DOI: 10.14529/jsfi250204

Prospects for Improving Computational Efficiency of Hydrodynamic Simulations on Supercomputers by Increasing the Number of GPUs per Compute Node

Sergei S. Khrapov[1], *Ekaterina O. Agafonnikova*[1], *Alexander V. Khoperskov*[1]

Hydrodynamic models for studying surface water dynamics on realistic topography place special demands on computational performance. Such simulations must cover large areas to ensure hydrological connectivity of the territory due to the influence of catchment areas. On the other hand, small topographic inhomogeneities on the scale of a meter are often the determining factors of fluid dynamics. Our analysis is based on a model of surface water and sediment dynamics for a large mountainous area of the Krasnodar region under rainfall/runoff conditions. The results of such large-scale models can be provided by parallel OpenMP-CUDA codes for computing systems with multi-GPU. We focus on different ways of transferring data between GPUs using both GPUDirect and HostCopy technologies on computing systems with one to eight GPUs. The parallel code with HostCopy is on average several times slower and less efficient compared to the GPUDirect approach. We propose to use auxiliary characteristics to analyze the efficiency of parallel implementation of a numerical algorithm. These values are calculated based on the average processing time of one computational cell and allow us to determine the optimal grid resolution in terms of performance.

Keywords: computational fluid dynamics, GPUs, efficiency, scalability, data transfer.

Introduction

Simulations of hydrological regimes for specific regions of the Earth often require outstanding computational resources, approaching the needs of climate modeling [2, 12, 21, 35]. This is due to at least three factors. First, the vastness of the simulated territory is determined by the length of the river system, the large area of drainage basins or hydrological landscape. Second, the need to use very fine grids is dictated by the presence of small-scale heterogeneities in topography and distributions of other physical characteristics that can significantly affect the results. The required spatial resolution in urban flooding conditions can be less than one meter, which implies hundreds of millions of cells in the digital elevation model (DEM) for specific urbanized areas [7]. Finally, the study of some hydrological processes requires the construction of long-term data series over a year or more ($T \sim 10^8$ sec), which already for two-dimensional models gives the number of integration time steps of the order of 10^9 in typical land surface hydrology problems. Flood forecast models are in high demand for assessing the consequences and identifying critical factors in the development of decision support systems and real-time emergency warning tools [2, 7, 34, 35]. Flood behavior is determined by the presence of physical factors that provide significant impacts at multiscale levels. Therefore, an interesting direction could be a 2D shallow water model using subgrid-scale topography for the 1D model as internal boundary conditions [33]. This provides additional savings in computing resources in multiscale problems.

The price-to-computing efficiency ratio dictates the need for a mass transfer of hydrodynamic simulation software to multi-GPU systems [9, 17, 21, 24, 37, 38]. This is facilitated by the presence of several reliable and convenient programming systems (CUDA, OpenCL and Ope-

[1]Volgograd State University, Volgograd, Russian Federation

nACC). The organization of parallel simulations on several GPUs has numerous peculiarities and subtleties that require special analysis. Improving the efficiency of code performance on GPUs involves special optimization for specific hardware, which reduces the portability of the software. However, hardware-oriented algorithms can provide higher performance if we take into account both the GPU architecture from different manufacturers (NVIDIA, AMD, Intel) and the specific computational task [12, 26, 31]. The authors [9] highlight the difficulties of organizing work with the memory of several CPUs/GPUs as a critical problem of parallelization.

A significant problem of multi-GPU simulations is the deterioration of scalability due to the complex data transfer between the host and device, which depends on the features of data communication between processes [1]. For example, unstructured CFDs are provided with more sophisticated algorithms to optimize the performance of parallel computing compared to simpler structured Cartesian grids [37]. There are studies showing a strong decrease in scalability in systems with a large number of GPUs [38], which requires special studies for each specific numerical model [9]. Parallelization of implicit numerical schemes on multi-GPU requires significant efforts and special algorithms, since global memory access and global dependence of variables are required [3]. Different programming languages and compilers imply the presence of additional ways to optimize calculations and parallelize codes [24]. There is some progress in the transition to new high-performance interconnect architectures for supercomputers, which improves the capabilities of the MPI interface [29].

Specific software implementations of CFD in different approximations for GPUs demonstrate their effectiveness. The wetland flooding model using multiple GPUs on the Wetland DEM software yields a speedup of 2.39x on four GPUs compared to a single unit [21]. Unstructured CFD simulations in [37] show good performance on a single GPU (Nvidia Tesla V100), achieving an average speedup of 13.4x compared to 28 CPU cores (Intel Xeon Gold 6132). The MPI-OpenACC method for parallel simulation of Huizhou City Mountain Flood achieves a speedup of over 800 using 8 GPUs [7]. Free-surface fluid flow models with complex solid boundary surfaces based on the SPH method show great parallelism potential on GPUs with 10^9 particles [6]. An example is the C++ and CUDA-based DualSPHysics software, which is widely used for both research and engineering tasks [5].

The purpose of this work is to determine the impact of different parallel communication methods in a multi-GPU computing system on minimizing the overhead associated with data transfer. The article is organized as follows. Section 1 contains descriptions of the mathematical model and numerical algorithms. In Section 2, we discuss the features of setting up computational experiments, OpenMP-CUDA algorithm, implementing CSPH-TVD (Combined Smoothed Particle Hydrodynamics – Total Variation Diminishin) method for hydrological simulations. The following section shows the result of simulations of the consequences of storm rain for the Southern area of the Krasnodar region. Section 4 is devoted to the analysis of the efficiency of our parallel code on multi-GPU computing systems. Finally, the conclusion summarizes the study and outlines potential future research topics.

1. Mathematical Model and Numerical Implementation

The equations of surface water dynamics at high velocities must describe the self-consistent movement of water and sediment [10, 22, 23]. The standard Saint-Venant system of equations

includes changes in the topography of b with time also [15, 20, 27, 30]

$$\frac{\partial H}{\partial t} + \nabla_\perp (H\mathbf{u}) = q\,, \tag{1}$$

$$\frac{\partial (H\mathbf{u})}{\partial t} + \nabla_\perp (H\mathbf{u}\otimes\mathbf{u}) = -gH\nabla_\perp(b+H) + H\mathbf{f}\,, \tag{2}$$

where $H(x,y,t)$ is the water depth, $\mathbf{u} = \{u(x,y,t), v(x,y,t)\}$ is the average velocity vector of surface water flow, $\nabla_\perp = \{\partial/\partial x, \partial/\partial y\}$, (x,y), $g = 9.81$ m/s^2, $b(x,y,t)$ is the digital elevation model (DEM), which defines the topography of the computational domain, $\mathbf{f}$ is the specific force of hydraulic resistance to the flow of water. The source function $q(x,y,t)$ determines the rate of inflow ($q > 0$) or outflow ($q < 0$) of liquid in the surface layer of water.

The temporal variability of the DEM is a result of sediment transport. A simple but effective model of sediment transport is the Exner equation [4, 10, 16, 22]

$$(1-\psi)\frac{\partial b}{\partial t} + \nabla_\perp \mathbf{J}_b = c_J\,\nabla_\perp\left(|\mathbf{J}_b|\nabla_\perp b\right)\,, \tag{3}$$

where $\mathbf{J}_b$ is the horizontal sediment flux, ψ is the soil porosity, c_J is an empirical value depending on the type and condition of the soil ($c_J \simeq$ 1–10 [SI system]), as an analogue of the diffusion coefficient [11]. The value of $\mathbf{J}_b$ on a flat bottom is determined by the Grass formula [20, 22, 27]:

$$\mathbf{J}_b = \begin{cases} a_J\,\mathbf{u}\,|\mathbf{u}|^{m_J}, & |\mathbf{u}| > u^{(cr)} \\ 0, & |\mathbf{u}| \le u^{(cr)} \end{cases}\,, \tag{4}$$

where $u^{(cr)}$ is the critical velocity determined by the Shamov formula [28], a_J and m_J ($-1 \le m_J \le 3$) are the adjustable parameters [10, 20, 27].

