Parallel algorithms for numerical solution of the initial-boundary value problems for the quasi-linear heat equation

Tokareva V. A.,[1, *] Streltsova O. I.,[2] and Zuev M. I.[2]

[1]*Department of Radio Engineering and Cybernetics, Moscow Institute of Physics and Technology, RU-141700 Dolgoprudny, Russia*

[2]*Laboratory of Nuclear Problems, Joint Institute for Nuclear Research, RU-141980 Dubna, Russia*

(Dated: September 5, 2015)

Abstract

During Summer Student Program 2015 at JINR:

- I've taken part in the tutorial "Parallel programming technologies on hybrid architectures" on MPI, CUDA, OpenMP and their applications 20 July — 10 August 2015;
- I've developed parallel algorithms for solving number of problems using locally one-dimensional schemes with technology CUDA and OpenMP;
- Developed algorithms have been tested and launched on heterogeneous cluster HybriLIT;
- I've carried out a comparative study of speed of the mentioned algorithms on different architectures, that is presented in this report.

Contents

INTRODUCTION

Actual problems of modern physics lead to the need for solving the equations of mathematical physics, described in [1–3]. One of the basic groups of such problems is related to the heat equation.

Therefore, it is actualy to develop and implement a variety of parallel algorithms for solving initial-boundary value problems for heat equations for different approaches to the selection of schemes for the numerical solution.

OBJECTIVES OF THE RESEARCH

Mathematical model

We seek a solution of the initial-boundary value problem for the two-dimensional heat equation:

$$c\rho\frac{\partial U}{\partial t} = \sum_{\alpha=1}^{2}\frac{\partial}{\partial x_\alpha}\left(k_\alpha(x,t)\frac{\partial U}{\partial x_\alpha}\right) + f(x,t), x \in D, t > 0, \tag{1}$$

where $U = U(x,t)$ is the temperature, which is the function of the spatial coordinates $x = (x_1, x_2)$ and time t, $k_\alpha(x,t)$ are the coefficients of temperature conductivity, $f(x,t)$ is the power density of the heat sources/sinks, D is the bounded rectangular area, $\rho = 1$ is density and $c = 1$ is constant . The solution is sought at the initial condition:

$$U(x,0) = U_0, \tag{2}$$

and the first-type boundary conditions:

$$\begin{cases} U(0,t) = \mu_1(t), \\ U(l,t) = \mu_2(t). \end{cases} \tag{3}$$

Selected instruments of implementation

For approximate numerical solution was chosen finite difference method, as one of the most versatile and common methods.

Locally-dimensional economic scheme was chosen for research, because it's combine the best features of explicit and implicit schemes. As implicit schemes, this scheme is absolutely stable, but it requires a relatively small amount of computation as the explicit schemes.

Implementation of algorithms for the two-dimensional heat equation was done by extending the separately developed parallel algorithm for the one-dimensional heat equation using the OpenMP technology and including the functions from the specialized libraries for solving systems of linear algebraic equations such as MKL [4] for CPU and . A study was conducted on how the solution time depends on the problem size and threads for this algorithm on CPU.

For the two-dimensional heat equation the calculation time dependence from the node number selection was studied for the CPU implementation with executing the algorithm on the IBM Power8 server and for the GPU implementation with executing on the GPU blades of the HybriLIT [5] clusters with the Tesla K40S graphic cards.

MEANS OF IMPLEMENTATION

Locally one-dimensional schemes

For the numerical solution of the problem (1) in the area $D \times [0 \leq t \leq T]$ we build a locally one-dimensional scheme [1–3].

