Proceedings Paper:
Titarenko, S orcid.org/0000-0002-4453-0180, Kulikov, I, Chernykh, I et al. (4 more authors) (2017) Multilevel Parallelization: Grid Methods for Solving Direct and Inverse Problems. In: Communications in Computer and Information Science: Supercomputing. RuSCDays 2016 (Russian Supercomputing Days), 26-27 Sep 2016, Moscow, Russia. Springer Verlag , pp. 118-131. ISBN 978-3-319-55668-0

https://doi.org/10.1007/978-3-319-55669-7_10

Multilevel Parallelization: Grid Methods for Solving Direct and Inverse Problems

Sofya S. Titarenko[1]✉, Igor M. Kulikov[2,3], Igor G. Chernykh[2], Maxim A. Shishlenin[2,3,4], Olga I. Krivorot'ko[2,3], Dmitry A. Voronov[2,3], and Mark Hildyard[1]

[1] School of Earth and Environment, University of Leeds, Leeds, LS2 9JT, UK
S.Titarenko@leeds.ac.uk
[2] Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, 630090, Russia
[3] Novosibirsk State University, Novosibirsk, 630090, Russia
[4] Sobolev Institute of Mathematics SB RAS, Novosibirsk, 630090, Russia

Abstract. In this paper we present grid methods which we have developed for solving direct and inverse problems, and their realization with different levels of optimization. We have focused on solving systems of hyperbolic equations using finite difference and finite volume numerical methods on multicore architectures. Several levels of parallelism have been applied: geometric decomposition of the calculative domain, workload distribution over threads within OpenMP directives, and vectorization. The run-time efficiency of these methods has been investigated. These developments have been tested using the astrophysics code AstroPhi on a hybrid cluster Polytechnic RSC PetaStream (consisting of Intel Xeon Phi accelerators) and a geophysics (seismic wave) code on an Intel Core i7-3930K multicore processor. We present the results of the calculations and study MPI run-time energy efficiency.

Keywords: High performance computing · Intel Xeon Phi accelerators · Grid-based numerical methods

1 Introduction

Numerical methods have become very powerful tools for modeling problems in physics and engineering. Many such problems can be described as a set of hyperbolic equations. In the last decade, a large number of numerical methods have been developed and improved, with finite difference and finite volume methods being almost the most popular. As models become more complex and often require high accuracies of calculation, the use of modern accelerators is more desirable. When moving towards exascale computing, energy consumption increases dramatically and the run-time energy efficiency of calculations becomes very important. In the recent past, geometrical decomposition of the solution domain of a problem (through message passing interface, MPI) was the only tool in parallelization. Since the release of multicore processors (e.g. Graphics Processing

Units and Xeon Phi processors), the combination of geometrical decomposition with multithread parallelization and vectorization of the calculations has become increasingly important.

Direct and inverse problems in geophysics are often impossible to solve analytically and hence numerical solution is the only option. One important example is forward modeling of wave propagation through an elastic medium. This problem was first solved numerically using a finite difference scheme by Alterman in 1968 [1]. Later this method was applied to generate synthetic seismograms by Kelly in 1976 [12]. A similar approach has been used to generate sound fields in acoustic problems [22, 24]. Solution of the direct wave propagation problem is widely used in full waveform inversion problems where a good initial guess is extremely important. This problem demands large computing resources and time and hence more and more scientists have optimized their codes using APIs, GPU and MPI parallelization. Examples of parallelizing large scale geophysical problems can be found in [20, 26, 6, 23, 2, 4].

When applying finite difference schemes to a problem it is necessary to calculate a derivative on a stencil type structure. Unfortunately, it is impossible to apply standard automatic vectorization techniques to a stencil type loop. In this work we present a method of memory rearrangement which allows vectorization with high level instructions only. This method is universal to any stencil type structure and its application considerably decreases the calculative time. Moreover, a derivative order has very little effect on CPU time. This allows a considerable increase in the accuracy of a scheme, without changing the CPU time. In this paper we discuss issues of using the proposed method together with OpenMP multithreading. We demonstrate the efficiency of the method on wave propagation through an elastic medium.

Another example of a complex problem which requires parallelization is the modeling of magnetic fields in astrophysics. Magnetic fields play a key role in the formation of astrophysical objects. Taking magnetic fields into account when modeling the evolution of interstellar turbulence makes a considerable difference to the results (see [18]). In recent years, modeling magnetohydrodynamic turbulence problems has helped our understanding of sub-alpha currents [19] and the rate of star formation [3]. A comparison of different codes for subsonic turbulence is presented in [13]. Classical methods for simulation of magnetohydrodynamic turbulence such as adaptive mesh refinement (AMR) and smoothed-particle hydrodynamics are still widely used, but in recent years an impressive range of new methods have been proposed (a good review of these methods can be found in [14–16]).

The inverse coefficient problem for a system of 2D hyperbolic equations has been studied in [17]. In this paper the acoustic tomography problem was reformulated as an inverse coefficient problem for a system of first order hyperbolic equations (system of acoustic equations). To solve the inverse problem the gradient method to optimize an objective functional was chosen. This method is widely used in inverse and ill-posed problem theory [9, 10, 7, 8, 11]. The main idea of the method is to solve direct and conjugate problems at every time step.