The choice of the hydraulic resistance model is not trivial, since it relies on subgrid physics as well as sediment transport modeling. The traditional description of hydraulic resistance only through the Manning roughness coefficient n_M [7, 10, 14, 33] requires its complex calibration. The additional influence of internal friction due to turbulence allows for a better fit of simulations with measured data [12, 15, 18], and we follow this approach of taking these factors into account here. The sum of b and H defines the water surface level $\eta(x,y,t) = b + H$, the measurements of which at gauging stations provide validation of numerical models by fitting measured and model time series $\eta(t)$. Thus, the value of $\mathbf{f}$ consists of two components and includes resistance due to roughness $\mathbf{f}^{(M)}$ and turbulence $\mathbf{f}^{(turb)}$ [12, 18]:

$$\mathbf{f} = \mathbf{f}^{(M)} + \mathbf{f}^{(turb)} = g\frac{n_M^2\,|\mathbf{u}|}{H^{4/3}}\mathbf{u} + \hat{\alpha}H^{\gamma}|\mathbf{u}|^{1-\gamma}\,\mathbf{u}\,, \tag{5}$$

where n_M, $\hat{\alpha}$, γ are the free parameters.

We use the Lagrangian-Eulerian numerical scheme Combined Smoothed Particle Hydrodynamics – Total Variation Diminishin (CSPH-TVD) [14, 16], which combines the strengths of both approaches at different steps of integration of the system of equations (1)–(3). A purely Lagrangian method SPH is complemented by the advantages of the grid algorithm for the exact calculation of the fluxes of physical quantities. Smoothed particles move within their cells under the action of hydrodynamic and external forces in the first step of the algorithm. Then, the mass and momentum fluxes are calculated through the boundaries of the cells of a fixed grid at the intermediate time $t_{n+1/2} = t_n + \Delta t_n/2$ using the modified TVD approach and an approximate

solution of the Riemann problem. The third stage involves the return of SPH particles to the cell centers, which gives the state of the system at time $t_{n+1} = t_n + \Delta t_n$.

The function $q(x, y, t)$ in (1) specifies the precipitation rate in the model. The rain front moves with the velocity $\mathbf{u}^{(r)}$ from left to right in the base model. We roughly reproduce the disaster in 2012, when a rain storm led to mountain torrents and severe flooding of Krymsk City and other settlements in the Krasnodar region [19]. We analyze different variants of rain characteristics. The base model is the velocity $\mathbf{u}^{(r)} = 30$ m sec^{-1} with the intensity 30–50 mm h^{-1}. This model is intended primarily to study the efficiency of parallelization and we do not solve the problem of exact reproduction of the flooding of this area. The uniform motion of the rain front ensures that the maximum number of cells are included in the calculations. The presence of $\mathbf{u}^{(r)}$ velocity and heterogeneity of topography lead to a complex structure of water flows and maximum computational load.

2. Parallel CUDA Implementation of the CSPH-TVD Scheme

2.1. Problem Statement

The analysis of computational efficiency depending on the amount of memory used and the method of data transfer between units is convenient when solving a problem with a computational domain in which the maximum number of grid cells contains water. Therefore, we consider the flooding of a large region with complex topography under the action of heavy rains. A good challenge is the choice of a mountainous area, which creates a complex spatial structure of surface water flows with a rapid change in the number of flooded cells due to rainfall/runoff. Examples of such extreme events are numerous and they are actively studied through numerical simulations [25, 32, 34]. We focus on a series of catastrophic events in the Krasnodar region as a geographic reference, when precipitation in the southern mountainous part created horrific flood phenomena, as in Krymsk City and other settlements in 2002 and 2012 [19].

Figure 1 shows the computational domain covering the Black Sea coast of the Krasnodar region. The basis is the digital elevation model $b_0(x_i, y_j) = b(x_i, y_j, t = 0)$ in the equations (2), (3) with a cell size of 15 meters. The flat part of the Krasnodar region is separated from the Black Sea by the Caucasus mountain ranges along the latitudinal direction. Storm rains in the mountains can create flood waves for settlements in the foothills, such as the cities of Krymsk and Abinsk, the Nizhnebakanskaya, Neberdzhaevskaya, Kholmskaya, Azovskaya, Severskaya, Verkhnebakansky, Ilsky, Erivansky, Sinegorsk settlements and others. In the simulations, we do not specify the initial water in the region of the Sea of Azov, where the DEM coincides with the sea surface level. We should note the work [34], which in detail analyzes flood situations in the area of the Dyurso River, located inside our computational domain slightly west off the Novorossiysk City (Fig. 1).

2.2. Computing System Topology

The development of a parallel algorithm of the CSPH-TVD method for CPU-multi-GPUs hybrid computing systems is based on OpenMP (CPU – Host level) and CUDA (GPU – Device level) technologies. Using OpenMP at the Host level allows us to create eight parallel threads (OpenMP Threads) that provide parallel data loading and launch of CUDA Kernels of the CSPH-TVD numerical algorithm on eight GPUs (GPU 0–7). We have created two versions of

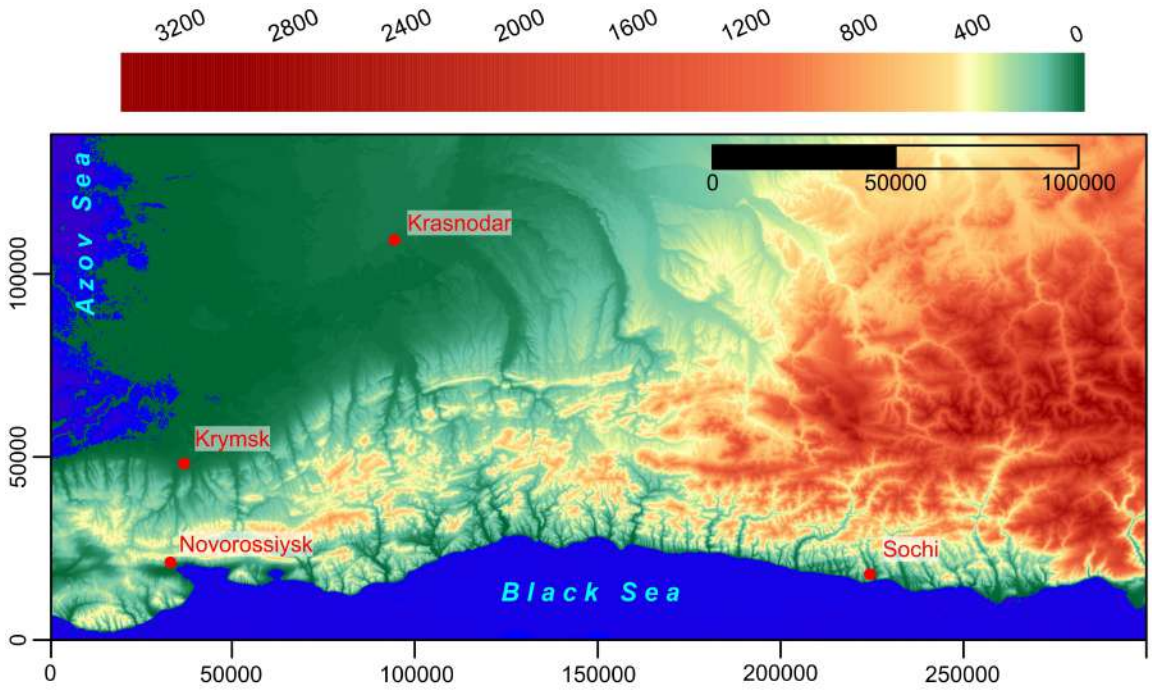


Figure 1. Topography of computational domain in the form of a digital elevation model

the parallel multi-GPU code. The first version uses the GPUDirect approach, which provides direct data exchange between GPUs via the NVLINK interface. Synchronization of calculations on different GPUs at each moment in time occurs directly due to the common address space of multi-GPUs, which is created programmatically at the Device level. This allows one GPU to directly access the memory of another and not involve the CPU (Host). The second version of our code uses the HostCopy approach instead of GPUDirect. This provides data synchronization between GPUs at each time step by copying data from the GPU to the CPU and back.

