Scientific Journal of Astana IT University
ISSN (P): 2707-9031 ISSN (E): 2707-904X
VOLUME 13, MARCH 2023

DOI: 10.37943/13SCQO3041

Altybay Arshyn
PhD, Senior Lecturer
arshyn.altybay@gmail.com, orcid.org/0000-0003-4939-8876
Al-Farabi Kazakh National University, Institute of mathematics and mathematical modeling, Almaty, Kazakhstan

Mekebayev Nurbapa
PhD, associate professor
Mekebaev.nurbapa@qyzpu.edu.kz, orcid.org/0000-0002-9117-4369
Kazakh national women's teacher training university, Almaty, Kazakhstan

Darkenbayev Dauren
PhD, Department of Computer Science
orcid.org/0000-0002-6491-8043
Faculty of Information Technologies,
Al-Farabi Kazakh National University, Almaty, Kazakhstan

A GPU IMPLEMENTATION OF THE TSUNAMI EQUATION

Abstract. In this paper, we consider numerical simulation and GPU (graphics processing unit) computing for the two-dimensional non-linear tsunami equation, which is a fundamental equation of tsunami propagation in shallow water areas. Tsunamis are highly destructive natural disasters that have a significant impact on coastal regions. These events are typically caused by undersea earthquakes, volcanic eruptions, landslides, and possibly an asteroid impact. To solve numerically, firstly we discretized these equations in a rectangular domain and then transformed the partial differential equations into semi-implicit finite difference schemes. The spatial and time derivatives are approximated by using the second-order centered differences following the Crank-Nicolson method and the calculation method is based on the Jacobi method; the computation is performed using the C++ programming language; and the visualization of numerical results is performed by Matlab 2021. The initial condition was given as a Gaussian, and the basin profile has been approximated by a hyperbolic tangent.

To accelerate the sequential algorithm, a parallel computation algorithm is developed using CUDA (Compute Unified Device Architecture) technology. CUDA technology has long been used for the numerical solution of partial differential equations (PDEs). It uses the parallel computing capabilities of graphics processing units (GPUs) to speed up the PDE solution. By taking advantage of the GPU's massive parallelism, CUDA technology can significantly speed up PDE computations, making it an effective tool for scientific computing in a variety of fields. The performance of the parallel implementation is tested by comparing the computation time between the sequential (CPU) solver and CUDA implementations for various mesh sizes. The comparison shows that our parallel implementation gives significant acceleration in the implementation of CUDA.

Keywords: Tsunami equation, finite difference scheme, numerical methods, parallel algorithm, CUDA.

Introduction

Since tsunami modelling is carried out over a very large area and a long period of time, it requires a large computational effort in the background of calculations. Therefore, in this case, it will be very important to use parallel calculations.

