A parallel hybrid implementation of the 2D acoustic wave equation

Arshyn Altybay
Al-Farabi Kazakh National University
17 PUBLICATIONS **105** CITATIONS
SEE PROFILE

Michael Ruzhansky
Ghent University and Queen Mary University of London
616 PUBLICATIONS **7,005** CITATIONS
SEE PROFILE

Niyaz Tokmagambetov
CRM Centre for Mathematical Research
110 PUBLICATIONS **1,105** CITATIONS
SEE PROFILE

A PARALLEL HYBRID IMPLEMENTATION OF THE 2D ACOUSTIC WAVE EQUATION

ARSHYN ALTYBAY, MICHAEL RUZHANSKY, AND NIYAZ TOKMAGAMBETOV

Abstract. In this paper, we propose a hybrid parallel programming approach for a numerical solution of a two-dimensional acoustic wave equation using an implicit difference scheme for a single computer. The calculations are carried out in an implicit finite difference scheme. First, we transform the differential equation into an implicit finite-difference equation and then using the ADI method, we split the equation into two sub-equations. Using the cyclic reduction algorithm, we calculate an approximate solution. Finally, we change this algorithm to parallelize on GPU, GPU+OpenMP, and Hybrid (GPU+OpenMP+MPI) computing platforms.

The special focus is on improving the performance of the parallel algorithms to calculate the acceleration based on the execution time. We show that the code that runs on the hybrid approach gives the expected results by comparing our results to those obtained by running the same simulation on a classical processor core, CUDA, and CUDA+OpenMP implementations.

1. Introduction

The reduction of computational time for long-term simulation of physical processes is a challenge and an important issue in the field of modern scientific computing. The cost of supercomputer, CPU clusters and hybrid clusters with a large number of GPUs are very expensive and they consume a lot of energy, which is inaccessible and ineffective to some small laboratories and individuals.

Nowadays, new generation computers are multi-core, hybrid architecture and their computational power is also quite high. For example, the Intel Xeon E5-2697 v2 (2S-E5) processors theoretically has computing power of about 19.56 GFLOPS, and, accordingly, the computational power of the NVIDIA TITAN Xp video card is about up to 379.7 GFLOPS. If we use the computing power of the CPU and GPU together, we can show good results.

The goal of this work is to develop a parallel hybrid implementation of the finite-difference method for solving two-dimensional wave equation using CUDA, CUDA + OpenMP and CUDA + OpenMP + MPI technologies and to study the parallelization efficiency by comparing the time to solve this problem with the above approaches.

2010 *Mathematics Subject Classification.* 35L05, 76B15, 68Q85.

Key words and phrases. Parallel computing, GPU, CUDA, MPI, Open MP, acoustic equation, wave equation.

The authors were supported by the FWO Odysseus 1 grant G.0H94.18N: Analysis and Partial Differential Equations. MR was supported in parts by the EPSRC Grant EP/R003025/1, by the Leverhulme Research Grant RPG-2017-151. AA was supported by the MESRK Grants AP08052028 and AP08053051 of the Ministry of Education and Science of the Republic of Kazakhstan.

Already for several years, GPUs have been used to accelerate well parallelizable computing, only with the advent of a new generation of GPUs with multicore architecture, this direction began to give palpable results.

For multidimensional problems, the efficiency of an implicit compact difference scheme depends on the computational efficiency of the corresponding matrix solvers. From this point of view, the ADI method [1] is promising because they can decompose a multidimensional problem into a series of one-dimensional problems. It has been shown that schemes acquired are unconditionally stable. For the proper assignment of large domains of modeling, two- or three-dimensional computational grids with a sufficient number of points are used. Calculations on such grids require more CPU time and computer memory resources. To accelerate the computation process, GPU, OpenMP, MPI technologies were used in this paper, which allows the program to operate on larger grids. With GPU becoming a viable alternative to CPU for parallel computing, the aforementioned parallel tridiagonal solvers and other hybrid methods have been implemented on GPUs [4]–[11]. In this paper, we propose three different parallel programming approaches using hybrid CUDA, OpenMP and MPI programming for personal computers. There are many examples in the literature of successfully using hybrid approaches for different simulation [12]–[17].

Here we study some issues in the numerical simulation of some problems in the propagation of acoustic waves on high performance computing systems.

