# A HARDWARE-ACCELERATED PARALLEL IMPLEMENTATION OF A TWO-DIMENSIONAL SCHEME FOR FREE SURFACE FLOWS

JACEK A. JANKOWSKI
*Department of Hydraulic Engineering in Inland Areas, Federal Waterways Engineering and Research Institute (BAW), Kussmaulstrasse 17, 76187 Karlsruhe, Germany*

This contribution concerns the verification and performance assessment of a hardware-accelerated parallel implementation of an algorithm for the semi-implicit finite difference method for solving the vertically integrated shallow water equations including a non-linear treatment of wetting and drying and conservative advection schemes. Instead of adapting an existing serial, OpenMP-, or MPI-parallelised code with all necessary compromises to be met, the selected approach is to write the code from scratch exposing the fine-grained parallelism of the scheme and execute the whole computational kernel of the code on a state-of-the-art streaming processor, i.e. a GPU. The reached speedups compared to a single CPU core are in the order of 20 or 30 for the double or single precision, respectively, which confirms the attractivity of the presently available advanced programming technologies for detailed, high-resolution river modelling applying commodity hardware.

## INTRODUCTION

There are still some uncertainities how to adapt the existing numerical codes to quickly changing computer architectures in order to profit from new hardware capabilities and how to apply emerging programming paradigms efficiently. One of the answers for this ambiguity is to invest efforts in exposing fine-grained parallelism in existing algorithms. This approach is especially promising due to the fact that the primary source of computational performance of presently available GPUs and APUs (*Graphic* and *Accelerated Processing Units*) are hardware-accelerated vector operations. In this contribution we arbitrarily choose one of the most promising approaches, which allows also a clear and undisturbed assessment what speedups are to be awaited when a non-trivial shallow water solver is adapted for a commodity, but state-of-the-art GPU applying new software technologies.

Due to the specific waterways engineering point of view (applications concerning the assessment of engineering measures affecting rivers, estuaries, coastal areas) and the present GPU properties (greatest efficiency by possibly simple memory access patterns) we concentrate on a general algorithm for simulation of shallow water flows applying structured meshes. In principle, the solution algorithm considered in this paper is based on the semi-implicit finite difference method for the vertically integrated shallow water equations as formulated first by Casulli in 1990 [1]. The rigorous analysis of the governing equations allows a precise determination of those terms, which have to be discretised implicitly, leaving the terms uncritical for the stability in the explicit form. In this way a remarkably stable, accurate and computationally efficient semi-implicit scheme results, which has been

consequently extended from two to three dimensions [2], including a non-hydrostatic version [3] without losing the robustness and simplicity. Consistently, the vertically integrated version is a limit case of the standard three-dimensional scheme, when only one mesh level is taken. The resulting code TRIM has been successfully applied for numerous practical projects and scientific investigations in the past [2,4]. In the present contribution this well-validated finite difference method is complemented with a noticeable new feature, namely introducing an extension for an iterative, non-linear and mass-conservative wetting and drying treatment (e.g. for tidal flats or flooded river fore-lands). A detailed analysis of this new non-linear method is available [5], although formulated for the scheme further development for orthogonal, unstructured meshes, the code UnTRIM [6,7,8].

Various approaches are available how to use the new programming techniques for GPUs in existing larger, complex (legacy) CFD numerical codes, with very different solutions presented up to date. Exemplarily, applied techniques vary from applying semi-automatic scripts for porting shared-memory-parallelised parts of an existing code to kernels running on GPUs [9], over adapting most performance-critical parts of a larger code [10], to thorough, complete re-implementations of complex solvers, e.g. [11]. In the class of the shallow water equations solvers, a few examples of advanced implementations can be found, like [12,13], however implementing only explicit schemes for hyperbolic equations (Roe-type) with typical applications for dam-breaking and flooding. The results of these efforts suggest that a thorough re-write of a previously existing code is the most promising in terms of gaining additional computational performance. Pre-studies applying OpenMP-like accelerator directives as well as CUDA Fortran [16] have confirmed that the aim of bringing the computationally critical larger code parts to the GPU is better achievable using a specialised programming language, like CUDA C/C++ [17] as well as the already available ready-to-use mathematical libraries making use of GPUs.

