THE USE OF PARALLEL PROGRAMMING TECHNOLOGIES FOR MODELING SEISMIC WAVES USING GRID-CHARACTERISTIC METHOD

[1]Andrey M. Ivanov, [1]Nikolay I. Khokhlov, [1,2]Igor B. Petrov

[1]Moscow Institute of Physics and Technology, http://mipt.ru
141701 Dolgoprudny, Moscow region, Russian Federation
[2]Scientific Research Institute of System Analysis, Russian Academy of Sciences, https://www.niisi.ru
117218 Moscow, Russian Federation
ip-e@mail.ru, k_h@inbox.ru, petrov@mipt.ru

Abstract. **The paper deals with the problem of modeling seismic wave propagation in solid deformable bodies. The state of these bodies can be represented as a solution of hyperbolic system of equations. Grid-characteristic method is used to find the solution. This formulation of physical problem is used to demonstrate the use of technologies of parallel programming for shared memory systems using OpenMP and POSIX Threads technology. To achieve high performance, vector SSE and AVX instructions of CPU is used. In addition, GPU implementation is written using CUDA and OpenCL technologies. Results of multi-GPU parallelisation is also obtained. Use of NVIDIA GPUDirect technology that allows GPUs to make data exchanges directly through the PCI Express bus is considered. The paper demonstrates achieved acceleration and percentage of the theoretically possible performance. The paper demonstrates achieved acceleration and percentage of the theoretically possible performance. The paper describes the optimizations, which allowed us to achieve the obtained acceleration results.**

The study was financially supported by RFBR as part of a research project number 15-07-01931 A.

Keywords: **mathematical modeling, parallel programming, grid-characteristic method, shared memory, seismic, vectorization, GPU, CUDA, OpenCL**

UDC 519.633

Bibliography - 20 references *Received 25.11.2016*

RENSIT, 2016, 8(2)185-195 **DOI: 10.17725/rensit.2016.08.185**

Contents

1. INTRODUCTION

The problem of wave propagation in linear-elastic medium is of interest in seismology and geophysics. With the increasing performance of modern computer systems and the possibility of parallel execution of the program code, it is possible to carry out computer modeling of seismic waves with great precision. The question is how to efficiently use the available computing resources.

The first solution to this problem is to use of technologies of parallel programming for shared memory systems. The article [1] considers the problem of modeling effects of parallelization occurring in the ground during an earthquake, using OpenMP. With regard to other problems of mathematical modeling, there is the paper [2]

about application of OpenMP technology to parallelize finite element method on unstructured grids.

The second approach is parallelization on modern GPUs. There are a lot of papers on this subject. For example, parallelization of finite difference method on the GPU with CUDA technology [4] is considered in the article [3]. There are studies [5, 6] where the problem of reverse time migration is solved on a cluster of GPUs.

Finally, a third approach is parallelism in distributed memory systems. CPUs and GPUs may also be used in such systems. In paper [7], finite-difference method of viscoelastic environment modeling is parallelized using MPI [8] technology. The work [9] considers spectral elements method for the simulation of wave propagation in the asteroid. In addition, the work [10] compares the performance of parallelizing on GPU with use of CPU clusters. The study [11] demonstrates the use of the hybrid OpenMP + MPI systems for seismic finite-difference modeling.

There are also other approaches, a detailed consideration of which is beyond the scope of this article. For example there are the use of the FPGA [12] and ASIC [13].

In this paper we consider the solution of hyperbolic systems of equations that describes the behavior of waves in elastic solid bodies. To find the solution of this system grid-characteristic method [14–16] is used. Calculations are made on the explicit scheme, because in this case program responds well to parallelization. Solution of the problem is parallelized using shared memory systems with OpenMP, POSIX Threads technologies [17] and on GPUs using CUDA, OpenCL [18].

2. MATHEMATICAL MODEL

Environment behavior is described by the model of an ideal isotropic linear-elastic material. We consider the two-dimensional problem. The following system of partial differential equations describes the state of the elementary volume of elastic material in the approximation of small deformations:

$$\rho\frac{\partial v_x}{\partial t}=\frac{\partial \sigma_{xx}}{\partial x}+\frac{\partial \sigma_{xy}}{\partial y},\quad \rho\frac{\partial v_y}{\partial t}=\frac{\partial \sigma_{xy}}{\partial x}+\frac{\partial \sigma_{yy}}{\partial y},$$

$$\frac{\partial \sigma_{xx}}{\partial t}=(\lambda+2\mu)\frac{\partial v_x}{\partial x}+\lambda\frac{\partial v_y}{\partial y},$$

$$\frac{\partial \sigma_{yy}}{\partial t}=\lambda\frac{\partial v_x}{\partial x}+(\lambda+2\mu)\frac{\partial v_y}{\partial y},$$

$$\frac{\partial \sigma_{xy}}{\partial t}=\mu\left(\frac{\partial v_x}{\partial x}+\frac{\partial v_y}{\partial y}\right),$$

where ϱ is the density of the medium; λ, μ are Lame parameters; v_x and v_y are the horizontal and vertical components of the velocity of the particles of the medium; σ_{xx}, σ_{yy}, σ_{xy} are the components of the stress tensor.