DOI: 10.37943/13SCQO3041

The introduction of the CUDA programming language in 2006 has made GPU programming on NVIDIA graphics cards easier. CUDA has a C-like syntax, which makes it easy to learn. GPUs are powerful processors that are highly parallel, multi-threaded, and multi-core. They have low cost, high bandwidth floating-point operations, and memory access bandwidth, which make them appealing to high-performance computing researchers [1]. Compared to cluster systems, GPUs are inexpensive and consume less power with equivalent performance. Scientists and researchers can increase their productivity by several orders of magnitude using graphics processors in many fields of science and technology. For instance, in work [2], the authors first time efficiently implemented on the CUDA platform sparse matrix-vector multiplication, and they proved that scientific computing in the CUDA environment performs very well. In [3], the authors implemented the most complex scientific simulation on GPUs at that time when they implemented fluid simulation on GPUs. In this study, we examine a GPU implementation for applications to tsunami modelling in the coastal area. Accordingly, we develop a parallel algorithm for solving numerically nonlinear tsunami equations by using the CUDA technology. In the parallel implementation of the numerical solution to the tsunami and shallow water equations, many researchers proposed their algorithms using various methods. Here, we will mention a few of them. In the pioneering work [4], Gidra et al. considered the parallelization of the tsunami equation based on the TUNAMI-N1 model. They proposed two parallel models (data-parallel and hybrid) to speed up TUNAMI-N1 and obtained good results, but the structure of the kernel functions was unclear. One of the wildly used systems describing tsunami propagation is proposed by Imamura et al. in [19]. Indeed, the described system in this manual book is a modification of the shallow water equation. That is why many researchers use shallow water equation for tsunami waves in the coastal zone. There are a lot of numerical methods developed to solve the shallow water or tsunami propagation equations. Most of the conventional tsunami models are based on finite difference leap-frog schemes, in paper [5], the authors, for the first time using finite difference leap-frog schemes, did tsunami simulation; in [6], the authors used the finite difference leap-frog schemes for the shallow-water-wave equations in the calculation of the evolution of breaking and nonbreaking waves on sloping beaches. The authors in work [7] performed numerical simulations of the 2004 Sumatra earthquake fault plane mechanisms and the Indian Ocean tsunami by utilizing a modified shallow water equation and a leap-frog scheme. Besides the finite-difference method, the finite volume method has become very popular for the numerical solution of the shallow water equation. Therefore, many parallelization approaches on GPU are based on the finite volume method; for instance, in [8], the authors proposed a hydrodynamic model based on GPU parallel computing that has been presented for tsunami simulations. In work [9], the authors presented a parallel CUDA implementation of 2D Shallow Water Equations based on the staggered grid. But here unclear parallelization process. In [10] proposed a single GPU and a multi-GPU implementation of Savage–Hutter type model using MPI and the CUDA framework over structured meshes. Parallelization of the two-layer shallow water system on GPU was proposed in [11]. Khrapov at. al. [12] proposed a parallel implementation by using OpenMP-CUDA technologies for self-consistent simulations of surface water and sediment dynamics; in this implementation they used several GPUs. In [13] presented the application of GPU-accelerated finite volume methods for simulating three-dimensional shallow water flows. Asuncion et al. [14] demonstrate the use of GPU-accelerated AMR-based application for modeling tsunamis caused by landslides through a two-layer shallow water system. Satria et al. [15] proposed the GPU Acceleration of the Tsunami Propagation Model, which is based on the two-step finite-difference MacCormack scheme. Kohei et al. [16] present high-performance and power-efficient computation of MOST for practical tsunami simulation with FPGA. Parent's work [17] presents a GPU implementation for the real-time solution of shallow

Scientific Journal of Astana IT University
ISSN (P): 2707-9031 ISSN (E): 2707-904X
VOLUME 13, MARCH 2023

water equations. Their approach is also based on a finite difference scheme. Zhai et al. [18] propose parallel numerical implementation of two-phase shallow granular ow equations on multiple GPUs. In this work, we propose a CUDA implementation to simulate tsunami wave propagation in the coastal area.

Governing equation and Finite difference schemes

In [19] fundamental equations of the tsunami propagation in shallow water are given in the following form

$$\frac{\partial \eta}{\partial t} + \frac{\partial M}{\partial x} + \frac{\partial N}{\partial y} = 0 \tag{1}$$

$$\frac{\partial M}{\partial t} + \frac{\partial}{\partial x}\left(\frac{M^2}{D}\right) + \frac{\partial}{\partial y}\left(\frac{MN}{D}\right) + gD\frac{\partial \eta}{\partial x} + \frac{gn^2}{D^{7/3}} M\sqrt{M^2+N^2} = 0 \tag{2}$$

$$\frac{\partial N}{\partial t} + \frac{\partial}{\partial x}\left(\frac{MN}{D}\right) + \frac{\partial}{\partial y}\left(\frac{N^2}{D}\right) + gD\frac{\partial \eta}{\partial y} + \frac{gn^2}{D^{7/3}} N\sqrt{M^2+N^2} = 0 \tag{3}$$

where is water surface elevation, h is the depth of the water, $D = h(x,y) + \eta$ the total water depth, M and N are discharge fluxes in the x and y directions, g is the gravitational constant, n is a coefficient of bottom friction. In what follows, we will suppose that the tsunami propagation area is sufficiently large such that assuming the Dirichlet boundary condition on its boundary is natural.

