Title: Lotic Water Hydodynamic Model

Author(s): Judi, David Ryan
Tasseff, Byron Alexander

Intended for: Report

Issued: 2015-01-23

Lotic Water Hydrodynamic Model

The Department of Homeland Security sponsored the production of this material under the Department of Energy contract for the management and operation of Los Alamos National Laboratory.

Table of Contents

Table of Figures

Introduction

Water-related natural disasters, for example, floods and droughts, are among the most frequent and costly natural hazards, both socially and economically. Many of these floods are a result of excess rainfall collecting in streams and rivers, and subsequently overtopping banks and flowing overland into urban environments. Floods can cause physical damage to critical infrastructure and present health risks through the spread of waterborne diseases. Los Alamos National Laboratory (LANL) has developed Lotic, a state-of-the-art surface water hydrodynamic model, to simulate propagation of flood waves originating from a variety of events. Lotic is a two-dimensional (2D) flood model that has been used primarily for simulations in which overland water flows are characterized by movement in two dimensions, such as flood waves expected from rainfall-runoff events, storm surge, and tsunamis. In 2013, LANL developers enhanced Lotic through several development efforts. These developments included enhancements to the 2D simulation engine, including numerical formulation, computational efficiency developments, and visualization. Stakeholders can use simulation results to estimate infrastructure damage and cascading consequences within other sets of infrastructure, as well as to inform the development of flood mitigation strategies.

Flood Modeling and Simulation Enhancements

In 2008, to meet the objectives of fast-response flood modeling and simulation, LANL developed a 2D flood model using a numerical approximation to the non-conservative shallow water equations (Judi, Burian and McPherson 2009). The shallow water equations are derived from the Navier-Stokes equations by integrating horizontal momentum and continuity equations over depth. With bed slope and bed shear stress (i.e., friction) source terms, the non-conservative form of the equations may be written as shown in Equation 1:

$$\begin{cases} h_t + (hu)_x + (hv)_y = R \\ u_t + uu_x + vu_y = -gw_x - \frac{\tau^x}{\rho}, \\ v_t + uv_x + vv_y = -gw_y - \frac{\tau^x}{\rho} \end{cases} \qquad (1)$$

where h is the water depth, w is the water surface elevation, u is the velocity in the x-direction, v is the velocity in the y-direction, t is time, g is the gravitational constant, R is the water source term, τ^x and τ^y are the bed shear stress components in the x- and y-directions, and ρ is the water density. The numerical approximation used to solve these equations was a first-order upwind finite difference method.

Because velocities in the non-conservative form of the shallow water equations are not subject to universal conservation laws, the model can develop spurious oscillations when subjected to shocks and hydraulic jumps. However, this formulation is subject to numerical challenges when simulating nuanced floodplains (e.g., urban areas). Thus, LANL revised the numerical implementation using new and emerging algorithms based on the conservative form of the shallow water equations.

Numerical Reformulation of the Shallow Water Equations

With bed slope and bed shear stress source terms, the conservative form of the shallow water equations may be defined as shown in Equation 2:

$$\begin{cases} h_t + (hu)_x + (hv)_y = R \\ (hu)_t + \left(hu^2 + \frac{1}{2}gh^2\right)_x + (huv)_y = -ghB_x - \frac{\tau^x}{\rho} \\ (hv)_t + (huv)_x + \left(hv^2 + \frac{1}{2}gh^2\right)_y = -ghB_y - \frac{\tau^x}{\rho} \end{cases}, \qquad (2)$$

where B describes the bottom topography (or bathymetry) (Chertock, et al. 2010). In vector form, this may be written more succinctly in Equation 3:

$$\boldsymbol{U}_t + \boldsymbol{F}(\boldsymbol{U}) + \boldsymbol{G}(\boldsymbol{U}) = \boldsymbol{S}_B(\boldsymbol{U}, \nabla B) + \boldsymbol{S}_f(\boldsymbol{U}), \qquad (3)$$

where $\boldsymbol{U}$ is the vector of conserved variables, $\boldsymbol{F}$ and $\boldsymbol{G}$ are fluxes in the x- and y-directions, respectively, and $\boldsymbol{S}_B$ and $\boldsymbol{S}_f$ are the bed slope and bed shear stress source terms, respectively (Brodtkorb, Sætra and Altinakar 2012).

