

Assessment of a parallel implementation of a two-dimensional scheme for free surface flows using a GPU

JACEK A. JANKOWSKI

*Department of Hydraulic Engineering in Inland Areas,
Federal Waterway Engineering and Research Institute (BAW),
Kußmaulstraße 17, 76187 Karlsruhe, Germany.*

Abstract

The rapid changes in the present computer architectures throw questions concerning the future of the hardware-oriented programming based on the usage of heterogeneous parallel computing resources, consisting of nodes of interconnected, but separate CPUs and GPUs. However, it is believed that the sustainability of the programming efforts nowadays is to be searched in the fact that the principle source of the performance of the present GPUs as well as the emerging APUs is the hardware acceleration of vector operations. Therefore, this contribution delivers an assessment of a hardware-accelerated parallel implementation of a principal algorithm for a vertically integrated finite difference scheme for free surface flows including non-linear treatment of wetting and drying. The chosen approach is to expose the fine-grained parallelism of the scheme and execute the whole computational kernel of the code on a presently available streaming vector processor – a state-of-the-art GPU. The reached speedups compared to a single CPU core are in the order 20 or 30 for the double or single precision, respectively, which opens new possibilities for high resolution flood modelling. However, in order to reach this speedup level the coding must be radically adapted for the vectorised execution.

1 Introduction

Simulation of geophysical phenomena is based on mathematical conceptual models consisting of systems of partial differential equations. Their treatment in a numerical algorithm based on mesh methods leads to the necessity of solving large, but usually sparse systems of equations, linear or non-linear. From the computational point of view, a large number of arithmetic operations (like matrix-vector operations) with the ratio of floating point operations per memory access, (i.e. the so-called arithmetic intensity) of 1:1 is to be performed. For this purpose, computer architectures allowing simultaneous (parallel) processing of long numerical data vectors with a larger memory bandwidth and capacity are the most adequate. Consequently, this fact led ca. 1980-2000 to the dominance of the specially designed vector computers in the area of the high-performance computing (HPC). Despite their clear advantages, these relatively expensive and specialised machines were ca. 1995-2005 systematically ousted and replaced by massively parallel computers with shared or distributed memory applying specialised serial processors. Since ca. 2005 with the appearance of relatively low-priced many-core microprocessors, the dominant HPC architecture is a distributed memory cluster of easily configurable nodes based on mass production, standard hardware. Indeed, during the last decade one could observe the relative decline in the production of machines specifically designed for high-performance scientific computing compared with a huge increase in the number and quality of the general purpose commodity hardware, stationary and mobile.

However, the peak performance of a modern standard CPU, a MIMD-processor (*multi instruction, multiple data*), is attained for ratios of floating point operation per memory access of 10:1, which does not make it the ideal choice for intensive operations on vectors and matrices. The present-day large microprocessor-based clusters are often unable to exploit parallelism finer than one sub-domain (part of data for one processor), with threading hindered by the memory bandwidth shared by the processor cores and by the memory access latency.

This awkward situation has been addressed by special accelerator or co-processor chip designs dedicated mostly to the vector computing, like reconfigurable FPGA (*field-programmable gate arrays*), Cell-BEs (*broad-band engines*) and larger vector, SIMD (*single instruction, multiple data*) units of modern CPUs. However, the tremendous increase in the performance of widely available GPUs (*Graphic Processing Units*) in the last decade, the new GPU chip and memory designs allowing standard float point computation and the full opening ca. 2007 of the general-purpose programming interface have the potential of making the streaming processors of GPUs

the hardware of choice for compute-intensive parallel applications – especially for people without access to large clusters.

Indeed, the present-day programmer of a parallel machine is confronted with three types of parallelism to be dealt with:

1. *Fine-grained*: compute cores of a GPU or another SIMD accelerator — mostly programming with the vector/matrix-operations methodology using a specialised language; e.g. with CUDA C (*Compute Unified Device Architecture* [25]), OpenCL [26], directives [27] and/or a vector instruction set for a SIMD unit of the CPU applying a specialised compiler.
2. *Medium-grained*: using heterogeneous resources within the computational node (CPU cores, caches, threads) — the largest programming flexibility, using serial programming and threading, e.g. with OpenMP [28].
3. *Coarse grained*: programming with the message-passing communication between nodes of a cluster applying e.g. MPI [29]. — data decomposition or partitioning methods, clustering.

While the available programming tools allow addressing these parallelism types separately, the combination of any two or all three in a hybrid system is difficult, case-dependent and the matter of present-day research. This is especially due to the limitations of the hardware concerning the data transfers across the PCIe bus, i.e. between the host (CPU cores) and the devices (a single or multiple GPU) memories and beyond one computational node of a cluster over the interconnecting network. An additional issue in 2011 is the emerging APU (*Accelerated Processing Unit*) technology, merging CPU and GPU on one chip, with common caches and accessing the same memory areas, which would alleviate data transfer latencies and radically simplify the memory management. With the performance of these new hybrid APUs compared to new GPU designs still to be assessed, there exists an uncertainty how to adapt the existing CFD codes to the quickly changing hardware architecture.

The author believes that the answer for this ambiguity is to invest effort in exposing the fine-grained parallelism in the existing algorithms because the primary source of the computational performance of the present GPUs and coming APUs are the hardware-accelerated vector operations. It is not tried in this paper to use the full range of presently available heterogeneous parallel resources, like computing in a tandem of CPU and GPU or applying multiple GPUs, but using the CPU for the steering only, while bringing the whole computationally intensive part of the code to the parallel execution on a GPU. Instead of re-implementing of an existing serial, OpenMP-, or MPI-parallelised code to a GPU with all necessary compromises to be met, only the principal algorithm of a vertically integrated shallow water flow solver is adapted completely for a GPU – with simplified initial and boundary conditions, but with all typical features and computational load of the scheme. This concentrated, radical approach allows a clear and undisturbed assessment what speedups are to be awaited when a non-trivial shallow water solver is adapted for a state-of-the-art vector-parallel processor, a GPU, using presently available programming tools and libraries.

2 The numerical scheme

2.1 Shallow water equations

For the purpose of reference and convenience, the well known two-dimensional, vertically averaged shallow water equations in the transient, non-conservative form for the non-rotating Cartesian reference frame are quoted:

$$\begin{aligned}
 H(u_t + uu_x + vu_y) &= -gH\eta_x + (vHu_x)_x + (vHu_y)_y + \gamma_T u_a - \gamma u \\
 H(v_t + uv_x + vv_y) &= -gH\eta_y + (vHv_x)_x + (vHv_y)_y + \gamma_T v_a - \gamma v \\
 \eta_t + (Hu)_x + (Hv)_y &= 0
 \end{aligned} \tag{1}$$

consisting of two momentum equations and the continuity equation for variables $u(x, y, t)$, $v(x, y, t)$ as the horizontal, vertically averaged velocity components in the x - and y -direction and $\eta(x, y, t)$ as the free surface level (t being the time). The total water depth is $H(x, y, t) = \max(0, \eta(x, y, t) + h(x, y))$, whereby $h(x, y)$ is the given bathymetry. The prescribed horizontal viscosity is denoted by ν and the friction terms represent the combined friction at the bottom and at the free surface due to the wind velocity (u_a, v_a) , evaluated as $\gamma_B u + \gamma_T(u - u_a) = (\gamma_B + \gamma_T)u - \gamma_T u_a = \gamma u - \gamma_T u_a$. The gravitational acceleration is denoted by g .

