Application of Graphics Processing Units for self-consistent modelling of shallow water dynamics and sediment transport

S. S. Khrapov¹, * and A. V. Khoperskov¹, **

(Submitted by A. A. Editor-name)

¹Volgograd State University, Volgograd, Russian Federation

Abstract—In this paper, we describe a numerical algorithm for the self-consistent simulations of surface water and sediment dynamics. The method is based on the original Lagrangian-Eulerian CSPH-TVD approach for solving the Saint-Venant and Exner equations, taking into account the physical factors essential for the understanding of the shallow water and surface soil layer motions, including complex terrain structure and its evolution due to sediment transport. Additional Exner equation for sediment transport has been used for the numerical CSPH-TVD scheme stability criteria definition. By using OpenMP-CUDA and GPUDirect technologies for hybrid computing systems (supercomputers) with several graphic coprocessors (GPUs) interacting with each other via the PCI-E / NVLINK interface we also develop a parallel numerical algorithm for the CSPH-TVD method. The developed parallel version of the algorithm demonstrates high efficiency for various configurations of Nvidia Tesla CPU + GPU computing systems. In particular, maximal speed up is 1800 for a system with four C2070 GPUs compare to the serial version for the CPU. The calculation time on the GPU V100 (Volta architecture) is reduced by 95 times compared to the GPU C2070 (Fermi architecture).

2010 Mathematical Subject Classification: 65M22, 76-04

Keywords and phrases: Parallel Computing, GPUs, Numerical Methods, Shallow Water Model

1. INTRODUCTION

Modelling of unsteady flows of the surface fluid requires the calculation of movable flow boundaries [1] where the presence of sediment transport complicates this problem because the boundaries between the fluid and sediment may not coincide. Another complicating factor, in addition to the heterogeneity of the bottom topography \( b(x, y) \), is the heterogeneity of soil characteristics (e.g., porosity, particle sizes). A detailed analysis of various models of sediment transport can be found in [2]. Usually, the mathematical model of a shallow water dynamics is based on hyperbolic equations for which Godunov-type solvers are often used [3]. The first family of solvers uses the Harten-Lax-Leer (HLL) scheme and its modifications [4, 5]. Another approach is based on Roe’s approximate solver [6]. Nowadays, another state-of-the-art approach for modelling sediment transport by water flow is the discrete least-squares meshless method [7].

Recently advantages of the architecture of modern GPUs have been successfully used to solve a wide range of mathematical problems [8–14] thus making usage of the parallel computing on GPUs even more topical for a broad range of academical and applied problems. In this work, we discuss a combined SPH-TVD method for the numerical integration of the Saint-Venant and Exner equations which was proposed previously in our works [15, 16]. In this paper, we extend our approach for the modelling of self-consistent dynamics of surface water and sediment. In particular, the main aim of our work is to develop an effective parallel code for various computing systems with GPUs.

* E-mail: khrapov@volsu.ru
** E-mail: khoperskov@volsu.ru
2. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

Из-за широкого распространения комплексного описания динамики поверхностного потока воды и транспорта осадочного материала требуется использование мультислойных моделей в приближении глубоких вод [17–19]. В этом подходе, уравнения Сент-Вена а должны быть записаны в форме уравнения Энер [7, 12, 20], которое позволяет моделировать динамику процессов почвенного выветривания и перемещения осадочного материала из-за движения воды. Деформации поверхности дна в слабых течениях сопровождаются образованием структур в виде волн, берегов, песков, и т. д., которые являются важными структурными элементами дна, оказывая значительное влияние на гидравлическое сопротивление [21]. Например, сильные течения приводят к значительному выветриванию дна, в частности, может образоваться канал при разрушении дамбы, а затем быстро уплотняется и расширяется под влиянием увеличения потока воды.

Уравнение Энер для одноосного модельного как:

$$\frac{(1-\psi)}{\partial t} \nabla J_b = q^+ - q^-,$$

где $J_b$ - вектор транспортировки осадочной массы, $q^+$ - источник материала из-за осадочного накопления, $q^-$ - функция утечки из-за выветривания дна, $\psi$ - пористость дна, $\nabla$ - компонента оператора нла в плоскости Земли, $x, y$ - оси.