To numerically approximate the conservative form of the shallow water equations, developers employed an explicit finite difference scheme outlined by Kurganov and Petrova (Kurganov and Petrova 2007). The scheme has several favorable qualities, including the preservation of stationary steady states (e.g., the “lake at rest”), the guarantee of non-negative water depths, and the ability to simulate domains with discontinuous bottom bathymetries.

Numerical schemes that preserve steady-state solutions are regarded as “well-balanced.” Developing such a scheme is difficult when using the vector of conserved variables $\boldsymbol{U} = [h, hu, hv]^T$ but manageable when $\boldsymbol{U} = [w, hu, hv]^T$, in which $w = h + B$. The conservative shallow water equations may thus be rewritten as in Equation 4:

$$\begin{cases} w_t + (hu)_x + (hv)_y = R \\ (hu)_t + \left(\frac{(hu)^2}{w-B} + \frac{1}{2}g(w-B)^2\right)_x + \left(\frac{(hu)(hv)}{w-B}\right)_y = -g(w-B)B_x - \frac{\tau^x}{\rho} \\ (hv)_t + \left(\frac{(hu)(hv)}{w-B}\right)_x + \left(\frac{(hv)^2}{w-B} + \frac{1}{2}g(w-B)^2\right)_y = -g(w-B)B_y - \frac{\tau^x}{\rho} \end{cases} \qquad (4)$$

The spatial discretization of the Kurganov-Petrova scheme is based on a staggered grid, where $\boldsymbol{U}$ is given as cell averages, B is an averaged surface defined by bathymetric (or topographic) values at the four cell corners, and fluxes are computed at midpoints of cell interfaces (Brodtkorb, Sætra and Altinakar 2012). The scheme also uses interfacial bathymetric heights; for a given cell center C_{ij}, these data are computed as shown in Equations 5 through 8.

$$B_{ij}^{E} = \frac{1}{2}\left(B_{i+\frac{1}{2},j+\frac{1}{2}} + B_{i+\frac{1}{2},j-\frac{1}{2}}\right) \qquad (5)$$

$$B_{ij}^{W} = \frac{1}{2}\left(B_{i-\frac{1}{2},j+\frac{1}{2}} + B_{i-\frac{1}{2},j-\frac{1}{2}}\right) \qquad (6)$$

$$B_{ij}^{N} = \frac{1}{2}\left(B_{i+\frac{1}{2},j+\frac{1}{2}} + B_{i-\frac{1}{2},j+\frac{1}{2}}\right) \qquad (7)$$

$$B_{ij}^{S} = \frac{1}{2}\left(B_{i+\frac{1}{2},j-\frac{1}{2}} + B_{i-\frac{1}{2},j-\frac{1}{2}}\right) \qquad (8)$$

where superscripts E, W, N, and S define cell interfaces to the east, west, north, and south of C_{ij}'s center.

Reconstructions of $\boldsymbol{U}$ must first be performed at grid cell interfaces. These reconstructions are shown in Equations 9 through 12.

$$\boldsymbol{U}_{ij}^{E} = \boldsymbol{U}_{ij} + \frac{\Delta x}{2}(\boldsymbol{U}_x)_{ij} \qquad (9)$$

$$\boldsymbol{U}_{ij}^{W} = \boldsymbol{U}_{ij} - \frac{\Delta x}{2}(\boldsymbol{U}_x)_{ij} \qquad (10)$$

$$\boldsymbol{U}_{ij}^{N} = \boldsymbol{U}_{ij} + \frac{\Delta y}{2}\left(\boldsymbol{U}_y\right)_{ij} \qquad (11)$$

$$\boldsymbol{U}_{ij}^{S} = \boldsymbol{U}_{ij} - \frac{\Delta y}{2}\left(\boldsymbol{U}_y\right)_{ij} \qquad (12)$$