Let the grid be $\omega = \omega_\tau \times \omega_{h_x h_y}, \quad \omega_\tau = \{t_j = j\tau, \quad j = 0, \ldots, N_t - 1\}$,

$$\omega_{h_x h_y} = \{(x_{i_1}, y_{i_2}): \quad x_{i_1} = x_l + i_1 h_x, \;\; i_1 = 0, \ldots, N_x - 1; \;\; y_{i_2} = y_l + i_2 h_y, \;\; i_2 = 0, \ldots, N_y - 1\}.$$

Let's find an approximate solution of the equation (1) by successive solving of one-dimensional equations

$$\frac{\partial \upsilon_{(\alpha)}}{\partial t} = L_\alpha \upsilon_{(\alpha)} + \varphi_\alpha, \qquad t_j \leq t \leq t_{j+1}, \tag{4}$$

where $L_\alpha \upsilon_{(\alpha)} = \frac{\partial}{\partial x_\alpha}\left(k_\alpha(x,t)\frac{\partial U}{\partial x_\alpha}\right), \;\; \sum_{\alpha=1}^{2} \varphi_\alpha = f$ (further we take $\varphi_1 = f$ and $\varphi_2 = 0$) and the following coupling conditions are fulfilled:

$$\begin{cases} \upsilon^j = \upsilon^j_{(1)}, \\ \upsilon^{j+1}_{(2)} = \upsilon^j_{(1)}. \end{cases} \tag{5}$$

Solution of the problem is sought as $L_1 \to L_2$.

We write out explicitly the difference scheme for the solution of the desired equation. In the equation (4) we approximate at each layer

$$\frac{\partial \upsilon_{(\alpha)}}{\partial t} \approx \frac{\upsilon^{i,j+1}_{(\alpha)} - \upsilon^{i,j}_{(\alpha)}}{\tau},$$

$$L_\alpha \approx \Lambda_\alpha,$$

where

$$\Lambda_\alpha \upsilon_{(\alpha)}^{i,j+1} = (a_\alpha(x_i,t)\upsilon_{\overline{x}})_{x,i} =$$

$$\frac{1}{h^2}\left(a_\alpha(x_{i+1},t)(\upsilon_{(\alpha)}^{i+1,j+1} - \upsilon_{(\alpha)}^{i,j+1})) - a_\alpha(x_i,t)(\upsilon_{(\alpha)}^{i,j+1} - \upsilon_{(\alpha)}^{i-1,j+1})\right),$$

where $a_\alpha(x_i,t) = \frac{k(x_i,t)+k(x_{i-1},t)}{2}$.

So we can write a chain of one-dimentional schemes:

$$\frac{\upsilon_1^{j+1} - \upsilon_1^j}{\tau} = \Lambda_1 \upsilon_1^{j+1} + \varphi_1, \quad \Lambda_1 \upsilon = (a_1(x_i,t)\upsilon_{\overline{x}})_{x,i}, \tag{6}$$

$$\frac{\upsilon_2^{j+1} - \upsilon_2^j}{\tau} = \Lambda_2 \upsilon_2^{j+1} + \varphi_2, \quad \Lambda_2 \upsilon = (a_2(x_i,t)\upsilon_{\overline{x}})_{x,i}, \tag{7}$$

Thus,

$$\Lambda_\alpha \upsilon_{(\alpha)}^{i,j+1} = \frac{1}{2h^2}\left((k(x_{i+1},t)+k(x_i,t))(\upsilon_{(\alpha)}^{i+1,j+1} - \upsilon_{(\alpha)}^{i,j+1})) - \right.$$
$$\left.(k(x_i,t)+k(x_{i-1},t))(\upsilon_{(\alpha)}^{i,j+1} - \upsilon_{(\alpha)}^{i-1,j+1})\right).$$

For the equation (4) we obtain the following representation in the form of an explicit scheme:

$$\frac{\upsilon_{(\alpha)}^{i,j+1} - \upsilon_{(\alpha)}^{i,j}}{\tau} = \frac{1}{2h^2}\left((k(x_{i+1},t)+k(x_i,t))(\upsilon_{(\alpha)}^{i+1,j+1} - \upsilon_{(\alpha)}^{i,j+1})) - \right.$$
$$\left.-(k(x_i,t)+k(x_{i-1},t))(\upsilon_{(\alpha)}^{i,j+1} - \upsilon_{(\alpha)}^{i-1,j+1})\right) + \varphi_\alpha.$$

Numerical experiments show that the symmetrized locally one-dimensional scheme $0.5\Lambda_1^{(1)} \to 0.5\Lambda_2^{(1)} \to 0.5\Lambda_2^{(1)} \to 0.5\Lambda_1^{(1)}$ has greater precision compared to the considered formal scheme.