This means a numerical method to solve the direct problem needs to be well optimized. Here we present a method of optimization which proved to be very efficient.

In the first two sections we discuss various levels of optimization for the above astrophysics problem. The third section explains difficulties of automatic vectorization when applied to finite difference schemes. A method is proposed which overcomes the difficulties and considerably improves performance. The fourth section discusses the importance of run-time energy efficiency of calculations and demonstrates impressive results for the AstroPhi code. The fifth section presents results from numerical experiments.

2 Geometric Decomposition Pattern

The use of a uniform mesh gives us a possibility to apply a generic Cartesian topology for decomposition of the calculative domain. This approach leads to potentially infinite scalability of the problem. As shown in [16], the AstroPhi code implements multilevel one-dimensional geometric decomposition of the calculative domain. The first coordinate corresponds to the MPI level of parallelization. Every MPI thread sends tasks to OpenMP threads, optimized for MIC architectures. This type of topology is related to the topology and architecture of the hybrid cluster RSC PetaSteam, which has been used for the numerical simulations.

Various levels of AstroPhi code scalability have been tested on Intel Xeon Phi 5120 D accelerators. A grid $512p \times 256 \times 256$ has been used (where p is a number of accelerators). Every accelerator has 4 logical cores. The calculative domain is divided into data chunks of equal size, then the chunks are sent to the accelerators. To study the scalability we have estimated the total run-time (in seconds) for different numbers of Intel Xeon Phi accelerators. At every time step a certain number of processes has to be completed. We calculate the total run-time as the sum of the run-times of all these processes. The scalability has been calculated according to the formula

$$T = \frac{\mathrm{Total}_1}{\mathrm{Total}_p}, \tag{1}$$

where Total_1 and Total_p are a run-time and a calculation time on a single processor respectively, and the problem runs on p processors. The results are presented in Table 1. From the table it is clear to see we have achieved an efficiency of 73% on 256 Intel Xeon Phi 5120 D processors.

3 Multicore Threading on Intel Xeon Phi Accelerators

Parallelization of the AstroPhi code on Intel Xeon Phi accelerators has been achieved through a standard technique:

1. decomposition of the calculative domain;

Table 1. Scalability T of the AstroPhi code on the hybrid cluster RSC PetaStream. Time is in seconds.

MIC	Total (SPb)	Scalability (SPb)
1	55.5742	1.0000
8	56.3752	0.9857
64	64.1803	0.8659
128	68.6065	0.8101
256	76.1687	0.7296

2. workload distribution amongst the available threads;

For this problem we applied decomposition to a 512×256^2 grid on a single Xeon Phi accelerator. To calculate the acceleration we have measured the calculation time of each function of the numerical simulation and then calculated its summation on one thread and on p threads. The acceleration has been calculated according to the formula

$$P = \frac{\text{Total}_1}{\text{Total}_K}, \tag{2}$$

where Total_1 is the calculative time for one logical core, Total_K is the calculative time for K logical cores. The results of testing the AstroPhi code on hybrid cluster PetaStream (SPb) are presented in the Table 2.

Table 2. Acceleration P with increasing numbers of logical cores (on a single Xeon Phi accelerator). The code has been tested on a hybrid cluster RSC PetaStream (SPb). Time is in seconds.

Threads	Total (SPb)	P (SPb)
1	219.7956	1.0000
8	27.7089	7.9323
32	7.9673	27.5872
128	2.6271	83.6647
240	2.5905	84.8467

4 Vectorization

The first processor supporting a SIMD (Single Instruction Multiple Data) instruction set was designed by Intel in 1999. This Streaming SIMD (SSE) extension accelerates the calculation due to the use of larger registers. The first SIMD registers were designed to hold four 32-bit floats/ two 64-bit doubles (128-bit registers). This means that 4 floats/ 2 doubles can be uploaded into a register

and the arithmetic and logic routine can be applied to a *vector* instead of a *single* value.

This simple idea has become very popular and nowadays almost all modern architectures support SIMD operations. The capacity of SIMD registers has also been considerably increased. For example the Sandy Bridge microarchitecture includes the AVX extension with 256-bit SIMD registers and the Skylake microarchitecture includes the AVX-512 (Xeon models only) extension which operates with 512-bit SIMD registers.

To be able to take advantage of automatic vectorization either an optimization flag (`/O2` and higher for Intel machines) needs to be switched on, or a microarchitecture specific flag (ex. `/QxSSE4.1`, `/QxAVX` for Intel) should be chosen. It is important to note that an automatic vectorization routine cannot be applied to any loop. The memory in the loop needs to be aligned and the vector length must be divisible by 4 (8 or 16 depending on the size of a SIMD register and the bit size of values operated with). It is important that there shouldn't be any read-write memory conflict, for example a cycle of the type

```
for(int i = 0; i < N; i++){
    a[i] = b[i] + c[i];
}
```

is automatically vectorizable (assuming the memory is aligned and N is divisible by 4). However a loop of the type

```
__assumed_aligned(a, 32);
__assumed_aligned(b, 32);
#pragma simd
for(int i = 0; i < N; i++){
    a[i] = b[i] + b[i + 2];
}
```

cannot be vectorized by high level instructions.