Полное моделирование процессов выветривания, транспорта почв и накопления является математической и вычислительной задачей [22, 23] и его решение возможно только при использовании феноменологического подхода, дополняемого тщательным предварительным анализом результатов симуляции и контрольных экспериментов [24, 25]. Уравнение (1) является нелинейным за счет многообразия физических процессов в нижнем слое. Например, транспортировка материала $J_b$ может зависеть как от сил трения воды о дно и уклона дна [26]:

$$J_b = J_0^b - C_J \text{sign}(J_0^b) |J_0^b| \nabla b,$$

где $J_0^b$ - транспортировка осадочной массы для нулевого уклона, $C_J = 1.5 \div 2.3$ - это произвольная эмпирическая постоянная, которая может достигать значений $C_J = 5$ [27]. Та, уравнение (1) содержит нелинейный диффузионный член в форме $\nabla |J_0^b| \nabla b$ учитывая отношение (2). Чтобы определить поток $J_b$, мы используем формулу Грацис [26]:

$$J_0^b = A_J \nu |\tilde{v}|^m,$$

где $\nu$ - средняя скорость потока воды над дном, совпадающая со скоростью жидкости в модели глубоких вод, $A_J, m$ - постоянные коэффициенты, например, $A_G = 0.001, m = 2$ для мелкого песка.

Значение $A_J$ в соотношении (3) зависит от характеристик движущихся частиц и может быть представлено в виде: [28]:

$$A_J = \frac{0.05 n_1^3}{(s-1)\sqrt{gHd_{50}}},$$

где $s = \rho_s/\rho$ - относительная массовая плотность, $d_{50}$ - средний размер частиц осадочных частиц. Выветривание дна происходит тогда, когда скорость превышает критическое значение, определенное по уравнению Шамова [29]:

$$v_k = C_{Sh} d_{50}^{1/3} H^{1/6}, \quad C_{Sh} \approx 4 \div 6.$$

Векторная форма системы уравнений модели глубоких вод и уравнение Энер может быть записано в виде:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = \Phi,$$
where

$$U = \begin{pmatrix} H \\ H_u \\ H_v \\ b \end{pmatrix}, \quad F = \begin{pmatrix} H_u \\ H_u^2 \\ H_{uv} \\ J_{bx} \end{pmatrix}, \quad G = \begin{pmatrix} H_v \\ H_{uv} \\ H_v^2 \\ J_{by} \end{pmatrix}, \quad \Phi = \begin{pmatrix} \sigma \\ gH \sqrt{\frac{1}{1 - \psi}} \end{pmatrix}, \quad \Psi = \begin{pmatrix} \frac{q_b - q_n}{1 - \psi} \end{pmatrix},$$

where $H$ is water depth, $u(x, y, t)$, $v(x, y, t)$ is the water velocity components, $\sigma$ is the function of water sources/drains, determined by the operating mode of hydraulic structures, rainfall/snowfall, the rainfall run-off, the infiltration/evaporation; $f$ is the external forces connected with the Earth rotation, the wind, the bottom friction, the turbulence.

We use the Combined Smoothed Particle Hydrodynamic method (CSPH-TVD), which contains the Lagrangian stage (modified Smoothed Particle Hydrodynamic, or SPH) and the Euler stage (modified Total Variation Diminishing, or TVD) [15]. The consistent use of these two stages during the integration provides several important positive properties that can be achieved with less computational resources [16, 30]. The equation (1) imposes additional constraints on the stability of the entire numerical CSPH-TVD algorithm compared to the single-layer shallow water model considered in [16, 30]. Nonlinear terms of the diffusion type in the equation (1) due to the relation (2) affect the stability condition, which takes the form:

$$\tau_n = K \min_i \left( \frac{h}{2v_p}, \frac{h^2}{v_s}, \frac{h^2}{2D} \right),$$

where $0 < K < 1$ is the Courant number, $h$ is cell size, $i$ is the vector spatial index, $v_p = \max_i |v_i|$, $v_s = \max_i (|v_i| + \sqrt{gH_i})$, $D = \max_i |J_i^0|/(1 - \psi)$.