We use $\upsilon_{(\alpha)}^{i,j} = \upsilon_{(\alpha-1)}^{i,j+1}$.

Thus, further we will consider

$$\frac{\upsilon_{(\alpha)}^{i,j+1} - \upsilon_{(\alpha-1)}^{i,j+1}}{\tau} = \frac{1}{4h^2}\left((k(x_{i+1},t)+k(x_i,t))(\upsilon_{(\alpha)}^{i+1,j+1} - \upsilon_{(\alpha)}^{i,j+1}) - \right.$$
$$\left.-(k(x_i,t)+k(x_{i-1},t))(\upsilon_{(\alpha)}^{i,j+1} - \upsilon_{(\alpha)}^{i-1,j+1})\right) + \varphi_\alpha.$$

Let us bring this system of equations to the form $Ax = B$, which is equivalent to

$$A_i x_{i-1} + C_i x_i + B_i x_{i+1} = F_i. \tag{8}$$

The coefficients for the different indices υ on the $(j+1)$-th layer will be:

$$\begin{aligned} \upsilon_{(\alpha)}^{i+1,j+1}: &\ \frac{(k(x_{i+1},t)+k(x_i,t))}{4h^2} \to A_i, \\ \upsilon_{(\alpha)}^{i-1,j+1}: &\ \frac{(k(x_i,t)+k(x_{i-1},t))}{4h^2} \to B_i, \\ \upsilon_{(\alpha)}^{i,j+1}: &\ \frac{1}{\tau} - \frac{(k(x_{i+1},t)+k(x_i,t))}{4h^2} - \frac{(k(x_i,t)+k(x_{i-1},t))}{4h^2} \to C_i, \\ &\ \varphi_\alpha + \frac{\upsilon_{(\alpha-1)}^{i,j+1}}{\tau} \to F_i. \end{aligned}$$

So we brought the problem to the form $Ax = B$ at each time layer, where A_i, B_i, C_i and F_i are coefficients of Thomas algorithm [2] (sweep method).

Thomas algorithm

The algorithm used for solving SLAEs is written as $Ax = B$, where A — tridiagonal matrix. It is a variantion of the method of successive elimination of unknowns.

Method is based on the assumption that the sought variables can be expressed using the following recurrence relation:

$$x_i = \alpha_{i+1}x_{i+1} + \beta_{i+1}, \tag{9}$$

where $i = n-1, n-2, \dots, 1$.

Using this relation, we can get $x_i - 1$ and x_i from $x_i + 1$, and substitute in equation (8):

$$(A_i\alpha_i\alpha_{i+1} + C_i\alpha_{i+1} + B_i)\, x_{i+1} + A_i\alpha_i\beta_{i+1} + A_i\beta_i + C_i\beta_{i+1} - F_i = 0,$$

where F_i is the right part of $i-$th equation.

It will be valid regardless of the solution, if

$$\begin{cases} A_i\alpha_i\alpha_{i+1} + C_i\alpha_{i+1} + B_i = 0, \\ A_i\alpha_i\beta_{i+1} + A_i\beta_i + C_i\beta_{i+1} - F_i = 0. \end{cases} \tag{10}$$

This implies:

$$\begin{cases} \alpha_{i+1} = \dfrac{-B_i}{A_i\alpha_i + C_i}, \\ \beta_{i+1} = \dfrac{F_i - A_i\beta_i}{A_i\alpha_i + C_i}. \end{cases} \tag{11}$$

From first equation in (11):

$$\begin{cases} \alpha_2 = -\dfrac{B_1}{C_1}, \\ \beta_2 = \dfrac{F_1}{C_1}. \end{cases} \tag{12}$$

After finding sweep coefficients α and β, we can solve SLAE, using equation (9).The solution is then obtained by substituting back:

$$x_i = \alpha_{i+1}x_{i+1} + \beta_{i+1}, \quad i = n-1, \dots, 1$$

$$x_n = \frac{F_n - A_n\beta_n}{C_n + A_n\alpha_n}.$$

HybriLIT computing cluster

Cluster HybriLIT consist of seven blades, including four nodes with GPUs NVIDIA Tesla K40, one node with coprocessors Intel Xeon Phi 7120P, and the node containing NVIDIA Tesla K20x and co-processor Intel Xeon Phi 5110P. All computing nodes contains two 12-core processors Intel Xeon E5-2695v2. The structure of heterogeneous computing cluster is shown in Fig.1.