When calculating the derivative on a stencil structure we get a loop which is not automatically vectorizable. However, the situation changes if we *rearrange the memory* in the way described in Fig. 1. In this case the compiler will be uploading and applying arithmetic and logical operations to all 4 values simultaneously and the necessary acceleration will be achieved.

In the case of larger than 128-bit registers or smaller bit size values it is always possible to rearrange the data in a such way the compiler would work with a vector of 8, 16, etc. In this case the values we use for calculation of one derivative i will be located with the step spacing 8, 16 etc. from each other.

A very important property of such rearrangement is that the speed of derivative calculation is almost unaffected by the order of the derivative. This property is especially important for problems where it is problematic to achieve the required degree of accuracy due to long run times.

We have tested this property by calculating 2^{d}, 4^{th}, 6^{th}, 8^{th} and 10^{th} order derivatives on the grid 8192×8192 (for 500 cycles). Note that the order of the

derivative increases the size of the stencil. Table 3 shows the calculation time *without* memory rearrangement and automatic optimization (flag `/Od` for Intel), *without* memory rearrangement and *with* aggressive automatic vectorization (flag `/O3` for Intel) and *with* memory rearrangement and *with* aggressive automatic vectorization. The table shows that in the initial case, run time is 74% greater for the 10th order than the 2nd order. When automatic vectorization is used without memory rearrangement, run time is much shorter but the difference between 10th order and second order becomes 200%. Finally, when vectorization is combined with memory rearrangement, run time becomes almost independent of the order of the derivative. The acceleration for 10^{th} order is ≈ 11.57 times, compared with ≈ 6.63 times for 2^{nd} order.

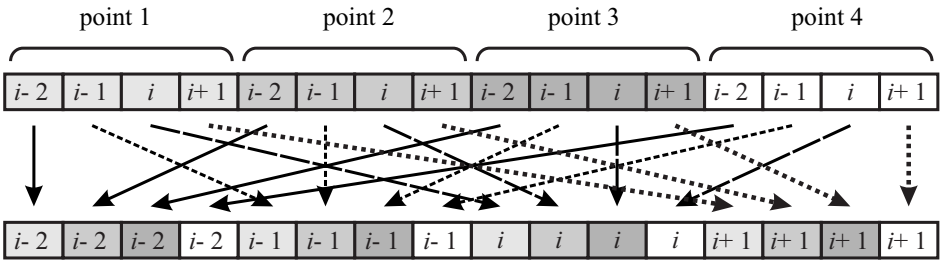

Fig. 1. Memory reorganization for high level vectorization.

Table 3. Applying different levels of optimization (see text for details) for derivatives of a high order. Run time is in seconds.

order	`/Od`	`/O3`, $[1 \times 1]$	`/O3`, $[4 \times 1]$
2	311.601	63.978	40.987
4	378.303	96.804	41.502
6	433.277	122.147	42.687
8	485.380	158.753	41.350
10	543.615	192.288	46.996

5 Study of Run-time Energy Efficiency

Nowadays, run-time energy efficiency is mostly used for commercial projects working with big data problems. However, the idea of exascale computing is becoming more and more popular and petascale computers are expected to be widely used in the near future. Exascale computing is capable of more then 10^{18}

calculations per second. The first petascale computers were released in 2008 and are considered to be very promising and powerful tools for solving big data problems, for example modeling for climate, in geophysics and astrophysics. Running such a computer efficiently is of great importance. If only 10 MW of energy for running an exascale supercomputer were used inefficiently it could cancel out all the advantage of using it. Overall the definition of run-time energy efficiency includes about 20 parameters, most of them related to run time efficiency. In this work we assume the code to be efficient if

1. CPU cores and CPUs are used in the most efficient way;
2. the data exchange between the CPU cores and CPUs is minimized;
3. the code has good balance.

By minimizing data exchange between CPU cores/CPUs we reduce the waiting time for a CPU/CPU cluster (the time while the CPU/CPU cluster doesn't work, awaiting the completion of all necessary processes). Good balance allows tasks to be distributed in between cores and accelerators evenly. By applying these ideas to the AstroPhi code we have reduced the time spent on data exchange by MPI instructions to 7–8% of the total run-time. The level of imbalance has been reduced to no more then 2–3% between all the threads. This helped us to achieve 72% efficiency (scalability in the "weak" sense) in parallelizing on a 256 Intel Xeon Phi accelerator (more then 50K cores). Modern accelerators help to achieve the maximum run-time efficiency by multithreading and vectorization. By applying vectorization to the AstroPhi code we have achieved 6.5 times acceleration and by run-time energy efficiency we have approached the efficiency of libraries like MAGMA MIC [5].