2.2 The semi-implicit finite-difference scheme

The considered solution algorithm is based on the general semi-implicit finite difference method for the shallow water equations as formulated by Casulli et al. [1, 2, 3, 4, 5]. The rigorous analysis of the governing equations (1) allows a precise determination of the terms, which have to be discretised in the implicit way, leaving the terms uncritical for the stability in the explicit form [1]. In this way a remarkably stable, accurate and computationally efficient scheme results, which can be also extended from two to three dimensions [2, 3, 4], including a non-hydrostatic version [5] without losing the robustness and simplicity. Consistently, the vertically integrated scheme (2Dxy) is a limit case of the standard 3D-scheme, when only one mesh level is taken. The resulting codes TRIM-2D and TRIM-3D have been applied for numerous practical projects and scientific investigations in the past, e.g. [3, 4, 6, 7].

In this contribution the mentioned above semi-implicit finite difference method is followed with a noticeable new feature: introducing an extension for an improved, mass conservative non-linear wetting and drying treatment (e.g. tidal flats, a river foreland, etc.). A detailed analysis of this non-linear method is available [13, 14], albeit formulated for the further development of the given scheme for the orthogonal unstructured mesh (code UNTRIM) [8, 9, 10, 12]. In the following, the notation applied is consistent with the references cited above for the sake of convenience.

2.3 The discrete non-linear scheme

The finite difference numerical scheme is formulated on a rectangular, staggered mesh consisting of $N_x \times N_y$ rectangular polygons of length Δx and width Δy , where N_x and N_y are the number of polygons in the x and y directions, respectively. So defined computational domain Ω is fixed and intended to cover the time-dependent domain covered by the fluid $\Omega(t) = \{(x, y) \in \Omega : H(x, y, t) > 0\}$ for the simulation time $t \geq 0$ thoroughly. The scheme is transient, with a discrete time step of Δt and time steps numbered with n . The mesh cells (rectangular polygons) are numbered with indices (i, j) at their centres, where also the bathymetry $h_{i,j}$, the position of the free surface level $\eta_{i,j}^n$ and the total water depth $H_{i,j}^n$ at the time step n are defined. The discrete velocity components u and v are defined at polygon edges, whereby in the discretised equations the numbering reflecting the position of the velocity points is chosen: u is defined at half integer i and full integer j , while v at full integer i and half integer j , so that at the time step n we have on the four polygon edges $u_{i\pm\frac{1}{2},j}^n$ and $v_{i,j\pm\frac{1}{2}}^n$. Accordingly, also the bathymetry and the variable total water depth are defined at the u and v edge positions with the same notation, e.g. $H_{i\pm\frac{1}{2},j}^n$ and $H_{i,j\pm\frac{1}{2}}^n$. The roughness factors γ_T , γ and the viscosity ν are also defined separately for u and v mesh edges. In principle, the scheme allows a representation of the partially wet and partially dry mesh cells (rectangular polygons), without the necessity to determine the precise distribution of flooded areas in the cell. For this purpose, the total water depth H is defined in a non-linear way as follows:

$$H(x, y, \eta^n) = \int_{-\infty}^{\eta^n} p(x, y, z) dz = \max(0, h(x, y) + \eta^n) \quad (2)$$

where an auxiliary function $p(x, y, z)$ (*porosity*) is introduced in order to represent the depth distribution:

$$p(x, y, z) = \begin{cases} 1 & \text{if } h(x, y) + z > 0 \\ 0 & \text{if } h(x, y) + z \leq 0 \end{cases} \quad (3)$$

The computational cells can be described as $\Omega_{i,j}(\eta^n) = \{(x, y) : x_{i-\frac{1}{2}} \leq x \leq x_{i+\frac{1}{2}}, y_{j-\frac{1}{2}} \leq y \leq y_{j+\frac{1}{2}}, \text{ where } H(x, y, \eta^n) > 0\}$; therefore the horizontal wet area of a partially flooded polygon $p_{i,j}$ corresponding to $\eta_{i,j}^n$ can be expressed in terms of the auxiliary function (3) as:

$$0 \leq p_{i,j}(\eta_{i,j}^n) = \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} p(x, y, \eta_{i,j}^n) dx dy \leq \Delta x \Delta y \quad (4)$$

In the consequence, the water depth can vary along the edge according to a given sub-grid distribution. The cross-sectional area of the water column above the wet edge part (e.g. for a u -edge) can be defined as:

$$A_{i+\frac{1}{2},j}^n = \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} H(x_{i+\frac{1}{2}}, y, \eta_{i+\frac{1}{2},j}^n) dy \quad (5)$$

we can define an average total depth along a wet or partially wet edge:

$$H_{i+\frac{1}{2},j}^n = \frac{A_{i+\frac{1}{2},j}^n}{\int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} p(x_{i+\frac{1}{2}}, y, \eta_{i,j}^n) dy} = \frac{A_{i+\frac{1}{2},j}^n}{\Delta y_{i+\frac{1}{2},j}^n} \quad (6)$$

Where the wetted edge length $\Delta y_{i+\frac{1}{2},j}^n$ can vary in time as well (note the difference to the constant spacing Δy). The volume in every polygon can be expressed applying the porosity function as:

$$0 \leq V_{i,j}(\eta_{i,j}^n) = \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} H(x,y, \eta_{i,j}^n) dx dy = \int_{-\infty}^{\eta_{i,j}^n} p_{i,j}(z) dz \quad (7)$$