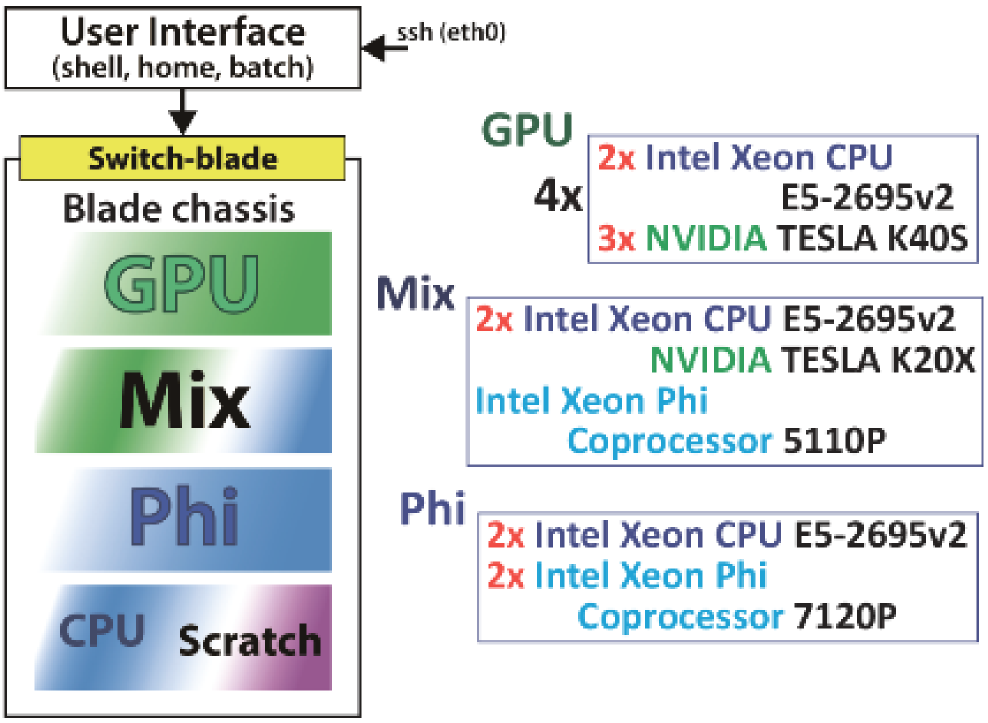
User Interface
(shell, home, batch)
ssh (eth0)
Switch-blade
Blade chassis
GPU
Mix
Phi
CPU
Scratch
GPU
4x
2x Intel Xeon CPU E5-2695v2
3x NVIDIA TESLA K40S
Mix
2x Intel Xeon CPU E5-2695v2
NVIDIA TESLA K20X
Intel Xeon Phi Coprocessor 5110P
Phi
2x Intel Xeon CPU E5-2695v2
2x Intel Xeon Phi Coprocessor 7120P

Figure 1: The structure of the heterogeneous cluster HybriLIT.

All the research stages have been conducted on a heterogeneous cluster HybriLIT, using the following software: tables I–III.

Table I: HybriLIT software configuration.

Operating system	Scientific Linux 6.6. Kernel version: 2.6.32-504
File system	NFS4
Task scheduler	SLURM-14.11.6-3
Modules	MODULES 3.2.10

Table II: Available technologies of parallel programming and their implementation

MPI	OpenMPI 1.6.5, 1.8.1
CUDA	CUDA 5.5, 6.0, 7.0
OpenMP	GNU 4.4.7, Intel 14.0.2, PGI 15.3
OpenCL	Intel 14.0.2, CUDA 6.0, 7.0

Table III: Available C/C++ compilers

GNU:	gcc, g++
Intel:	icc, icpc, mpiicc, mpiicpc
CUDA:	nvcc
OpenMPI:	mpicc, mpicxx

