T$ and the reflective ones are defined as $U_b = [h, 0, 0]^T$, where h, u, v are values at the inner neighbour of each boundary point.

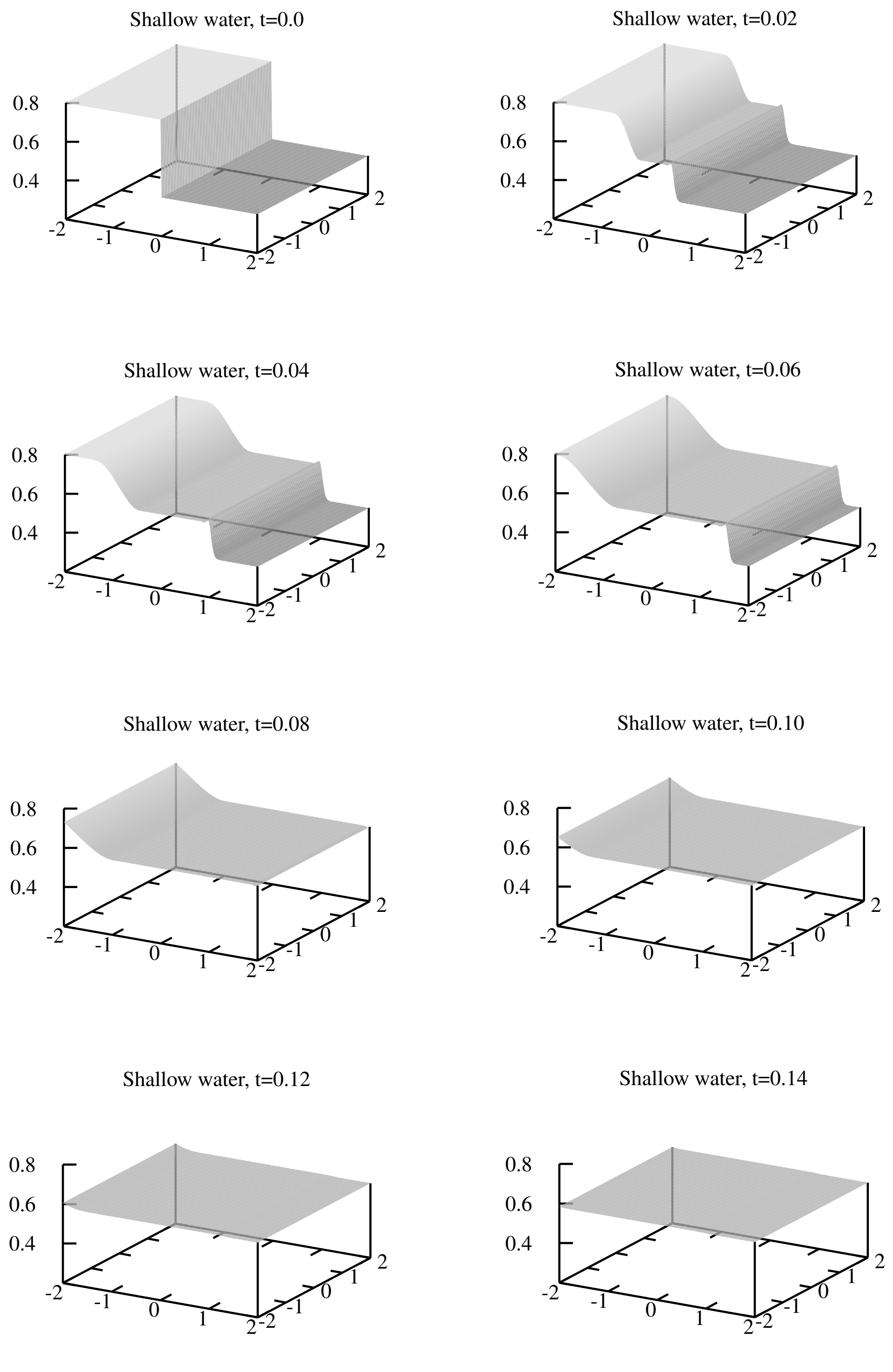


FIG. 5.1. *Shallow water, transmissive boundary condition.*

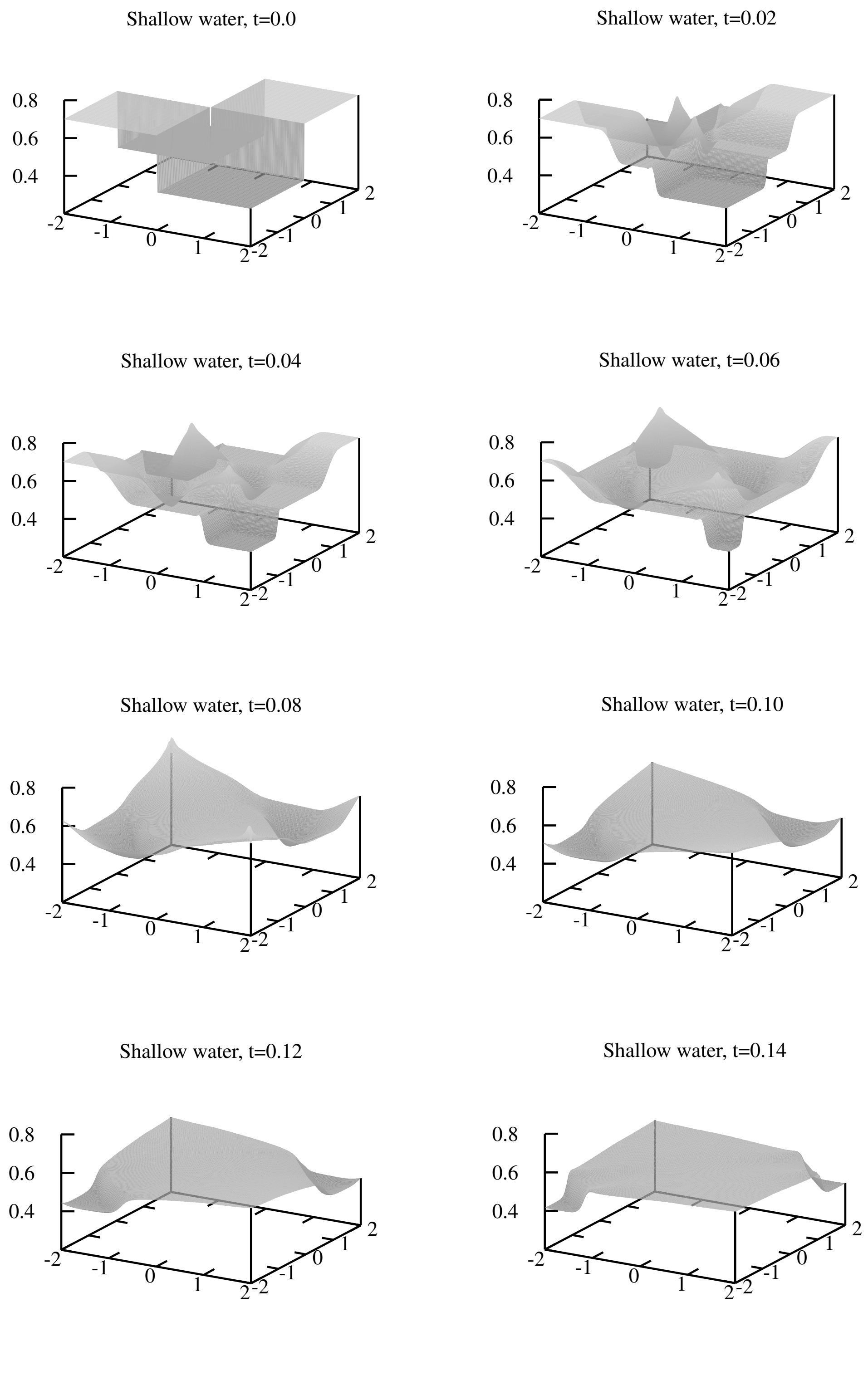


FIG. 5.2. *Shallow water, transmissive boundary condition.*

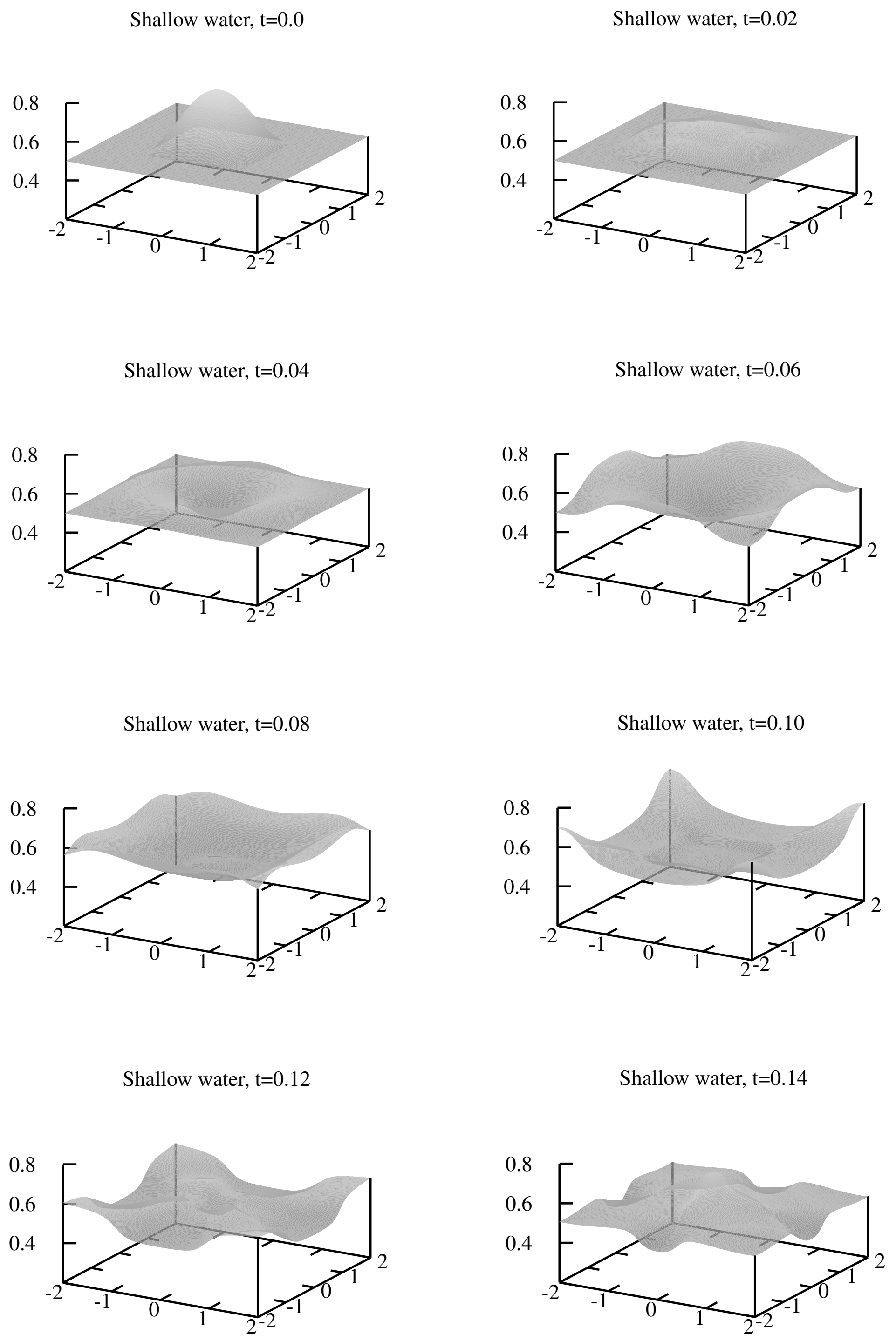


FIG. 5.3. *Shallow water, reflective boundary condition.*

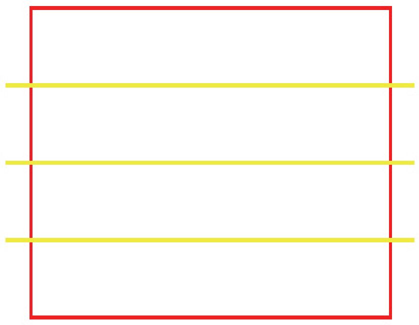


FIG. 6.1. *Used splitting of the space domain.*

6. Parallelization of numerical algorithms. The main purpose of our work was to compare the efficiency of parallel algorithms for numerical solution of the shallow water equation on systems with shared memory. Parallel