## PARALLEL IMPLEMENTATION

### The algorithm to be implemented

The precise mathematical description of the scheme is omitted in this short contribution and we concentrate only on the computational aspects of the scheme. The interested reader is re-directed to consult the references mentioned in the previous sections [1,2,3,4,5,6,7,8]. For clarity, the same notation as in these references is applied. The algorithm to be implemented can be sketched as follows.

- Geometry and staggered mesh setup
- Initialisations and allocations
- The time loop:
  - Finding water depths per $i,j$-cell $H^n_{i,j}$ and per edge (in staggered positions) $H^n_{i\pm1/2,j}$ and $H^n_{i,j\pm1/2}$, including wetting and drying treatment
  - Setting the new time step $\Delta t$ for the time level $n$
  - Explicit advection and diffusion operators for velocity $\boldsymbol{Fu}^n_{i\pm1/2,j}$, $\boldsymbol{Fv}^n_{i,j\pm1/2}$

- Assembling the $\boldsymbol{T}$ and the right hand forcing vector $\boldsymbol{b}$ in the non-linear free surface equation $\boldsymbol{V}(\zeta^m) + \boldsymbol{T}\,\zeta^m = \boldsymbol{b}$ (refer to [5])
- Newton iterations over the index $m$ for the new elevation $\zeta^m = \eta^{n+1}_{i,j}$ until the given accuracy ε is reached, i.e. $|\Delta\zeta| = |\zeta^m - \zeta^{m-1}| < \varepsilon$:
  - Computing with $\zeta^m$ the right hand vector in the canonical form of the Newton scheme $\boldsymbol{f}(\zeta^m) = \boldsymbol{V}(\zeta^m) + \boldsymbol{T}\,\zeta^m - \boldsymbol{b}$
  - Setting the derivative of the nonlinear part of the free surface matrix $\boldsymbol{V'}(\zeta^m)$ and adding it to the linear part $\boldsymbol{T}$
  - Solving the linear equations set $\boldsymbol{f'}(\zeta^m)\Delta\zeta = [\boldsymbol{V'}(\zeta^m) + \boldsymbol{T}]\Delta\zeta = \boldsymbol{f}(\zeta^m)$
- Obtaining new velocity components for edges $u^n_{i\pm1/2,j}$ , $v^n_{i,j\pm1/2}$
- Optional output

The so sketched scheme (numerical kernel) is compact enough to be quickly implemented using any appropriate computer language or mathematical software, which allows experimenting with various implementation approaches as programming paradigms evolve. With an exception for the new non-linear wetting and drying treatment [5] and the optional application of a conservative upwind scheme for the velocity advection [14] instead of the Lagrangian scheme, the algorithm is in principle very similar to the numerical kernel of the TRIM code [1], which means that it can be embedded in the already existing sophisticated modelling software for general waterway engineering purposes [4,7].

**Implementation**

As mentioned above, we choose to re-implement the numerical code kernel thoroughly for the execution on a GPU. For simplicity of memory accesses, the finite-difference algorithm version for structured, staggered meshes is taken, with a possible future adaptation for unstructured ones. Some of the existing optimisations for the serial, vectorised and shared-memory parallelised (OpenMP) versions of TRIM [4], as well as the OpenMP- and MPI-parallelised UnTRIM-versions [7,8] are taken into account. The code structure allows ongoing message-passing parallelisation using MPI with simple mesh partitioning methods in the framework of further development efforts. The presented code has been written in CUDA C language with C++ extensions [17] from scratch, using a basic serial code for the reference in terms of securing the solution correctness in all development stages. We apply THRUST [18] and CUSP [19] libraries for vector-based arithmetic operations, consistent host and device memory management and a specialised conjugate gradient linear equation solver. THRUST is a library of parallel linear algebra algorithms based on vector operations with an application interface resembling the C++ Standard Template Library. Similarly, CUSP is a library of generic parallel algorithms for sparse matrix and graph computations. Both libraries are written in CUDA C with C++ extensions and apply well-optimised CUDA kernels for arithmetically intensive operations. However, some specific parts of the code have been realised as CUDA kernels, also alternatively to the ready-to-use THRUST and CUSP solutions. This concerns especially more complex parts of the code, like the advection schemes, for which CUDA C kernels without using the mentioned above libraries