6 Results

6.1 Modeling of Wave Propagation through an Elastic Medium

To model wave propagation through two-dimensional elastic media the second order wave equation is often reduced to a system of first order partial differential equations

$$\begin{cases} \rho\frac{\partial u}{\partial t} = \frac{\partial \sigma_{11}}{\partial x_1} + \frac{\partial \sigma_{12}}{\partial x_2}, \quad \rho\frac{\partial v}{\partial t} = \frac{\partial \sigma_{22}}{\partial x_2} + \frac{\partial \sigma_{12}}{\partial x_1}, \\ \frac{\partial \sigma_{11}}{\partial t} = \left(\lambda + 2\mu\right)\frac{\partial u}{\partial x_1} + \lambda\frac{\partial v}{\partial x_2}, \quad \frac{\partial \sigma_{12}}{\partial t} = \mu\frac{\partial u}{\partial x_2} + \mu\frac{\partial v}{\partial x_1}, \\ \frac{\partial \sigma_{22}}{\partial t} = \lambda\frac{\partial u}{\partial x_1} + \left(\lambda + 2\mu\right)\frac{\partial v}{\partial x_2}, \end{cases} \tag{3}$$

where (u, v) is the velocity vector, σ_{ij} is the stress tensor, and λ and μ are Lamé parameters.

It is also usual to move from an ordinary grid to a *staggered grid*. The method was first proposed in [25] to solve elastic wave propagation problems and has been proved to have better stability and dispersion behavior for 4^{th} order accuracy schemes [21].

Let us define

$$\begin{aligned}
&\delta u \equiv u^{i-2} - 27u^{i-1} + 27u^{i} - u^{i+1},\\
&\delta v \equiv v^{j-2} - 27v^{j-1} + 27v^{j} - v^{j+1},\\
&\delta\sigma_{11} \equiv \sigma_{11}^{i-1} - 27\sigma_{11}^{i} + 27\sigma_{11}^{i+1} - \sigma_{11}^{i+2},\\
&\delta\sigma_{22} \equiv \sigma_{11}^{j-1} - 27\sigma_{11}^{j} + 27\sigma_{11}^{j+1} - \sigma_{11}^{j+2},\\
&\delta u^{+} \equiv u^{j-1} - 27u^{j} + 27u^{j+1} - u^{j+2},\\
&\delta v^{+} \equiv v^{i-1} - 27v^{i} + 27v^{i+1} - v^{i+2}.
\end{aligned} \tag{4}$$

Then according to Finite Difference rules the new expression for σ_{11}, σ_{22}, σ_{12} and velocities u, v for every time step can be found as

$$\begin{aligned}
&\sigma_{11}^{t} = \sigma_{11}^{t-1} + \tfrac{(\lambda+2\mu)\Delta t}{24\Delta x_1}\delta u + \tfrac{\mu\Delta t}{24\Delta x_2}\delta v,\\
&\sigma_{22}^{t} = \sigma_{22}^{t-1} + \tfrac{\mu\Delta t}{24\Delta x_1}\delta u + \tfrac{(\lambda+2\mu)\Delta t}{24\Delta x_2}\delta v,\\
&\sigma_{12}^{t} = \sigma_{12}^{t-1} + \tfrac{\Delta t\mu}{24\Delta x_2}\delta u^{+} + \tfrac{\Delta t\mu}{24\Delta x_1}\delta v^{+},\\
&u^{t+1/2} = u^{t-1/2} + \tfrac{\Delta t}{24\rho\Delta x_1}\delta\sigma_{11} + \tfrac{\Delta t}{24\rho\Delta x_2}\left(\sigma_{12}^{j-2} - 27\sigma_{12}^{j-1} + 27\sigma_{12}^{j} - \sigma_{12}^{j+1}\right),\\
&v^{t+1/2} = v^{t-1/2}\tfrac{\Delta t}{24\rho\Delta x_1}\left(\sigma_{12}^{i-2} - 27\sigma_{12}^{i-1} + 27\sigma_{12}^{i} - \sigma_{12}^{i+1}\right) + \tfrac{\Delta t}{24\rho\Delta x_2}\delta\sigma_{22}.
\end{aligned} \tag{5}$$

Fig. 2 shows the general scheme for vectorization. It is clear to see that application of boundary conditions for the problem will lead to divergence. This happens because the boundary conditions will be applied to the internal points of the problem. To eliminate this bottleneck we introduce *virtual* blocks. The boundary conditions will be applied to virtual blocks instead and at every time step we have to copy the data from internal points to the virtual ones to achieve convergence.

Table 4 shows the acceleration of the problem for different sizes. If the flag `/O1` is switched on the compiler applies automatic optimization, but not vectorization. Flag `/O2` and higher enables automatic vectorization.

Table 4. Comparison of run time for problems of different sizes ($N \times N$) without optimization and with flags `/O1` and `/O3`. The problems have been calculated on one thread. Time is in seconds.