Finite difference schemes. We rely on a rectangular time-space grid. The time discretization is $t^k = k\Delta t$ and the spatial grid is $x_i = i\Delta x, y_i = j\Delta y$. The particle velocity vector field and the discharge flux are dened at integer spatial and time nodes: $\eta_{i,j}^k, M_{i,j}^k, N_{i,j}^k$ Let

$$L_x = h_1 Q_1, L_y = h_2 Q_2, T = \tau P,$$

Where L_x and L_y are the length of the domain in the x and y directions, T is the total time, Q_1,; Q_2, the grid size, $h_1; h_2; \tau$, respectively, the spatial and time step. For simplicity, we put $Q := Q_1 = Q_2$ and denote $h := h_1; h_2$. We used finite difference method to approximate 2D Tsunami. The spatial derivatives were approximated by using the second-order centered differences. The time derivatives were approximated by using the second-order centered differences following the Crank-Nicolson method. The equation (1), therefore, can be approximated by

$$\frac{\eta_{i,j}^{k+1} - \eta_{i,j}^k}{\tau} + \frac{M_{i+1,j}^{k+1} + M_{i-1,j}^{k+1}}{2h} + \frac{N_{i,j+1}^{k+1} + N_{i,j-1}^{k+1}}{2h} = 0 \tag{4}$$

$$\frac{M_{i,j}^{k+1} - M_{i,j}^k}{\tau} + \frac{(\frac{M^2}{D})_{i+1,j}^k - (\frac{M^2}{D})_{i-1,j}^k}{2h} + \frac{(\frac{MN}{D})_{i,j+1}^k - (\frac{MN}{D})_{i,j-1}^k}{2h} + gD_{i,j}^k \frac{\eta_{i+1,j}^k - \eta_{i-1,j}^k}{2h} + \frac{gn^2}{D_{i,j}^{7/3}} M_{i,j}^k \sqrt{M_{i,j}^2 - N_{i,j}^2} = 0 \tag{5}$$

$$\frac{N_{i,j}^{k+1} - N_{i,j}^k}{\tau} + \frac{(\frac{MN}{D})_{i+1,j}^k - (\frac{MN}{D})_{i-1,j}^k}{2h} + \frac{(\frac{N^2}{D})_{i,j+1}^k - (\frac{N^2}{D})_{i,j-1}^k}{2h} + gD_{i,j}^k \frac{\eta_{i+1,j}^k - \eta_{i-1,j}^k}{2h} + \frac{gn^2}{D_{i,j}^{7/3}} N_{i,j}^k \sqrt{M_{i,j}^2 - N_{i,j}^2} = 0 \tag{6}$$

for $(i,j,k) \in \omega_{h_1,h_2}^{\tau}$, $h_{i,j}^k = h(ih, jh)$,

DOI: 10.37943/13SCQO3041

$$\eta_{i,j}^0 = \varphi_{i,j}, N_{i,j}^0 = \varphi_{i,j}, M_{i,j}^0 = 0 \quad (7)$$

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

$$\begin{aligned} &\eta_{0,j}^k = 0, \eta_{N,j}^k = 0, \eta_{i,0}^k = 0, \eta_{i,N}^k = 0, \\ &N_{0,j}^k = 0, N_{N,j}^k = 0, N_{i,0}^k = 0, N_{i,N}^k = 0, \\ &M_{0,j}^k = 0, M_{N,j}^k = 0, M_{i,0}^k = 0, M_{i,N}^k = 0, \end{aligned} \quad (8)$$