3. PARALLEL IMPLEMENTATION OF NUMERICAL MODEL

The computational module of the software package for simulation of dynamics of the water surface and sediment based on the CSPH-TVD method consists in four main blocks: Control Block — Main; Computing Block — Solver; State Recording Block — Save; Screen Output Block — Screen (Fig. 1). The Control Block Main provides loading of input data (parameters of the numerical model, the digital elevation model $b(x, y)$, the distributions of bottom friction coefficient $\eta_M(x, y)$ and absorption coefficient $\beta(x, y)$) and creating of the initial state. This unit is stored on the CPU, and the computation is organized as follows: the cyclical launch of the computing cores from the Solver block located on the GPU; the cyclical recording of calculation states by calling routines from the block Save (CPU); the cyclical display of the data necessary for monitoring the correct operation of the application by accessing the block Screen (CPU).

The Solver block contains 5 main computing kernels organized in the form of subprograms and functions where the CSPH-TVD numerical algorithm for computation the shallow water dynamics on irregular terrain and the sediment transport is written in CUDA C. Since the stages (kernels) of the task should be performed sequentially (i.e. the data obtained at each individual stage are used in the subsequent one), we chose a parallelization model for the algorithm implementation. Implementation of this approach on GPUs allows us to speed-up the program by hundreds and thousands of times, depending on the configuration of the computing cluster in comparison to the CPU. The task parallelization model (CUDA streams) can be used to separate computation processes and copy data between the CPU and GPU. Since the implementation of the algorithm focuses on the minimization of the traffic between the CPU and the GPU (data exchange between the CPU and GPU occurs only after 100–1000 time iterations), the parallelization can increase productivity by $3 – 10\%$ at best. The speed of the data transfer process between the CPU and the GPU is an order of magnitude lower than the computations time on the GPU. The number of simultaneously running computing kernels for each stage of the SPH-TVD method in the case of the parallel OpenMP version (for the CPU) is equal to the number of processor kernels $k$ (for
The number of simultaneously running kernels for each stage of the SPH-TVD method for a CUDA program on a GPU does not depend on the number of scalar GPU kernels (Tesla C1060 — 240 kernels, Tesla C2070 — 448 kernels, Tesla K20 — 2496 kernels, Tesla K40 — 2880 kernels, Tesla K80 — 4992 kernels, Tesla P100 — 3584 kernels, Tesla V100 — 5120 kernels) and it can be set based on the principle that “the more – the better” the loading of the GPU. In our case, this number is equal to the number of cells of the computational grid, i.e. each thread of the GPU processes a single element. The GPU load of our task is about 99.9%, and the speed-up of the program (compared to the serial version of the CPU) reaches $\sim \alpha k$ times ($k$ is the number of scalar GPU cores, $\alpha$ is the ratio of CPU frequency to GPU frequency). The efficient usage of the shared and global memory, the data alignment, and the vectorized data types for the Fermi, Kepler, Pascal, and Volta architectures allow speeding up the program in $\sim (2.5 \div 3) \alpha k$ times, respectively.

Our program has been developed in CUDA C language using the CUDA Runtime libraries. To improve the performance of the numerical algorithm we also adopted a hierarchical grid system (HGS). The HGS-based approach implies that the cells of the computational grid of size $N \times M$ are grouped into blocks of size $L \times K$ which in turn are joint into regions (Fig. 2). Similar to the computation module for the CPU, the application of the HGS approach to the CUDA program leads to a significant speed-up of the computations, especially in case of large sections area without water. Execution of computational kernels in the CUDA version of the code is organized as a grid of thread blocks. Figure 2 demonstrates that stream blocks in CUDA are analogous to blocks in HGS. Thereby to implement the HGS algorithm in the GPU version of the program, it is enough to enter the array $\text{Index\_block}[N\_block]$ (where $N\_block$ is the number of CUDA blocks in the...
CUDA grid), which determines the number of cells without water in the CUDA block at a given time, and to supplement the algorithm by new CUDA-kernel, in which the values of the elements of this array are derived. The body of the corresponding computational core is launched and fully loads the streaming multiprocessor (SM) of the GPU if the condition $\text{Index\_block}[\text{k\_block}] > 0$ is satisfied (where $k\_block = blockIdx.x + blockIdx.y \times \text{gridDim.x}$ is the current number of the CUDA block, determined through the built-in CUDA Runtime variables), otherwise SM is freed to start another CUDA kernel from the CUDA Runtime queue. Code snippets for computing kernels using HGS are shown below.