Figure 2 shows the flow diagram of our code for hydrological simulations based on the system of equations (1)–(3), which are solved by the CSPH-TVD method. The software is intended for CPU-multi-GPU computing systems. The maximum number of GPUs is eight (GPU 0, GPU 1, ..., GPU 7). The flow diagram demonstrates the principle of organizing computations on a hybrid CPU + 8 GPU system using parallel OpenMP-CUDA technologies and different approaches to data transfer between GPUs (GPUDirect and HostCopy). The initial loading of the source data in the form of arrays of hydrodynamic quantities is performed at the Host level. Arrows of different colors show the method of memory transfer between GPUs. The HostCopy approach uses low-speed PCI Express (PCIe) and QuickPath Interconnect (QPI) connections. According to NVIDIA DGX-1 documentation, each V100 GPU has a 16-channel connection to PCIe Switches, which provides a data exchange rate of ~ 16 GB/s. The processor E5-2698v4 (CPUs) in the NVIDIA DGX-1 system has two QPI links with a speed of ~ 9.6 GT/s (T=Transfer). One unit of one transfer has a width of = 80 bit. Thus, the QPI connection is characterized by a data transfer rate, which is: 2(link) $\times$ 9.6 GT/s $\times$ 80 bit = 1536 Gbit/s = 192 GB/s. GPUDirect technology enables higher data transfer rates of up to 50 GB/s via the NVLink interface. The data transfer rate can vary from 25 GB/s (red arrows in Fig. 2) to 50 GB/s (magenta arrows in Fig. 2) depending on the connection sequence of GPUs on a multi-GPU platform. For example,

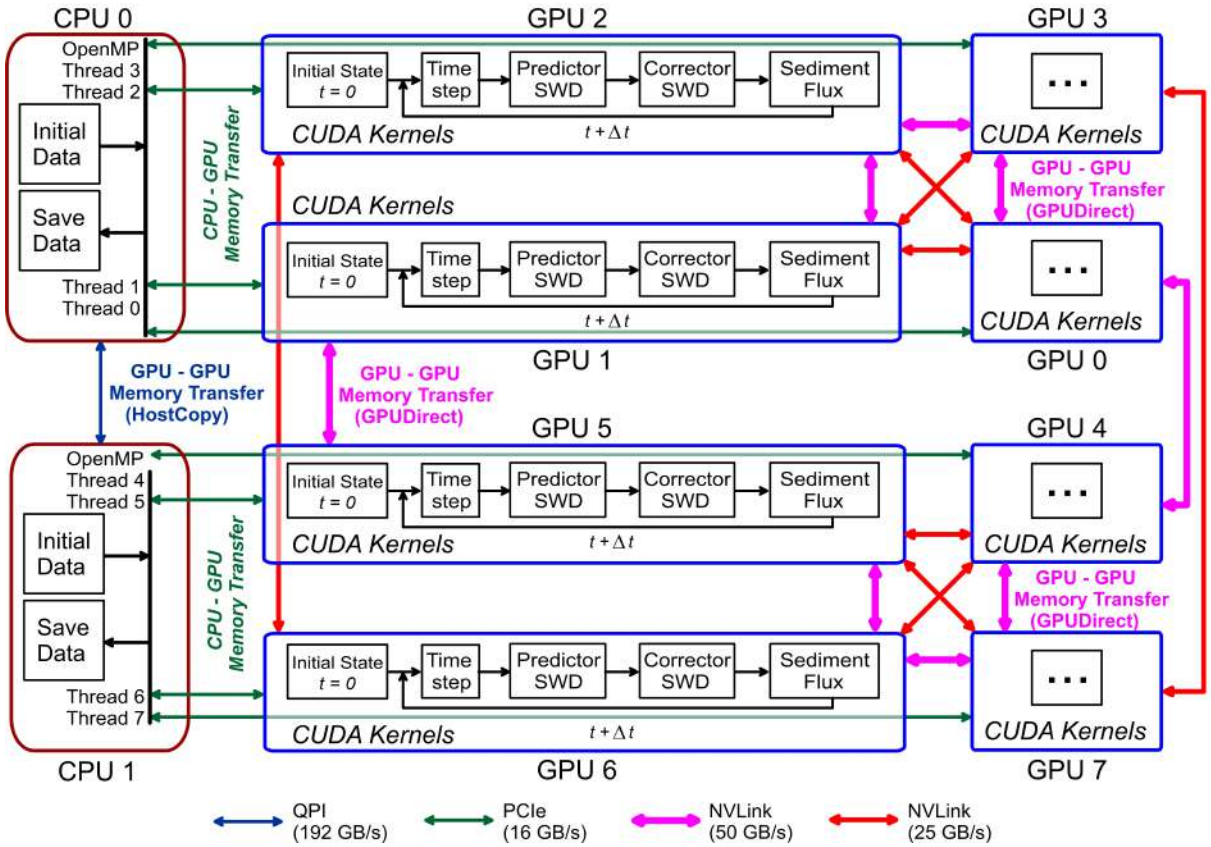


Figure 2. Flow diagram of the parallel OpenMP-CUDA algorithm of CSPH-TVD method for CPU-multi-GPU computing systems

the maximum data transfer rate (50 GB/s) for 8 GPUs is achieved on the NVIDIA DGX-1 supercomputer with the following connection method: GPU 0 – GPU 3 – GPU 2 – GPU 1 – GPU 5 – GPU 6 – GPU 7 – GPU 4.

The implementation of the numerical algorithm CSPH-TVD is based on the following main CUDA Kernels.

1) The initial configuration of water flow is created by the CUDA Kernel ≪Initial State≫.

2) The numerical stability condition of the CSPH-TVD algorithm gives the time step of integration in the CUDA Kernel ≪Time Step≫.

3) CUDA Kernel ≪Predictor SWD≫ ensures the execution of the predictor step when integrating the system of equations (1)–(3) at the Lagrangian stage of the CSPH-TVD method. The result are intermediate values of hydrodynamic quantities (water depth, current velocity, surface-relief height) in grid cells on the time layer $t + \Delta t/2$.

4) The corrector for calculating shallow water dynamics of the Lagrangian and Eulerian stages of the CSPH-TVD method is performed in ≪Corrector SWD≫. The result is an update of the final values of the hydrodynamic quantities (depth and current velocity) in all cells on the time layer $t + \Delta t$.

5) CUDA Kernel ≪Sediment Flux≫ is designed to calculate the sediment flux density and update the topography (function b) in all grid cells on the time layer $t + \Delta t$.