2. Problem Statement and Numerical Scheme

We consider 2D acoustic wave equation with the positive "speed" function c and the source term f

$$\frac{\partial^2 u}{\partial t^2} - c(t)\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f(t,x,y),\ (t,x,y) \in [0;T]\times[0;l]\times[0;l], \tag{2.1}$$

subject to the initial conditions

$$u(0,x,y) = \varphi(x,y),\ x,y \in [0,l], \tag{2.2}$$

$$\frac{\partial u(0,x,y)}{\partial t} = \psi(x,y),\ x,y \in [0,l], \tag{2.3}$$

and boundary conditions

$$u(t,x,0) = 0,\ u(t,x,l) = 0,\ t \in [0,T],\ x \in [0,l], \tag{2.4}$$

$$u(t,0,y) = 0,\ u(t,l,y) = 0,\ t \in [0,T],\ y \in [0,l]. \tag{2.5}$$

In what follows, we take all data, namely, the coefficient c, the source function f, the initial functions φ and ψ, smooth enough. Due to the notion of "very weak solutions" and the approach developed by Garetto and Ruzhansky in [18], we can deal with 2D acoustic wave equation with singular (δ–like) data approximating them by smooth functions. For more details on these techniques and applications, we refer to the papers [18, 19, 20, 21, 22, 23, 24].

For numerical simulations, we introduce a space-time grid with steps h_1, h_2, τ respectively, in the variables x, y, t :

$$\omega^{\tau}_{h_1,h_2} = \{t_k = k\tau, k = \overline{1,M};\ x_i = ih_1, i = \overline{1,N_1};\ y_j = jh_2, j = \overline{1,N_2}\}, \tag{2.6}$$

and

$$\Omega^{\tau}_{h_1,h_2} = \{t_k = k\tau, k = \overline{0, M};\ x_i = ih_1, i = \overline{0, N_1};\ y_j = jh_2, j = \overline{0, N_2}\}, \tag{2.7}$$

where $h_1 = l/N_1$, $h_2 = l/N_2$ and $\tau = T/M$.

On this grid we approximate the problem (2.1)–(2.5) using the finite difference method. For simplicity, we put $N := N_1 = N_2$ and denote $h := h_1 = h_2$. Consider the Crank-Nicolson scheme for the problem (2.1)–(2.5)

$$\begin{aligned} &\frac{u^{k+1}_{i,j} - 2u^k_{i,j} + u^{k-1}_{i,j}}{\tau^2} - \frac{c^{k+1}}{2h^2}(u^{k+1}_{i+1,j} - 2u^{k+1}_{i,j} + u^{k+1}_{i-1,j} + u^{k+1}_{i,j+1} - 2u^{k+1}_{i,j} + u^{k+1}_{i,j-1}) \\ &\qquad - \frac{c^{k-1}}{2h^2}(u^{k-1}_{i+1,j} - 2u^{k-1}_{i,j} + u^{k-1}_{i-1,j} + u^{k-1}_{i,j+1} - 2u^{k-1}_{i,j} + u^{k-1}_{i,j-1}) = f^k_{i,j}, \end{aligned} \tag{2.8}$$

for $(k, i, j) \in \omega^{\tau}_{h_1,h_2}$, with initial conditions

$$u^0_{i,j} = \varphi_{i,j},\ \ u^1_{i,j} - u^0_{i,j} = \tau\psi_{i,j}, \tag{2.9}$$

for $(i, j) \in \overline{0, N} \times \overline{0, N}$, and with boundary conditions

$$u^k_{0,j} = 0,\ \ u^k_{N,j} = 0,\ \ u^k_{i,0} = 0,\ \ u^k_{i,N} = 0, \tag{2.10}$$

for $(j, k) \in \overline{0, N} \times \overline{0, M}$ and $(i, k) \in \overline{0, N} \times \overline{0, M}$, respectively.