This system can be presented in the matrix form:

$$\frac{\partial u_p}{\partial t}+A_{pq}\frac{\partial u_q}{\partial x}+B_{pq}\frac{\partial u_q}{\partial y}=0, \quad (1)$$

where $\mathbf{u}$ is the vector of 5 independent variables $\mathbf{u} = (\sigma_{xx}, \sigma_{yy}, \sigma_{xy}, v_x, v_y)^T$. The explicit form of the matrices A_{pq}, B_{pq} is presented in [19]. Hereinafter we mean summation over repeated indices. The eigenvalues of matrices A_{pq} and B_{pq} are as follows: $s_1 = -c_p$, $s_2 = -c_s$, $s_3 = 0$, $s_4 = c_s$, $s_5 = c_p$, where c_p and c_s are the propagation speeds of elastic P-waves and S-waves in the medium.

3. NUMERICAL METHOD

Using coordinate-wise splitting we can reduce the problem of constructing a difference scheme for the system of equations (1), to the problem of constructing a difference scheme for system of the form:

$$\frac{\partial u_p}{\partial t}+A_{pq}\frac{\partial u_q}{\partial x}=0, \quad (2)$$

For hyperbolic system of equations (2) matrix $\mathbf{A}$ can be represented as $\mathbf{A} = \mathbf{R}\Lambda\mathbf{R}^{-1}$, where Λ is a diagonal, the elements of which are the eigenvalues of $\mathbf{A}$, and $\mathbf{R}$ is the matrix consisting of right eigenvectors of $\mathbf{A}$. We introduce new

variables: $\mathbf{w} = \mathbf{R}^{-1}\mathbf{u}$ (the so-called Riemann invariants). Then the system of equations (2) will be reduced to a system of 5 independent scalar advection equations.

Let's reduce a third-order accuracy scheme to the numerical solution of one-dimensional linear advection equation $u_t + au_x = 0$, $a > 0$, $\sigma = a\tau/h$, τ is a time step, h is step on coordinate:

$$u_m^{n+1} = u_m^n + \sigma(\Delta_0 + \Delta_2)/2 + \sigma^2(\Delta_0 - \Delta_2)/2 + \\ + \frac{\sigma(\sigma^2 - 1)}{6}(\Delta_1 - 2\Delta_0 + \Delta_2), \qquad (3)$$

$$\Delta_0 = u_{m-1}^n - u_m^n,$$

$$\Delta_1 = u_{m-2}^n - u_{m-1}^n,$$

$$\Delta_2 = u_m^n - u_{m+1}^n.$$

Scheme (3) is stable to Courant numbers not bigger than 1. A grid-characteristic criterion of monotony is used, it is based on the characteristic property of the accurate solutions:

$$\min(u_{m-1}^n, u_m^n) \le u_m^{n+1} \le \max(u_{m-1}^n, u_m^n).$$

In places where this criterion is met, the order of scheme falls to the second one.

Once the values of the Riemann invariants on the next time step are found, the solution: $\mathbf{u}^{n+1} = \mathbf{R}\mathbf{w}^{n+1}$.

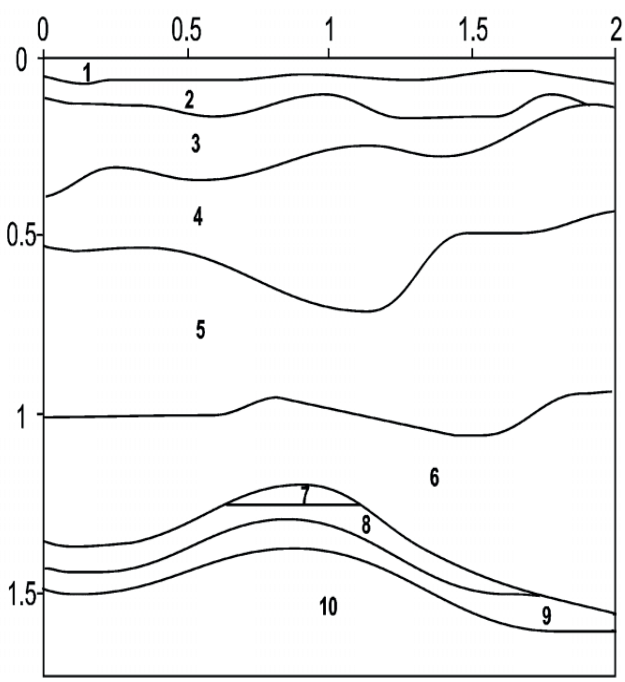

Fig. 1. *Geological model of anticlinal trap* [20].

4. STATEMENT OF PROBLEM

Test model is shown in **Fig. 1**. The dimensions are given in kilometers. A non-reflecting boundary condition is set for bottom and side borders and free boundary is set for the top. The source of perturbations is a vertical force applied to the site from 925.7 m to 974.1 m on the day surface; its amplitude is set by Ricker pulse frequency of 40 Hz. The calculation results are presented in **Fig. 2**.