The decomposition of the computational domain is carried out for a selected number of GPUs. Figure 3 shows an example of decomposition for the conditions in Fig. 2 with eight computational subdomains for individual GPUs. The process of performing calculations on the GPU

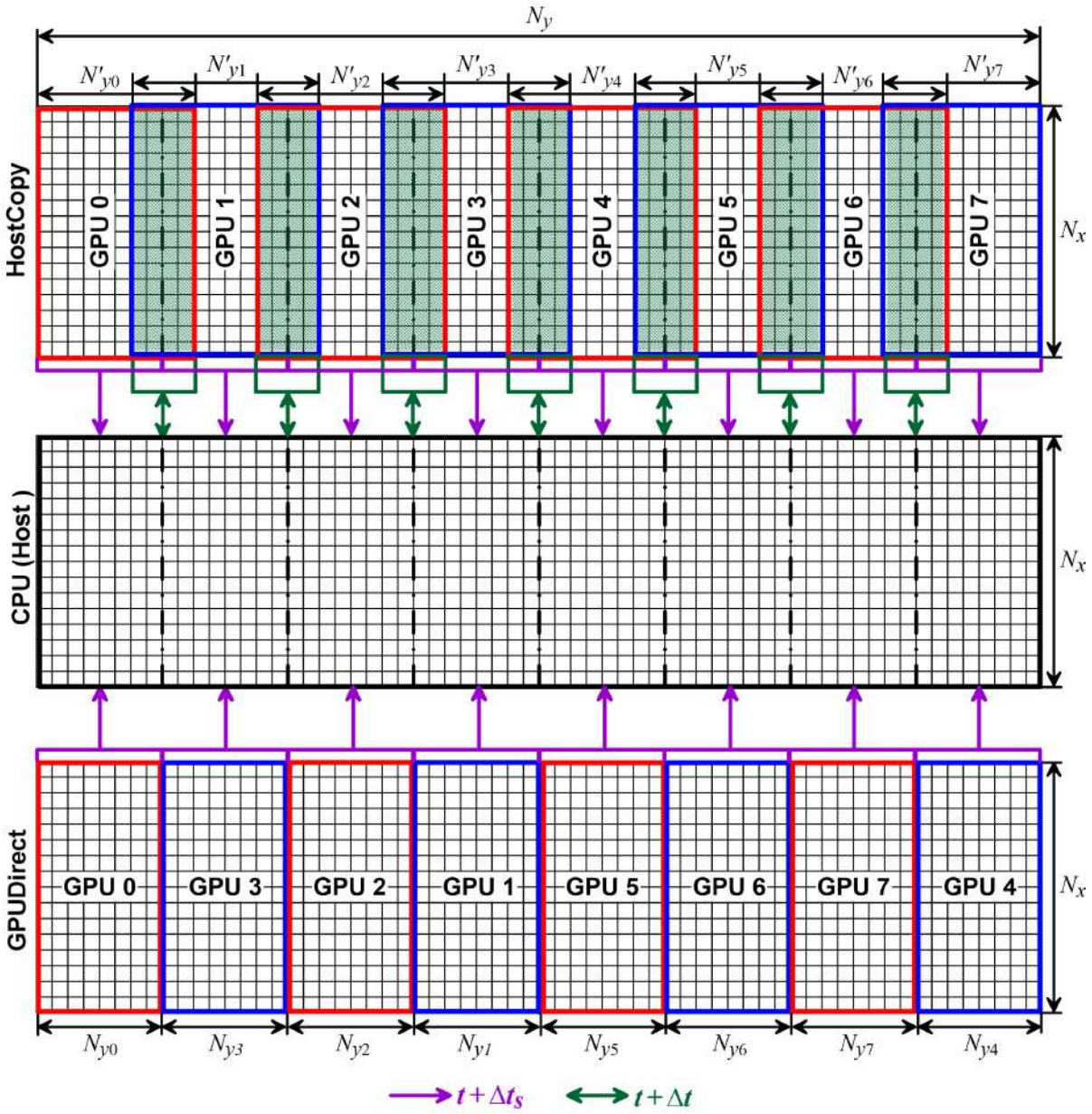


Figure 3. Decomposition of the computational domain for computing system

must be accompanied by data exchange (GPUDirect or HostCopy) to synchronize calculations in the CUDA Kernels. Unloading of newly calculated data back to the Host occurs periodically after a specified number of iterations

$$n_t = \frac{\Delta t_s}{\Delta t} \gg 1\,, \tag{6}$$

where Δt_s is the time interval between recording the states of the simulations, Δt is the time step of integration from the stability condition of the numerical algorithm [15].

The main differences between the GPUDirect and HostCopy approaches are shown in Fig. 3. The size of the computational subdomains $(N_x \times N'_{yk})$ in the code with HostCopy on each GPU is larger than in the code with GPUDirect support $(N_x \times N_{yk})$, and the condition $N'_{yk} > N_{yk}$ is satisfied. This is due to the synchronization of computations on different GPUs due to the use of neighboring cells located on another GPU in the CSPH-TVD numerical algorithm. Such synchronization in the GPUDirect approach is ensured by direct access to the memory of the

Table 1. A set of models with different computational grids

Model	Nx	Ny	N	Model	Nx	Ny	N
G_0	9 216	18 432	169 869 312	-	-	-	-
G_{x1}	8 192	18 432	150 994 944	G_{y1}	9 216	16 384	150 994 944
G_{x2}	7 168	18 432	132 120 576	G_{y2}	9 216	14 336	132 120 576
G_{x3}	6 144	18 432	113 246 208	G_{y3}	9 216	12 288	113 246 208
G_{x4}	5 120	18 432	94 371 840	G_{y4}	9 216	10 240	94 371 840
G_{x5}	4 096	18 432	75 497 472	G_{y5}	9 216	8 192	75 497 472
G_{x6}	3 072	18 432	56 623 104	G_{y6}	9 216	6 144	56 623 104
G_{x7}	2 048	18 432	37 748 736	G_{y7}	9 216	4 096	37 748 736
G_{x8}	1 024	18 432	18 874 368	G_{y8}	9 216	2 048	18 874 368
G_{x9}	512	18 432	9 437 184	G_{y9}	9 216	1 024	9 437 184
G_{x10}	256	18 432	4 718 592	G_{y10}	9 216	512	4 718 592
G_{x11}	128	18 432	2 359 296	G_{y11}	9 216	256	2 359 296
G_{x12}	128	9 216	1 179 648	G_{y12}	4 608	256	1 179 648
G_{x13}	128	4 608	589 824	G_{y13}	2 304	256	589 824
G_{x14}	128	2 304	294 912	G_{y14}	1 152	256	294 912

neighboring GPU (GPU k $\leftrightarrow$ GPU k′). Using the HostCopy approach to synchronize computations requires first copying neighboring cells from the neighboring GPU to the CPU at each time layer (GPU k $\rightarrow$ CPU and GPU k′ $\rightarrow$ CPU) and then back (CPU $\rightarrow$ GPU k′ and CPU $\rightarrow$ GPU k). Therefore, the HostCopy case requires overlapping computational subdomains on neighboring GPUs. Since CSPH-TVD is a five-point scheme, the total number of cells copied to the Host from each boundary between neighboring GPUs is $4 \times N_x$.

The analysis of parallelization efficiency in Section 4 is based on a set of grids with different numbers of cells of up to $1.7 \cdot 10^8$, which allows us to study the effect of the data load size for different numbers of GPUs. The memory capacity of a single GPU limits the size of the computational domain to a grid with $N \leq 2.5 \cdot 10^8$ cells. Table 1 contains characteristics of the computational grids used to analyze the efficiency of the simulations. Each grid G_{xn} or G_{yn} differs in the number of cells N_x along either the x coordinate (the vertical direction in Fig. 1) or the number N_y along the y coordinate (the horizontal direction in Fig. 1). The set (G_{xn}, G_{yn}) forms a hierarchy of computational domains with different numbers of cells along the two coordinates. This allows us to consider different loading and data transfer scenarios between CPUs/GPUs when using different numbers of GPUs. Since the cell sizes are the same in these grids, the corresponding areas are different.

3. Flash Flood Simulations

Figure 4 shows the result of the simulations at $t = 3$ hours, where the blue color highlights the water distribution. In the mountainous areas, the structure of the flows in the form of numerous streams in accordance with the topography is clearly visible. The region near the Krymsk City is highlighted with a red frame. The main part of the rain mass from the mountainous area goes to the flat northern part or flows into the Black Sea.