It is well-known, that the implicit scheme is unconditionally stable and it has accuracy order $O(\tau + |h|^2)$, see, for example [25]. We solve the difference equation (2.8) by the alternating direction implicit (ADI) method, namely, dividing it into two sub-problems

$$\begin{aligned} &\frac{u^{k+1/2}_{i,j} - 2u^k_{i,j} + u^{k-1/2}_{i,j}}{\tau^2} - \frac{c^{k+1/2}}{2h^2}(u^{k+1/2}_{i+1,j} - 2u^{k+1/2}_{i,j} + u^{k+1/2}_{i-1,j}) \\ &- \frac{c^{k-1/2}}{2h^2}(u^{k-1/2}_{i+1,j} - 2u^{k-1/2}_{i,j} + u^{k-1/2}_{i-1,j}) = f^k_{i,j}, \end{aligned} \tag{2.11}$$

and

$$\begin{aligned} &\frac{u^{k+1}_{i,j} - 2u^{k+1/2}_{i,j} + u^k_{i,j}}{\tau^2} - \frac{c^{k+1}}{2h^2}(u^{k+1}_{i,j+1} - 2u^{k+1}_{i,j} + u^{k+1}_{i,j-1}) \\ &- \frac{c^k}{2h^2}(u^k_{i,j+1} - 2u^k_{i,j} + u^k_{i,j-1}) = f^{k+1/2}_{i,j}. \end{aligned} \tag{2.12}$$

3. Hybrid parallel computing model

High-performance computing uses parallel computing to achieve high levels of performance. In parallel computing, the program is divided into many subroutines, and then they are all executed in parallel to calculate the required values. In this section, we will propose a hybrid parallel approach numerically solving a two-dimensional wave equation, for this, we use CUDA, MPI OpenMP technologies.

3.1. **CUDA approach.** The graphics processing unit (GPU) is a highly parallel, multi-threaded, and multi-core processor with enormous processing power. Its low cost and high bandwidth floating point operations and memory access bandwidth are attracting more and more high performance computing researchers [32]. In addition, compared to cluster systems, which consist of several processors, computing on a GPU is inexpensive and requires low power consumption with equivalent performance. In many disciplines of science and technology, users were able to increase productivity by several orders of magnitude using graphics processors [2], [3]. The year 2007, with the appearance of the CUDA programming language, programming GPUs on NVIDIA graphics cards became significantly simpler because its syntax is similar to C[26].

It is designed so that its constructions allow a natural expression of concurrency at the data level. A CUDA program consists of two parts: a sequential program running on the CPU, and a parallel part running on the GPU [3], [31]. The parallel part is called the kernel. A CUDA program automatically uses more parallelism on GPUs that have more processor cores.

A C program using CUDA extensions hand out a large number of copies of the kernel into available multiprocessors to be performed simultaneously.

The CUDA code consists of three computational steps: transferring data to the global GPU memory, running the CUDA core, and transferring the results from the GPU to the CPU memory. We have designed a CUDA program based on cyclic reduction method, whose full CR function codes are located in [29]. The algorithm for solving the problem (2.1)–(2.5) is shown in Algorithm 1.

Algorithm 1 Implementation of 2D wave equation

```
compute initial function matrix $U0$
from initial condition (2.2) we get $u = U0$
**while** $(t < t_{end})$ **do**
    for $j = 0, \dots, n$
    for $i = 0, \dots, n$
    calculate tridiagonal system elements $a_i, b_i, c_i, f_i$
    call function $CR(a_i, b_i, c_i, f_i, y_i, n)$
    calculate matrix $Ux$
    for $i = 0, \dots, n$
    for $j = 0, \dots, n$
    calculate tridiagonal system elements $aj, bj, cj, fj$
    call function $CR(a_j, b_j, c_j, f_j, y_j, n)$
    calculate matrix $Uy$
    swap $(u, Ux)$
    swap $(U0, Uy)$
    $t \leftarrow t + \triangle t$
```

Here, u, $U0$, Ux, Uy denote $u_{i,j}^{k-1/2}$, $u_{i,j}^{k}$, $u_{i,j}^{k+1/2}$, $u_{i,j}^{k+1}$, respectively. The $CR()$ function includes 3 device functions, namely, $CRM\_forward(), cr\_div()$, $andCRM\_backward()$, and one host function $calc\_dim()$. First we have to calculate the block size according to the size of the matrix and step numbers of forward and backward sub-steps. For this, we use one cycle

```
for $(i = 0; i < \log 2(n + 1) - 1; i + +)$ {
$stepNum = (n - pow(2.0, i + 1))/pow(2.0, i + 1) + 1;$
$calc\_dim(stepNum, \&dimBlock, \&dimGrid);$
$CRM\_forward <<< dimGrid, dimBlock >>> (d\_a, d\_b, d\_c, d\_f, n, stepNum, i);$
}
```