5. PARALLELIZATION TEST CONDITIONS

All the CPU tests were carried out on the one test task. The two-dimensional test problem with the number of nodes 4000×4000 is considered. 100 time steps were carried out. All calculations were performed both with a single, and a double accuracy. In this paper only double-precision charts are given. Each grid point stored 5 floating point variables. Grid size in memory is 305 MB for computing with single precision and 610 MB for the calculations with double accuracy.

The next compiler optimization fl were used: -fno-tree-vectorize to restrict vectorization, -fopenmp and -pthread to specify parallelization technology, -O2 to use standard optimizations,

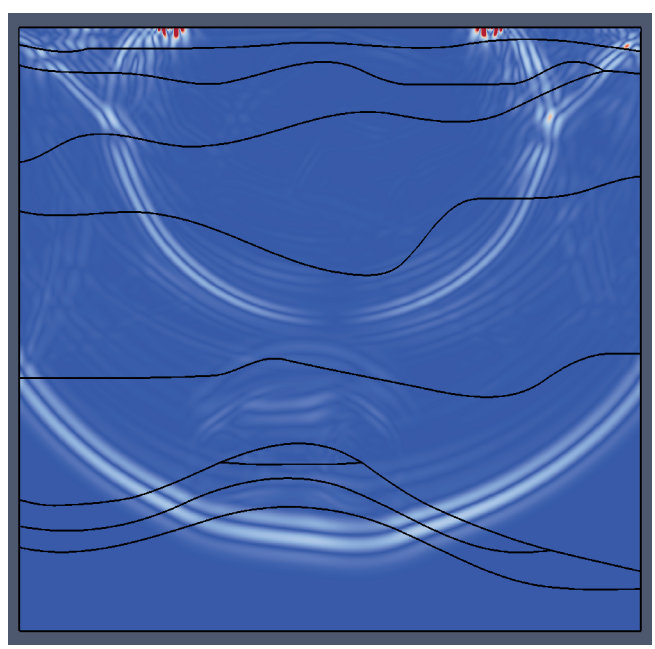

Fig. 2. *The calculation result is the wave pattern in time moment t = 0.38 s.*

Table 1

CPUs are used for parallelization tests

Name (notation)	Clock rate, GHz	Cache, kB	Processors and cores	Compilers	Performance, GFLOPS
AMD Opteron 6272 (a64)	2.1	2048	8 processors with 8 cores	gcc4.4.6	1075.2
Intel Xeon E5-2697 (i24)	2.7	30720	2 processors with 12 cores	gcc4.4.7, icc15.0.0	518.4
AMD Opteron 8431 (a48)	2.4	512	8 processors with 6 cores	gcc4.4.7	921.6

-msse, -mavx to specify a set of vectorization instructions.

Calculations were made on CPUs presented in **Table 1**. Compilers on charts are denoted as gcc and icc. Vector instructions sets are denoted as sse or avx.

All the CPU tests were carried out with another parameters. The two-dimensional test problem with the number of nodes 4096×4096 is considered. 6500 time steps were carried out. All calculations were performed both with a single (SP), and a double (DP) accuracy. Each grid point stored 5 floating point variables. Grid size in memory is 320 MB for computing with single precision and 640 MB for the calculations with double accuracy. Characteristics of used GPUs is shown in **Table 2**.

6. OPTIMIZATION PROCESS

The sequential version of the algorithm was written first. Recalculation of grid nodes was carried out in two steps. In the step *X* algorithm bypasses the grid by the columns, then by the rows. In the step *Y* it firstly bypasses the grid by the rows, then by the columns. It was measured the number of arithmetic operations in the code and using this value we obtained the theoretical value of required FLOPS. It was 262 to recalculate one grid node. On the charts, this version is denoted as simple.

Table 2

Characteristics of GPUs

GPU	Cores	Clock rate, MHz	GFLOPS (SP)	SP:DP	GFLOPS (DP)
GeForce GT 640	384	900	691	24	29
GeForce GTX 480	480	1401	1345	8	168
GeForce GTX 680	1536	1006	3090	24	129
GeForce GTX 760	1152	980	2258	24	94
GeForce GTX 780	2304	863	3977	24	166
GeForce GTX 780 Ti	2880	876	5046	24	210
GeForce GTX 980	2048	1126	4612	32	144
Tesla M2070	448	1150	1030	2	515
Tesla K40m	2880	745	4291	3	1430
Tesla K80	2496	562	2806	1.5	1870
Radeon HD 7950	1792	800	2867	4	717
Radeon R9 290	2560	947	4849	8	606

Next, an attempt was made to optimize the step *Y*. The sequence of nodes access was changed so that it coincided with the access order in step *X*. Thus the memory access was became sequential, leading to a more efficient use of cache memory. Furthermore, by replacing the arithmetical operations by equivalent, the required amount of FLOPS has been reduced to 190 for single node recalculation. The designation of this version on the charts is cf.

Then the cycles where grid nodes are recalculated were vectorized. It was done with the help of the OpenMP technology directives (#pragma omp simd), appearing in the standard 4.0. Explicitly defined a set of instructions to be used: SSE or AVX. These measurements were made only on Intel Xeon E5-2697.

Based on the frequency of each individual processor core, its peak performance can be obtained. Thus, the percentage of peak performance was calculated for serial version for each test (**Fig. 3**).