OpenMP

Where is algorithm for solving the problem using the OpenMP [6] technology on CPU:

1. Initialization, i.e. define variables, allocate memory on CPU, initialization of zero time layer;
2. Then for each time layer from 1 to N_t:
   (a) Calculating the lower, upper and main diagonals and the right side of SLAEs corresponding to the three-point difference equations (6), $i = \overline{1, N_y - 1}$;
   (b) Parallel solving of $N_y - 1$ tridiagonal SLAEs using the dgesv(int* n, int* nrhs, double* a, int* lda, int* ipiv, double* b, int* ldb, int* info) function in the direction of x;

   (c) Calculating the lower, upper and main diagonals and the right side of SLAEs corresponding to the three-point difference equation (7), $i = \overline{1, N_x - 1}$;

   (d) Parallel solving of $N_x - 1$ tridiagonal SLAE using the dgesv(int* n, int* nrhs, double* a, int* lda, int* ipiv, double* b, int* ldb, int* info) function in the direction of y;

3. Counting elapsed time, printing resuts.

Note that for the non-parallel version of the computational scheme the tridiagonal SLAEs were solved by the tridiagonal matrix algorithm [2, 3].

CUDA

The algorithm for solving the problem (1)-(3) on GPU using the CUDA technology [7] is the following:

1. Initialize the input data: define variables, allocate memory on GPU, duplicate the parameters of the problem, computational methods and other ones in the constant memory of GPU;

2. Initialize descriptors of cuSPARSE library [8] functions;

3. Then for each time layer:

   (a) Calculating the lower, upper and main diagonals and the right side of SLAEs corresponding to the three-point difference equations (6), $i = \overline{1, N_x - 1}$;

   (b) Parallel solving of $N_x - 1$ tridiagonal SLAEs using the cusparseDgtsvStrided-Batch() function in the direction of y;

   (c) Calculating the lower, upper and main diagonals and the right side of SLAEs corresponding to the three-point difference equations (7), $i = \overline{1, N_y - 1}$;

   (d) Parallel solving of $N_y - 1$ tridiagonal SLAEs using the cusparseDgtsvStrided-Batch() function in the direction of x;

   (e) Transferring calculated elements v_2 to v_1 with the cudaMemcpy() function.

TESTING

As part of implementation a test of the parallel algorithm was conducted for one-dimensional heat equation with first-type boundary conditions.

The problem is formulated as follows:

$$\begin{cases} \dfrac{\partial U}{\partial t} = \dfrac{\partial^2 U}{\partial x^2} + f(x,t), \quad x \in [x_r, x_l], t > 0, \\ U(x,0) = U_0, \\ U(x_r,t) = \mu_1, \\ U(x_l,t) = \mu_2. \end{cases} \tag{13}$$

where $U = U(x,t)$ is the temperature as the function of coordinate $x \in [x_r, x_l]$ and time $t > 0$, $f(x,t)$ is the power density of sources/sinks of heat, x_r, x_l are the borders of the closed segment.

It was solved by using the computational difference scheme with weights [1, p.416]:

$$\frac{\upsilon^{n+1} - \upsilon^n}{\tau} = \sigma\Lambda(\upsilon^{n+1}) + (1-\sigma)\Lambda(\upsilon^n) + f, \quad t = 0, ..., N_t - 1, \quad \sigma \in [0,1],$$

leading to the system of linear algebraic equations (SLAE) of the following kind

$$\frac{\sigma}{h^2}\upsilon_{i+1}^{n+1} - \left(\frac{2\sigma}{h^2} + \frac{1}{\tau}\right)\upsilon_{i+1}^{n+1} + \frac{\sigma}{h^2}\upsilon_{i-1}^{n+1} = -\left(f_i + (1-\sigma)\frac{\upsilon_{i+1}^n - 2\upsilon_i^n + \upsilon_{i-1}^n}{h^2} + \frac{\upsilon_i^n}{\tau}\right),$$
$$i = 1, ..., N-1, \quad t = 0, ..., N_t - 1, \quad \sigma \in [0,1],$$

with grid steps h and τ on x and t respectively, solved by the tridiagonal matrix algorithm described in [3].