With so defined geometry of wetting and drying, one can discretise in the semi-implicit way the momentum equations of shallow water equations (1) applying the θ -method for the free surface gradients, i.e. weighting with the implicitness factor θ between time levels $n+1$ and n [4]:

$$\begin{aligned} \left(H_{i+\frac{1}{2},j}^n + \Delta t \gamma_{i+\frac{1}{2},j}^n \right) u_{i+\frac{1}{2},j}^{n+1} &= \mathbf{F} u_{i+\frac{1}{2},j}^n \\ -g \frac{\Delta t}{\Delta x} H_{i+\frac{1}{2},j}^n \left[\theta \left(\eta_{i+1,j}^{n+1} - \eta_{i,j}^{n+1} \right) + (1-\theta) \left(\eta_{i+1,j}^n - \eta_{i,j}^n \right) \right] &+ \Delta t \gamma_T u_a \end{aligned} \quad (8)$$

$$\begin{aligned} \left(H_{i,j+\frac{1}{2}}^n + \Delta t \gamma_{i,j+\frac{1}{2}}^n \right) v_{i,j+\frac{1}{2}}^{n+1} &= \mathbf{F} v_{i,j+\frac{1}{2}}^n \\ -g \frac{\Delta t}{\Delta y} H_{i,j+\frac{1}{2}}^n \left[\theta \left(\eta_{i,j+1}^{n+1} - \eta_{i,j}^{n+1} \right) + (1-\theta) \left(\eta_{i,j+1}^n - \eta_{i,j}^n \right) \right] &+ \Delta t \gamma_T v_a \end{aligned} \quad (9)$$

where $\mathbf{F}u = Hu^* + \Delta t[(vu_x)_x + (vu_y)_y]^*$ is an explicit, non-linear finite difference operator containing explicit discretisation of the advective and viscous terms, to be discussed later. The term γ_T concerns the wind friction with a prescribed wind velocity (u_a, v_a) and the term $\gamma = r_B \sqrt{u_*^2 + v_*^2}$ represents the bottom friction evaluated after applying the advection and diffusion operator on the velocity. r_B is friction coefficient, which can be defined in many ways, e.g. in the Chézy formulation $r_B = g/C_z^2$. Due to the fact that $\eta_t = H_t$, one obtains by integrating the continuity equation of (1) in the cell area the following expression:

$$\left[\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} H_{i,j} dx dy \right]_t + \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} \left[(Hu)_{i+\frac{1}{2}} - (Hu)_{i-\frac{1}{2}} \right] dy + \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \left[(Hv)_{j+\frac{1}{2}} - (Hv)_{j-\frac{1}{2}} \right] dx = 0 \quad (10)$$

Noticing that the first integral in (10) represents the volume and the averaged u is constant along the y edge (and v is constant along the x edge) we get a finite volume form of the continuity equation representing a precise volume balance in a cell (i, j) :

$$\begin{aligned} V_{i,j}(\eta_{i,j}^{n+1}) &= V_{i,j}(\eta_{i,j}^n) - \theta \Delta t \left[A_{i+\frac{1}{2},j}^n u_{i+\frac{1}{2},j}^{n+1} - A_{i-\frac{1}{2},j}^n u_{i-\frac{1}{2},j}^{n+1} + A_{i,j+\frac{1}{2}}^n v_{i,j+\frac{1}{2}}^{n+1} - A_{i,j-\frac{1}{2}}^n v_{i,j-\frac{1}{2}}^{n+1} \right] \\ &- (1-\theta) \Delta t \left[A_{i+\frac{1}{2},j}^n u_{i+\frac{1}{2},j}^n - A_{i-\frac{1}{2},j}^n u_{i-\frac{1}{2},j}^n + A_{i,j+\frac{1}{2}}^n v_{i,j+\frac{1}{2}}^n - A_{i,j-\frac{1}{2}}^n v_{i,j-\frac{1}{2}}^n \right] \end{aligned} \quad (11)$$

whereby the velocity is discretised with the θ -method [4]. Introducing the equations (8-9) to (11) and taking account of the equation (6) one obtains an equation system for $\eta_{i,j}^{n+1}$:

$$\begin{aligned} V_{i,j}(\eta_{i,j}^{n+1}) - \theta^2 g \frac{\Delta t^2}{\Delta x} \left[\left(\frac{\Delta y H^2}{H + \Delta t \gamma} \right)_{i+\frac{1}{2},j}^n \left(\eta_{i+1,j}^{n+1} - \eta_{i,j}^{n+1} \right) - \left(\frac{\Delta y H^2}{H + \Delta t \gamma} \right)_{i-\frac{1}{2},j}^n \left(\eta_{i,j}^{n+1} - \eta_{i-1,j}^{n+1} \right) \right] \\ - \theta^2 g \frac{\Delta t^2}{\Delta y} \left[\left(\frac{\Delta x H^2}{H + \Delta t \gamma} \right)_{i,j+\frac{1}{2}}^n \left(\eta_{i,j+1}^{n+1} - \eta_{i,j}^{n+1} \right) - \left(\frac{\Delta x H^2}{H + \Delta t \gamma} \right)_{i,j-\frac{1}{2}}^n \left(\eta_{i,j}^{n+1} - \eta_{i,j-1}^{n+1} \right) \right] \\ = b_{i,j}^n \end{aligned} \quad (12)$$

with the right hand side vector $b_{i,j}^n$ containing all known terms in equation (12) at the time level n . Denoting for brevity $G_x = \mathbf{F}u + \Delta t \gamma_T u_a$ and $G_y = \mathbf{F}v + \Delta t \gamma_T v_a$ one obtains for $b_{i,j}^n$:

$$\begin{aligned} b_{i,j}^n &= V_{i,j}(\eta_{i,j}^n) \\ &- \theta \Delta t \left[\left(\frac{\Delta y H G_x}{H + \Delta t \gamma} \right)_{i+\frac{1}{2},j}^n - \left(\frac{\Delta y H G_x}{H + \Delta t \gamma} \right)_{i-\frac{1}{2},j}^n + \left(\frac{\Delta x H G_y}{H + \Delta t \gamma} \right)_{i,j+\frac{1}{2}}^n - \left(\frac{\Delta x H G_y}{H + \Delta t \gamma} \right)_{i,j-\frac{1}{2}}^n \right] \\ &- (1 - \theta) \Delta t \left[(\Delta y H u)_{i+\frac{1}{2},j}^n - (\Delta y H u)_{i-\frac{1}{2},j}^n + (\Delta x H v)_{i,j+\frac{1}{2}}^n - (\Delta x H v)_{i,j-\frac{1}{2}}^n \right] \end{aligned} \quad (13)$$

The non-linearity of (12) is caused by the definition of $V_{i,j}(\eta_{i,j}^n)$, being the integral of the piece-wise constant, but discontinuous function p . Introducing a new variable $\zeta_{i,j}^m = \eta_{i,j}^{m+1}$, the equation set (12) can be written in a compact form:

$$\mathbf{V}(\zeta) + \mathbf{T}\zeta = \mathbf{b} \quad (14)$$

where \mathbf{V} is the $N_x N_y$ -long vector of cell volumes and \mathbf{b} is defined by (13). The structure of the $N_x N_y \times N_x N_y$ matrix \mathbf{T} can be analysed taking (12) into account. Denominating for clarity $R = \Delta x H^2 / (H + \Delta t \gamma)$ and $S = \Delta y H^2 / (H + \Delta t \gamma)$:

$$\begin{aligned} t_{i,j} &= \theta^2 g \frac{\Delta t^2}{\Delta x} (S_{i+\frac{1}{2},j} + S_{i-\frac{1}{2},j}) + \theta^2 g \frac{\Delta t^2}{\Delta y} (R_{i,j+\frac{1}{2}} + R_{i,j-\frac{1}{2}}), \\ t_{i+\frac{1}{2},j} &= -\theta^2 g \frac{\Delta t^2}{\Delta x} S_{i+\frac{1}{2},j}, & t_{i-\frac{1}{2},j} &= -\theta^2 g \frac{\Delta t^2}{\Delta x} S_{i-\frac{1}{2},j} \\ t_{i,j+\frac{1}{2}} &= -\theta^2 g \frac{\Delta t^2}{\Delta y} R_{i,j+\frac{1}{2}}, & t_{i,j-\frac{1}{2}} &= -\theta^2 g \frac{\Delta t^2}{\Delta y} R_{i,j-\frac{1}{2}} \end{aligned} \quad (15)$$