7. PARALLEL ALGORITHM

The algorithm has been parallelized using OpenMP and POSIX Threads technologies. Each program version has been parallelized separately to be able to observe the effects of

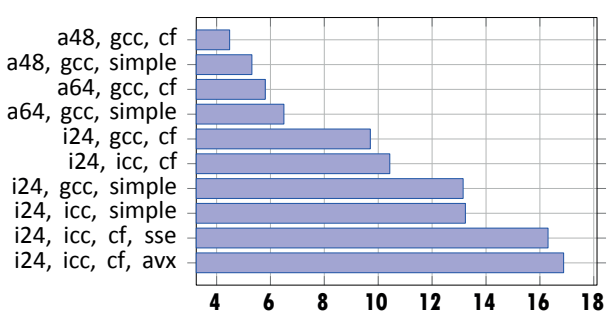

Fig. 3. *Percentage of peak performance.*

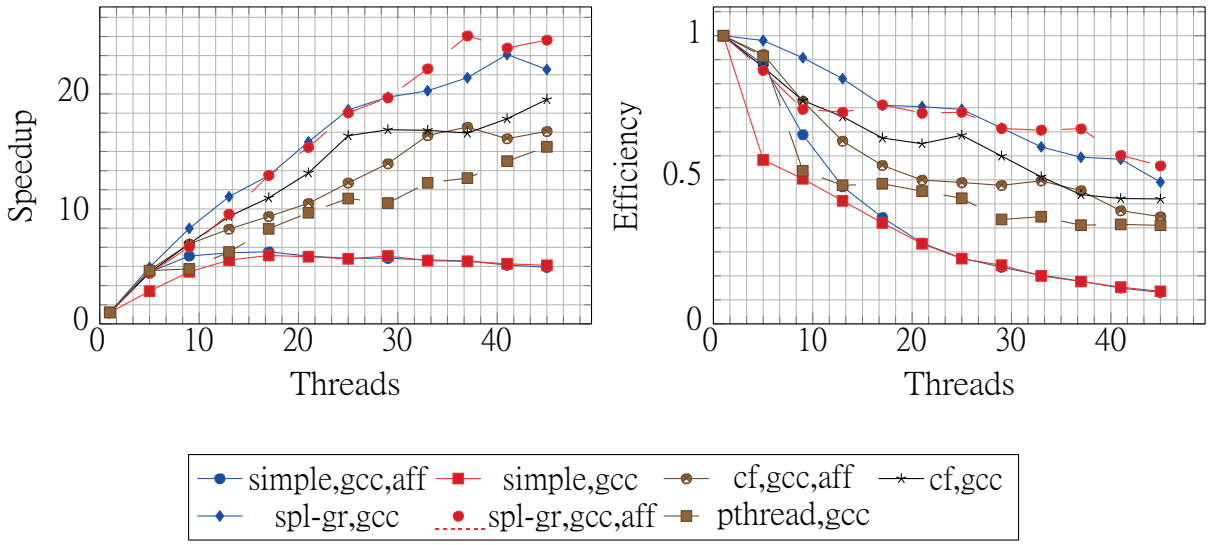

Fig. 4. *Parallelization results on AMD Opteron 8431.*

sequential code optimization on the efficiency of its parallelization.

Optimization of already parallel code has been made. Each thread allocates memory, which is only necessary to recalculate nodes in its part of grid. On the charts, this version is indicated as spl-gr. In the theory, this should lead to an increase in performance in NUMA systems, since a memory which is allocated on a specific thread is placed in the local memory of this thread according to OpenMP standard. Performance is increased because local memory access consumes less amount of time.

Separate measurements, where change of processor cores was forbidden for threads, were made. In other words a "binding" of threads to specifi processor cores was made. This is done to reduce the time costs. Without this binding, the optimization where the grid is divided into separate parts for different processes ceases to make sense. The results are shown in **Fig. 4 - 7**.

Next, OpenMP version of the code with grid splitting between the threads has been rewritten to use POSIX Threads as a parallelization technology.

8. GPU OPTIMIZATIONS

The CPU optimized version of this program was taken as the basis for the algorithm implementation on GPUs. The most