The numerical derivatives $\boldsymbol{U}_x$ and $\boldsymbol{U}_y$ are at least first-order approximations of $\boldsymbol{U}_x(i,j,t)$ and $\boldsymbol{U}_y(i,j,t)$, respectively, and are computed using a nonlinear limiter. In this scheme, slopes are computed using the generalized minmod flux limiter, defined by Equations 13 and 14:

$$U_z = \text{minmod}(\theta f, c, \theta b) \qquad (13)$$

$$\text{minmod}(a,b,c) = \begin{cases} \min(a,b,c), & \{a,b,c\} > 0 \\ \max(a,b,c) & \{a,b,c\} < 0 \\ & 0 \end{cases} \qquad (14)$$

where $\theta = 1.3$, and f, c, and b are the forward, central, and backward difference approximations to the derivative, respectively (Brodtkorb, Sætra and Altinakar 2012). However, this reconstruction does not guarantee non-negativity of h. To handle dry zones, Kurganov and Petrova propose to correct the slopes of w, such that values of h at integration points become non-negative. These corrections are made in the following four cases, shown in Equations 15 through 18:

$$w_{ij}^{E} < B_{ij}^{E} \rightarrow w_{ij}^{E} = B_{ij}^{E}, \quad w_{ij}^{W} = 2\overline{w}_{ij} - B_{ij}^{E} \qquad (15)$$

$$w_{ij}^{W} < B_{ij}^{W} \rightarrow w_{ij}^{W} = B_{ij}^{W}, \quad w_{ij}^{E} = 2\overline{w}_{ij} - B_{ij}^{W} \tag{16}$$

$$w_{ij}^{N} < B_{ij}^{N} \rightarrow w_{ij}^{N} = B_{ij}^{N}, \quad w_{ij}^{S} = 2\overline{w}_{ij} - B_{ij}^{N} \tag{17}$$

$$w_{ij}^{S} < B_{ij}^{S} \rightarrow w_{ij}^{S} = B_{ij}^{S}, \quad w_{ij}^{N} = 2\overline{w}_{ij} - B_{ij}^{S} \tag{18}$$

where the average slope is defined by Equation 19.

$$\overline{w}_{ij} = \frac{w_{ij}^{E}+w_{ij}^{W}}{2} = \frac{w_{ij}^{N}+w_{ij}^{S}}{2} \tag{19}$$

This correction guarantees the reconstruction of w is conservative; hence, the water depth, h, will be non-negative. However, the values of h may still be very small (or zero). Thus, computing velocities (e.g., $u = (hu)/h$) could result in round-off errors as h approaches zero, leading to erroneously large velocities. To avoid this, Chertock et al. provided a velocity correction for $h < \kappa$, shown in Equation 20:

$$u = \frac{2(hu)}{h^2+\max(h^2,\kappa)} \tag{20}$$

with the (extrapolated) suggestion by Brodtkorb et al. for k shown in Equation 21:

$$\kappa = \sqrt{K_0 \max\{1, \min(\Delta x, \Delta y)\}} \tag{21}$$

with $K_0 = 10^{-2}$ for single-precision calculations. Even with these corrections, however, round-off errors may still lead to negative water depths. Thus, as an extra precaution, water depths less than ϵ_m are set to zero, where ϵ_m is near machine precision.

Using the newly constructed interfacial values of h, u, and v, vectors $\boldsymbol{U}^{E,W,N,S}$ are constructed accordingly. Finally, interfacial flux values (i.e., $\boldsymbol{F}(\boldsymbol{U})$ and $\boldsymbol{G}(\boldsymbol{U})$) are computed using the quantities defined within vectors $\boldsymbol{U}^{E,W,N,S}$.