computation is based on concepts for data exchange, shared and distributed memory. Shared memory means that all data are saved in the memory that can be accessed by all CPUs. This concept is used by OpenMP API (see [2]). Distributed memory means that in a multiprocessor system each processor has its own memory and after the processing of required computation tasks the data have to be reassembled. This concept is used by Message Passing Interface (MPI) (see [1]) and is applied similarly to [8].

All the presented numerical schemes are explicit. This enables to split the space domain into p parts and solve each part on one processing unit. The OpenMP shares the memory, therefore, after computing values at time n, all CPUs start computing the $n + 1$ time level. However, with distributed memory systems, it is necessary to send values from the boundary of each part to all neighbors after completing every time step computation.

The domain splitting is illustrated in Figure 6.1, where the square represents the border of the space domain and dash lines represent borders of parts used for the computation on each processing unit. It can be seen that each middle part has only two neighbors. This simplifies communication needs for parallel computation. One can admit that this splitting is not minimizing the number of data sent. For example, if we use the domain splitting with the square parts, then we will need to send only two thirds of the data needed for this case. But this splitting is independent of the number of processing units, which allow us to measure the efficiency with the same program on any number of processors.

7. Results of the efficiency measurement. The efficiency measurement was performed for the shallow water equation (2.4) on the following parallel systems. The first three computers are shared memory parallel systems and the last two computers are distributed parallel systems.

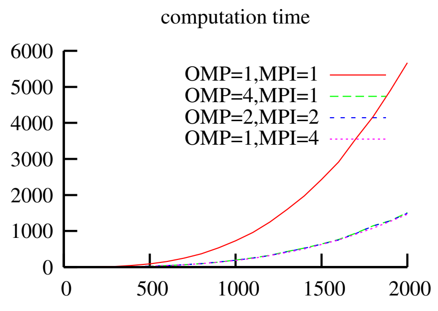
- Hewlett Packard C8000, 4xCPU HP PA-RISC - 1GHz, 12GB RAM
- SGI ALTIX 3700, 24xCPU Intel Itanium II - 1,3 GHz, 64GB RAM
- Linux PC, 2xCPU Intel Pentium II - 700 MHz, 1GB RAM
- Linux grid, 8xCPU Intel Pentium 4 - 2.4 GHz, 512MB RAM
- Linux grid, 12xCPU Intel Pentium 4 - 2.4 GHz, 512MB RAM