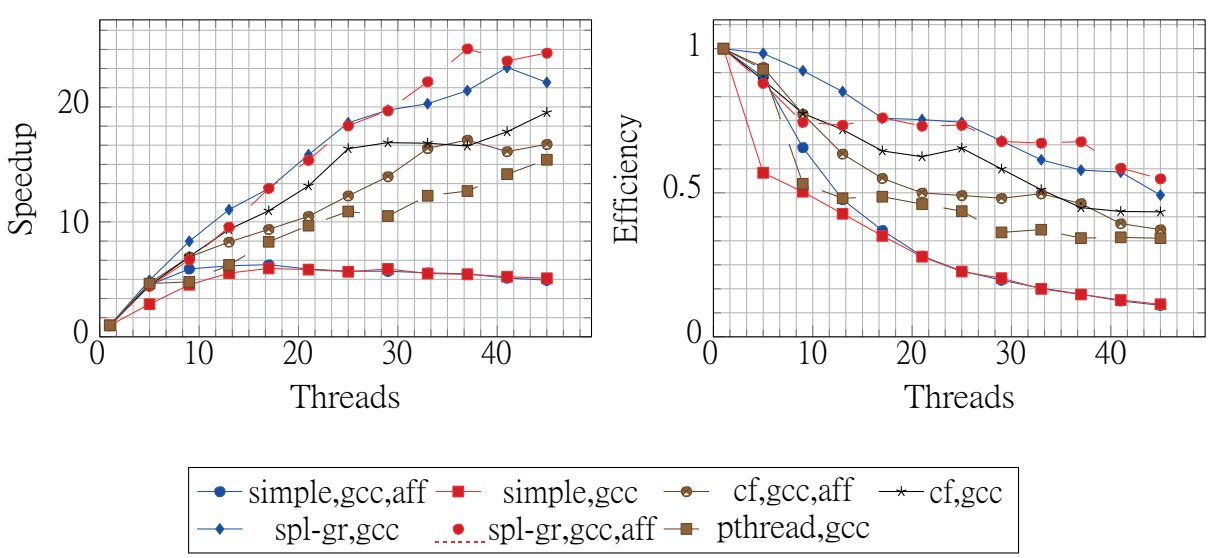

Fig. 5. *Parallelization results on AMD Opteron 6272.*

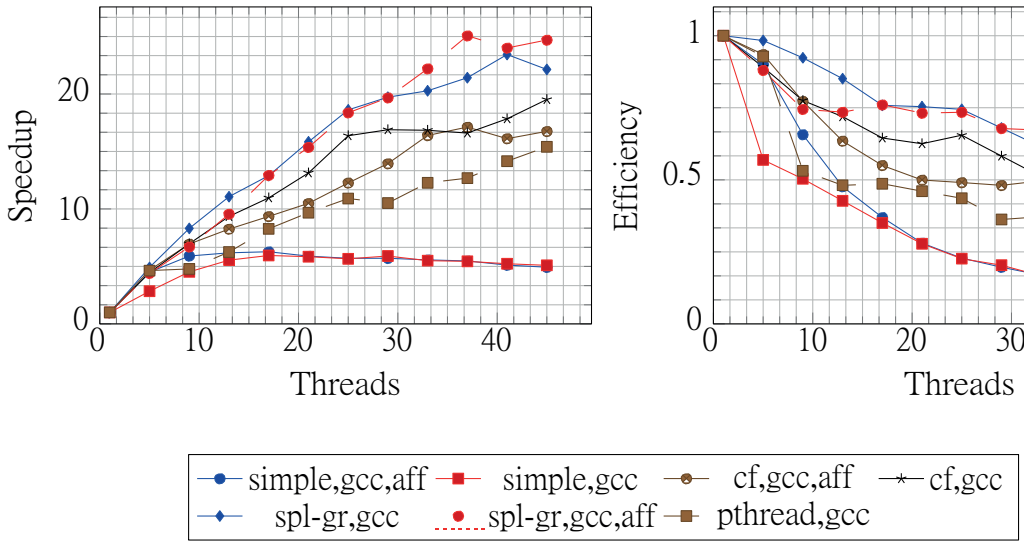

Fig. 6. *Parallelization results on Intel Xeon E5-2697 (without affinity).*

computationally expensive parts of the algorithm were optimized. As the spatial coordinate splitting was used, two steps were required to transfer the entire grid: on the *X* axis and *Y* axis. At that the number of arithmetic floating point operation was calculated, required for the conversion of one grid point in two steps – 190 Flops. Therefore, by knowing the number of grid nodes, the number of time steps, the theoretical amount of GFlops consumed by algorithm, can be determined. Next, knowing the number of stream processors in GPU, its clock frequency and the number of FMA (fused multiply-add) processors in a single processor, we calculated a peak performance for each GPU. The real algorithm tests on GPUs have shown lower values of performance.

Fig. 8 shows a test result. The percentage of the peak performance of algorithms was estimated as the ratio between two values – theoretically required amount of Flops for grid recalculation and real consumed amount of Flops.

8.1. Transferring implementation from CPU to GPU – cuda1

In the original version, the algorithm was redesigned for execution on GPUs using CUDA technology, but it is not optimized for execution on GPUs. In this technology, the graphic processor was assigned 2 times more memory than required to store the computational grid.