have been written together with their serial equivalents (for CPU). This approach and the fact that codes using THRUST and CUSP libraries are able to run also on serial (CPU) hosts without change allow straightforward performance comparisons between a single CPU core and the GPU. At the beginning of the code execution, mesh, bathymetry and elevation initialisations are realised on the host (CPU), then transferred to the device (GPU), whereby the geometry data residing on the host is applied later for interpolations on the staggered mesh and for input/output. Although generally controlled by the CPU (as a characteristic feature of CUDA programming paradigm), the whole time loop of the scheme is executed solely on the GPU with an exception of output for a set of time steps; for this purpose results must be transferred from GPU to CPU.

## VERIFICATION

Verification and validation of the given algorithm with in its different implementations has been already performed with numerous test cases and applications, presented e.g. in references [1,2,4,6,7]. For completeness, a clearly non-linear verification test case based on the analytical solution presented by Thacker [15] is considered here, concerning a periodic motion of a circular shallow water body, contained in a basin with an axisymmetric paraboloid bottom. The initial free surface profile is also paraboloid, bulged upward. The initial velocity is set to zero; under the influence of gravity acceleration $g$, a periodic motion develops, with the period of $T=2\pi/\omega$ (where $\omega$ is the motion frequency). The position of the circular shoreline oscillates axisymmetrically, advancing and retreating from the natural rest position. Taking as a reference the rest position of the system, i.e. the horizontally flat free surface, and denominating as $h_0$ the water depth in the middle of the body, as $L$ the radius of the equilibrium shoreline and as $\eta_0$ the initial displacement of the initial bulged free surface in the middle of the domain, the equations of motion can be derived as follows [15]:

$$\eta(x,y,t) = h_0\left\{\frac{\sqrt{1-A^2}}{1-A\cos(\omega t)} - 1 - \frac{x^2+y^2}{L^2}\left[\frac{1-A^2}{(1-A\cos(\omega t))^2} - 1\right]\right\} \quad (1)$$

$$u(x,y,t) = \frac{1}{1-A\cos(\omega t)}\left(\frac{1}{2}\omega x A \sin(\omega t)\right) \quad (2)$$

$$v(x,y,t) = \frac{1}{1-A\cos(\omega t)}\left(\frac{1}{2}\omega y A \sin(\omega t)\right) \quad (3)$$

where the frequency of the motion $\omega$ and the parameter $A$ are:

$$\omega = \frac{\sqrt{8gh_0}}{L} \quad A = \frac{(h_0+\eta_0)^2 - h_0^2}{(h_0+\eta_0)^2 + h_0^2} \quad (4)$$