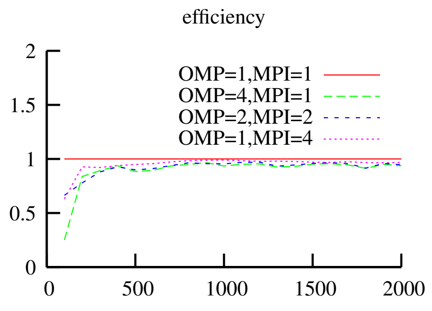
The results of efficiency measurement are presented in Tables 7.1 and 7.1. The tables have the following structure. The first column contains the grid dimension. The second column contains the time of sequence program in seconds, this means the time of the computation made by only one processing unit. The remaining columns contain the time of parallel program and the efficiency of this program in the brackets. In the header of these columns there is specified how many MPI and OpenMP processes were used in the computation. The time duration of each computation was measured by the C $gettimeofday()$ function as a difference between the start and the end time. The times listed in here are the times needed for the computation only. This means the times needed for the value initialization and result saving is excluded. The efficiency is calculated from the following formula:

$$\text{efficiency} = \frac{\text{sequence time}}{\text{parallel time} \times \text{number of processors}} \tag{7.1}$$

grid		OMP=4,MPI=1	OMP=2,MPI=2	OMP=1,MPI=4
100×100	0.660	0.661(0.250)	0.249(0.664)	0.262(0.631)
200×200	5.705	1.704(0.837)	1.833(0.778)	1.540(0.926)
300×300	19.436	5.445(0.892)	5.525(0.879)	5.283(0.920)
400×400	46.568	12.432(0.936)	12.564(0.927)	12.411(0.938)
500×500	89.915	25.429(0.884)	24.972(0.900)	23.695(0.949)
600×600	154.264	43.074(0.895)	42.324(0.911)	40.290(0.957)
700×700	248.072	66.403(0.934)	66.086(0.938)	63.709(0.973)
800×800	369.939	97.479(0.949)	95.952(0.964)	94.033(0.984)
900×900	533.809	138.156(0.966)	139.469(0.957)	134.933(0.989)
1000×1000	726.617	194.368(0.935)	190.263(0.955)	183.260(0.991)
1100×1100	960.135	252.363(0.951)	248.010(0.968)	244.049(0.984)
1200×1200	1250.901	327.944(0.954)	322.275(0.970)	319.292(0.979)
1300×1300	1600.657	432.835(0.925)	429.117(0.933)	408.119(0.981)
1400×1400	1975.505	530.362(0.931)	523.026(0.944)	506.207(0.976)
1500×1500	2432.951	639.059(0.952)	635.187(0.958)	626.386(0.971)
1600×1600	2918.703	762.497(0.957)	754.847(0.967)	753.314(0.969)
1700×1700	3562.641	943.562(0.944)	931.938(0.956)	914.391(0.974)
1800×1800	4169.705	1142.250(0.913)	1139.587(0.915)	1078.108(0.967)
1900×1900	4900.167	1289.412(0.950)	1278.466(0.958)	1269.950(0.965)
2000×2000	5668.034	1513.491(0.936)	1495.630(0.947)	1462.494(0.969)

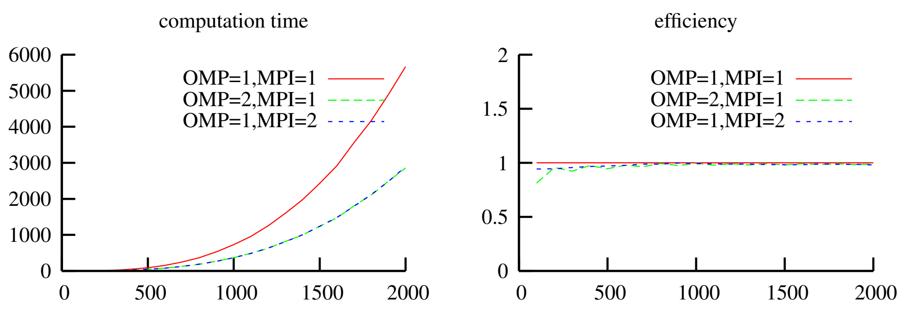
TABLE 7.1
Time and efficiency of parallel program using composite scheme computed on sugar.fjfi.cvut.cz.