N	`/Od`	`/O1`	`/O3`	acceleration
$32 \times 32 \times 2$	216.761	54.143	26.133	8.295
$64 \times 32 \times 2$	974.981	186.679	81.356	11.984
$128 \times 32 \times 2$	3789.820	705.704	309.508	12.245

The problem can easily be parallelized on available CPU cores by applying OpenMP directives. The whole domain is divided into blocks. Each block has so-called *buffer* points. At every time step the values from the *buffer* points are updated by copying from the internal grid points of corresponding blocks. The blocks are run in a random order. For our problem we have found the following OpenMP structure gives the best acceleration

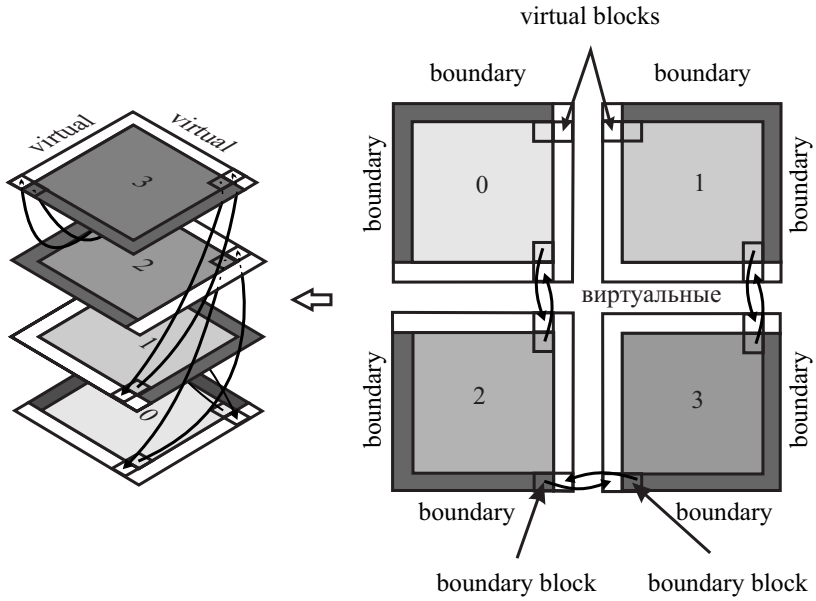

Fig. 2. Copying of data from real blocks to virtual.

```
#pragma omp parallel
{
    for (int j = 0; j < numberOfSteps; j++){
        int i;
#pragma omp for private (i) schedule(auto)
        //calculate velocity
#pragma omp for private (i) schedule(auto)
        //copy velocities
#pragma omp for private (i) schedule(auto)
        //calculate stress
#pragma omp for private (i) schedule(auto)
        //copy stress
#pragma omp for private (i) schedule(auto)
        //calculate boundaries
#pragma omp for private (i) schedule(auto)
        //copy virtual blocks
    }
}
```

It should be mentioned, that vectorization blocks can be arranged in a different order and the order has a small influence on acceleration. For this problem we studied $[4 \times 1]$, $[1 \times 4]$, $[2 \times 2]$ structures. They are presented in Fig. 3.

Experiments and results from VTune Intel Amplifier profiling proved the best vectorization structure to be $[4 \times 1]$. This structure is used in Table 5 for OpenMP parallelization.

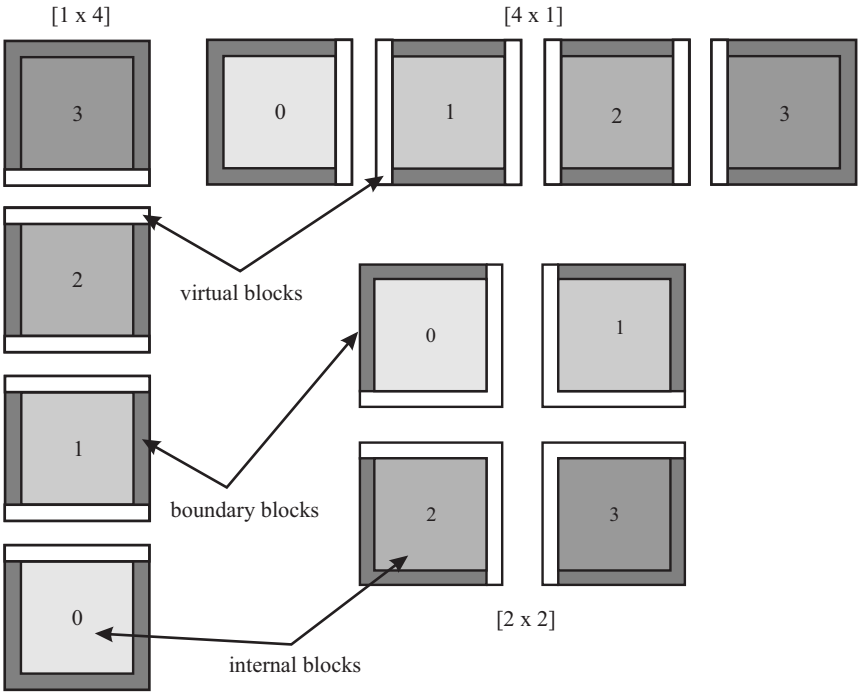

Fig. 3. Types of vectorization structure.