Here $\log 2(n + 1) - 1$ is a step number and a variable of *stepNum*. It is to identify the block size. Therefore, we need the function *calc_dim*(), which is identifying the block sizes. After that the function *CRM_forward*() runs $\log 2(n + 1) - 1$ times. Consequently, the system reduces to one equation. After that we synchronize the device and call the function *cr_div*(). This function calculates two unknowns. Then we use one cycle

```
for $(i = \log 2(n + 1) - 2; i >= 0; i - -)$ {
$stepNum = (n - pow(2.0, i + 1))/pow(2.0, i + 1) + 1;$
$calc\_dim(stepNum, \&dimBlock, \&dimGrid);$
$CRM\_backward <<< dimGrid, dimBlock >>> (d\_a, d\_b, d\_c, d\_f, d\_x, n, stepNum, i);$
}
```

Here the backward substitution runs $\log 2(n+1)-2$ times because the first backward substitution sub-step is calculated by the function *calc_dim*(). Thus, we calculate *d_x* array. After that we copy the calculated data *d_x* from the device to the host using

$cudaMemcpy(y, \text{d\_x}, sizeof(double) * n, cudaMemcpyDeviceToHost)$.

3.2. **OpenMP+CUDA approach.**

OpenMP (Open Multi-Processing) was introduced to provide the means to implement shared memory concurrency in FORTRAN and C/C ++ programs. In particular, OpenMP defines a set of environment variables, compiler directives and library procedures that will be used for parallelization with shared memory. OpenMP was specifically designed to use certain characteristics of shared memory architectures, such as the ability to directly access memory throughout a system with low latency and very fast shared memory locking [27].

We can easy parallelize loops by using MPI thread libraries and invlove the OpenMP compilers. In this way, the threads can obtain new tasks, the unprocessed loop iterations directly from local shared memory. The basic idea behind OpenMP is data-shared parallel execution. It consists of a set of compiler directives, callable runtime library routines and environment variables that extend FORTRAN, C, and C++ programs. The working unit of OpenMP is a thread. It works well when accessing shared data costs you nothing. Every thread can access a variable in a shared cache or RAM.

In this paper, we use OpenMP to solve an initial boundary value problem. Since do deal with it we use two cycles and calculate one matrix. Moreover, OpenMP parallel computing model is very convenient to implement, and it has low latency and high bandwidth.

3.3. **Hybrid approach.**

The message passing interface (MPI) is a standardized and portable programming interface for exchanging messages between multiple processors executing a parallel program in distributed memory.

MPI works well on a wide variety of distributed storage architectures and is ideal for individual computers and clusters. However, MPI depends on explicit communication between parallel processes which requires mesh decomposition in advance due to data decomposition. Therefore, MPI can cause load balancing and consume extra time.

Since MPICH2 is freely accessible here in our implementations we use it. In [8] the authors used a compact implementation of the MPI standard for message-passing for distributed-memory applications. MPICH is a free software. Also, it is available for most types of UNIX and Microsoft Windows systems. MPI is standardized on many levels. Indeed, it provides many advantages for the users. One of them makes you sure that MPI codes can be executed in any MPI implementation launching on your architecture, even if the syntax has been standardized. Since the functional behavior of the MPI calls is also standardized, its should behave in the same way whatever of implementation, which ensures the portability of the parallel programs.

We use MPI technology to calculate the elements of the tridiagonal matrix system, i.e ai, bi, ci, fi because these values can be calculated independently, so we can successfully apply MPI technology here.

Listing code 1 shows the program code.

Listing 1. Calculation of ai, bi, ci, fi

```
 i1 = (n*rank) / size;
 i2 = (n*(rank + 1)) / size;
for (i = i1; i <i2; i++)
  {
   a_m[kk] = tau*tau;
   c_m[kk] = tau*tau;
   b_m[kk] = 2 * tau*tau + h*h;
   f_m[kk] = h*h*Unn[i] - 2 * h*h*uu0[i];
   kk++;
  }

  MPI_Gather(a_m, n/size, MPI_DOUBLE, a, n/size, MPI_DOUBLE, 0,
      MPI_COMM_WORLD);
  MPI_Gather(b_m, n/size, MPI_DOUBLE, b, n/size, MPI_DOUBLE, 0,
      MPI_COMM_WORLD);
  MPI_Gather(c_m, n/size, MPI_DOUBLE, c, n/size, MPI_DOUBLE, 0,
      MPI_COMM_WORLD);
  MPI_Gather(f_m, n/size, MPI_DOUBLE, f, n/size, MPI_DOUBLE, 0,
      MPI_COMM_WORLD);

  if (rank == 0)
  {
   MPI_Bcast(a, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
   MPI_Bcast(b, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
   MPI_Bcast(c, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
   MPI_Bcast(f, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
  }
```