CUDA-kernels code for Lagrangian stage

```c
__global__ void kernel_LS(double *H, double *Zb, double2 *HV,
int *Index_block,...){
if(Index_block[blockIdx.x+blockIdx.y*gridDim.x] > 0){
[----//---- COMPUTING KERNEL BODY] } }
```

CUDA-kernels code for the Euler stage

```c
__global__ void kernel_ES(double *H, double *Zb, double2 *HV,
int *Index_block,...){
it ib=blockIdx.x+blockIdx.y*gridDim.x;
it xbl=blockIdx.x-1; if(xbl<0) { xbl=0;}
it ibl=xbl+blockIdx.y*gridDim.x;
it xbr=blockIdx.x+1;
it(xbr>gridDim.x){xbr=gridDim.x;}
it ibr=xbr+blockIdx.y*gridDim.x;
```
int ybl=blockIdx.y-1;  if(ybl<0){ybl=0;}  
int ibb=blockIdx.x+ybl*gridDim.x;  
int ybr=blockIdx.y+1;  
if(ybr>gridDim.y){ybr=gridDim.y;}  
int ibt=blockIdx.x+ybr*gridDim.x;  
if(Index_block[ib] > 0 || Index_block[ibl] > 0  
|| Index_block[ibr] > 0 || Index_block[ibb] > 0  
|| Index_block[ibt] > 0){  
    [----//---- COMPUTING KERNEL BODY]  
} }

CUDA-kernel code Index_block

__global__ void kernel_Index_block(double *H, int *Index_block,...){
if(H[ind]>Eps || Index_Q[ind]>0) Sij[ind_thb]=1;
else Sij[ind_thb]=0; __syncthreads();
int k = ithbx*ithby/2;
while(k != 0){
if(ind_thb < k) Sij[ind_thb] += Sij[ind_thb+k];
__syncthreads(); k /= 2; }
if(ind_thb == 0) Index_block[ib] = Sij[0];}

It should be noted that at the Lagrangian stage it is enough to check the presence of water only in  
the current CUDA block. Meanwhile, the Euler stage requires the presence of water in neighbouring  
CUDA blocks to be checked well because fluxes of mass and momentum are calculated at the cell  
boundaries and water can flow from one block to another. Solution of any applied problems with  
a large number of calculations is indeed requires the hybrid computing systems with GPUs. For  
instance, MPI technology provides parallelization of applications on computing systems with shared  
memory. From other hand, for systems with global memory based on a hybrid configuration of CPU  
+k×GPU (along with CUDA technology) it is convenient to use OpenMP because it is necessary  
to create a separate threads on the CPUs to load each GPU. We used NVIDIA GPUDirect (Peer-  
to-Peer Communication) technology to exchange data between GPUs without copying them into  
the CPU when implementing a program adapted for computing on hybrid supercomputers (CPU  
+k×GPU) using a two-level parallelization scheme OpenMP–CUDA. This approach leads to an  
almost linear speed-up with an increase in the number of GPUs used. The following is an example  
of a code fragment that is based on the parallel computing technologies mentioned above.

Sample Code OpenMP–CUDA

#pragma omp parallel for schedule(static,1) shared(deviceCount, 
threads,dt)  
private(deviceId,devId_b,devId_t)  
for(deviceId=0; deviceId<deviceCount; deviceId++){
  cudaMemcpy(deviceId);  
dim3 blocks(Nx/ithbx,Ny_dev[deviceId]/ithby);  
if(deviceId==0){devId_t=1; devId_b=deviceCount-1;  
}else if(deviceId==deviceCount-1){devId_t=0; devId_b=deviceId-1;  
}else{devId_t=deviceId+1; devId_b=deviceId-1;}
  cudaMemcpyPeerAccess(devId_t, 0);  
cudaDeviceEnablePeerAccess(devId_b, 0);  
kernel_forces_predictor<<<blocks, threads>>>( dev_outSrc[deviceId],  
dev_inSrc[deviceId],dev_inSrc[devId_b],dev_inSrc[devId_t],
  dt,deviceId,deviceCount);  }