grid		OMP=2,MPI=1	OMP=1,MPI=2
100×100	0.660	0.407(0.812)	0.350(0.943)
200×200	5.705	2.979(0.957)	3.018(0.945)
300×300	19.436	10.542(0.922)	10.168(0.956)
400×400	46.568	23.888(0.975)	24.081(0.967)
500×500	89.915	47.564(0.945)	46.331(0.970)
600×600	154.264	78.828(0.978)	78.992(0.976)
700×700	248.072	128.731(0.964)	125.554(0.988)
800×800	369.939	185.831(0.995)	187.054(0.989)
900×900	533.809	274.079(0.974)	267.076(0.999)
1000×1000	726.617	364.812(0.996)	365.749(0.993)
1100×1100	960.135	490.989(0.978)	484.739(0.990)
1200×1200	1250.901	629.919(0.993)	632.366(0.989)
1300×1300	1600.657	815.476(0.981)	809.857(0.988)
1400×1400	1975.505	994.999(0.993)	1002.577(0.985)
1500×1500	2432.951	1240.065(0.981)	1238.091(0.983)
1600×1600	2918.703	1470.887(0.992)	1484.173(0.983)
1700×1700	3562.641	1809.891(0.984)	1799.128(0.990)
1800×1800	4169.705	2101.875(0.992)	2113.957(0.986)
1900×1900	4900.167	2491.581(0.983)	2485.624(0.986)
2000×2000	5668.034	2858.638(0.991)	2891.638(0.980)

TABLE 7.2
Time and efficiency of parallel program using composite scheme computed on sugar.fjfi.cvut.cz.



8. Conclusion. The efficiency results show that both OpenMP and MPI are suitable for paralleling programs computing numerical solution of the shallow water equation because its values for the grid that was large enough were never less than 85% during the testing. For a small grid there the communication time is higher than the processing time and therefore the efficiency is too small. One important result is that efficiency of the program using MPI was higher than efficiency of the program using OpenMP. This is due to the ideal conditions over the measuring. Any load of one processing unit will enlarge the computational time of the program using MPI while the program using OpenMP will be minimally affected. OpenMP contains built-in tools for load balancing that can be easily used even if the programmer did not originally intend to use it. The MPI programmer has to write his own load balancing which will enlarge the programming time.

Acknowledgment. The work was partly supported by the project No. LC06052 ”Jindřich Nečas Center for Mathematical Modelling” of the Ministry of Education, Youth and Sports of the Czech Republic.

REFERENCES

[1] *MPI: A Message-Passing Interface Standard.* http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html.

[2] *OpenMP Application Program Interface.* http://www.openmp.org/drupal/mp-documents/spec25.pdf.

[3] D. L. GEORGE, *Numerical Approximation of the Nonlinear Shallow Water Eguations with Topography and Dry Beds: A Godunov-Type Scheme.* University of Washington, Washington (2004). Master Theses.

[4] R. J. LEVEQUE, *Numerical Methods for Conservation Laws.* Birkhauser Verlag, Basel (1990).

[5] R. J. LEVEQUE, *Nonlinear Conservation Laws and Finite Volume Methods for Astrophysical Fluid Flow.* Springer-Verlag, University of Washington (1998).

[6] R. LISKA, AND B. WENDROFF, *Composite Schemes for Conservation Laws.* SIAM J. Numer. Anal. **35**, no. 6 (1998), 2250–2271, .

[7] R. LISKA, AND B. WENDROFF, *2D Shallow Water Equations by Composite Schemes.* Int. J. Numerical Methods in Fluids **30** (1999), 461–479.

[8] M. ŠENKÝŘ, J. MIKYŠKA, AND M. BENEŠ, Application of Parallel Computing Techniques for Problems of Degenerate Diffusion. In *Numerical Mathematics and Advanced Applications, ENUMATH 2003 (peer reviewed proceedings)*, pages 756–766, Springer Verlag, Berlin, 2004. eds. M. Feistauer, VDolejší, P. Knobloch, K. Najzar, ISBN 3-540-21460-7.