6.2 Modeling of Magneto-hydrodynamics Turbulence Evolution

This numerical model is based on coupling of equations for multidimensional magneto-gas-dynamics, the ordinary differential equation for the evolution of the concentration of ionized hydrogen, and a special form for external force. External force is found from the mass conservation law and Poisson equation. Its time derivative can be described by the Cauchy–Kovalevskaya equation. By using this mathematical model it becomes possible to formulate a generalized parallelization calculation method [15], which is based on a combination of an operator-splitting method, Godunov method, and a piecewise-parabolic approximation on a regular grid cell. Fig. 4 shows the result of the numerical simulations described above. The figure shows the high density area of a "palm tree" shape, which resembles the nebular NGC 6188. Fig. 5 shows the correlation of $M \sim n^2$ (white line) and most of a nebular cloud $n > 10$ m^{-3} is in the super-Alfvenic speed area. Contours of the cosine of the colinear angle between velocity vector

Table 5. Parallelization of seismic wave problem through OpenMP API. The size of the problem is $[4096 \times 4096]$, the vectorization structure is $[4 \times 1]$. In all cases the most aggressive automatic vectorization has been applied. Time is in seconds.

N threads	run time	acceleration
1	82.300	1.000
4	28.157	2.923
6	25.286	3.255

and magnetic field have a saddle shape (see Fig. 6). This means that compression occurs along magnetic field lines.

7 Conclusions

In this paper we have demonstrated the following.

1. It is important to apply all levels of parallelization to big data problems: vectorization, multithread parallelization within shared memory API (OpenMP directives in the test cases) and clusters.
2. Run-time energy efficiency is very important in parallelization as demonstrated by application to modeling of magnetic fields in astrophysics problems.
3. It is impossible to apply high level vectorization methodology to a stencil type loop for a standard memory structure. We have presented a new method of memory rearrangement which overcomes this bottleneck and allows automatic vectorization to take place. In application to finite difference schemes we have demonstrated the method to be particular efficient when using high order derivatives. We proved that if the suggested memory organization is used, the run-time for derivative calculations is almost independent of its order (which is not the case for a standard memory structure!).
4. We have presented the results of astrophysics code AstroPhi efficiency, (tested on the hybrid cluster RSC PetaStream based on Intel Xeon Phi accelerators), vectorization and multithreading withing OpenMP of seismic wave propagation problem (tested on Intel Core i7-3930K machine) and the study on AstroPhi code run-time energy efficiency. The results from numerical simulations are presented.

Acknowledgments. The authors would like to thank the colleagues from Intel (Nikolai Mester and Dmitri Petunin) and RSC Group (Alexader Moscowski, Pavel Lavrenko and Boris Gagarinov) for access to RSC PetaStream cluster and helpful instructions for its use. Authors acknowledge the support of Russian Foundation for Basic Research 14-01-00208, 15-31-20150 mol-a-ved, 15-01-00508, 16-01-00755, 16-31-00382, 16-29-15120 and 16-31-00189, grant of the President of Russian Federation MK-1445.2017.9 and Ministry of Education and Science of Russian Federation. The work has been also done with the support of RCUK

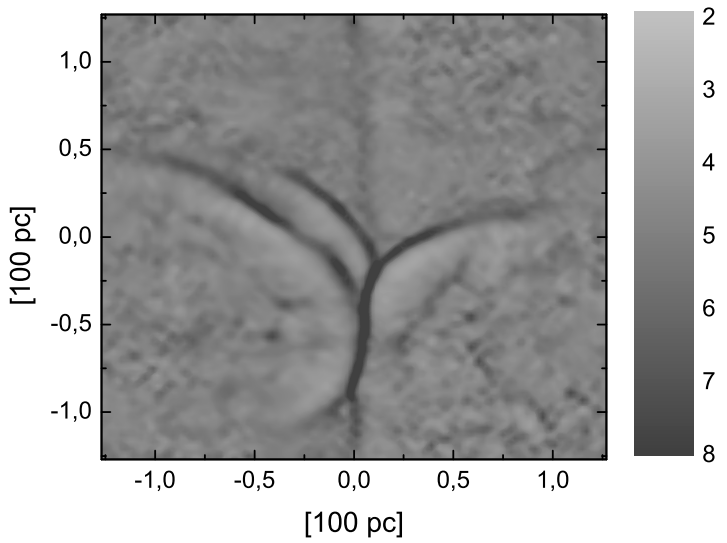

Fig. 4. Numerical simulation for magneto-gas-dynamics. Gas concentration is in cm^{-3} at a time $t = 15$ million years (back from now).

grant NE/L000423/1 and NERC and Environment Agency and Radioactive Waste Management Ltd.