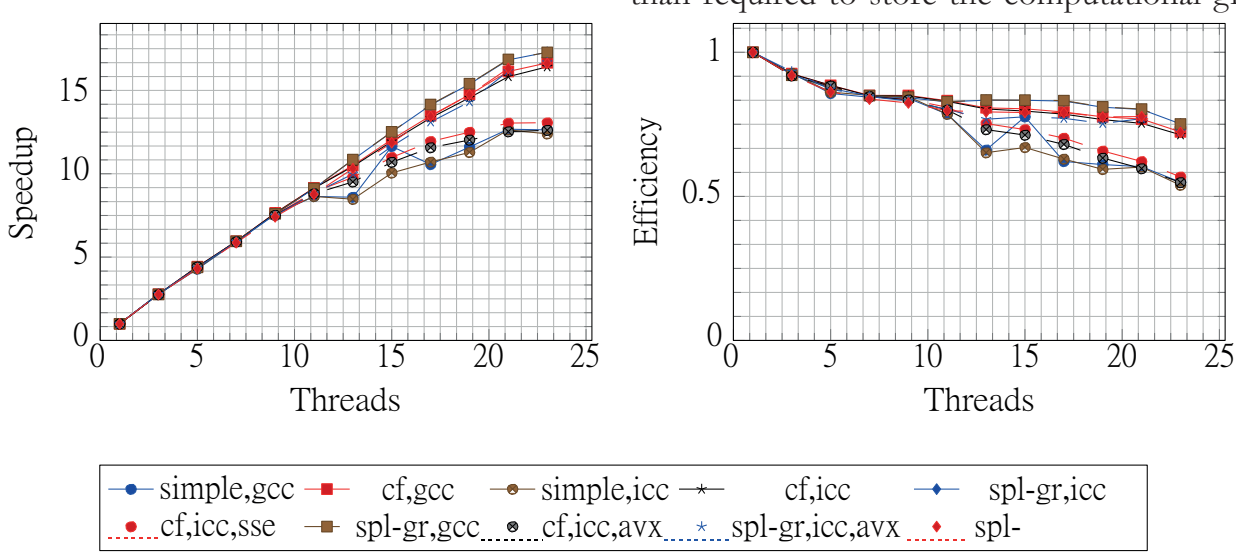

Fig. 7. *Parallelization results on Intel Xeon E5-2697 (with affinity).*

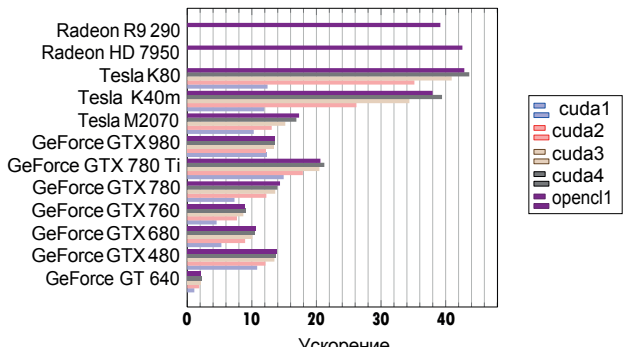

Fig. 8. *Percentage of peak performance.*

It is a standard practice when working with the algorithm designing technologies for GPUs. As a result, only the synchronization is performed between function calls that run on GPU (CUDA kernels). This was done due to the fact that global synchronization of all graphics processor causes large time delays, so these delays were inserted between calls of kernels. Thus, it became possible to reduce the number of global synchronization up to two times by one time step. Architecture of CUDA stream processors means that in one CUDA unit, flows that perform the same code on different areas of memory are executed simultaneously, if there are no branches in the program code. Therefore, a situation when all flows wait for completion of one occurs only if this flow executes any operations that differ from the rest.

All operations on the memory assigning on GPU and calls of functions running on the graphic processor, are produced by the host – CPU. Operations of memory grid copping from the host memory and back require a lot of time, so the grid is copied once from the host memory to GPU memory, before the start of the main computing, and once at the end from the GPU memory to host memory. Number of steps by the time and computational grid size was such that the time required for calculations far exceeds the time required to copy it.

Data can be stored in grid in two ways: in the form of the array of structures and in the form of arrays structure. The structure in this case means vector **u**, consisting of 5 components. In the original version, data in the computational grid were organized as an array of structures, i.e. data in a particular node are stored sequentially in the memory. To set the boundary conditions of the task, the behavior other than required by other flows at the boundary of the computational grid was required. To handle the grid boundaries in the kernel code, the code section with five conditional blocks "if-else" is inserted in the algorithm. Several options of the unit size were considered for this version. It was found that the optimal size is 16x16, at which operation time was minimal.

8.2. Structure of arrays and sequential memory access – cuda2

Shared memory was used in the next optimization of CUDA block. This is due to the fact that the latency during the interaction with this memory is less than the interaction with the global memory. Optimization means there is only one reading from the global memory at each step along X and Y at the beginning of a function call that is running on the GPU and the data is copied in total memory. Immediately after that all the flows are synchronized in the block; after that, all calculations are performed on the data in the shared memory block. At the end of kernel, the result is written to the second copy of the computational grid in the global memory. Another optimization was the selection of a different storing method of the calculated grid in GPU memory – array structure. After that, calls to the global memory become coalesced which led to the increase in performance. The above mentioned optimizations reduced the number of if-else blocks for grid boundaries processing up to 2.

8.3. Kernel call parameters – cuda3

Before that, additional information, such as grid size, material, variables values, resulting from intermediate calculations valid along all time

steps were passed to the structure in the GPU global memory as a pointer. All flows constantly addressed the same memory section. Such data were calculated on the CPU in CUDA3 algorithm version once before the execution of code on GPU and were transferred through the parameters of kernel call. It was expected that each flow would create a local copy of these values.

8.4. Block sizes – cuda4

It was important to choose the dimensions of the blocks so that the graphics processor would be constantly loaded, that is, there were no part-time loaded flow processors. It was also important to choose the block sizes multiple of the warp sizes, as the sequential access to memory is carried out within a warp. Another reason is that the flows of a warp can execute code simultaneously on different memory sections, if there are no branches.