The matrix \mathbf{T} is clearly penta-diagonal with the main diagonal being a sum of contributions from all surrounding edges of a cell (i, j) . The values stemming from neighbouring mesh cells reside in the side diagonals. So structured matrix satisfies conditions for the convergence of the Newton method of solving non-linear equations. Rewriting (14) in the canonical form:

$$\mathbf{f} = \mathbf{V}(\zeta) + \mathbf{T}\zeta - \mathbf{b} = 0 \quad (16)$$

and denoting with m the Newton iteration index one obtains the classical form of the scheme:

$$\zeta^{m+1} = \zeta^m + \Delta \zeta = \zeta^m - [\mathbf{f}'(\zeta^m)]^{-1} \mathbf{f}(\zeta^m) = \zeta^m - [\mathbf{V}'(\zeta^m) + \mathbf{T}]^{-1} [\mathbf{V}(\zeta^m) + \mathbf{T}\zeta^m - \mathbf{b}] \quad (17)$$

The derivative of the cell volume $\mathbf{V}'(\zeta) = p_{i,j}(\zeta_{i,j}^m)$ according to the definition (7) is a single-diagonal matrix with non-negative values, to be added to \mathbf{T} in (17). Indeed, due to (7) we have at the diagonal of \mathbf{V}' values of $0 < p_{i,j} < \Delta x \Delta y$ for partially wet, $p_{i,j} = \Delta x \Delta y$ for wet cells and zero values $p_{i,j} = 0$ otherwise. In the exceptional cases of cells (i, j) being completely dry and surrounded by dry cells, a line from the system matrix must be eliminated (or the diagonal term just set to one). Each iteration of the Newton method (17) can be performed applying the conjugate gradient method solving for $\Delta \zeta$ in $[\mathbf{V}'(\zeta^m) + \mathbf{T}]\Delta \zeta = \mathbf{f}$, with the pre-conditioning necessity depending on the numbering scheme of the cells.

The iterations are continued until a given accuracy ε is reached in terms of a vector norm, $|\Delta \zeta| \leq \varepsilon$. Due to the mild non-linearity of the system, the number of iterations required is very moderate, with only one iteration if the wet-dry pattern does not change. The formal discussion the nonlinear method can be found in [13] (for an unstructured mesh).

After finding the new water levels $\eta_{i,j}^{m+1}$, the the velocity components can be computed from the momentum equations (8-9) and the new water depths per edge are found applying (6).

2.4 The discrete linear scheme

One can obtain easily a version of the free surface equation (12) which does not take the sub-grid geometry into account by setting the constant spacing instead of the variable wetted edge length $\Delta y_{i\pm\frac{1}{2},j}^n = \Delta y$, $\Delta x_{i,j\pm\frac{1}{2}}^n = \Delta x$. Assuming that we consider the the completely wet cells only we can divide both sides of equation (12) by the total cell area $\Delta x \Delta y$:

$$\begin{aligned} \eta_{i,j}^{n+1} & - \theta^2 g \frac{\Delta t^2}{\Delta x^2} \left[\left(\frac{H^2}{H + \gamma \Delta t} \right)_{i+\frac{1}{2},j}^n \left(\eta_{i+1,j}^{n+1} - \eta_{i,j}^{n+1} \right) - \left(\frac{H^2}{H + \gamma \Delta t} \right)_{i-\frac{1}{2},j}^n \left(\eta_{i,j}^{n+1} - \eta_{i-1,j}^{n+1} \right) \right] \\ & - \theta^2 g \frac{\Delta t^2}{\Delta y^2} \left[\left(\frac{H^2}{H + \gamma \Delta t} \right)_{i,j+\frac{1}{2}}^n \left(\eta_{i,j+1}^{n+1} - \eta_{i,j}^{n+1} \right) - \left(\frac{H^2}{H + \gamma \Delta t} \right)_{i,j-\frac{1}{2}}^n \left(\eta_{i,j}^{n+1} - \eta_{i,j-1}^{n+1} \right) \right] \\ & = b_{i,j}^n \end{aligned} \quad (18)$$

whereby the right hand side $b_{i,j}^n$ is written as follows:

$$\begin{aligned} b_{i,j}^n = \eta_{i,j}^n & - \theta \frac{\Delta t}{\Delta x} \left[\left(\frac{HG_x}{H + \gamma \Delta t} \right)_{i+\frac{1}{2},j}^n - \left(\frac{HG_x}{H + \gamma \Delta t} \right)_{i-\frac{1}{2},j}^n \right] \\ & - \theta \frac{\Delta t}{\Delta y} \left[\left(\frac{HG_y}{H + \gamma \Delta t} \right)_{i,j+\frac{1}{2}}^n - \left(\frac{HG_y}{H + \gamma \Delta t} \right)_{i,j-\frac{1}{2}}^n \right] \\ & - (1 - \theta) \frac{\Delta t}{\Delta x} \left[(Hu)_{i+\frac{1}{2},j}^n - (Hu)_{i-\frac{1}{2},j}^n \right] - (1 - \theta) \frac{\Delta t}{\Delta y} \left[(Hv)_{i,j+\frac{1}{2}}^n - (Hv)_{i,j-\frac{1}{2}}^n \right] \end{aligned} \quad (19)$$

Defined on a finite difference mesh of $N_x \times N_y$ polygons, the equations (18-20) form a linear equation system of $N_x N_y$ equations with a sparse, penta-diagonal matrix for $\eta_{i,j}^{n+1}$. Due to the fact that the water depths $H_{i\pm\frac{1}{2},j}^{n+1}$, $H_{i,j\pm\frac{1}{2}}^{n+1}$ are non-negative, the system is positive-definite and symmetric and can be solved by conjugate gradient methods even without applying the nonlinear Newton iteration scheme. Completely dry cells have to be detected and removed from the equation system. However, even when the cell edges are treated completely wet or completely, the non-linear scheme described in the previous section 2.3 does not loose its generality and opens the possibility of moving the shoreline more than one (whole) cell during a single time-step.

After obtaining the new water levels $\eta_{i,j}^{n+1}$, the velocity components can be computed from the momentum equations (8-9) and the new water depths per edge are found applying (6).