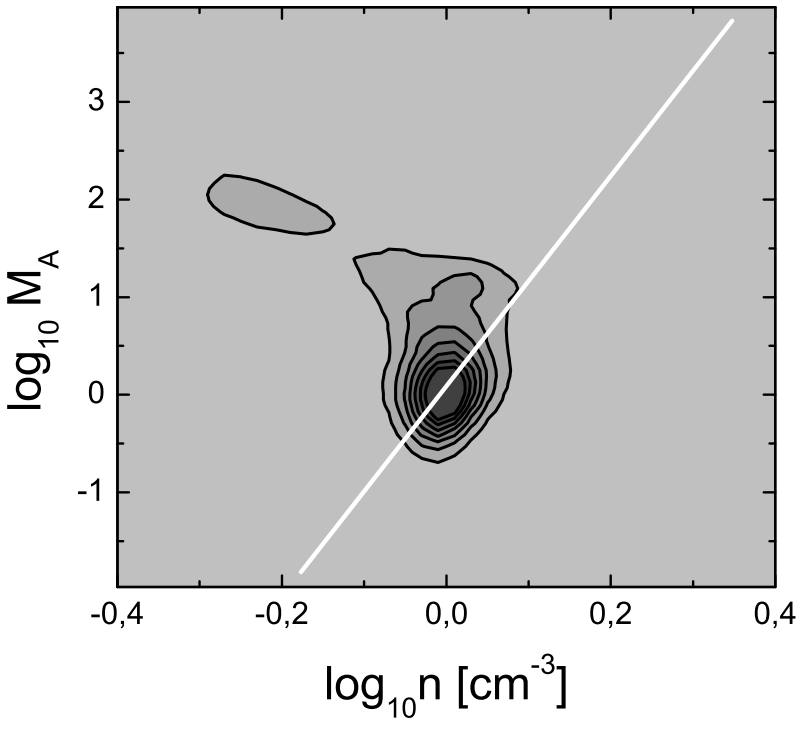

Fig. 5. Dependence of Alfven velocity (M_A) on gas density (n).

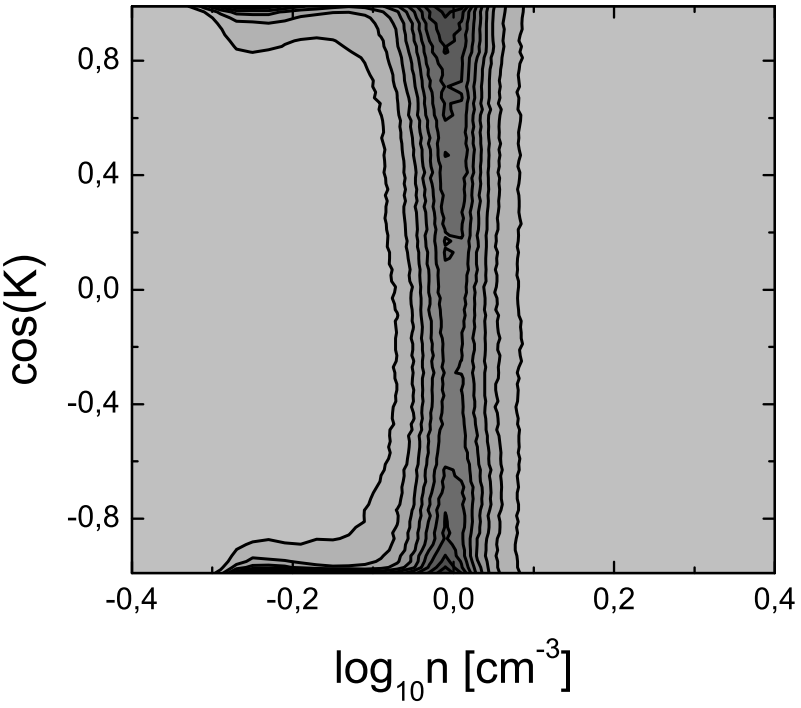

Fig. 6. Dependence of the cosine of colinear angle between velocity vector and magnetic field from gas density.