Parallel implementation of the algorithm was performed using the MKL library, supported with OpenMP technology. The implementation of Intel 14.0.2 using the icpc compiler was utilized to run the application on the CPU.

Testing of the computing scheme was conducted on problems with the exact solution. Accuracy of calculations using the scheme was considered by the equation:

$$\varepsilon_n = \max_{0 \le i \le N} \mid \upsilon_i - U_i \mid$$

for each time layer. When testing, the number of nodes of the grid N and time layers N_t was varied.

The average execution time of the algorithm with different number of grid nodes for different number of threads has been studied to identify their optimal number for the future calculations.

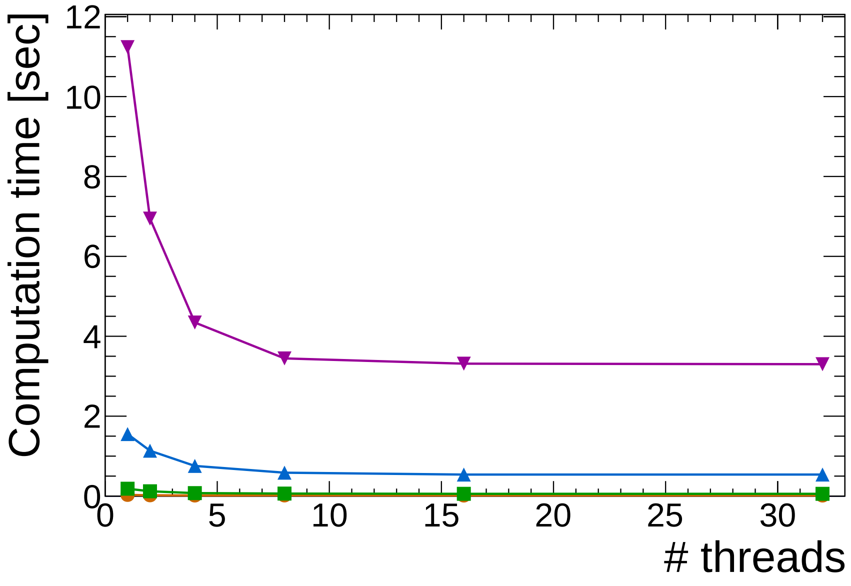

Figure 2: Calculation time for the two-dimensional heat equation vs. the number of threads, ● — 2^9 nodes for X, ■ — 2^{10} ones, ▲ — 2^{11} ones, ▼ — 2^{12} ones.

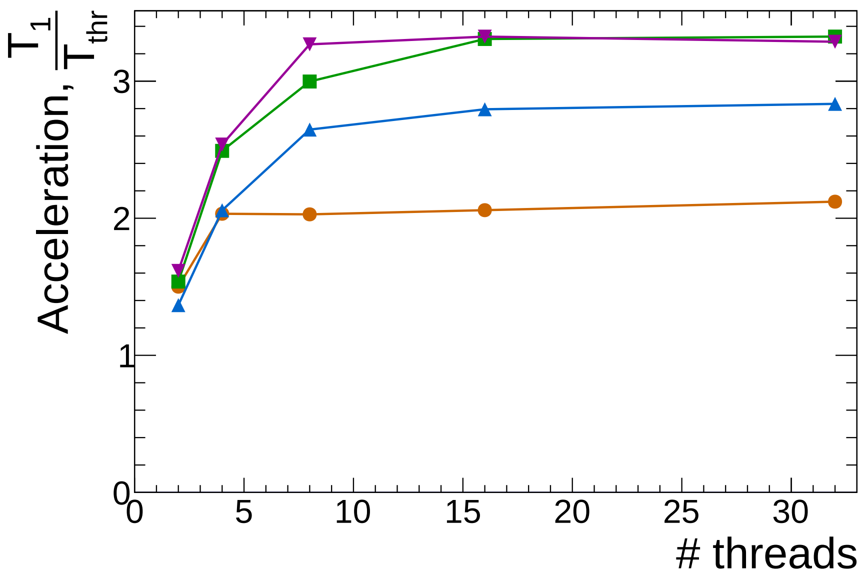

Figure 3: Acceleration (defined as calculation time with one thread divided by calculation time with multiple threads) for the two-dimensional heat equation vs. the number of threads, ● — 2^9 nodes for X, ■ — 2^{10} ones, ▲ — 2^{11} ones, ▼ — 2^{12} ones.