2.5 Discretisation of advective and viscous terms

The advective and viscous terms are discretised using an explicit formulation. If it is appropriate to neglect the advection and horizontal viscosity terms, the operator \mathbf{F} in (8) and (9) reduces to $\mathbf{F}u_{i+\frac{1}{2},j}^n = H_{i+\frac{1}{2},j}^n u_{i+\frac{1}{2},j}^n$.

If the Courant-number-independent Eulerian-Lagrangian discretisation for the advection is applied (i.e. the method of characteristics), with a simplification the viscosity terms $(1/H)\nabla \cdot (Hv\nabla u)$ to the form $\nabla \cdot (v\nabla u)$, one can exemplarily write for the u component:

$$\mathbf{F}u_{i+\frac{1}{2},j}^n = H_{i+\frac{1}{2},j}^n u_{i+\frac{1}{2},j}^* + \Delta t \left[(vu_x)_x + (vu_y)_y \right]_{i+\frac{1}{2},j}^* \quad (20)$$

where $u_{i+\frac{1}{2},j}^*$ is the velocity at the time level t^n interpolated from the surrounding grid points at the foot of the Lagrangian trajectory obtained by integrating the velocity starting at the velocity node $(i + \frac{1}{2}, j)$ backward in time from t^{n+1} to t^n . $[(vu_x)_x (vu_y)_y]_{i+\frac{1}{2},j}^*$ is the discretisation of the horizontal viscosity term also at the foot of the Lagrangian trajectory. The methodology is described in detail for rectangular meshes in [1, 2] for linear interpolation and in [7] for the quadratic interpolation. If a classical upwind method (with the Courant number limitation on the time step) is applied for the advection [1], \mathbf{F} is a sum of advective \mathbf{F}_{adv} and viscous \mathbf{F}_{visc} contributions.

For the purposes of the performance-oriented investigations described in this paper, one chooses for the advection an upwind scheme with constant or variable $\Delta t < \Delta x / |u|_{max}$:

$$\begin{aligned}
A_m &= \frac{1}{2} \left(\frac{1}{2} (u_{i+\frac{3}{2},j}^n + u_{i+\frac{1}{2},j}^n) - \frac{1}{2} |u_{i+\frac{3}{2},j}^n + u_{i+\frac{1}{2},j}^n| \right) \\
A_p &= \frac{1}{2} \left(\frac{1}{2} (u_{i-\frac{1}{2},j}^n + u_{i+\frac{1}{2},j}^n) + \frac{1}{2} |u_{i-\frac{1}{2},j}^n + u_{i+\frac{1}{2},j}^n| \right) \\
B_m &= \frac{1}{2} \left(\frac{1}{2} (v_{i,j+\frac{3}{2}}^n + v_{i,j+\frac{1}{2}}^n) - \frac{1}{2} |v_{i,j+\frac{3}{2}}^n + v_{i,j+\frac{1}{2}}^n| \right) \\
B_p &= \frac{1}{2} \left(\frac{1}{2} (v_{i,j-\frac{1}{2}}^n + v_{i,j+\frac{1}{2}}^n) + \frac{1}{2} |v_{i,j-\frac{1}{2}}^n + v_{i,j+\frac{1}{2}}^n| \right)
\end{aligned} \tag{21}$$

$$\begin{aligned}
\mathbf{F}_{adv} u_{i+\frac{1}{2},j}^n &= u_{i+\frac{1}{2},j}^n - \frac{\Delta t}{\Delta x} \left[A_m (u_{i+\frac{3}{2},j}^n - u_{i+\frac{1}{2},j}^n) + A_p (u_{i+\frac{1}{2},j}^n - u_{i-\frac{1}{2},j}^n) \right] \\
&\quad - \frac{\Delta t}{\Delta y} \left[B_m (u_{i+\frac{1}{2},j+1}^n - u_{i+\frac{1}{2},j}^n) + B_p (u_{i+\frac{1}{2},j}^n - u_{i+\frac{1}{2},j-1}^n) \right] \\
\mathbf{F}_{adv} v_{i,j+\frac{1}{2}}^n &= v_{i,j+\frac{1}{2}}^n - \frac{\Delta t}{\Delta x} \left[A_m (v_{i+1,j+\frac{1}{2}}^n - v_{i,j+\frac{1}{2}}^n) + A_p (v_{i,j+\frac{1}{2}}^n - v_{i-1,j+\frac{1}{2}}^n) \right] \\
&\quad - \frac{\Delta t}{\Delta y} \left[B_m (v_{i,j+\frac{3}{2}}^n - v_{i,j+\frac{1}{2}}^n) + B_p (v_{i,j+\frac{1}{2}}^n - v_{i,j-\frac{1}{2}}^n) \right]
\end{aligned} \tag{22}$$

and for the viscous terms central differences:

$$\begin{aligned}
\mathbf{F}_{visc} u_{i+\frac{1}{2},j}^n &= + \frac{v\Delta t}{\Delta x^2} \left(u_{i-\frac{1}{2},j}^n - 2u_{i+\frac{1}{2},j}^n + u_{i+\frac{3}{2},j}^n \right) \\
&\quad + \frac{v\Delta t}{\Delta y^2} \left(u_{i+\frac{1}{2},j-1}^n - 2u_{i+\frac{1}{2},j}^n + u_{i+\frac{1}{2},j+1}^n \right) \\
\mathbf{F}_{visc} v_{i,j+\frac{1}{2}}^n &= + \frac{v\Delta t}{\Delta y^2} \left(v_{i,j-\frac{1}{2}}^n - 2v_{i,j+\frac{1}{2}}^n + v_{i,j+\frac{3}{2}}^n \right) \\
&\quad + \frac{v\Delta t}{\Delta x^2} \left(v_{i-1,j+\frac{1}{2}}^n - 2v_{i,j+\frac{1}{2}}^n + v_{i+1,j+\frac{1}{2}}^n \right)
\end{aligned} \tag{23}$$

It must be noted that there are numerous possibilities for defining the explicit advection \mathbf{F}_{adv} and diffusion \mathbf{F}_{visc} operators, whereby the conservative advection schemes valid for all flow regimes by the presence of wetting and drying are especially attractive for models applying staggered finite difference grids [15, 16].