These parallel technologies, CUDA, OpenMP and MPI can be combined to form a multi-layered hybrid structure, the premise is that the system has several CPU cores and at least one graphics processor. Under this hybrid structure (Figure 1), we can make better use of the advantages of another programming model.

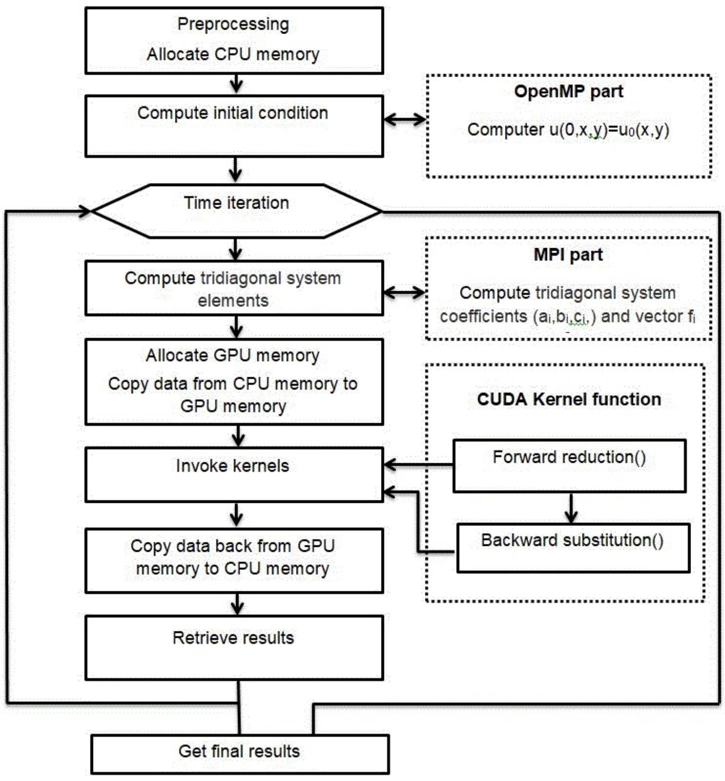
Preprocessing
Allocate CPU memory
Compute initial condition
OpenMP part
Computer u(0,x,y)=u₀(x,y)
Time iteration
Compute tridiagonal system elements
MPI part
Compute tridiagonal system coefficients (aᵢ,bᵢ,cᵢ,) and vector fᵢ
Allocate GPU memory
Copy data from CPU memory to GPU memory
CUDA Kernel function
Invoke kernels
Forward reduction()
Backward substitution()
Copy data back from GPU memory to CPU memory
Retrieve results
Get final results

Figure 1. Flowchart of hybrid approach

4. Experimental Results

In this section we show the results obtained on a desktop computer with configuration 4352 cores GeForce RTX 2080 TI, NVIDIA GPU together with a CPU Intel Core(TM) i7-9800X, 3.80 GHz, RAM 64Gb. Simulation parameters are configured as follows. Mesh size is uniform in both directions with $\Delta x = \Delta y = 0.01$, coefficients $c = 1$ and numerical time step Δt is 0.02, and simulation time is $T = 1.0$, therefore the total numerical time step is 50.

Using the implicit sub-scheme (2.11), the cyclic reduction [30] method is performed in the x direction, with the result that we get the grid function $u_{i,j}^{k+1/2}$. In the second fractional time step, using the sub-scheme (2.12), the cyclic reduction method is performed in the direction of the y axis, as the result we get the grid function $u_{i,j}^{k+1}$.

To present more realistic data we test four cases with large domain sizes of $1024 \times 1024, 2048 \times 2048, 4096 \times 4096$ and 8192×8192. In Table 1 we report the execution times in seconds for serial (CPU time), CUDA (GPU time), GPU+OpenMP, and CUDA+OpenMP+MPI implementations of the cyclic reduction method to the discrete problem (2.8)–(2.10).