In CUDA4 version, block sizes are selected so as to satisfy the above mentioned requirements and minimize the number of nodes which require memory exchanging between the memory blocks. At the step on *X* axis to convert each node of the computational grid, values in two adjacent nodes on the *X* axis are required, and at step on *Y*-axis, two adjacent nodes on the *Y*-axis are required. Therefore, one should choose the block sizes so that the number of nodes that require values in the adjacent blocks would be smaller. It was necessary, since the adjacent blocks of nodes require more memory in a general memory block.

If, for example, we take a block of $M \times N$ times, then in step *X* for storage of nodes of adjacent blocks $4N$, additional nodes are required, and in step Y – $4M$. Therefore, in step *X*, the size equal to 256×1 has been selected, as a result, block required only 4 additional nodes in memory. In step *Y*, the block size was set as 16×16 to reach a compromise between the number of additional memory (64 grid nodes) and the requirements for the serial memory access.

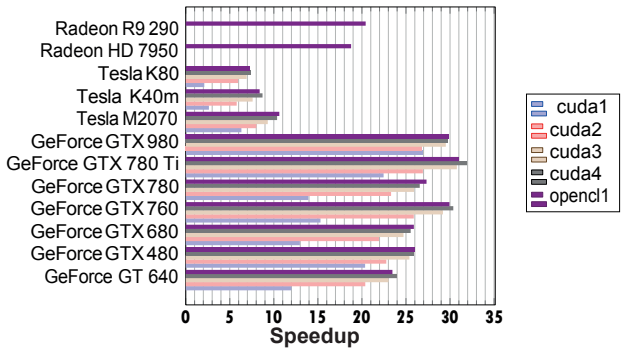

Fig. 9. *Acceleration on GPU compared to CPU core.*

8.5. Using opencl1

The next step was to create the OpenCL implementation of this algorithm. Optimized version of the CUDA4 algorithm was taken as the basis for this. Acceleration test results for all implementations are shown in **Fig. 9.** The maximum obtained acceleration compared with a single CPU on one graphic CPU – 55 times on the GeForce GTX 780 Ti in the computations of single precision and 44 times at Tesla K80 in double precision computations.

It good results for AMD devices should be noted, in spite of the fact that cheap desktop card were used, they showed good results on implementations of single-precision and double-precision.

9. MULTIPLE GPUs

The algorithm has been parallelized to run on multiple graphics processors. The most optimized version was used to perform several unused GPU. At that the computational grid was divided into several equal-sized rectangular areas. The division was made on the *Y* axis, due to the selected size of the block on *X* axis.

Tests with multiple GPUs were carried out only for the same GPUs. This was made due to the fact that by using GPUs of different performance, faster processors will cause downtime of slower graphics processors, and the effect of their simultaneous use may be less noticeable. Synchronization of computational grid between GPUs was carried out by sharing through host memory (CPU). Moreover synchronization was performed only once at each

time step before the step on *Y* axis. If the grid size is $M \times N$, and the number of processors is **D**, the number required for the synchronization of grid nodes equals $4M(D-1)$.

Also, the version based on GPUDirect was implemented. The main advantage of GPUDirect technology in problem solution is the ability to transfer data, which is located in GPU memory directly without the involvement of the host via PCI Express bus, i.e., there is no need to copy data from the first graphics processor to the host, and then from the host to the other GPU. Test results are shown in **Fig. 10.**

The result of using CUDA technology does not differ significantly, when host memory isused as an exchange buffer. The result for OpenCL technology is a little worse.

10. CONCLUSION

This paper shows how the capabilities of CPU are involved in the solution of problems with seismic grid-characteristic method.

Highest percentage of peak performance of the processor has been obtained in sequential version. It was 22% for the calculations in single precision and 17% for double precision computations with cache memory access optimizations and use of AVX instruction set.

The maximum speedups was on AMD Opteron 6272 – up to 37 times with 64 cores, on AMD Opteron 8431 – up to 25 times on 48 cores, and on Intel Xeon E5-2697 – up to 1710 times on 24 cores.

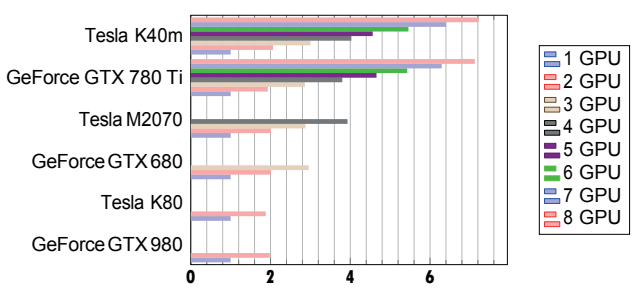

Fig. 10. *Acceleration using GPUDirect.*

Thus, we demonstrated, that in systems with shared memory, the main limitation to the growth of speedup with the increase of number of threads is the maximum speed of interaction with the memory. Separation of memory between threads allows processor cores to access only local memory, so memory access is faster. Binding threads to specific cores reduces the number of memory used to maintain the coherence of cores cache, which is also leads to increased performance. It should be noted that the implementation of a parallel version using POSIX Threads is more complicated than using of OpenMP, but does not provide a significant performance increase or efficiency of parallelization.