3 Parallel implementation

3.1 The algorithm to be implemented

The algorithm to be implemented can be sketched as follows:

- Mesh and geometry setup
- Initialisations and allocations
- The time loop:
 - Finding water depths $H_{i,j}^n, H_{i\pm\frac{1}{2},j}^n, H_{i,j\pm\frac{1}{2}}^n$, including wetting and drying
 - Setting the new time step Δt for the time level n
 - Explicit advection and diffusion operators $\mathbf{F}u_{i\pm\frac{1}{2},j}^n, \mathbf{F}v_{i,j\pm\frac{1}{2}}^n$
 - Assembling the linear parts of free surface matrix \mathbf{T} and the right hand vector \mathbf{b}
 - Newton iterations for the new elevation $\zeta^m = \eta_{i,j}^{n+1}$ until $|\Delta\zeta| = |\zeta^m - \zeta^{m-1}| \leq \varepsilon$:
 - * Computing with ζ^m the right hand vector $\mathbf{f}(\zeta^m) = \mathbf{V}(\zeta^m) + \mathbf{T}\zeta^m - \mathbf{b}$
 - * Setting the nonlinear part of the free surface matrix $\mathbf{V}'(\zeta^m)$ and adding it to \mathbf{T}
 - * Solving iteratively the linear equations set $[\mathbf{V}'(\zeta^m) + \mathbf{T}]\Delta\zeta = \mathbf{f}(\zeta^m)$

- Obtaining new velocities $u_{i\pm\frac{1}{2},j}^{n+1}, v_{i,j\pm\frac{1}{2}}^{n+1}$
- Optional output

The principle semi-implicit algorithm is compact enough to be quickly implemented using any appropriate computer language or mathematical software, which allows experimenting with various implementation approaches as the programming paradigms evolve. With an exception for the new non-linear wetting and drying the algorithm is similar to the numerical kernel of the TRIM-2D code [2, 3] which can be embedded in a sophisticated modelling software for general purposes [6].

3.2 Approach

Although there exist a serial implementation of a similar scheme as the numerical kernels of the TRIM-2D and TRIM-3D codes [2, 3], as well as special vectorised and shared-memory parallelised (OpenMP) versions [6], the approach chosen in this paper is to write a new code almost from a scratch using the previous implementations as a reference. One reason for this that these legacy FORTRAN codes do not implement the non-linear treatment of wetting and drying which has been introduced for the successor program UNTRIM applying non-structured meshes [8, 9, 13, 14]. Another reason is that pre-studies with these codes applying OpenMP-like accelerator directives as well as CUDA FORTRAN [27] have shown that up to date, the radical aim of bringing possibly the whole code to the GPU is better achievable using CUDA C [25] or OpenCL [26] due to the larger experience of the scientific community with these languages and already existing, ready-to use, but computer language-specific vector-parallel numerical libraries. This approach does not exclude a re-implementation of this relatively compact piece of code in the form of a numerical kernel into other languages later on as the programming paradigms evolve, especially in sight of the new hardware developments, like APUs (*Accelerated Processing Units*).

The principle aim of the study is to assess the economy of the hardware-oriented programming in a setting allowing experiments with different hardware-aware approaches taking advantage of the presently available tools. Confronted with promising results of hardware-aware programming paradigms, there is an ongoing discussion how to deal with larger, complex (legacy) CFD codes, with different proposed solutions. Exemplarily, the approaches vary from adapting the fairly most computationally intensive parts [17], semi-automatic scripts for porting parts of the code to kernels running on GPUs [18] to thorough re-implementations – from simpler (explicit) codes [19] to complete re-writes of more complex solvers [20].

In this particular case the code has been written in C++ language from a scratch, using a basic serial code for the reference in terms of solution correctness in all stages of the development. So far as possible, the presently available versions of THRUST [24] and CUSP [23] numerical libraries have been applied. 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 CUDA kernels for arithmetically intensive operations. It must be noted, that codes using THRUST and CUSP libraries are able to run also on serial hosts without change. Both libraries have been applied for a consistent host and device memory management, transferring single loops of the serial code into GPU kernels, as well as for a specialised conjugate gradient solvers. For more complex parts of the code, like the advection scheme, CUDA C kernels without using the mentioned above libraries have been written together with their serial equivalents for performance comparisons. The main reason of applying the ready-to-use numerical libraries is the fact that they simplify maintaining a clearly vector-oriented code structure and that they can be easily adapted to other programming language than the hardware-specific CUDA C by just replacing the kernels for basic operations.

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, the whole time loop is executed solely on the GPU with an exception of input/output: results must be transferred from GPU to CPU, because only CPU can read/write files. The issue of the necessity of the costly data transfer from the device to the host for the output can presently be solved e.g. by applying CUDA streaming technology, allowing hiding host-device transfer latencies by overlapping of computing and data transfer and using the host for post-processing the data for output. However, I/O issues are less relevant for a study concerning a computational core, which is supposed to be embedded in an industrial modelling software. Additionally, the data transfer latencies should be alleviated by the emerging APU (*Accelerated Processing Unit*) technology, merging CPU and GPU on one chip, which would also remove the necessity of the specific management of the separate host and device memories. As mentioned above, the key

issue is that for both GPU and APU as well as SIMD-processors of present CPUs the vectorisation of the code is crucial for performance.

Due to the application of the vector-oriented THRUST and CUSP libraries and introduction of the compact CUDA kernels, the structure of the code is thoroughly different compared to the reference serial code: virtually every loop concerning data vectors changed, there are separate allocations for the host and device and explicit data transfers device-host appeared. The resulting code resembles legacy codes for vector computers applying intensively standard mathematical libraries and/or special routines guaranteeing the correct vectorisation of the code, like e.g. TELEMAC [21]. It must be noted that the vectorisation of the code using THRUST and CUSP libraries has brought a slow-down in the purely serial execution by about 10% and in specific cases even up to 20% (clearly because of CPU cache misses) – this is the price of re-writing the code for stream processors, which introduces data structures inadequate for a cache-oriented CPU.

4 Results

4.1 Verification

The verification and validation of the algorithm with in its different implementations has been performed with numerous test cases, presented e.g. in references [2, 3, 6, 11, 14]. For the sake of completeness, a clearly non-linear verification test case based on the analytical solution presented by Thacker [22] is considered here, concerning a periodic motion of a circular shallow water body, 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, a periodic motion develops, with a period of $T = 2\pi/\omega$, where ω is the frequency of the ongoing motion. 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 de-nominating as h_0 the water depth in the middle of the body, as L the radius of the equilibrium shoreline and as η_0 the initial displacement of the bulged free surface in the middle of the domain, the equations of motion can be derived as follows [22]:

$$\begin{aligned}\eta(x, y, t) &= h_0 \left\{ \frac{\sqrt{1-A^2}}{1-A \cos \omega t} - 1 - \frac{x^2 + y^2}{L^2} \left[\frac{1-A^2}{(1-A \cos \omega)^2} - 1 \right] \right\} \\ u(x, y, t) &= \frac{1}{1-A \cos \omega t} \left(\frac{1}{2} \omega x A \sin \omega t \right) \\ v(x, y, t) &= \frac{1}{1-A \cos \omega t} \left(\frac{1}{2} \omega y A \sin \omega t \right)\end{aligned}\tag{24}$$

where the frequency of the motion ω and the parameter A are:

$$\omega = \frac{\sqrt{8gh_0}}{L}, \quad A = \frac{(h_0 + \eta_0)^2 - h_0^2}{(h_0 + \eta_0)^2 + h_0^2}\tag{25}$$