For verification purposes, the geometrical parameters have been chosen to $L$ = 4.0 m, $h_0$ = 0.2 m, $\eta_0$ = 0.01 m. The square computational domain measures 10 m x 10 m, with both coordinates $x$, $y$ varying between -5 m and +5 m. By the resolution of 2000 x 2000 cells one obtains the mesh spacing of $\Delta x = \Delta y$ = 0.005 m. For this set of parameters, the resulting period of motion is $T$ = 6.34 s. The simulation is performed with the time step $\Delta t$ = 0.005 s for 50 s, i.e. for over 7 motion periods. The case is inviscid and frictionless. The conjugate gradient solver accuracy is set to $10^{-10}$ and for the Newton iterations $10^{-12}$, resulting in a moderate number of inner (CG) iterations *O(10)* and up to 3 outer (Newton) iterations due to wetting and drying. The free surface movement is shown in Figure 1 in the form of time series for two de-central positions at *(x,y)* = (-2,0) m and (-1,0) m as well as the central position (0,0) m, where also the analytical solution, Eq. (1) for $\eta$ is plotted for the reference. One obtains a very good agreement of the period and amplitude of the motion compared to the theory. However, the amplitude of the motion diminishes slightly to below 98% of the initial value after 7 periods, which is (besides numerical diffusion) due to the chosen implicitness parameter $\theta$ = 0.65 (weighting between previous and next time step variable values taken in the semi-implicit scheme) and the imperfection in representing the axisymmetric case (initial and boundary conditions, circular shoreline) using a mesh of rectangles.

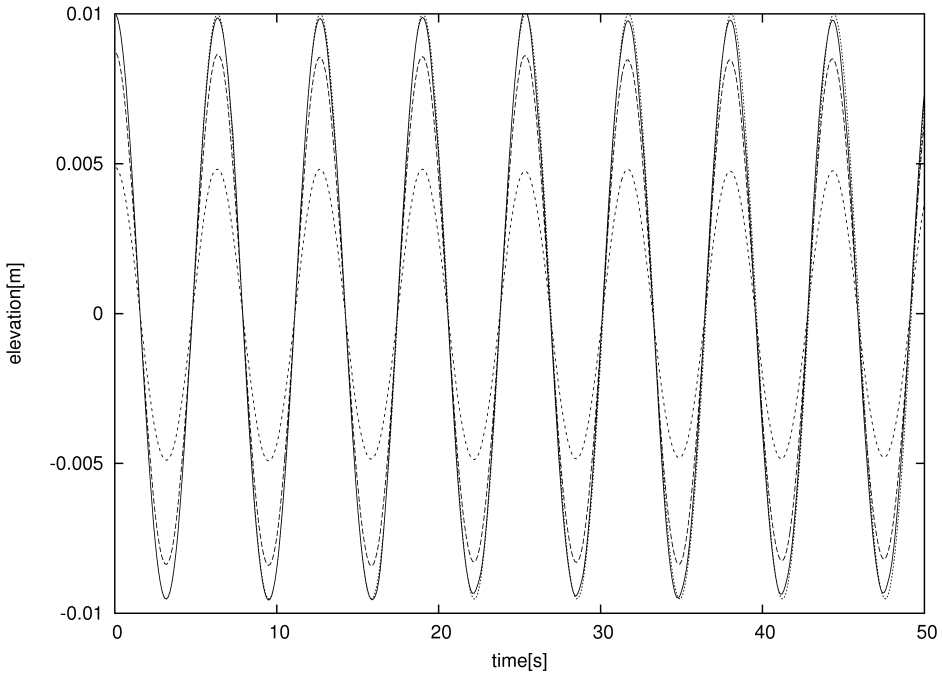


Figure 1. The elevation changes in three points of the domain, for $y$ = 0 m and two de-central positions $x$= -2 m and $x$ = -1 m (dashed lines) and the central position $x$ = 0 m (solid line). For the central position the analytical solution is plotted for comparison (dotted line).