A central-upwind semi-discretization of the conservative shallow water equations results in the system of ordinary differential equations (ODE) in Equation 22:

$$\frac{d}{dt}\boldsymbol{U}_{ij}(t) = -\frac{\boldsymbol{H}_{ij}^{E}(t) - \boldsymbol{H}_{i-1,j}^{E}(t)}{\Delta x} - \frac{\boldsymbol{H}_{ij}^{N}(t) - \boldsymbol{H}_{i,j-1}^{N}(t)}{\Delta y} + \boldsymbol{S}_{B,ij}(t) + \boldsymbol{S}_{f,ij}(t) \tag{22}$$

The numerical fluxes $\boldsymbol{H}_{ij}^{E}$ and $\boldsymbol{H}_{ij}^{N}$ are given by Equations 23 and 24:

$$\boldsymbol{H}_{ij}^{E} = \frac{a^{+}\boldsymbol{F}\left(\boldsymbol{U}_{ij}^{E}\right) - a^{-}\boldsymbol{F}\left(\boldsymbol{U}_{i+1,j}^{W}\right)}{a^{+} - a^{-}} + \frac{a^{+}a^{-}}{a^{+} - a^{-}}\left(\boldsymbol{U}_{i+1,j}^{W} - \boldsymbol{U}_{ij}^{E}\right) \qquad (23)$$

$$\boldsymbol{H}_{ij}^{N} = \frac{b^{+}\boldsymbol{G}\left(\boldsymbol{U}_{ij}^{N}\right) - b^{-}\boldsymbol{G}\left(\boldsymbol{U}_{i,j+1}^{S}\right)}{b^{+} - b^{-}} + \frac{b^{+}b^{-}}{b^{+} - b^{-}}\left(\boldsymbol{U}_{i,j+1}^{S} - \boldsymbol{U}_{ij}^{N}\right) \qquad (24)$$

Where a^{+}, a^{-}, b^{+}, b^{-} are described in Equations 25-28, respectively:

$$a^{+} = \max\left\{u_{ij}^{E} + \sqrt{gh_{ij}^{E}}, u_{i+1,j}^{W} + \sqrt{gh_{i+1,j}^{W}}\right\} \qquad (25)$$

$$a^{-} = \min\left\{u_{ij}^{E} - \sqrt{gh_{ij}^{E}}, u_{i+1,j}^{W} - \sqrt{gh_{i+1,j}^{W}}\right\} \qquad (26)$$

$$b^{+} = \max\left\{v_{ij}^{N} + \sqrt{gh_{ij}^{N}}, v_{i,j+1}^{S} + \sqrt{gh_{i,j+1}^{S}}\right\} \qquad (27)$$

$$b^{-} = \min\left\{v_{ij}^{N} - \sqrt{gh_{ij}^{N}}, v_{i,j+1}^{S} - \sqrt{gh_{i,j+1}^{S}}\right\} \qquad (28)$$

Upon obtaining $\frac{d}{dt}\boldsymbol{U}_{ij}(t)$, the system of ODEs may be integrated with an ODE solver of desired order. Brodtkorb et al. suggest using a standard second-order total variation diminishing Runge-Kutta integrator, shown in Equation 29 and 30:

$$\boldsymbol{U}_{ij}^{*} = U_{ij}^{n} + \Delta t\left(\frac{d}{dt}\boldsymbol{U}_{ij}^{n}\right) \qquad (29)$$

$$\boldsymbol{U}_{ij}^{n+1} = \frac{1}{2}U_{ij}^{n} + \frac{1}{2}\left[\boldsymbol{U}_{ij}^{*} + \Delta t\left(\frac{d}{dt}\boldsymbol{U}_{ij}^{*}\right)\right] \qquad (30)$$

where the time step, Δt, is computed based on the Courant–Friedrichs–Lewy condition in Equation 31:

$$\Delta t = \min\left\{\frac{\Delta x}{4a}, \frac{\Delta y}{4b}\right\} \qquad (31)$$

where a and b are defined in Equation 32, respectively:

$$a = \max\left\{a^{+}{}_{j+\frac{1}{2},k}, -a_{j+\frac{1}{2},k}^{-}\right\} \qquad (32)$$

$$b = \max\left\{b^{+}{}_{j,k+\frac{1}{2}}, -b_{j,k+\frac{1}{2}}^{-}\right\} \qquad (33)$$

where a and b are global maxima over the entire domain (Chertock, et al. 2010).