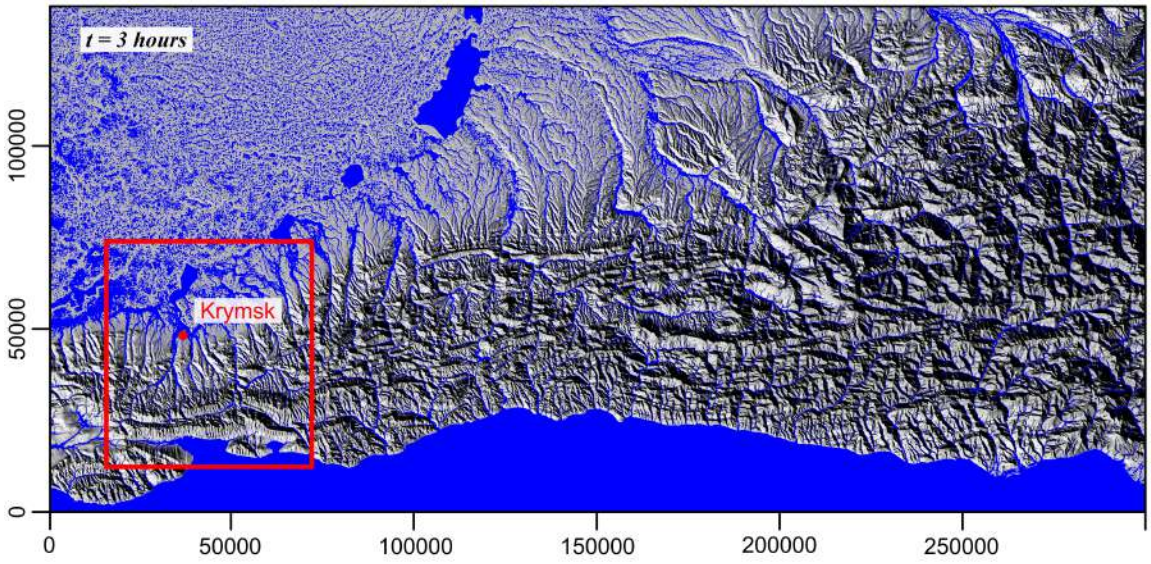
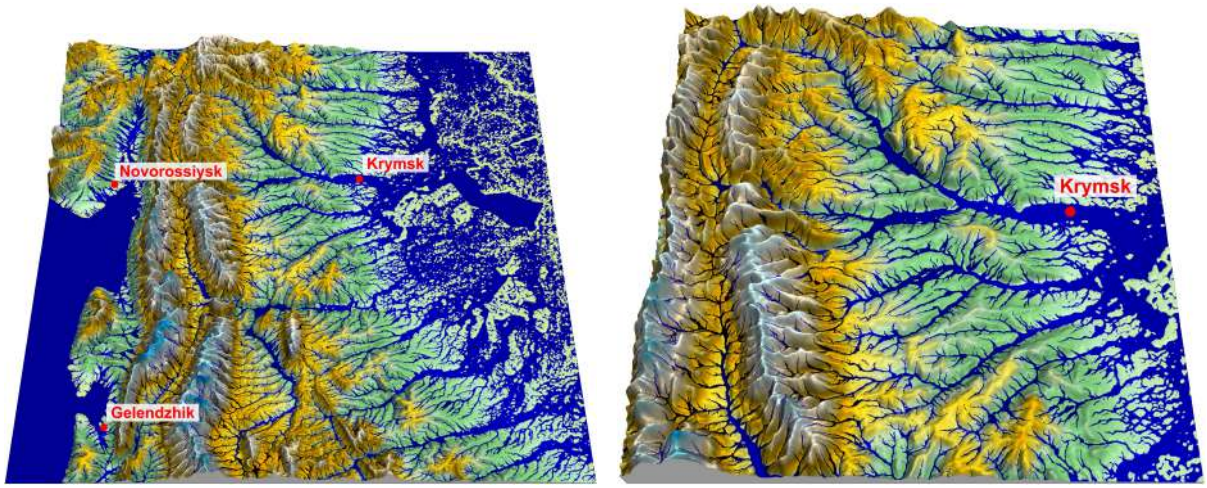


Figure 4. Water distribution 3 hours after the start of a storm rain in one of the models



(a) Zone 55 km × 60 km (b) Zone 30 km × 30 km

Figure 5. Flooding in the area of the Krymsk City

A more detailed picture of flooding on grid G_0 for the Krymsk catchment area is shown in Fig. 5. We use the 3D relief image as a background to highlight water flows. The specific features of the relative positions of channel structures on the relief and the Krymsk City are the cause of possible catastrophic flooding as in July 6–7, 2012 [19]. There is a bottleneck effect, when the power of several flows combines to form the main water flow in the urban area. The capacity of the channel and floodplain of the Adagum River may not ensure the passage of water under flash flood conditions. Negative factors also include infrastructural structures such as road and railway bridges of the Adagum River with the additional influence of uprooted trees and anthropogenic

debris brought by a powerful flow of water. Our model reproduces a catastrophic rise in water level and flooding of the Krymsk City. These results are preliminary and provide only a general qualitative picture, since the main task is to analyze the efficiency of parallelization in Section 3.

4. Efficiency of Parallelization on Multi-GPU

The computational performance of two implementations of our parallel OpenMP-CUDA code (GPUDirect & HostCopy) for simulating self-consistent surface water and sediment dynamics was carried out on two hybrid supercomputers with the following characteristics.
1) NVIDIA DGX-1 consists of 2×CPU (IntelXeon E5-2698v4, DDR4 512Gb) + 8×GPU (NVIDIA TESLA V100, NVLINK, HBM2 256 Gb).
2) Lomonosov-2 supercomputer (Volta 1 and Volta 2 with GPU, Lomonosov Moscow State University) consists of computing nodes that are implemented on the platform 1×CPU (Intel Xeon Gold 6142, DDR4 96Gb) + 2×GPU (NVIDIA TESLA V100, NVLINK, HBM2 64 Gb) [36].

Figure 6 shows the scalability of the parallel OpenMP-CUDA code (GPUDirect & HostCopy) computational performance from the total number of cells N in the numerical models G_0, G_{y1}–G_{y14} for different numbers of GPUs on the NVIDIA DGX-1 supercomputer. The simulations with the GPUDirect code were performed using the fastest NVLink connection (50 GB/s).

We propose to evaluate the efficiency of the implementation of the numerical algorithm by calculating the average processing time of one computational cell:

$$\tau_{nGPU}(N) = \frac{t_{nGPU}(N)}{N}\,, \tag{7}$$

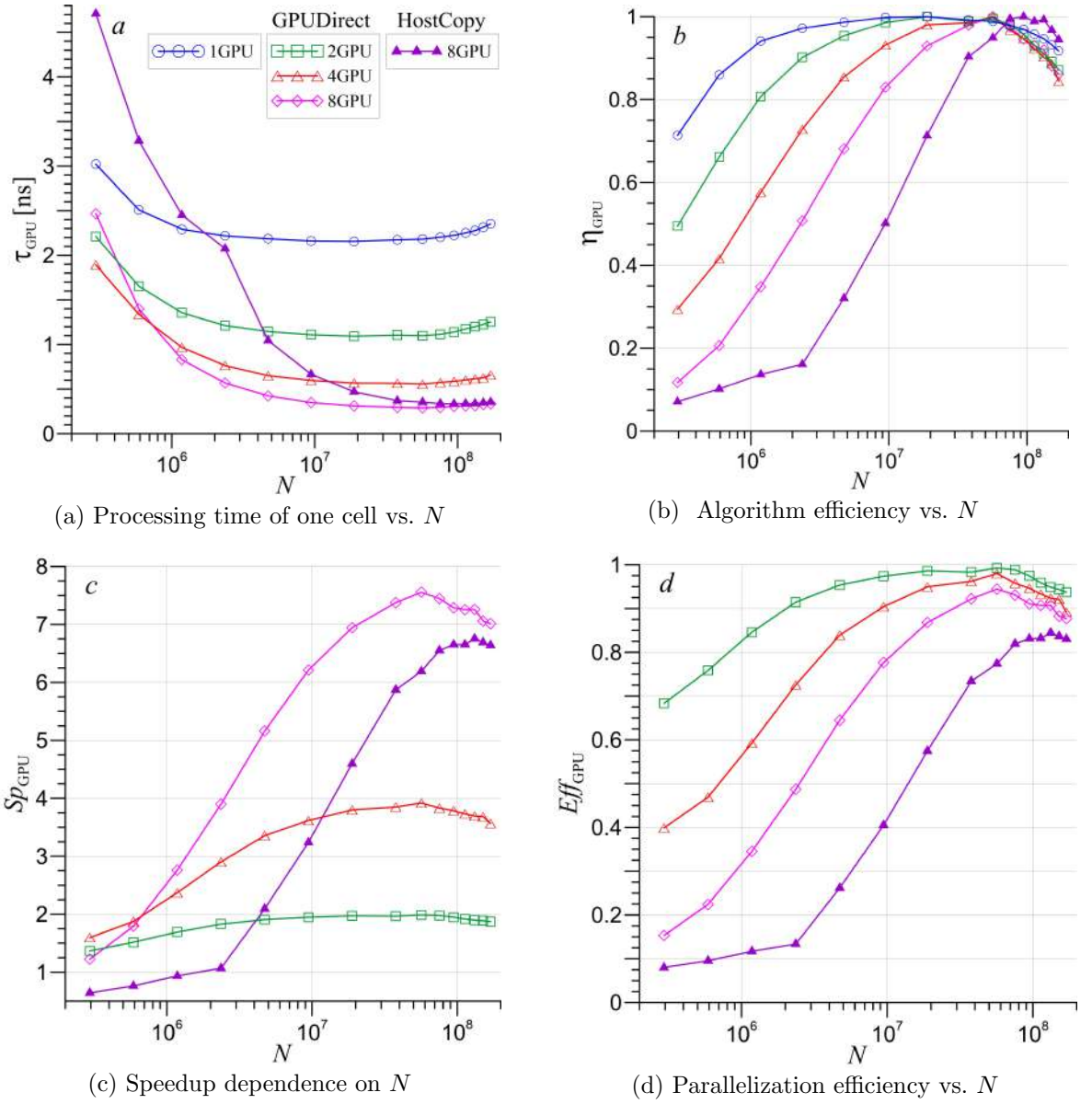
where t_{nGPU} is the average execution time of one computational iteration per time step Δt. The dependences of τ_{nGPU} on the total number of computational cells N for one and several GPUs are shown in Fig. 6a. The value of τ_{nGPU} decreases rather rapidly with increasing N and reaches its minimum value at $N > 10^7$. The model for one GPU (blue line with circles) is characterized by an almost constant level in the region of $10^7 < N < 10^8$ with small deviations within about 3 percent. There is a more significant increase in the time τ_{GPU} up to 10 percent in the region of $N > 10^8$. The G_0 model includes $N \approx 1.7 \cdot 10^8$ cells. We believe that this behavior is due to the additional cost of transferring large amounts of data between the GPU's global memory and the internal memory (register, shared) of its stream multiprocessors (SMs). GPU V100 has 80 such SMs, each containing 32 scalar cores (SCs).

Using 2, 4 and 8 GPUs reduces the time τ_{nGPU} while maintaining the characteristic shape of the dependence $\tau_{nGPU}(N)$ as for 1GPU (see Fig. 6).

Based on the value τ_{GPU}, we can determine the efficiency of using the GPU computing resources, which we will call the efficiency of parallel implementation of a numerical algorithm or, for short, algorithm efficiency

$$\eta_{nGPU}(N) = \frac{\tau_{nGPU}^{(min)}}{\tau_{nGPU}(N)}\,, \tag{8}$$

where $\tau_{nGPU}^{(min)} = \tau_{nGPU}(N_*)$ is the minimum processing time of one cell at maximum GPU load, corresponding to $N = N_*$. The model with 1GPU yields $\tau_{nGPU}^{(min)} \approx 2.16$ ns and $N_* \approx 2 \cdot 10^7$. The value η_{nGPU} allows one to estimate the efficiency of parallel implementation of the numerical algorithm for a fixed number of GPUs depending on N. The scalability effects described above for $\tau_{nGPU}(N)$ are more pronounced in the $\eta_{nGPU}(N)$ dependencies (see Fig. 6b). GPU load



(a) Processing time of one cell vs. N

(b) Algorithm efficiency vs. N

(c) Speedup dependence on N

(d) Parallelization efficiency vs. N

Figure 6. Comparison of the computational performance of parallel OpenMP-CUDA code (GPUDirect & HostCopy) depending on the total number of grid cells N in numerical simulations for 1, 2, 4 and 8 GPUs

efficiency or GPU computational resource utilization efficiency strongly depends on both the number of computational cells N and the number of GPUs. For example, the G_{y14} model for 1 GPU with $N \approx 3 \cdot 10^5$ yields algorithm efficiency $\eta_{nGPU} \approx 0.7$. The efficiency of the G_{y8} model with $N \approx 2 \cdot 10^7$ reaches its maximum value of $\eta_{nGPU} \approx 1$. It decreases to $\eta_{nGPU} \approx 0.92$ in the G_0 model ($N \approx 1.7 \cdot 10^8$). With an increase in the number of GPUs, the algorithm efficiency decreases several times for small values of $N < 10^6$, and its maximum shifts to the region of large N ($N_* \approx 6 \cdot 10^7$). This is due to both a decrease in the number of computational cells per GPU in multi-GPU parallelization and the cost of data transfer between GPUs.

First of all, the values τ_{nGPU} and η_{nGPU} are intended to evaluate the performance of the computational algorithm on a single GPU and allow one to determine the most optimal grid resolution for $\tau_{nGPU}(N) = \tau_{nGPU}^{(min)}$. In multi-GPU parallelization, these characteristics can also be used to find the most optimal number of computational cells in each subdomain on 1 GPU.

The positions of the extrema of the functions $\tau_{nGPU}(N)$ and $\eta_{nGPU}(N)$ determine the optimal grid resolution that ensures maximum computational performance. Figures 6a and 6b show shifts of such extrema to the right toward larger N with an increase in the number of GPUs used. The calculated dependence $\tau_{nGPU}(N)$ for one GPU (see Fig. 6a, blue circles and line) makes it possible to estimate the efficiency of multi-GPU parallelization for any number of GPUs even without performing these simulations.

The impact of data transfer between GPUs and the load on one GPU during multi-GPU parallelization can be conveniently analyzed using standard quantities, such as speedup (Sp_{nGPU}) and parallelization efficiency Eff_{nGPU}:

$$Sp_{nGPU} = \frac{t_{1GPU}}{t_{nGPU}}, \qquad Eff_{nGPU} = \frac{Sp_{GPU}}{nGPU}. \tag{9}$$

The quantity Eff_{nGPU} differs from η_{nGPU} in that it allows one to estimate the efficiency of multi-GPU parallelization of a computational algorithm depending on the number of GPUs used in the simulations for a fixed value of N. Figure 6c, d shows the $Sp_{nGPU}(N)$ and $Eff_{nGPU}(N)$ dependences for 2, 4 and 8 GPUs in the GPUDirect code. We plot the corresponding values for 8 GPUs in the HostCopy code for comparison, which on average turns out to be several times slower and less efficient than the GPUDirect code.