Table 1. Performance of the scheme.

| Example | Precision | Resolution | Speedup |
|---|---|---|---|
| (1) *Nonlin* | single | 1000x1000 | 27 |
| | single | 2000x2000 | 29 |
| | single | 3000x3000 | 28 |
| (1) *Nonlin* | double | 1000x1000 | 21 |
| | double | 2000x2000 | 20 |
| (2) *Sloshing* | single | 1000x1000 | 32 |
| | single | 2000x2000 | 32 |
| | single | 3000x3000 | 30 |
| (2) *Sloshing* | double | 1000x1000 | 23 |
| | double | 2000x2000 | 22 |
| (3) *Lake* | single | 1000x1000 | 32 |
| | single | 2000x2000 | 34 |
| | single | 3000x3000 | 33 |
| (3) *Lake* | double | 1000x1000 | 24 |
| | double | 2000x2000 | 23 |
| (4) *Waves* | single | 1000x1000 | 30 |
| | single | 2000x2000 | 43 |
| | single | 3000x3000 | 42 |
| (4) *Waves* | double | 1000x1000 | 20 |
| | double | 2000x2000 | 20 |

**PERFORMANCE**

The computer system used for performance testing is a good quality, but mass-production PC-system equipped with the 4-core Intel Xeon CPU X5570 working at 2.93 GHz with 8 MB cache working in tandem with the Nvidia graphic card GeForce GTX 480 (Fermi chip) working at 1.4 GHz, with 1.5 GB memory, 15 multiprocessors and 480 stream processing units (CUDA compute capability 2.0). Due to the fact that we are interested in the performance of the computational kernel only, the reached speedups are given exclusively for the time loop part of the algorithm presented in the previous sections and ignoring output to files. With approx. 1.5 GB GPU memory available, a model resolution of up to ca. 4.5 and 9.5 million mesh cells in double and in single precision, respectively, is possible.

The results presented in Table 1 concern speedups reached for four schematic test cases taking into account the typical features of the given scheme: (1) *Nonlin* - the validation test case described in the previous validation section, but computed implicitly, with non-zero viscosity and bottom roughness; (2) *Sloshing* - a small amplitude standing wave in a rectangular basin with an initial flat, sloped free surface (no wetting and drying); (3) *Lake* - similar sloshing, but in a parabolic-bottom round lake (drying and wetting on all shores) but in contrast to case (1) with an initial flat, sloped free surface and in a deeper domain;

(4) *Waves* - waves generated by an initial bump on the free surface attacking shores of a Gaussian hill shaped island with wetting-drying and reflections from sides of a rectangular domain. In all cases the same, quadratic mesh covers the rectangular domain of 10 m x 10 m with water depths for the cases (2-4) in the range of 1m; the time step $\Delta t$ varies between 0.001 s and 0.005 s; 100 time steps are executed. The resolution $\Delta x = \Delta y$ can vary between 0.0033 m (3000x3000 cells) and 0.01 m (1000x1000). The solver accuracies are set exactly as in the verification test case described above, and the implicitness factor $\theta = 1$. In all cases a relatively small bottom roughness of Manning 0.001[-] and small horizontal viscosities $v_x = v_y = 10^{-4}$ m²/s are applied. The reached speedups for the time loop are almost consistently *O(20)* or *O(30)* for the execution applying double and single precision floating point numbers, respectively, reaching 40 in one test case, all compared to the execution of the same code on one of the CPU cores.

## CONCLUSIONS

It is demonstrated that a successful ground-up re-implementation of a non-trivial numerical, two-dimensional, free surface flows scheme for the execution on the GPUs can be obtained applying presently available software tools. The critical issue is not only to adapt the most performance-critical parts of the code, but its possibly large, contiguous parts to be executed on the GPU completely. In this case such contiguous part contains the whole time loop with the computationally most intensive parts of the scheme. The reached speedup compared to the execution on a single CPU core (in the order of 20 or 30 for the double or single precision, respectively) and the fact, that for this relatively compact computational scheme simulations using a few millions of cells on a single GPU are possible, opens a clear potential to increase the time and space resolution in waterways engineering applications using commodity hardware.

## ACKNOWLEDGEMENTS

The author wishes to thank all colleagues who helped directly or indirectly by this work, however, especially Markus Brückner and Frank Platzek for their expert hardware and software support. This paper concerns the BAW R&D internal project A39530270001.