For the verification purposes, the geometrical parameters have been chosen to $L = 4.0$, $h_0 = 0.2\text{m}$, $\eta_0 = 0.01\text{m}$. The square computational domain measures $10\text{m} \times 10\text{m}$, with both co-ordinates x, y varying between -5m and $+5\text{m}$, with the resolution 2000×2000 one obtains $\Delta x = \Delta y = 0.005\text{m}$. For this set of parameters, the resulting period of motion is $T = 6.34\text{s}$ and computation is performed with the time step $\Delta t = 0.005\text{s}$ for 50s , i.e. for over 7 periods of motion. The case is inviscid, $\nu = 0$ and without roughness, $\gamma = \gamma_T = 0$. The conjugate gradient solver accuracy is set to 10^{-10} and for the Newton iterations 10^{-12} , resulting in 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 time series for two de-central positions at $(x, y) = (-2, 0)\text{m}$, $(-1, 0)\text{m}$ and the central position $(x, y) = (0, 0)\text{m}$, where also the analytical solution (24) for η 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 $\theta = 0.65$ and the imperfection in representing the axisymmetric case (initial and boundary conditions, circular shoreline) using a mesh of rectangles.

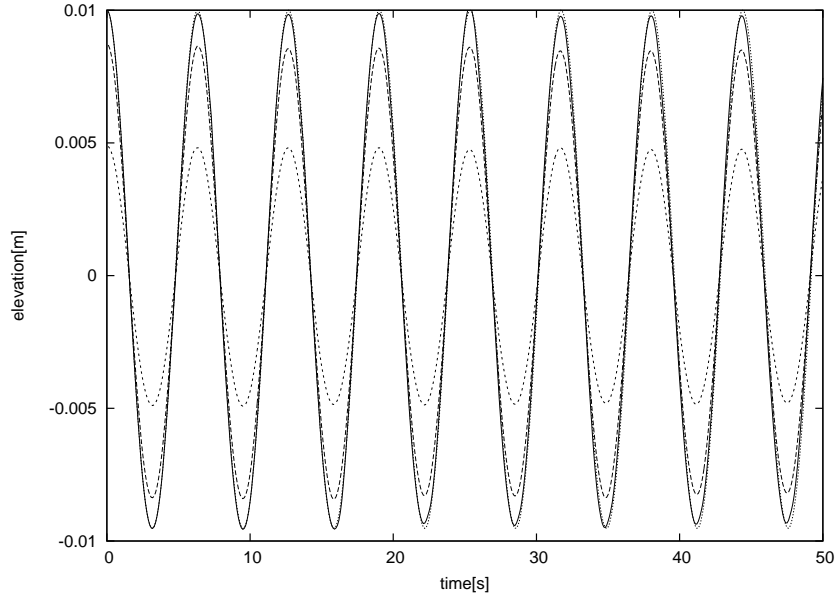


Figure 1: The elevation changes in three points of the domain, for $y = 0\text{m}$ and two de-central positions $x = -2\text{m}$, and $x = -1\text{m}$ (dashed lines) and the central position with 0m , solid line. For the central position $(x, y) = (0, 0)\text{m}$ the analytical solution is plotted for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

4.2 Performance

The computer system applied for performance testing is a good quality, but mass-production PC equipped with a 4-core Intel Xeon CPU X5570 working at 2.93GHz with 8MB cache in tandem with a off-the-shelf commodity graphic card GeForce GTX 480 working at 1.4 GHz, with 1.5GB memory, 15 multiprocessors and 480 stream processing units, having CUDA compute capability 2.0. This graphic card, featuring the Nvidia Fermi chip, allows computing in double precision without larger performance penalties compared to the single precision execution which were typical for the predecessor Nvidia GPUs (e.g. GT200 series).

Because of the interest on the performance of the computational kernel only, the reached speedups are given only for the time loop part of the algorithm presented in section 3.1 and without output. The comparisons ignore therefore also the GPU initialisation time, for this type up to ca. 1.5s. By ca. 1.5GB GPU memory of the given GPU type, the resolution up to ca. 4.5 million mesh cells in double precision and 9.5 million cells in single precision is possible.

The results are presented in Table 1 concern speedups reached applying four test cases, all of them relatively simple and schematic, but taking into account the typical features of the given scheme: (1) *Nonlin* – the test case described above in the section 4.1, but computed implicitly and with 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 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 $10\text{m} \times 10\text{m}$ with water depths for the cases (2-4) in the range of 1m ; the time step Δt varies between 0.001s and 0.005s ; 100 time steps are executed. The resolution $\Delta x = \Delta y$ can be deduced from the Table 1, it varies between 0.0033m and 0.01m . 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 $\nu_x = \nu_y = 10^{-4}\text{m}^2/\text{s}$ are applied.

The reached speedups for the time loop are consistently $O(20)$ or $O(30)$ for the execution applying double and single precision floating point numbers, respectively, compared to the execution of the same code on one of the CPU cores. The speedups $O(40)$ for the *waves* test case are ignored due to the fact that the serial reference case performed sub-optimally due to cache misses of the vectorised code on the CPU.

Table 1: Performance of the 2Dxy scheme

<i>example</i>	<i>precision</i>	<i>resolution</i>	<i>speedup</i>
(1) <i>Nonlin</i>	<i>single</i>	1000 × 1000	27
		2000 × 2000	29
		3000 × 3000	28
(1) <i>Nonlin</i>	<i>double</i>	1000 × 1000	21
		2000 × 2000	20
(2) <i>Sloshing</i>	<i>single</i>	1000 × 1000	32
		2000 × 2000	32
		3000 × 3000	30
(2) <i>Sloshing</i>	<i>double</i>	1000 × 1000	23
		2000 × 2000	22
(3) <i>Lake</i>	<i>single</i>	1000 × 1000	32
		2000 × 2000	34
		3000 × 3000	33
(3) <i>Lake</i>	<i>double</i>	1000 × 1000	24
		2000 × 2000	23
(4) <i>Waves</i>	<i>single</i>	1000 × 1000	30
		2000 × 2000	43
		3000 × 3000	42
(4) <i>Waves</i>	<i>double</i>	1000 × 1000	20
		2000 × 2000	20

5 Conclusions and outlook

5.1 Conclusions

It is demonstrated that a successful ground-up adaptation of a non-trivial two-dimensional numerical scheme for free surface flows for the streaming processors (GPU) is possible with a relatively small effort, even if one relies for the performance on the ready-to use numerical libraries. The critical issue is to adapt not only 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 assembling of matrices, solution of a linear equation inside outer, non-linear iterations set as well as all necessary depth and velocity updates. The considerable speedup (in the order of 20 or 30 for the double or single precision, respectively) compared to a single CPU core and the fact, that for this simple computational scheme computations using a few millions of cells on a single, commodity GPU are possible, opens a clear potential to increase the time and space resolution applying standard, off-the-shelf hardware. However, for this purpose, the algorithm adaptation radically changes the code structure towards a vector-oriented one, which brings some drawbacks for the serial execution of the same code. This confirms the well-known fact that a thorough re-programming of crucial parts of legacy codes is necessary for performance on the GPU only. On the other hand, it is believed that this vector-oriented programming method – also with application of numerical libraries – is sustainable enough in the light of the upcoming new computer architectures, where the performance is to be searched not only in the massive parallelism in terms of the processor unit numbers, but also in the hardware-acceleration of vector operations. The remaining issues concerning the specific software implementation as the programming paradigm evolves should not be critically large for a small, compact computational kernel.