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

GPU Implementation

CUDA (Compute Unified Device Architecture) was developed to enable multi-core parallelization on NVIDIA's GPUs. It includes a software model and a set of compilation tools that support Many-Core GPUs. A CUDA program consists of two parts: a sequential program that runs on the CPU and a parallel part that runs on the GPU. The parallel part is called the kernel, and a C program with CUDA extensions distributes multiple copies of the kernel to available multiprocessors for concurrent execution. The CUDA code has three computational stages: transferring data to global GPU memory, running the CUDA core, and transferring results from the GPU to CPU memory. Due to the CFL condition and the need to capture all topographic details, implementing the method efficiently is critical. While most processors have multithreading capabilities, it is not straightforward to use all cores. To make the iterative process parallelizable, it can be divided into different sub-processes that can be processed concurrently by different units, provided the loop contains straight-line code and has no data dependency between iterations. If the data flow does not satisfy these conditions, the method for processing elements to make them parallelizable must be restructured. In this work, we used the latest NVIDIA computing architecture, including the Turing TU102 architecture and NVIDIA GeForce RTX 2080Ti with 11 GB of GDDR6 memory.

Fig. 1 shows the general steps of the parallel algorithm, here, we calculate M, N, η on the graphics processor. In this study, we use 3 kernel functions such as M, N, η, and M, N, η kernel functions are performed independently of each other, after getting M, N then synchronization is performed. Then η is calculated by the obtained results of M, N.

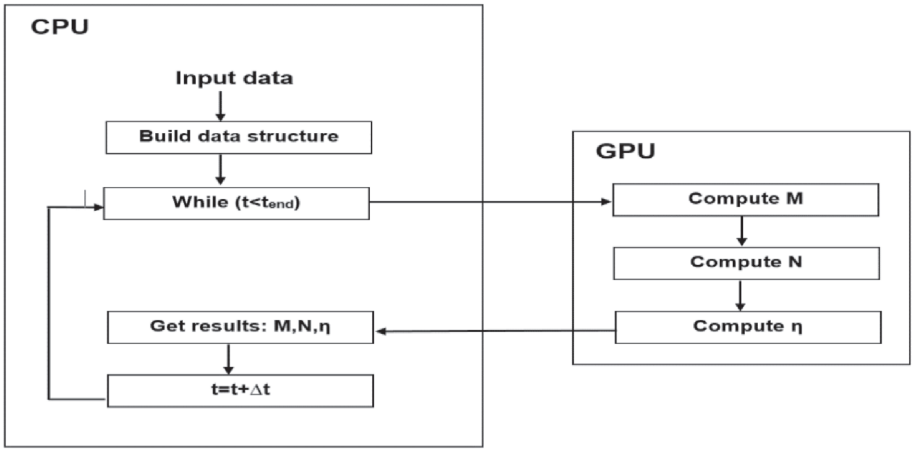

Figure 1. General steps of the parallel algorithm implemented in CUDA

CUDA implementation. In 2007, NVIDIA introduced CUDA, an extension to C programming language, for general purpose computing on graphics processors. The design of the algorithm

Scientific Journal of Astana IT University
ISSN (P): 2707-9031 ISSN (E): 2707-904X
VOLUME 13, MARCH 2023

enables a straightforward way of expressing parallelism in terms of data level. The steps involved in solving equations (4)-(6) are outlined in Algorithm 1.

Algorithm 1 Implementation of 2D tsunami equation

```
compute initial condition matrix M_0, N_0, η_0
repeat
  for j ← 0 to n do
    for i ← 0 to n do
      copy host to device M_0, N_0, η_0
      call Device functions M(M_0, n),N(N_0, n),η(M, N, η_0, n)
      copy results M, N, η from device to host
    end for
  end for
  swap(M_0, M)
  swap(N_0, N)
  swap(η_0, η)
  t ← t + Δt
until t <= T
```

Experiment

This section presents the outcomes achieved on a desktop computer with a configuration of 4352 cores GeForce RTX 2080 TI, NVIDIA GPU, and an Intel Core(TM) i7-9800X CPU running at 3.80 GHz, with 64 GB RAM. The simulation parameters are set as follows. Mesh size is uniform in both directions with $\Delta x = \Delta y = 0.5$, and numerical time step Δt is 0.05 s, and simulation time is $\mathrm{T} = 5.0$ s, therefore the total number of time steps is 100. To provide more accurate information, we conducted experiments on five scenarios with varying domain sizes: 200×200, 500×500, 1000×1000, 2000×2000, and 4000×4000.