Number of nodes for x, N	Number of threads					
	1	2	4	8	16	32
512	0.02958	0.01970	0.01457	0.01644	0.01399	0.01410
1024	0.18419	0.12021	0.07418	0.06266	0.05633	0.05623
2048	1.55204	1.13537	0.75476	0.58538	0.53834	0.53874
4096	11.2372	6.94576	4.34454	3.44586	3.31430	3.30106

Table IV: Computation time for OpenMP realization depending on problem size and number of threads (in seconds).

PARALLEL IMPLEMENTATION

Testing the developed algorithm and the subsequent calculations were performed on the JINR hybrid computing cluster HybriLIT and the IBM Power8 server.

Each blade of the hybrid computing cluster has the characteristics described above and contains 128 GB of RAM. Calculations on a cluster were carried out on the GPU blades with the graphics processors Nvidia Tesla K40S.

At the IBM Power8 server calculations were carried out on the sites with features 2×10-core 3.42 GHz POWER8 Processor Card, 32 GB of RAM, 2 x NVIDIA Tesla K40m 12 Gb 2880 CUDA-cores, the operating system Ubuntu 14.10.

Testing of the computing scheme was conducted using problems with the exact solution. Accuracy of calculations of the difference scheme (2), (3) was checked by using the Runge method by calculation on the sequence of the condensing grids. Algorithm launches were made with different numbers of time layers and grid steps in x and y.

The study was aimed to determine the relation between the threads (blocks), minimizing the time of calculations. It was found that the minimum is reached at a ratio of 32×32. These values were chosen for all subsequent calculations.

Here are the results of calculations for various dimensions of the problem, with the number of time layers $N_t = 100$.

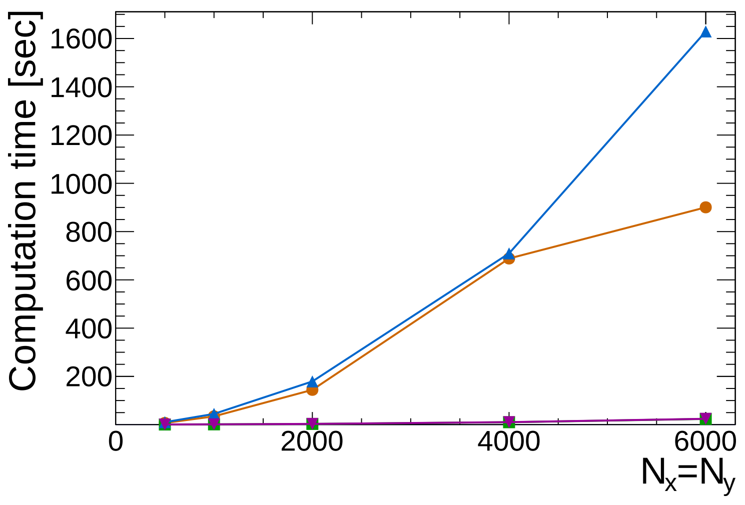

Figure 4: Calculation time for the two-dimensional heat equation vs. the number of nodes, ▼ — on HybriLIT CPU blades, 🟩 — on IBM Power8 CPUs,● — on HybriLIT GPUs, ▲ — on IBM Power8 GPUs.