Table 1. Execution Time (Seconds)

N (mesh size)	CPU	GPU	GPU/OpenMP	GPU/OpenMP/MPI		
				2 CPU core	4 CPU core	8 CPU core
1024×1024	48.13	24.104	24.151	24.432	23.232	22.61
2048×2048	189.677	45.033	45.01	35.133	33.571	30.261
4096×4096	755.614	122.24	59.996	58.797	54.223	51.413
8192×8192	3272.305	435.854	239.556	173.45	168.876	159.501

5. Conclusion and Future Work

In this paper, we proposed the numerical solution of the 2D acoustic wave equation based on an implicit finite difference scheme using the cyclic reduction method. And, we constructed a heterogeneous hybrid programming environment for a single PC by combining the message passing interface MPI, OpenMP, and CUDA programming. Also, implemented parallelization of the cyclic reduction method on the graphic processing unit. Finally, we showed how we accelerated the cyclic reduction method on the NVIDIA GPU. From the test results of table 1 it can be seen that the acceleration method proposed by us gives a good result. Our hybrid CUDA/OpenMP/MPI implementation obtained up to 2.75 times faster results than CUDA implementation.

In future work, we are planning to improve and adapt our hybrid approach for GPU cluster and test it.

References

[1] D. W. Peaceman, H. H. Rachford. The Numerical Solution of Parabolic and Elliptic Differential Equations, *Journal of the Society for Industrial and Applied Mathematics*, 3.1, 1955, issn: 03684245. url: http://www.jstor.org/stable/2098834

[2] N. Bell, M. Garland. Efficient sparse matrix-vector multiplication on CUDA, *NVIDIA Technical Report*, 2008.

[3] E. Elsen, P. LeGresley, E. Darve. Large calculation of the flow over a hypersonic vehicle using a GPU, *J. Comput. Phys*, 227,1014810161, 2008.

[4] Y. Zhang, J. Cohen, J. Owens, Fast tridiagonal solvers on the GPU, *ACM Sigplan Notices*, 45:5,127136,2010

[5] Y. Zhang, J. Cohen, A. Davidson, J. Owens, A hybrid method for solving tridiagonal systems on the GPU, *GPU Computing Gems Jade Edition*, 117,2011.

[6] A. Davidson, J. Owens Register packing for cyclic reduction: a case study, Proceedings of the FourthWorkshop on General Purpose Processing on Graphics Processing Units, ACM, 4,2011.

[7] A. Davidson, Y. Zhang, J. Owens An auto-tuned method for solving large tridiagonal systems on the GPU, *Parallel and Distributed Processing Symposium (IPDPS), IEEE International, IEEE*, 2011,956965,2011.

[8] D. Goddeke, R. Strzodka. Cyclic reduction tridiagonal solvers on GPUs applied to mixed-precision multigrid, *Parallel and Distributed Systems, IEEE Transactions*, 22:1, 2232, 2011.

[9] H. Kim, S.Wu, L. Chang, W. Hwu. A scalable tridiagonal solver for GPUs, *Parallel Processing (ICPP), 2011 International Conference on, IEEE*, 444453, 2011.

[10] N. Sakharnykh. Tridiagonal solvers on the GPU and applications to fluid simulation, *GPU Technology Conference*, 2009.

[11] Z. Wei, B. Jang, Y. Zhang, Y.Jia. Parallelizing Alternating Direction Implicit Solver on GPUs, *International Conference on Computational Science, ICCS, Procedia Computer Science* 18, 389398, 2013.

[12] F. Bodin, S. Bihan. Heterogeneous multicore parallel programming for graphics processing units, *J. Sci. Programming* 17:4, 325336, 2009. doi:10.3233/SPR-2009-0292.

[13] C.T. Yang, C. L. Huang, C. F. Lin. Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters, *Computer Physics Communications* 182,266269,2011

[14] Y. Liu, R. Xiong. A MPI + OpenMP + CUDA Hybrid Parallel Scheme for MT Occam Inversion, *International Journal of Grid and Distributed Computing* Vol. 9,9,67-82,2016 http://dx.doi.org/10.14257/ijgdc.2016.9.9.07

[15] A. L. D, J.E. Roman. MPI-CUDA parallel linear solvers for block-tridiagonal matrices in the context of SLEPcs eigensolvers, *Parallel Computing* 118, 2017

[16] D. Mu, P, Chen, L. Wang. Accelerating the discontinuous Galerkin method for seismic wave propagation simulations using multiple GPUs with CUDA and MPI. *Earthq Sci* 26(6):377393,2013 DOI 10.1007/s11589-013-0047-7

[17] P. Alonso, R. Cortina, F.J. Martnez-Zaldvar, J. Ranilla, Neville elimination on multi- and many-core systems: OpenMP, MPI and CUDA, *J. Supercomputing*, in press, doi:10.1007/s11227-009-0360-z, SpringerLink Online Date: Nov. 18, 2009.