In the following, we demonstrate numerical simulations. All calculations and visualization are done in Matlab by using the simple iterative method. For all simulations $\triangle t = 0.05$, $\triangle x = \triangle y = 0.5$. For the simulations, we use initial conditions

$$\tau(x, y, 0) = e^{-\frac{(x-30)^2}{10} - \frac{(x-30)^2}{20}},$$

and Dirichlet boundary conditions. The sea basin is modeled in the form of the hyperbolic tangent. The depth prole is given as

$$h(x, y) = 50 - 45\ tahn\ (\frac{-(x-50)^2}{20}).$$

The result of the numerical simulation is illustrated in Fig. 2.

DOI: 10.37943/13SCQO3041

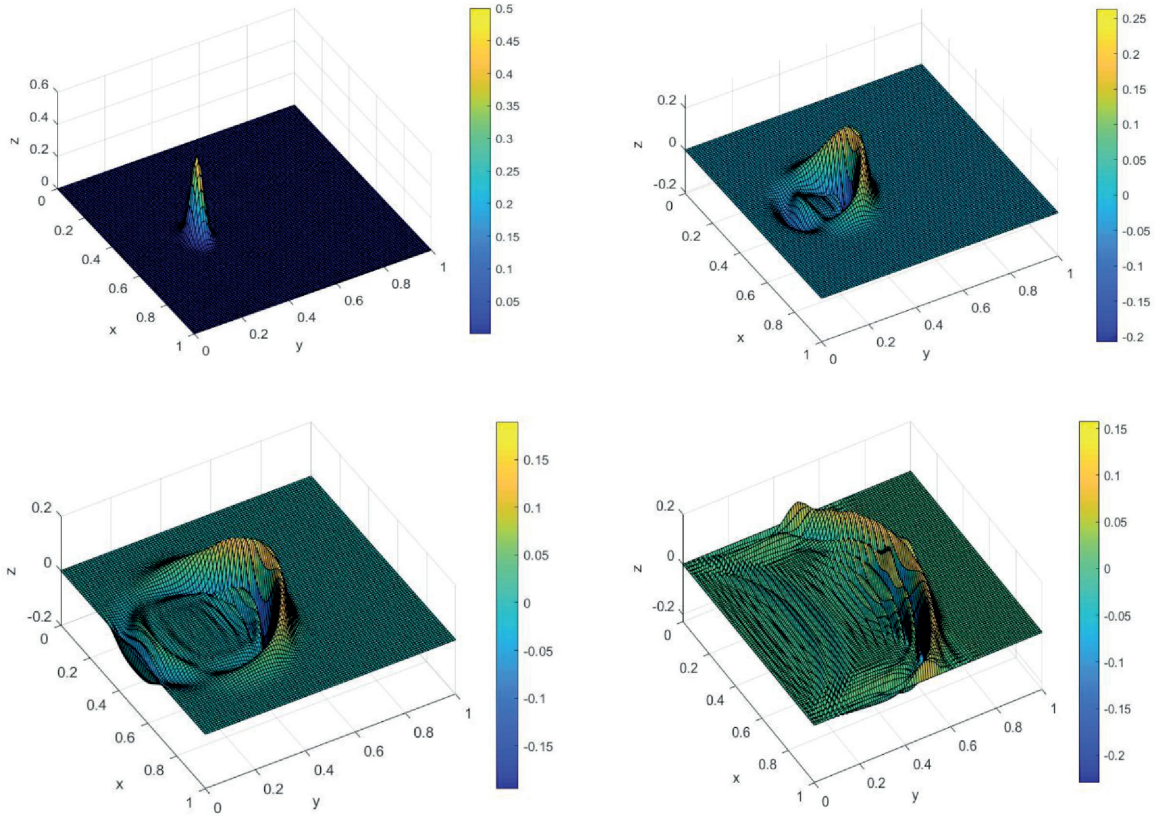

Figure 2. Displacement of the wave at different times (t = 0; t = 0.5; t = 1.0; t = 2.0).

Table 1 shows the CPU time and GPU time in seconds for solving the problem (1)-(4) using serial and CUDA methods. The speedup of the process can be seen in Fig. 3.