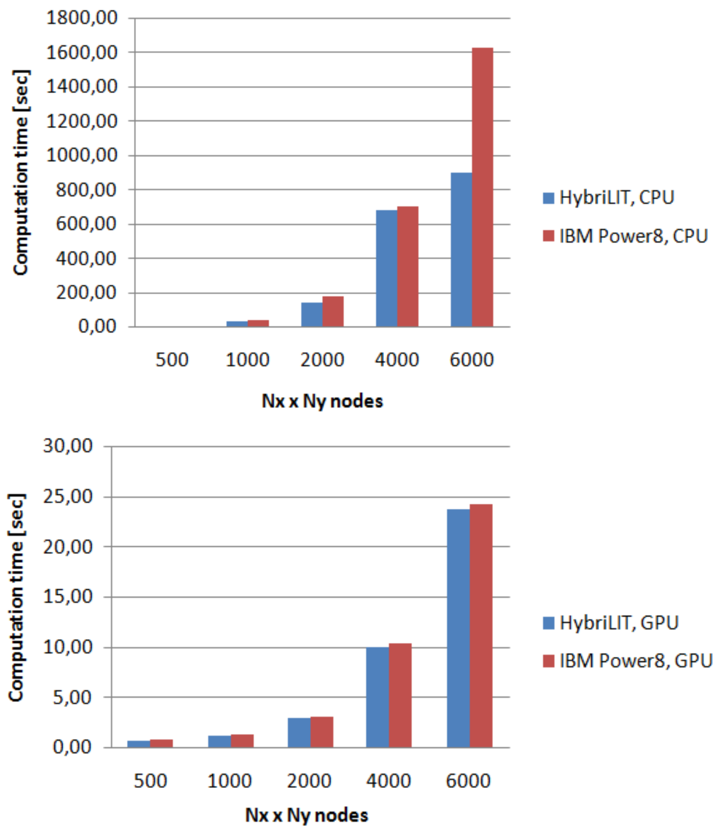

Figure 5: Calculation time for the two-dimensional heat equation vs. the number of nodes, top panel — CPUs, bottom panel — GPUs.

Table V: Calculation time for the task on CPU and GPU (seconds)

Number of nodes,	HybriLIT		IBM Power8	
$N_x \times N_y$	CPU	GPU	CPU	GPU
500×500	7.14	0.761	10.34	0.877
1000×1000	34.52	1.312	44.65	1.353
2000×2000	144.316	3.045	179.183	3.214
4000×4000	688,498	10.121	709.579	10.455
6000×6000	900.56	23.791	1629.26	24.326

SUMMARY

Lectures were listened about such parallel programming technologies as CUDA, OpenMP, MPI. I have received skills on working at HybriLIT and IBM Power8 servers.

The resulting knowledge has been applied to developing parallel algorithms for one-dimensional and two-dimensional heat equations. I performed their parallel implementation using OpenMP and CUDA technologies and tested them using resources provided by the cluster.

An acceleration has been studied for various numbers of grid nodes and threads on the CPU and GPU architectures.

Parallel algorithms allow reducing the time of calculations by the factor of 3.3 using the multicore component of the cluster (where OpenMP is the algorithm realization) and by the factor of up to 70 using the graphical accelerators (where CUDA is the algorithm realization).

* Electronic address: `tokareva.v.a@yandex.ru`

[1] Samarskii A.A., Tichinov A.N. *Equations of Mathematical Physics.* — Courier Corporation, 1963. — p. 765

[2] Samarskii A.A. *Introduction into theory of difference schemes.* — M.: Nauka, 1971. — p. 552.

[3] Samarskii A.A., Gulin A.V. *Numerical methods.* — M.: Nauka, 1989. — p. 432.

[4] Intel Math Kernel Library, web source – http://software.intel.com/en-us/intel-mkl

[5] Heterogeneous cluster HybriLIT. Official site. – http://hybrilit.jinr.ru/

[6] Official OpenMP web source: – http://www.openmp.org/

[7] Cuda parallel computing, official Nvidia web source – http://www.nvidia.ru/object/cuda-parallel-computing-ru.html

[8] Cuda toolkit documentation: cuSPARSE – http://docs.nvidia.com/cuda/cusparse/#abstract

[9] Boreskov A.B., Charlamov A.V. *Basics of work with CUDA.* — M.: DMKPress, 2010. — 232 p.

[10] Levin M.P. *Parallel programming with using OpenMP.* - M: Binom, 2012. — p. 118.

[11] Lupin S.A., Posypkin M.A. *Technologies of parallel programming.* — M.: Infra-M, 2011. — p. 202.

[12] Summer Student Program 2014 at Joint Institute of Nuclear Research. http://students.jinr.ru/index.php/ru/summer-2014