Alternatively, an Euler integrator may be used by using Equation 34:

$$\boldsymbol{U}_{ij}^{n+1} = U_{ij}^{n} + \Delta t \left(\frac{d}{dt} \boldsymbol{U}_{ij}^{n} \right) \qquad (34)$$

The previously described equations were implemented on traditional CPU hardware utilizing shared memory and parallel computing, and to the GPU. The implementation was benchmarked against previously used case studies and will be described briefly in a later section.

Domain Tracking on the GPU

Although parallel computation greatly increases the efficiency of the shallow water model, additional considerations can further reduce model runtime. In real-world scenarios (e.g., dam breaches), many cells within a domain never become "wet." Computations in these dry regions can thus be neglected, as their solutions are known a priori. Tracking wet and dry cells focuses only on important computations (Judi, Burian and McPherson 2011).

The CPU implementation of the shallow water model was adjusted to only perform computations on wet cells and dry cells adjacent to wet cells. To map this concept to the GPU, the hierarchy of NVIDIA's Compute Unified Device Architecture (CUDA) architecture was exploited as per recommendations of Sætra (Sætra 2013). Understanding this implementation, however, requires a general understanding of the GPU implementation.

CUDA's computational hierarchy consists of three major components, shown pictorially in Figure 1. To summarize these components, the CUDA programming guide is quoted (NVIDA 2014):

> *"The CUDA architecture is built around a scalable array of multithreaded Streaming Multiprocessors (SMs). When a CUDA program on the host CPU invokes a kernel grid, the blocks of the grid are enumerated and distributed to multiprocessors with available execution capacity. The threads of a thread block execute concurrently on one multiprocessor, and multiple thread blocks can execute concurrently on one multiprocessor. As thread blocks terminate, new blocks are launched on the vacated multiprocessors."*

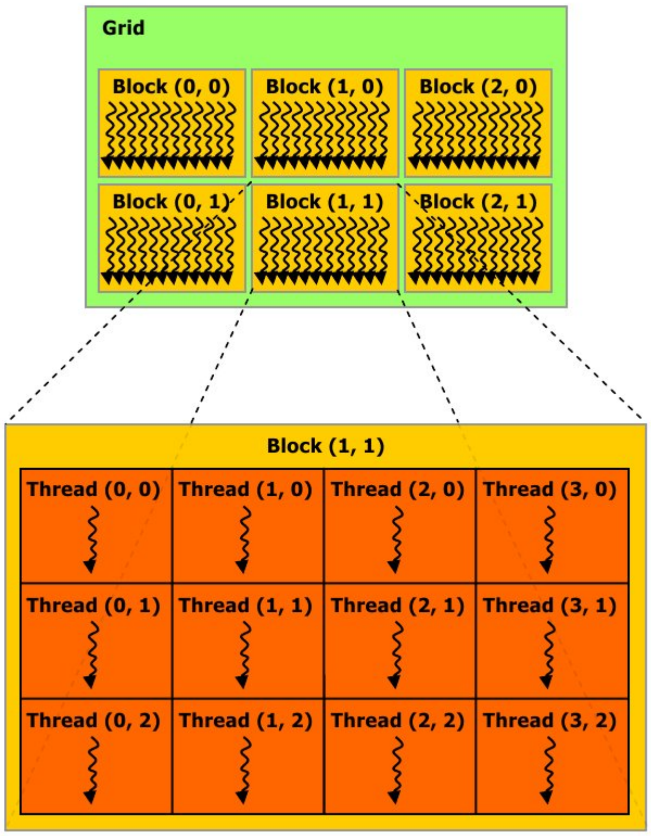

Figure 1. CUDA computational components.