Table 1. Execution time and speedup with the Intel Core(TM) i7-9800X, 3.80 GHz, NVIDIA RTX 2080 TI

Domain size	CPU time	GPU time
200 × 200	1.128	0.52
500 × 500	6.883	2.47
1000 × 1000	27.324	9.5
2000 × 2000	107.21	23.62
4000 × 4000	434.502	79.66

Table 1 shows that the execution time in the GPU is significantly faster than the CPU, noting the domain size.

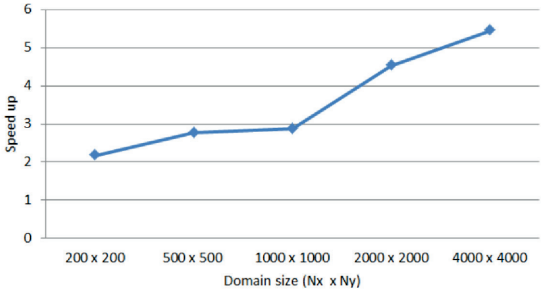

Figure 3. Speedup

Scientific Journal of Astana IT University
ISSN (P): 2707-9031 ISSN (E): 2707-904X
VOLUME 13, MARCH 2023

From this figure 3, it can be seen that the speedup slowly increases for domain sizes up to 1000*1000 and increases rapidly for domain sizes greater than 1000*1000, this is due to the transfer time for copying data between the host and the device.

Conclusion

In this paper, we have introduced a parallel implementation of a numerical solution to the two-dimensional tsunami wave equation using a graphic processing unit. The numerical method using finite differences was parallelized, modified to be compatible with the GPU, and coded using the CUDA framework in order to utilize the GPU's parallel processing capability. The simulation time was compared to CPU implementation by varying the mesh size. We observe that GPU simulations are much faster than CPU simulations.

In the future, we plan to solve this issue on GPU clusters with an aim to simulate wave propagation over larger domain sizes. We also aim to explore the possibility of using fully implicit finite difference schemes and solving them via the parallel cyclic reduction method or the Thomas algorithm.

References

1. Klockner, A., Warburton, T., Bridge, J., & Hesthaven, J.S. (2009). Nodal discontinuous Galerkin methods on graphics processors, *Journal of Computational Physics*, *228*(21),7863-7882. https://doi.org/10.1016/j.jcp.2009.06.041
2. Bell, N., & Garland, M. (2008). *Efficient sparse matrix-vector multiplication on CUDA* (Vol. 2, No. 5). Nvidia Technical Report NVR-2008-004, Nvidia Corporation.
3. Elsen, E., LeGresley, P., & Darve, E. (2008) Large calculation of the flow over a hypersonic vehicle using a GPU, *Journal of Computational Physics*, 227, 10148-10161. https://doi.org/10.1016/j.jcp.2008.08.023
4. Gidra, H., Haque, I., Kumar, N.P., Sargurunathan, M., Gaur, M.S., Laxmi, V., ... & Singh, V. (2011, September). Parallelizing TUNAMI-N1 Using GPGPU. In *2011 IEEE International Conference on High Performance Computing and Communications* (pp. 845-850). IEEE. https://doi.org/10.1109/HPCC.2011.120
5. Goto, C., Ogawa, Y., Shuto, N., & Imamura, F. (1997). Numerical method of tsunami simulation with the leap-frog scheme. *IOC Manuals and Guides*, *35*, 130.
6. Titov, V. V., & Synolakis, C. E. (1995). Modeling of breaking and nonbreaking long-wave evolution and runup using VTCS-2. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, *121*(6), 308-316. https://doi.org/10.1061/(ASCE)0733-950X(1995)121:6(308)
7. Wang, X., & Liu, P.L.F. (2006). An analysis of 2004 Sumatra earthquake fault plane mechanisms and Indian Ocean tsunami. *Journal of Hydraulic Research*, *44*(2), 147-154. https://doi.org/10.1080/00221686.2006.9521671
8. Amouzgar, R., Liang, Q., Clarke, P.J., Yasuda, T., & Mase, H. (2016). Computationally efficient tsunami modeling on graphics processing units (GPUs). *International Journal of Offshore and Polar Engineering*, *26*(02), 154-160. https://doi.org/10.17736/ijope.2016.ak10
9. Arnoldy, A., & Adytia, D. (2019, July). Performance of Staggered Grid Implementation of 2D Shallow Water Equations using CUDA Architecture. In *2019 12th International Conference on Information & Communication Technology and System (ICTS)* (pp. 286-290). IEEE. https://doi.org/10.1109/ICTS.2019.8850930
10. Asunción, M., Castro, M.J., Mantas, J.M., & Ortega, S. (2016). Numerical simulation of tsunamis generated by landslides on multiple GPUs. *Advances in Engineering Software*, *99*, 59-72. https://doi.org/10.1016/j.advengsoft.2016.05.005
11. Asunción, M., Mantas, J.M., & Castro, M.J. (2010). Programming CUDA-based GPUs to simulate two-layer shallow water flows. In *Euro-Par 2010-Parallel Processing: 16th International Euro-Par Conference, Ischia, Italy, August 31-September 3, 2010, Proceedings, Part II 16* (pp. 353-364). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-15291-7_32