We increase the total number of CUDA kernels of the calculation module to 8 (see Fig. 1) by  
adding a new CUDA kernel to implement the HGS approach which includes the calculations of the  
time step and the updating the system at the Eulerian stage in separate CUDA kernels. This extra  
module allows us to use more efficiently the resources of the graphics processor when modeling the  
joint dynamics of shallow water and sediment transport. Below we list the CUDA kernels of the  
calculation module:
K1 determines the presence of water in the CUDA block;
K2 calculates the forces at the Lagrangian stage of the CSPH-TVD method at time $t_n$ (predictor);
K3 calculates the time step $dt_n$;
K4 determines the positions of particles and their integral characteristics on the time layer $t_{n+1/2}$ (predictor);
K5 calculates the forces at the Lagrangian stage of the CSPH-TVD method at the time $t_{n+1/2}$ (corrector);
K6 determines the positions of the particles and their integral characteristics for the next time layer $t_{n+1}$ (corrector);
K7 calculates the fluxes of physical quantities through the cell boundaries at the Eulerian stage in $t_{n+1/2}$;
K8 updates the state of the system at the Eulerian stage at time instant $t_{n+1}$, calculating the flow parameters in the cells of the computational grid.

The parallelization efficiency of the CUDA-algorithm of the CSPH-TVD method was studied by simulating the dynamics of seasonal and catastrophic flooding of the Volga-Akhtuba floodplain (VAF). The calculations were performed on grids $1024 \times 1024$, $2048 \times 2048$ and $4096 \times 4096$ by using Nvidia Tesla GPUs: C2070, K20, K40, K80 (2 GPU), P100 and V100.

Fig. 3 shows the distribution of the execution time of the numerical algorithm between CUDA kernels. The most computationally expensive part is CUDA-kernel 7 (58.5%) where the fluxes of physical quantities at the boundaries of the Euler cells are calculated. This is due to the fact that for each of the x- and y-directions, a piecewise linear reconstruction of the grid functions is carried out and the Riemann problem is solved using the HLL method. In addition, when calculating the slopes of piecewise-linear distributions of grid functions, TVD-limiters are used, which, on the one hand, suppress oscillations in the numerical solution, and, on the other hand, increase the computation time due to the use of conditional operators in the CUDA software implementation. The figure shows that the fraction of the time required for the Lagrangian stage (CUDA kernels K2 – K6) is 30%, the Euler stage (CUDA kernels K7 – K8) accounts for 63.5%, and the CUDA core K1 that implements the HGS approach, uses only 6.5%, while it can increase the calculation speed by a factor of ten and more. Thereby in order to increase the computational performance of the algorithm, it is necessary firstly to optimize the kernel K7. Fig. 4 demonstrates the results of simulations of strong spring flooding of the Volga-Akhtuba interfluve, showing changes in the terrain structure due to sediment transport, which are consistent with the analysis of long-term
Figure 4. Example of modeling the joint dynamics of shallow water and sediment transport in the Volga-Akhtuba floodplain. Top — Shaded Relief Map. The inset is the change in bottom topography for the 28th day from the beginning of the spring flood. There are the bottom changes within $\Delta b = -1.5 \div 1.5$ m). Below is the distribution of water in the floodplain at the same time ($|H|=m$).

observation series for VAF [31]. Fig. 5 shows a comparison of the computational efficiency of our parallel algorithm of the CSPH-TVD method for various GPUs. We see that the computation time decreases exponentially ($\sim \exp(-t/\tau)$) with the transition to more modern GPUs. For example, the calculation time on the GPU V100 (Volta architecture) is reduced by 95 times compared to
the GPU C2070 (Fermi architecture). We have estimated the characteristic time scale $\tau \approx 1.82$ for our CUDA-algorithm on the grid $1024 \times 1024$. The parameter $\tau$ decreases with increasing grid resolution (the number of cells) because of the more efficient loading of new generations of GPUs increases.

We have tested two parallel CUDA versions of the calculation module. The first version can be used on single GPU, in particular on any NVIDIA GPUs with the level of computing capabilities of at least 1.3 (for example, Tesla C1060). The second version is designed for calculations on NVIDIA Tesla C2070, K20, K40, K80, P100 and V100 GPUs with the level of computing capabilities of at least 2.0. Since this version is based on the OpenMP, CUDA and GPUDirect technologies, in principle, it can utilize all the GPUs and CPU cores of the supercomputer. A particular attention we paid to the effective work with global and shared GPU memory, as well as to the optimization of the data exchange process between the CPUs and GPUs for both versions of our code. For example, the maximum speed-up is 1800 on a supercomputer with four C2070 GPUs compare to the serial version for the CPU and 200 compared to the parallel OpenMP version for the CPU.