## REFERENCES

[1] Casulli V., “Semi-implicit finite difference methods for the two-dimensional shallow water equations”, *Journal of Computational Physics*, Vol. 86, (1990), pp. 56-74.

[2] Casulli V., Cheng R., “Semi-implicit finite difference methods for three-dimensional shallow water flow”, *International Journal for Numerical Methods in Fluids*, Vol. 15, (1992), pp. 629-648.

[3] Casulli V., “A semi-implicit finite difference method for non-hydrostatic, free surface flows”, *Int. J. for Numerical Methods in Fluids*, Vol. 30, (1999), pp. 425-440.

[4] Bundesanstalt für Wasserbau (BAW). Editor: Lang G., "*HN-Verfahren TRIM-2D. Validierungsdokument, Version 2.0*", Technical report, Bundesanstalt für Wasserbau, Hamburg, (1998).

[5] Casulli V., "A high-resolution wetting and drying algorithm for free-surface hydrodynamics", *Int. J. for Numerical Methods in Fluids*, Vol. 60, (2009), pp. 391-408.

[6] Casulli V., Zanolli P., "Semi-implicit numerical modelling of non-hydrostatic free-surface flows for environmental problems", *Mathematical and Computer Modelling*, Vol. 36, (2002), pp. 1131-1149.

[7] Casulli V., Lang G., "*Mathematical model UnTRIM, validation document, Version 1.0*". Technical report, Bundesanstalt für Wasserbau (BAW), Hamburg, 2004.

[8] Jankowski J.A., "Parallel implementation of a non-hydrostatic model for free surface flows with semi-Lagrangian advection treatment", *Int. J. for Numerical Methods in Fluids*, Vol. 59, (2009), pp. 1157-1179.

[9] Corrigan A., Camelli F.F., Löhner R., Mut F., "Semi-automatic porting of a large-scale Fortran CFD code to GPUs", *Int. J. for Numerical Methods in Fluids,* online, (2011).

[10] Griebel M., Zaspel P., "A multi-GPU accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations", *Computer Science - Research and Development*, Vol. 25, Nr. 1-2, (2010), pp. 65-73.

[11] Cohen J., Molemaker M.J., "A Fast Double Precision CFD Code Using CUDA", *Proceedings of 21st International Conference on Parallel Computational Fluid Dynamics*, Moffett Field, California, USA, May 2009.

[12] Brodtkorb A.R., Saetra M.L., Altinakar M., "Efficient shallow water simulations on GPUs. Implementation, visualisation, verification and validation", *Computers & Fluids,* (2011), in print.

[13] Castro M.J., Ortega S., de la Asuncion M., Mantas J.M., "GPU computing for shallow water flows simulation based on finite volume schemes", *Comptes Rendus Mecanique,* Vol. 339, No. 2-3, (2011), pp. 165-189.

[14] Stelling G.S., Duinmeijer S.P., "A staggered conservative scheme for every Froude number in rapidly varied shallow water flows", *Int. J. for Numerical Methods in Fluids*, Vol. 43, (2003), pp. 1329-1354.

[15] Thacker W.C., "Some exact solutions to the non-linear shallow water equations", *Journal of Fluid Mechanics*, Vol. 107, (1981), pp. 499-508.

[16] The Portland Group, "*PGI Accelerator Programming Model for Fortran & C Version 1.3*". Available from http://www.pgroup.com/resources/accel.htm

[17] NVIDIA Corporation. "*NVIDIA CUDA C Programming Guide*", Version 4.0, (2011). Available from: http://developer.nvidia.com/nvidia-gpu-computing-documentation.

[18] Hoberock J., Bell N., "*THRUST: A Parallel Template Library. Version 1.3.0*", (2010), Software, http://code.google.com/p/thrust/.

[19] Bell N., Garland M., "*CUSP: Generic Parallel Algorithms for Sparse Matrix and Graph Computations. Version 0.1.2*", (2010), Software, http://code.google.com/p/cusp-library/.