DOI: 10.37943/13SCQO3041

12. Khrapov, S.S., & Khoperskov, A.V. (2020). Application of Graphics Processing Units for self-consistent modelling of shallow water dynamics and sediment transport. *Lobachevskii Journal of Mathematics*, *41*, 1475-1484. https://doi.org/10.1134/S1995080220080089
13. Boubekeur, M., Benkhaldoun, F., & Seaid, M. (2017). GPU accelerated finite volume methods for three-dimensional shallow water flows. In *Finite Volumes for Complex Applications VIII-Hyperbolic, Elliptic and Parabolic Problems: FVCA 8, Lille, France, June 2017 8* (pp. 137-144). Springer International Publishing. https://doi.org/10.1007/978-3-319-57394-6_15
14. Asunción, M., & Castro, M. J. (2017). Simulation of tsunamis generated by landslides using adaptive mesh refinement on GPU. *Journal of Computational Physics*, *345*, 91-110. https://doi.org/10.1016/j.jcp.2017.05.016
15. Satria, M.T., Huang, B., Hsieh, T.J., Chang, Y.L., & Liang, W.Y. (2012). GPU acceleration of tsunami propagation model. *IEEE Journal of Selected Topics in Applied Earth Observations and remote Sensing*, *5*(3), 1014-1023. https://doi.org/10.1109/JSTARS.2012.2199468
16. Nagasu, K., Sano, K., Kono, F., & Nakasato, N. (2017). FPGA-based tsunami simulation: Performance comparison with GPUs, and roofline model for scalability analysis. *Journal of Parallel and Distributed Computing*, *106*, 153-169. https://doi.org/10.1016/j.jpdc.2016.12.015
17. Parna, P., Meyer, K., & Falconer, R. (2018). GPU driven finite difference WENO scheme for real time solution of the shallow water equations. *Computers & Fluids*, *161*, 107-120. https://doi.org/10.1016/j.compuid.2017.11.012
18. Zhai, J., Liu, W., & Yuan, L. (2016). Solving two-phase shallow granular flow equations with a well-balanced NOC scheme on multiple GPUs. *Computers & Fluids*, *134*, 90-110. https://doi.org/10.1016/j.compuid.2016.04.0
19. Imamura F. & Yalcine A.C. (2006). Tsunami Modeling Manual, 58 pages. Retrieved from http://www.tsunami.civil.tohoku.ac.jp/hokusai3/J/projects/manual-ver-3.1.pdf
20. NVIDIA TURING GPU ARCHITECTURE. Graphics Reinvented. Retrieved from https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf
21. Altybay, A., Ruzhansky, M., & Tokmagambetov, N. (2020). A parallel hybrid implementation of the 2D acoustic wave equation. *International Journal of Nonlinear Sciences and Numerical Simulation*, *21*(7-8), 821-827. https://doi.org/10.1515/ijnsns-2019-0227