In addition, we described methods that allow achieving the highest performance of the algorithm when computing on the GPU. The problem of effective GPU memory use both in case of using one GPU, and in case of multiple GPUs is also studied. The influence of different optimizations on the performance of the algorithm was considered. The maximum acceleration obtained on graphic CPU compared with a single CPU – 55 times on GeForce GTX 780 Ti when computing with a single precision and 44 times at Tesla K80 at double precision computations. We managed to achieve the performance of 460 GFlops for single accuracy, and the maximum performance of 138 GFlops for double accuracy was obtained. Maximum achieved acceleration of the graphics processors is 7.1 times for 8 GPUs for double precision. GPUDirect technology raised acceleration to 10% of what has been achieved without the calculations with a single precision.

Based on these results we can draw come to a conclusion that GPUs can be used for the solution of such problems. The results are similar for other tasks, which use similar numerical methods (finite volume method, finite difference methods). It is also worth noting that there are good results for the AMD

processors and OpenCL technology, while CUDA technology for NVidia GPUs is used in most works.

REFERENCES

1. Caserta A, Ruggiero V, Lanucara P. Numerical modelling of dynamical interaction between seismic radiation and near-surface geological structures: a parallel approach. *Computers and geosciences,* 2002, 28, 9:1069-1077.
2. Guo X, Lange M, Gorman G, Mitchell L, Weiland M. Developing a scalable hybrid MPI/OpenMP unstructured finite element model. *Computers and Fluids*, 2015, 110:227-234.
3. Micikevicius P. 3D finite difference computation on GPUs using CUDA. *Proceedings of 2nd workshop on general purpose processing on graphics processing units.* ACM, 2009:79-84.
4. Nickolls J, Buck I, Garland M, Skadron K. Scalable parallel programming with CUDA. *Queue*, 2008, 6, 2:40-53.
5. Abdelkhalek R, Calendra H, Coulaud O, Latu G, Roman J. Fast seismic modeling and reverse time migration on a GPU cluster. *HPCS'09 Intern. Conf. on High Performance Computing and Simulation*, IEEE, 2009:36-43.
6. Foltinek D, Eaton D, Mahovsky J, Moghaddam P, McGarry R. Industrial-scale reverse time migration on GPU hardware. *2009 SEG Annual Meeting - Society of Exploration Geophysicists.* 2009.
7. Bohlen T. Parallel 3-D viscoelastic finite difference seismic modelling. *Computers and Geosciences*, 2002, 28, 8:887-899.
8. Gropp W, Lusk E, Doss N, Skjellum A. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel computing*, 1996, 22, 6:789-828.
9. Martin R, Komatitsch D, Blitz C, Le Goff N. Simulation of seismic wave propagation in an asteroid based upon an unstructured MPI spectral-element method: blocking and non-blocking communication strategies. *Intern. Conf. on High Performance Computing for Computational Science.* Springer, Berlin Heidelberg, 2008:350-363.
10. Rostrup S, De Sterck H. Parallel hyperbolic PDE simulation on clusters: Cell versus GPU. *Computer Physics Communications*, 2010, 181, 12:164-179.
11. Aochi H, Dupros F. MPI-OpenMP hybrid simulations using boundary integral equation and finite difference methods for earthquake dynamics and wave propagation: Application to the 2007 Niigata Chuetsu-Oki earthquake. *Procedia Computer Science,* 2011, 4:1496-1505.
12. Vanderbauwhede W, Benkrid K. *High-performance computing using FPGAs*. New York, Springer, 2013.
13. Krueger J, Donofrio D, Shalf J, Mohiyuddin M, Williams S, Oliker L, Pfreundt FJ. Hardware/software co-design for energy-efficient seismic modeling. *Proceedings of 2011 Intern. Conf. for High Performance Computing, Networking, Storage and Analysis.* ACM, 2011, 73 p.
14. Petrov IB, Favorskaya AV, Sannikov AV, Kvasov IE. Grid-characteristic method using high-order interpolation on tetrahedral hierarchical meshes with a multiple time step. *Mathematical Models and Computer Simulations,* 2013, 5, 5:409-415.
15. Golubev VI, Petrov IB, Khokhlov NI. Numerical simulation of seismic activity by the grid-characteristic method. *Computational Mathematics and Mathematical Physics*, 2013, 53, 10:1523-1533.
16. Beklemysheva KA, Petrov IB, Favorskaya AV. Numerical simulation of processes in solid deformable media in the presence of dynamic contacts using the grid-characteristic method. *Mathematical Models and Computer Simulations,* 2014, 6, 3:294-304.

17. Butenhof DR. *Programming with POSIX threads*. Addison-Wesley Professional, 1997, 400 p.
18. Munshi A. The opencl specification. *2009 IEEE Hot Chips 21 Symposium (HCS)*. IEEE, 2009, 314 p.
19. LeVeque RJ. *Finite volume methods for hyperbolic problems*. Cambridge university press, 2002, 558 p.
20. Carcione JM, Herman GC, ten Kroode APE. Review Article: Seismic modeling. *Geophysics,* 2002, 67, 4:1304-1325.