[18] C. Garetto and M. Ruzhansky. Hyperbolic Second Order Equations with Non-Regular Time Dependent Coefficients. *Arch. Rational Mech. Anal.*, 217(1):113–154, 2015.

[19] M. Ruzhansky and N. Tokmagambetov. Wave equation for operators with discrete spectrum and irregular propagation speed. *Arch. Ration. Mech. Anal.*, 226(3):1161–1207, 2017.

[20] M. Ruzhansky, N. Tokmagambetov. Very weak solutions of wave equation for Landau Hamiltonian with irregular electromagnetic field. *Lett. Math. Phys.*, 107:591–618, 2017.

[21] M. Ruzhansky and N. Tokmagambetov. On a very weak solution of the wave equation for a Hamiltonian in a singular electromagnetic field. *Math. Notes*, 103(5–6):856–858, 2018.

[22] J. C. Munoz, M. Ruzhansky and N. Tokmagambetov. Wave propagation with irregular dissipation and applications to acoustic problems and shallow waters. *J. Math. Pures Appl.*, 123:127–147, 2019.

[23] J. C. Munoz, M. Ruzhansky and N. Tokmagambetov, Acoustic and Shallow Water Wave Propagation with Irregular Dissipation. *Funct. Anal. Appl.*, 53(2):153–156, 2019.

[24] M. Ruzhansky and N. Tokmagambetov. Wave Equation for 2D Landau Hamiltonian. *Appl. Comput. Math.*, 18(1):69-78, 2019.

[25] A.A. Samarskii. The Theory of Difference Schemes. *CRC Press*, 2001.

[26] NVIDIA, Nvidia, http://www.nvidia.com/, Accessed 2019.

[27] G. Karniadakis, R. M. Kirby. Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation *Cambridge University Press, PA-P/CDR edition*, 17-30, 2003

[28] D. Goddeke, R. Strzodka, J. Mohd-Yusof, P. McCormick, S. Buijssen, M. Grajewski, S. Tureka, Exploring weak scalability for FEM calculations on a GPU-enhanced cluster, *Parallel Comput.* 33, 685699, 2007.

[29] 2D wave GPU implementation https://github.com/Arshynbek/2Dwave-GPU-implementation

[30] R. W. Hockney. A fast direct solution of Poissons equation using Fourier analysis. *Journal of the ACM*, 12:1, 95113, 1965.

[31] J. Nickolls, I. Buck, M. Garland, K.Skadron. Scalable parallel programming with cuda. *Queue*, 6:2,4053,2008. doi:http://www.doi.acm.org/10.1145/ 1365490.1365500.

[32] A. Klockner, T. Warburton, J. Bridge, and J.S. Hesthaven. Nodal discontinuous Galerkin methods on graphics processors, *J. Comput. Phys.*, 228: 21,78637882,2009.

Arshyn Altybay:
Al-Farabi Kazakh National University
71 Al-Farabi avenue
050040 Almaty
Kazakhstan
and
Department of Mathematics: Analysis, Logic and Discrete Mathematics
Ghent University, Belgium

and
Institute of Mathematics and Mathematical Modeling
125 Pushkin str., Almaty, 050010
Kazakhstan,
E-mail address arshyn.altybay@gmail.com

Michael Ruzhansky:
Department of Mathematics: Analysis, Logic and Discrete Mathematics
Ghent University, Belgium
and
School of Mathematical Sciences
Queen Mary University of London
United Kingdom
E-mail address michael.ruzhansky@ugent.be

Niyaz Tokmagambetov:
Department of Mathematics: Analysis, Logic and Discrete Mathematics
Ghent University, Belgium
and
al–Farabi Kazakh National University
71 al–Farabi ave., Almaty, 050040
Kazakhstan,
E-mail address niyaz.tokmagambetov@ugent.be