In the case of the Kurganov-Petrova implementation described previously, each thread is assigned a grid cell, while each block is assigned 16 × 12 threads. Instead of tracking individual wet cells, wet blocks are tracked during each iteration. A block is marked as active if any individual cell within the block is wet, or if the block is adjacent to another "wet" block, shown visually in Figure 2. During each iteration, active blocks are launched by kernel functions that describe the numerical scheme. At the end of each iteration, the list and number of active blocks are updated for use during the next iteration.

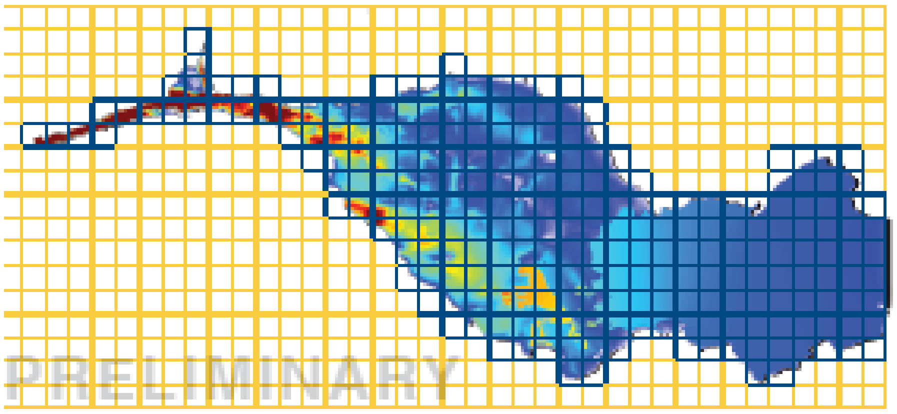

Figure 2. Pictorial description of CUDA blocks marked as active (blue) and inactive (yellow).

Verification and Computational Performance Benchmarking

A hypothetical circular dam break was used to benchmark the accuracy and performance of GPU implementations. This hypothetical simulation is commonly used to benchmark new shallow water equation simulations in the literature (Alcrudo and Garcia-Navarro 1993). The scenario is defined by a flat, square domain with dimensionality of 50 × 50 cells; each cell is 1 meter. A circular column of water lies at the center of this domain with a depth of 10 meters and a radius of 11 meters. The surrounding cells are initialized with a water depth of 1 meter. Under the influence of gravity, this column of water propagates in all directions, gradually encompassing the domain. The results for the simulation at *t* = 0.69 seconds are shown in Figure 3.

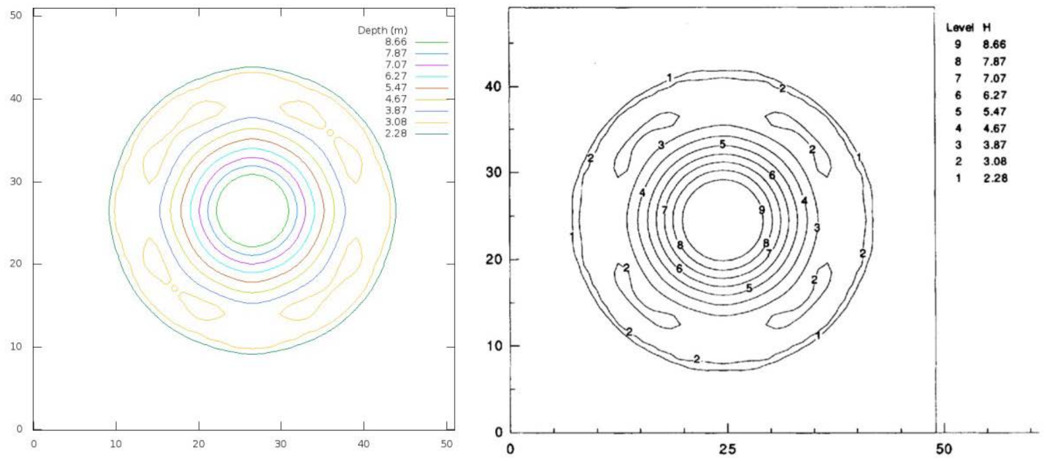

Figure 3. Comparison of water depths at *t* = 0.69 seconds between Lotic (left) and Alcrudo and Garcia-Navarro (right).