If the behavior of the curves $\eta_{nGPU}(N)$ and $Eff_{nGPU}(N)$ in Fig. 6b, d is compared for 2, 4 and 8 GPUs, these dependencies correlate well with each other. This allows using two approaches to analyzing the performance of calculations (algorithm efficiency and parallelization efficiency) to estimate the data transfer time between GPUs from the following equations:

$$t_{1GPU} = \frac{N \cdot \tau_{1GPU}^{(min)}}{\eta_{1GPU}(N)}, \qquad t_{nGPU} = \frac{N \cdot \tau_{1GPU}}{nGPU \cdot \eta_{1GPU}(N_{GPU})} + N_B \cdot \tau_{nGPU}^{(MT)}, \tag{10}$$

where $\tau_{nGPU}^{(MT)}$ is the specific time (per cell) of memory transfer between $nGPU$, $N_{GPU} = N/nGPU$ is the number of computational cells processed on one GPU, N_B is the number of computational cells along the grid boundary between GPUs. The expressions (10) predict the efficiency of multi-GPU parallelization depending on N, $nGPU$, N_B and parallelization technologies (GPUDirect, HostCopy). It is sufficient to determine the dependence $t_{1GPU}(N)$ in simulations with one GPU for different values of N, then, based on this dependence, calculate $\tau_{1GPU}^{(min)}$ and $\eta_{1GPU}(N)$, and also determine $\tau_{nGPU}^{(MT)}$ for one fixed value of N.

An important characteristic of the computational algorithm is the memory size m_{cell} allocated to the GPU in the simulation per cell, which limits the size of the computational domain in the model. Our parallel OpenMP-CUDA algorithm of the CSPH-TVD method requires $m_{cell} \approx 128$ Bit/cell. This gives the following limit on the size of the computational domain for different numbers of GPU V100: $N \simeq nGPU \cdot 2.6 \cdot 10^8$.

We analyzed the influence of the high-speed NVLink connection type (25 or 50 GB/s) related to the sequence of GPUs used in decomposition of the computational domain. Simulations show that the performance of our GPUDirect code at $N > 10^7$ is practically independent of the choice of the NVLink connection type between GPUs. The deviations are no more than 0.5 percent, which corresponds to the level of thermal noise. Simulation variants with NVLink 50 GB/s at small values of $N \leq 10^6$ turn out to be $\sim$ 5–10 percent more efficient than with NVLink 25 GB/s.

Similar estimates were obtained for different decomposition variants (see models G_{yn} and G_{xn} in Tab. 1). Models G_{xn} also turn out to be $\simeq$ 5–10 percent more efficient than models with grids G_{yn} for values of $N < 10^7$.

We can compare the NVIDIA DGX-1 system and the Lomonosov-2 supercomputer with each other only for simulations on one and two GPUs [36]. Our analysis showed that using Lomonosov-2 is several percent more efficient than NVIDIA DGX-1. This may be due to the presence of a more powerful CPU and a more efficient cooling system on Lomonosov-2.

Our performance analysis of various hydrodynamic codes for CPUs shows that parallel OpenMP versions of these codes on modern CPUs with 40 cores (DGX-1) are 100–1000 times slower, than parallel OpenMP-CUDA programs, depending on the number of GPUs [8, 13].

Conclusion

We studied numerical hydrodynamic models that require large computational grids with the number of cells $N > 10^8$ using the example of simulating a flash flood over a large area of $\sim 5 \cdot 10^4$ km^2 with a good spatial resolution ~ 15 m, which is provided by a digital elevation model based on satellite data. Such numerical models require about 20–30 GB of GPU memory to store them, which limits their use on a single GPU and can lead to a decrease in computational efficiency.

Our numerical rainfall/runoff simulations for the southern mountainous part of the Krasnodar region are aimed at studying the consequences of catastrophic floods that have repeatedly occurred in the past. We have shown the formation of merging channel flows, the power of which increases in the mountainous area. A powerful flow is formed in Krymsk City as a result of the bottleneck effect, which can cause emergency situations for this and other settlements.

We compared the computational efficiency of parallel simulations on CPU-multi-GPU computing systems for two implementations of parallel OpenMP-CUDA using GPUDirect and HostCopy technologies. The maximum speedup value Sp_{nGPU} in the parallel multi-GPU code with GPUDirect for eight GPUs reaches $\sim$ 7.6, which provides parallelization efficiency $Eff_{nGPU} \sim 0.95$. Multi-GPU code with HostCopy requires more operations compared to GPUDirect, which leads to delays in calculations, and as a result, reduces the efficiency of this implementation several times.

We propose to use a special dimensionless coefficient to estimate the efficiency of a numerical algorithm for solving evolutionary problems using the example of computational fluid dynamics for the grid approach. This characteristic is defined through the average processing time of one computational cell τ_{nGPU}. The value τ_{nGPU} has a non-monotonic dependence on the total number of cells N and has a minimum $\tau_{GPU}^{(min)}$ for some $N = N_*$. We define the algorithm efficiency η_{nGPU} to evaluate the efficiency of using the computational resources of a fixed number of GPUs depending on N. The value $\eta_{nGPU} = 1$ means the maximum possible GPU load for a specific parallel implementation of the numerical algorithm. It is important that the calculation of the standard characteristic of computational performance Eff_{nGPU} together with the algorithm efficiency η_{nGPU} makes it possible to predict the scalability of the computational algorithm with an increase in the number of cells and the number of GPUs in numerical simulations.

Note that the value η_{nGPU} can be used to evaluate the efficiency of a numerical algorithm on other computing platforms, including CPU.

Acknowledgements

This work is supported by the Russian Science Foundation (grant no. 23-71-00016, https://rscf.ru/project/23-71-00016/. The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University.

References

1. Aamodt, T.M., Fung, W.W.L., Rogers, T.G.: General-Purpose Graphics Processor Architectures. Springer Cham (2018)

2. Belikov, V., Borisova, N., Vasileva, E., *et al.*: A Numerical Hydrodynamic Model of a Long Segment of the Ural River and Its Application to Assessing the Inundation Risk of Residential Areas by Floods and Breakthrough Waves. Water Resources 51(5), 654–665 (2024). https://doi.org/10.1134/S0097807824701008

3. Bocharov, A., Evstigneev, N., Petrovskiy, V., *et al.*: Implicit method for the solution of supersonic and hypersonic 3D flow problems with Lower-Upper Symmetric-Gauss-Seidel preconditioner on multiple graphics processing units. Journal of Computational Physics 406, 109189 (2020). https://doi.org/10.1016/j.jcp.2019.109189

4. Diaz, M.C., Fernandez-Nieto, E., Ferreiro, A., *et al.*: Two-dimensional sediment transport models in shallow water equations. a second order finite volume approach on unstructured meshes. Computer Methods in Applied Mechanics and Engineering 198(33-36), 2520–2538 (2009). https://doi.org/10.1016/j.cma.2009.03.001

5. Dominguez, J.M., Fourtakas, G., Altomare, C., *et al.*: DualSPHysics: from fluid dynamics to multiphysics problems. Computational Particle Mechanics 9, 867–895 (2022). https://doi.org/10.1007/s40571-021-00404-2

6. Dominguez, J., Crespo, A., Valdez-Balderas, D., *et al.*: New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters. Computer Physics Communications 184(8), 1848–1860 (2013)

7. Dong, B., Huang, B., Tan, C., *et al.*: Multi-GPU parallelization of shallow water modelling on unstructured meshes. Journal of Hydrology 657, 133105 (2025). https://doi.org/10.1016/j.jhydrol.2025.133105

8. Dyakonova, T., Khoperskov, A., Khrapov, S.: Numerical Model of Shallow Water: The Use of NVIDIA CUDA Graphics Processors. Communications in Computer and Information Science 687, 132–145 (2016). https://doi.org/10.1007/978-3-319-55669-7_11

9. Gorobets, A., Bakhvalov, P.: Heterogeneous CPU+GPU parallelization for high-accuracy scale-resolving simulations of compressible turbulent flows on hybrid supercomputers. Computer Physics Communications 271, 108231 (2022). https://doi.org/10.1016/j.cpc.2021.108231