5.2 Outlook

The promising results for the vertically integrated 2D scheme suggest that an extension of this methodology to three dimensions [2, 5] should also bring good results. With a specific pre-sorting of the mesh adequate for the streaming processor structure [19], the adaptation of the equivalent unstructured scheme [9] should be feasible. The developed code is a useful and ready-to-use tool for testing implementations of further developments, like the very promising sub grid methods [13, 14] or conservative advection schemes [15, 16]. The aimed application area of the resulting codes are studies concerning flooding and drying in highly resolved larger domains, requiring excellent numerical efficiency while maintaining the detail.

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 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* 1990; **86**: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* 1992; **15**:629–648.
- [3] Cheng R, Casulli V, Gartner J. Tidal, Residual, Intertidal Mudflat (TRIM) model and its applications to San Francisco Bay, California. *Estuarine, Coastal and Shelf Science* 1993; **36**:235–280.
- [4] Casulli V, Cattani E. Stability, accuracy and efficiency of a semi-implicit method for three-dimensional shallow water flow. *Computers Math. Applic.* 1994; **27**(4):99–112.
- [5] Casulli V. A semi-implicit finite difference method for non-hydrostatic, free surface flows. *International Journal for Numerical Methods in Fluids* 1999; **30**:425–440.
- [6] Bundesanstalt für Wasserbau (BAW). Editor: Lang G. *HN-Verfahren TRIM-2D. Validierungsdokument, Version 2.0*. Technical report, Bundesanstalt für Wasserbau, Hamburg, 1998.
- [7] Deußfeld N. *Ein Lagrange-Verfahren 2.Ordnung zur Large-Eddy Simulation*. Diploma work, Bundesanstalt für Wasserbau (BAW), Hamburg, 1999.
- [8] Casulli V, Walters R. An unstructured grid, three dimensional model based on the shallow water equations. *International Journal for Numerical Methods in Fluids* 2000; **32**:331–348.
- [9] Casulli V, Zanolli P. Semi-implicit numerical modelling of non-hydrostatic free-surface flows for environmental problems. *Mathematical and Computer Modelling* 2002; **36**:1131–1149.
- [10] Casulli V, Lang G. *Mathematical model UnTRIM, validation document, Version 1.0*. Technical report, Bundesanstalt für Wasserbau (BAW), Hamburg, 2004.
- [11] Casulli V, Zanolli P. Comparing analytical and numerical solution of nonlinear two and three-dimensional hydrostatic flows *International Journal for Numerical Methods in Fluids* 2007; **53**:1049–1062.
- [12] Jankowski J.A. Parallel implementation of a non-hydrostatic model for free surface flows with semi-Lagrangian advection treatment. *International Journal for Numerical Methods in Fluids* 2009; **59**:1157–1179. doi: 10.1002/fld.1859
- [13] Casulli V. A high-resolution wetting and drying algorithm for free-surface hydrodynamics. *International Journal for Numerical Methods in Fluids* 2009; **60**:391–408. doi: 10.1002/fld.1896.
- [14] Casulli V, Stelling G.S. Semi-implicit subgrid modelling of three-dimensional free-surface flows. *International Journal for Numerical Methods in Fluids* 2010. **67**:441–449. doi: 10.1002/fld.2361.
- [15] Stelling G.S, Duinmeijer S.P.A. A staggered conservative scheme for every Froude number in rapidly varied shallow water flows. *International Journal for Numerical Methods in Fluids* 2003. **43**:1329–1354. doi: 10.1002/fld.537
- [16] Kramer S.C, Stelling G.S. A conservative unstructured scheme for rapidly varied flows. *International Journal for Numerical Methods in Fluids* 2008. **58**:183–212. doi: 10.1002/fld.1722
- [17] Griebel M, Zaspel P. A multi-GPU accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations *Computer Science - Research and Development*, 2010. **25**(1–2):65–73. doi: 10.1007/s00450-010-0111-7

- [18] Corrigan A, Camelli F.F, Löhner R, Mut F. Semi-automatic porting of a large-scale Fortran CFD code to GPUs. *International Journal for Numerical Methods in Fluids* 2011. Published online, doi: 10.1002/flid.2560
- [19] Corrigan A, Camelli F.F, Löhner R, Wallin, J. Running unstructured grid-based CFD solvers on modern graphic hardware. *International Journal for Numerical Methods in Fluids* 2010; **66**:221—229. doi: 10.1002/flid.2254
- [20] Cohen J, Molemaker M.J. A Fast Double Precision CFD Code Using CUDA. In: Proceedings of 21st International Conference on Parallel Computational Fluid Dynamics, Moffett Field, California, USA, May 2009.
- [21] Hervouet J-M. *Hydrodynamic of Free Surface Flows. Modelling with the finite element method*. 2007, John Wiley & Sons, Chichester, 341pp.
- [22] Thacker WC. Some exact solutions to the non-linear shallow water equations. *Journal of Fluid Mechanics* 1981; **107**:499–508.
- [23] Bell N, Garland M. CUSP: Generic Parallel Algorithms for Sparse Matrix and Graph Computations. Version 0.1.2, 2010, <http://code.google.com/p/cusp-library/>.
- [24] Hoberock J, Bell N. THRUST: A Parallel Template Library. Version 1.3.0, 2010. <http://code.google.com/p/thrust/>.
- [25] NVIDIA Corporation. *NVIDIA CUDA C Programming Guide*, Version 4.0, 2011. Available from: <http://developer.nvidia.com/nvidia-gpu-computing-documentation>.
- [26] Khronos OpenCL Working Group. Editor: Munshi A. *The OpenCL Specification. Version 1.0*, 2008. Available from: <http://www.khronos.org/opencl/>.
- [27] The Portland Group. *PGI Accelerator Programming Model for Fortran & C, Version 1.3*, 2010. Available from: <http://www.pgroup.com/resources/accel.htm>
- [28] OpenMP Architecture Review Board. *OpenMP Application Program Interface. Version 3.0*, 2008. Available from: <http://www.openmp.org/>.
- [29] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*. Version 2.2, September 4, 2009. University of Tennessee, Knoxville. Available from: <http://www.mpi-forum.org/docs/docs.html>.