4. CONCLUSION

In conclusion, we summarize the main results of the work. We have described a new stable numerical algorithm for simulation of self-consistent dynamics of the surface water and sediment process. The method is based on the CSPH-TVD approach by taking into account the process of fluid and grains transport. The implemented parallel CUDA-algorithm CSPH-TVD has also been parallelized by means of direct GPU technology for high-performance supercomputers with GPUs, where the interconnection between nodes is based on PCI-E / NVLINK standards. We demonstrate the high computational efficiency of our algorithm for several different GPU Nvidia Tesla hardware (C2070, K20 / 40/80, P100, V100).

The code described above has been developed for the fast high-resolution simulation of the dynamics of flooding of vast territories. Apart from the calculation module, based on the parallel CSPA-algorithm CSPH-TVD method, our software also includes several routines for the initial conditions preprocessing and visualization of the simulations results. By using this software package, we made a set of new dynamical models of the hydrological regime of the Volga-Akhtuba floodplain were we also take into account sediment transport. We demonstrate the deepening of the bottom of the Volga channel, which, in turn, negatively affects the efficiency of the floodplain territories flooding during the spring floods.
Acknowledgments. The authors were supported by the Ministry of Science and Higher Education of the Russian Federation (government task No. 0633-2020-0003). Numerical simulations were carried by using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University supported by the project RFMEFI62117X0011.