10. Hafiyyan, Q., Harlan, D., Adityawan, M.B., *et al.*: 2D Numerical Model of Sediment Transport Under Dam-break Flow Using Finite Element. International Journal on Advanced Science Engineering and Information Technology 11(6), 2476–2481 (2021). https://doi.org/10.18517/ijaseit.11.6.14484

11. Jayaratne, R., Takayama, Y., Shibayama, T.: Applicability of suspended sediment concentration formulae to large-scale beach morphological changes. In: Lynett, P., McKee Smith, J.e. (eds.) Coastal Engineering Proceedings, vol. 1 (33), pp. 1–15. Coastal Engineering Research Council (2012). https://doi.org/10.9753/icce.v33.sediment.57

12. Khoperskov, A., Khrapov, S., Klikunova, A., *et al.*: Efficiency of Using GPUs for Reconstructing the Hydraulic Resistance in River Systems Based on Combination of High Performance Hydrodynamic Simulation and Machine Learning. Lobachevskii Journal of Mathematics 45(7), 3085–3096 (2024). https://doi.org/10.1134/S199508022460376X

13. Khrapov, S.: Numerical modeling of two-dimensional gas-dynamic flows in multicomponent nonequilibrium media. Mathematical Physics and Computer Simulation 28(1), 60–88 (2025)

14. Khrapov, S., Pisarev, A., Kobelev, I., *et al.*: The numerical simulation of shallow water: Estimation of the roughness coefficient on the flood stage. Advances in Mechanical Engineering 2013, 787016 (2013). https://doi.org/10.1155/2013/787016

15. Khrapov, S.: Numerical modeling of hydrodynamic accidents: Erosion of dams and flooding of territories. Vestnik of the St. Petersburg University: Mathematics 56(2), 261–272 (2023). https://doi.org/10.1134/s1063454123020085

16. Khrapov, S., Khoperskov, A.: Application of graphics processing units for self-consistent modelling of shallow water dynamics and sediment transport. Lobachevskii Journal of Mathematics 41(8), 1475–1484 (2020). https://doi.org/10.1134/S1995080220080089

17. Khrapov, S., Khoperskov, A.: Study of the Effectiveness of Parallel Algorithms for Modeling the Dynamics of Collisionless Galactic Systems on GPUs. Supercomputing Frontiers and Innovations 11(3), 27–44 (2024). https://doi.org/10.14529/jsfi240302

18. Klikunova, A., Polyakov, M., Khrapov, S., *et al.*: Problem of building high-quality predictive model of river hydrology: the combined use of hydrodynamic simulations and intelligent computing. Communications in Computer and Information Science 1909, 191–205 (2023). https://doi.org/10.1007/978-3-031-44615-3_13

19. Kotlyakov, V.M., Desinov, L.V., Dolgov, S.V., *et al.*: Flooding of July 6-7, 2012, in the town of Krymsk. Regional Research of Russia 3, 32–39 (2013). https://doi.org/10.1134/S2079970513010061

20. Li, W., Hu, P., Pahtz, T., *et al.*: Limitations of empirical sediment transport formulas for shallow water and their consequences for swash zone modelling. Journal of Hydraulic Research 55(1), 114–120 (2017). https://doi.org/10.1080/00221686.2016.1212942

21. Liu, T., Trim, S.J., Ko, S.B., *et al.*: The multi-GPU Wetland DEM Ponding Model. Computers & Geosciences 199, 105912 (2025). https://doi.org/10.1016/j.cageo.2025.105912

22. de Luna, T.M., Diaz, M.J.C., Madronal, C.P.: On a sediment transport model in shallow water equations with gravity effects. In: Kreiss, G., Lotstedt, P., Malqvist, A., Neytcheva, M. (eds.) Numerical Mathematics and Advanced Applications, pp. 655–661. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11795-4_70

23. Macca, E., Avgerinos, S., Castro-Diaz, M.J., *et al.*: A semi-implicit finite volume method for the Exner model of sediment transport. Journal of Computational Physics 499, 112714 (2024). https://doi.org/10.1016/j.jcp.2023.112714

24. McKevitt, J., Vorobyov, E.I., Kulikov, I.: Accelerating Fortran codes: A method for integrating Coarray Fortran with CUDA Fortran and OpenMP. Journal of Parallel and Distributed Computing 195, 104977 (2025). https://doi.org/10.1016/j.jpdc.2024.104977

25. Mignot, E., Paquier, A., Haider, S.: Modeling floods in a dense urban area using 2D shallow water equations. Journal of Hydrology 327(1-2), 186–199 (2006). https://doi.org/10.1016/j.jhydrol.2005.11.026

26. Narasiman, V., Shebanow, M., Lee, C.J., *et al.*: Improving GPU performance via large warps and two-level warp scheduling. In: 2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Porto Alegre, Brazil, pp. 308–317. IEEE (2011)

27. Ndengna, A.R.N., Njifenjou, A.: A well-balanced PCCU-AENO scheme for a sediment transport model. Ocean Systems Engineering 12(3), 359–384 (2022). https://doi.org/10.12989/ose.2022.12.3.359

28. Shao, X., Wang, X.: Introduction to River Dynamics. Tsinghua University Press Co., Beijing, China (2005)

29. Simonov, A.S., Semenov, A.S., Shcherbak, A.N., *et al.*: The High Performance Interconnect Architecture for Supercomputers. Supercomputing Frontiers and Innovations 10(2), 127–136 (2023). https://doi.org/10.14529/jsfi230208

30. Siviglia, A., Vanzo, D., Toro, E.: A splitting scheme for the coupled Saint-Venant-Exner model. Advances in Water Resources 159, 104062 (2022). https://doi.org/10.1016/j.advwatres.2021.104062

31. Sukhinov, A.I., Protsenko, E.A., Protsenko, S.V.: WAVEWATCH III Hybrid Parallelization for Azov Sea Wave Modeling. Supercomputing Frontiers and Innovations 11(1), 81–96 (2024). https://doi.org/10.14529/jsfi240104

32. Taccone, F., Antoine, G., Delestre, O., *et al.*: A new criterion for the evaluation of the velocity field for rainfall-runoff modelling using a shallow-water model. Advances in Water Resources 140, 103581 (2020). https://doi.org/10.1016/j.advwatres.2020.103581

33. Valles, P., Fernandez-Pato, J., Morales-Hernandez, M., *et al.*: A 2D shallow water flow model with 1D internal boundary condition for subgrid-scale topography. Advances in Water Resources 189, 104716 (2024). https://doi.org/10.1016/j.advwatres.2024.104716

34. Vasileva, E.S., Aleksyuk, A.I., Belyakova, P.A., *et al.*: Numerical modeling of the behavior of a destructive rain flood on a mountain river. Water Resources 46(1), 45–55 (2019). https://doi.org/10.1134/S0097807819070169

35. Vatyukova, O., Klikunova, A., Vasilchenko, A., *et al.*: The problem of effective evacuation of the population from floodplains under threat of flooding: algorithmic and software support with shortage of resources. Computation 11(8), 150 (2023). https://doi.org/10.3390/computation11080150

36. Voevodin, V., Antonov, A., Nikitenko, D., *et al.*: Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community. Supercomputing Frontiers and Innovations 6(2), 4–11 (2019). https://doi.org/10.14529/jsfi190201

37. Zhang, X., Guo, X., Weng, Y., *et al.*: Hybrid MPI and CUDA paralleled finite volume unstructured CFD simulations on a multi-GPU system. Future Generation Computer Systems 139, 1–16 (2023). https://doi.org/10.1016/j.future.2022.09.005

38. Zolfaghari, H., Becsek, B., Nestola, M.G., *et al.*: High-order accurate simulation of incompressible turbulent flows on many parallel GPUs of a hybrid-node supercomputer. Computer Physics Communications 244, 132–142 (2019). https://doi.org/10.1016/j.cpc.2019.06.012