References

1. Alterman, Z., Karal, F.C.: Propagation of elastic waves in layered media by finite-difference methods. Bulletin of the Seismological Society of America **58**, 367–398 (1968)
2. 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 (mw6.6). Procedia Computer Science **4**, 1496–1505 (2011). DOI 10.1016/j.procs.2011.04.162
3. Federrath, C., Klessen, R.S.: The star formation rate of turbulent magnetized clouds: Comparing theory, simulations, and observations. The Astrophysical Journal **761**(2), 156 (2012). DOI 10.1088/0004-637X/761/2/156
4. Guyau, A., Cupillard, P., Kutsenko, A.: Accelerating a finite-difference time-domain code for the acoustic wave propagation. In: 35th Gocad Meeting — 2015 RING Meeting. ASGA (2015)
5. Haidar, A., Dongarra, J., Kabir, K., Gates, M., Luszczek, P., Tomov, S., Jia, Y.: HPC programming on Intel many-integrated-core hardware with MAGMA port to Xeon Phi. Scientific Programming **23** (2015). DOI 10.3233/SPR-140404
6. Hernández, M., Imbernón, B., Navarro, J.M., García, J.M., Cebrián, J.M., Cecilia, J.M.: Evaluation of the 3-D finite difference implementation of the acoustic diffusion equation model on massively parallel architectures. Computers & Electrical Engineering **46**, 190–201 (2015). DOI 10.1016/j.compeleceng.2015.07.001
7. Kabanikhin, S., Shishlenin, M.: Quasi-solution in inverse coefficient problems. Journal of Inverse and Ill-Posed Problems **16**(7), 705–713 (2008)
8. Kabanikhin, S., Shishlenin, M.: About the usage of the a priori information in coefficient inverse problems for hyperbolic equations. Proceedings of IMM of UrB RAS **18**(1), 147–164 (2012)
9. Kabanikhin, S.I., Gasimov, Y.S., Nurseitov, D.B., Shishlenin, M.A., Sholpanbaev, B.B., Kasenov, S.: Regularization of the continuation problem for elliptic equations. Journal of Inverse and Ill-Posed Problems **21**(6), 871–884 (2013)
10. Kabanikhin, S.I., Nurseitov, D.B., Shishlenin, M.A., Sholpanbaev, B.B.: Inverse problems for the ground penetrating radar. Journal of Inverse and Ill-Posed Problems **21**(6), 885–892 (2013)
11. Kabanikhin, S.I., Shishlenin, M.A., Nurseitov, D.B., Nurseitova, A.T., Kasenov, S.E.: Comparative analysis of methods for regularizing an initial boundary value problem for the helmholtz equation. Journal of Applied Mathematics **2014** (2014). DOI 10.1155/2014/786326
12. Kelly, K.R., Ward, R.W., Tritel, S., Alford, R.M.: Synthetic seismograms: A finite-difference approach. Geophysics **41**, 2–27 (1976)
13. Kritsuk, A.G., Nordlund, Å., Collins, D., Padoan, P., Norman, M.L., Abel, T., Banerjee, R., Federrath, C., Flock, M., Lee, D., Li, P.S., Muller, W.C., Teyssier, R., Ustyugov, S.D., Vogel, C., Xu, H.: Comparing numerical methods for isothermal magnetized supersonic turbulence. The Astrophysical Journal **737**(1), 13 (2011). DOI 10.1088/0004-637X/737/1/13
14. Kulikov, I.: GPUPEGAS: A new GPU-accelerated hydrodynamic code for numerical simulations of interacting galaxies. The Astrophysical Journal Supplement Series **214**(1), 12 (2014). DOI 10.1088/0067-0049/214/1/12
15. Kulikov, I., Vorobyov, E.: Using the PPML approach for constructing a low-dissipation, operator-splitting scheme for numerical simulations of hydrodynamic flows. Journal of Computational Physics **317**, 318–346 (2016). DOI 10.1016/j.jcp.2016.04.057

16. Kulikov, I.M., Chernykh, I.G., Snytnikov, A.V., Glinskiy, B.M., Tutukov, A.V.: AstroPhi: A code for complex simulation of the dynamics of astrophysical objects using hybrid supercomputers. Computer Physics Communications **186**, 71–80 (2015). DOI 10.1016/j.cpc.2014.09.004
17. Kulikov, I.M., Novikov, N.S., Shishlenin, M.A.: Mathematical modeling of ultrasound wave propagation in 2D medium: direct and inverse problem. Siberian Electronic Mathematical Reports **12**, 219–228 (2015). In Russian
18. Mason, J., Perez, J.C., Cattaneo, F., Boldyrev, S.: Extended scaling laws in numerical simulations of magnetohydrodynamic turbulence. The Astrophysical Journal Letters **735**(2), L26 (2011). DOI 10.1088/2041-8205/735/2/L26
19. McKee, C.F., Li, P.S., Klein, R.I.: Sub-Alfvénic non-ideal MHD turbulence simulations with ambipolar diffusion. II. Comparison with observation, clump properties, and scaling to physical units. The Astrophysical Journal **720**(2), 1612–1634 (2010). DOI 10.1088/0004-637X/720/2/1612
20. Michéa, D., Komatitsch, D.: Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards. Geophysical Journal International **182**(1), 389–402 (2010). DOI 10.1111/j.1365-246X.2010.04616.x
21. Moczo, P., Kristek, J., Bystrický, E.: Stability and grid dispersion of the P-SV 4th-order staggered-grid finite-difference schemes. Studia Geophysica et Geodaetica **44**(3), 381–402 (2000). DOI 10.1023/A:1022112620994
22. Sakamoto, S., Seimiya, T., Tachibana, H.: Visualization of sound reflection and diffraction using finite difference time domain method. Acoustical Science and Technology **23**(1), 34–39 (2002). DOI 10.1250/ast.23.34
23. Sheen, D.H., Kagan-Tuncay, Baag, C.E., Ortoleva, P.J.: Parallel impelemntation of a velocity-stress staggered-grid finite-difference method for 2-D poroelastic wave propagation. Computer & Geoscience **32**, 1182–1191 (2006)
24. Siltanen, S., Robinson, P.W., Saarelma, J., Pätynen, J., Tervo, S., Savioja, L., Lokki, T.: Acoustic visualizations using surface mapping. The Journal of the Acoustical Society of America **135**(6), EL344–EL349 (2014). DOI 10.1121/1.4879670
25. Virieux, J.: P-SV wave propagation in heterogeneous media; velocity-stress finite-difference method. Geophysics **51**(4), 889–901 (1986). DOI 10.1190/1.1442147
26. Zhang, Y., Gao, J.: A 3D staggered-grid finite difference scheme for poroelastic wave equation. Journal of Applied Geophysics **109**, 281–291 (2014). DOI 10.1016/j.jappgeo.2014.08.007