REFERENCES

1. O. V. Bulatov, T. G. Elizarova, “Regularized shallow water equations for numerical simulation of flows with a moving shoreline,” Computational Mathematics and Mathematical Physics 56, 661–679 (2016).
2. G. Parker, Transport of Gravel and Sediment Mixtures. In Sedimentation Engineering; Garcia, M., Ed.; American Society of Civil Engineers: Reston, VA, USA, 165–251 (2008).
3. A. G. Kulikovskii, N. V. Pogorelov and A. Yu. Semenov, Mathematical Aspects of Numerical Solution of Hyperbolic Systems (Chapman and Hall/CRC, London, 2019).
4. A. Harten and J. M. Hyman, “Self adjusting grid methods for one-dimensional hyperbolic conservation laws,” Journal of Computational Physics 50, 235–269 (1983).
5. E. F. Toro, “The HLLC Riemann solver,” Shock Waves 29, 1065–1082 (2019).
6. D. Ambrosi, “Approximation of shallow water equations by Roe’s Riemann solver,” International Journal for Numerical Methods in Fluids 20, 157–168 (1995).
7. F. E. Erami, A. R. Firoozjaee, “Numerical solution of bed load transport equations using discrete least squares meshless (DLSM) method,” Applied Mathematical Modelling 77, 1095–1109 (2020).
8. J.-T. Becerra-Sagredo, C. Málaga and F. Mandujano, “A GPU-based multi-level algorithm for boundary value problems,” Journal of Computational and Applied Mathematics 368, 112481 (2020).
9. R. Chisholm, S. Maddock and P. Richmond, “Improved GPU near neighbours performance for multi-agent simulations,” Journal of Parallel and Distributed Computing 137, 53–64 (2020).
10. K. Yu. Erofeev, E. M. Khramchenkov and E. V. Biryal’tsev, “High-performance Processing of Covariance Matrices Using GPU Computations,” Lobachevskii Journal of Mathematics 40, 547–554 (2019).
11. D. Shaikhislamov, A. Sozykin and V. Voevodin, “Survey on Software Tools that Implement Deep Learning Algorithms on Intel/x86 and IBM/Power8/Power9 Platforms,” Supercomputing Frontiers and Innovations 6(4), 57–83 (2019).
12. S. Dazzi, V. Voevodin and V. Voevodin, “Survey on Software Tools that Implement Deep Learning Algorithms on Intel/x86 and IBM/Power8/Power9 Platforms,” Supercomputing Frontiers and Innovations 6(4), 57–83 (2019).
13. S. Dazzi, V. Voevodin and V. Voevodin, “Survey on Software Tools that Implement Deep Learning Algorithms on Intel/x86 and IBM/Power8/Power9 Platforms,” Supercomputing Frontiers and Innovations 6(4), 57–83 (2019).
14. S. Khrapov, A. Pisarev, I. Kobelev et al., “The Numerical Simulation of Shallow Water: Estimation of the Roughness Coefficient on the Flood Stage,” Advances in Mechanical Engineering 5, 787016 (2013).
15. A. Khoperskov, S. Khrapov, A Numerical Simulation of the Shallow Water Flow on a Complex Topography. In: Numerical Simulations in Engineering and Science / Edited by Srinivasa Rao 237–254 (IntechOpen, 2018).
16. T. Broecker, K. Teuber, V. S. Gollo, G. Nützmann, J. Lewandowski and R. Hinkelmann, “Integral Flow Modelling Approach for Surface Water-Groundwater Interactions along a Rippled Streambed,” Water 11, 1517 (2019).
17. A. I. Sukhinov, A. E. Chistyakov and V. V. Sidoryakina, “Parallel Solution of Sediment and Suspension Transportation Problems on the Basis of Explicit Schemes,” Communications in Computer and Information Science 910, 306–321 (2018).
18. T. G. Elizarova, A. V. Ivanov, “Regularized Equations for Numerical Simulation of Flows in the Two-Layer Shallow Water Approximation,” Computational Mathematics and Mathematical Physics 58, 714–734 (2018).
19. G. D. Alan and D. Th. Peter, “Advances in the Study of Moving Sediments and Evolving Seabeds,” Surveys in Geophysics 29 (1), 1–36 (2008).
20. T. Dyakonova and A. Khoperskov, “Bottom friction models for shallow water equations: Manning’s roughness coefficient and small-scale bottom heterogeneity,” Journal of Physics: Conference Series 973, 012032 (2018).
21. A.A. Ipatov, L.A. Igumnov, S.Y. Litvinchuk et al., “Modification of Numerical Inversion of Laplace Transform in Solving Problems of Poroviscoelasticity via BEM,” Lobachevskii Journal of Mathematics 40, 304–310 (2019).
22. D.V. Anuprienko and I.V. Kapyrin, “Modeling Groundwater Flow in Unconfined Conditions: Numerical Model and Solvers’ Efficiency,” Lobachevskii Journal of Mathematics 39, 867–873 (2018).
24. A. Coronel, S. Berres and R. Lagos, “Calibration of a sedimentation model through a continuous genetic algorithm,” Inverse Problems in Science and Engineering 27, 1263–1278 (2019).
25. G.T. Török, J. Józsa and S. Baranya, “Validation of a Novel, Shear Reynolds Number Based Bed Load Transport Calculation Method for Mixed Sediments against Field Measurements,” Water 11, 2051 (2019).
26. X. Liu, Ph.d. diss., University of Illinois at Urbana-Champaign (2008).
27. R. Jayaratne, Y. Takayama, T. Shibayama, “Applicability of suspended sediment concentration formulae to large-scale beach morphological changes,” Coastal Engineering Proceedings: Proceedings of 33rd Conference on Coastal Engineering, Santander, Spain 33, 57 (2012).
28. E. I. Debolskaya, V. K. Debolsky, I. I. Grtsuk, O. Y. Maslikova and D. N. Ionov, “Modeling of river bed deformation composed of frozen sediments with increasing environmental temperature,” Ice and Snow 53, 104–110 (2013).
29. B. Przedwojski, K. W. Pilarczyk and R. Blazejewski, River Training Techniques: Fundamentals, Design and Applications (Rotterdam, Balkema, 1995).
30. T. Dyakonova, A. Khoperskov and S. Khrapov, “Numerical Model of Shallow Water: The Use of NVIDIA CUDA Graphics Processors,” Communications in Computer and Information Science 687, 132–145 (2016).
31. I.I. Isaeva, A.A. Voronin, A.V. Khoperskov, K.E. Dubinko, A.Yu. Klikunova, “Decision Support System for the Socio-Economic Development of the Northern Part of the Volga-Akhtuba Floodplain (Russia),” Communications in Computer and Information Science 1083, 63–77 (2019).