The hypothetical circular dam break was also used to benchmark computation performance with and without domain tracking, as well as the performance of a multi-core CPU implementation with domain tracking. To accomplish this, the domain was modified to increase the number of computational cells. The scenario is defined by a flat, square domain with dimensionality of 10,000 × 10,000 meters. A circular column of water lies at the center of this domain with a depth of 500 meters and a radius of 100 meters. To benchmark model implementations as a function of domain size, the cell size was altered as a function of the number of cells used to describe the domain. Performance was measured similarly to Brodtkorb, *S*ætra, and Altinakar, in units of megacells per second. To produce these measurements, each simulation was run with 10,000 time-steps, and the execution time of each simulation was recorded. The CPU implementation used a multithreaded OpenMP code, using 12 Intel Xeon x5670 cores, each at 2.93 gigahertz (GHz). The GPU implementation employed one NVIDIA Tesla C2050, containing 448 CUDA cores, each at 1.15 GHz. In all cases, boundary condition updates were inactive. Figure 4 shows the results of these performance comparisons.

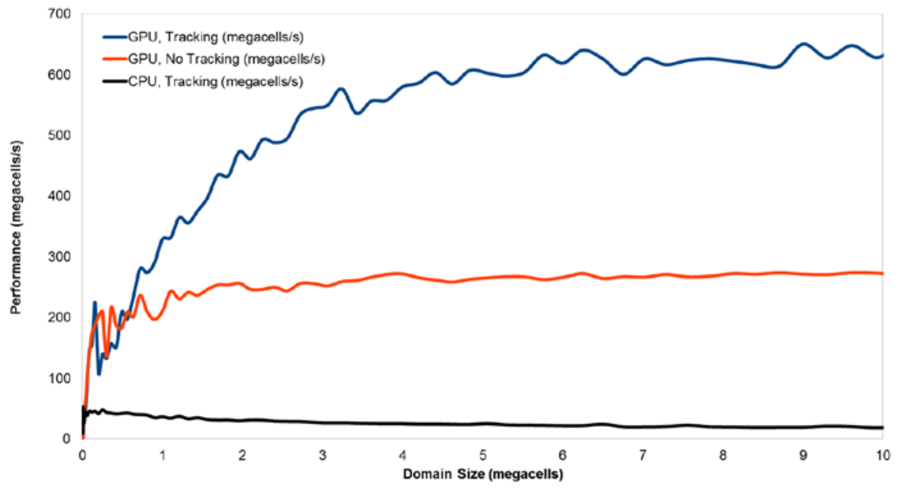

Figure 4. Performance comparison of CPU and GPU implementations with and without domain tracking as a function of domain size.

Clearly, both GPU implementations outperform the CPU implementation with domain tracking. In this example, computational performance is increased an order of magnitude. Comparing GPU implementations, a speedup of near 3 is observed when comparing the domain tracking implementation with the non-tracking implementation.[1] Additional speedup is anticipated using the domain tracking algorithm as overhead involved in wet cell/block tracking is reduced.

Multi-GPU

As simulation scenarios increase in size and duration, memory and performance limitations may be quickly reached, which a concern when performing wide-area hydrologic simulations. As an example, the NVIDIA Tesla C2050s used in benchmark comparisons throughout this report contain three gigabytes of global on-device memory. For a minimal simulation (e.g., Euler integration and no hydrological modeling), this

[1] In the field of computer architecture, speedup is a metric for relative performance improvement when executing a task. The notion of speedup was established by Amdahl's law, which was particularly focused in the context of parallel processing.

provides a theoretical maximum grid size of roughly 65 million cells. If more memory is required for higher-order integration and/or hydrological modeling, this maximum size decreases substantially. However, in emergency-response scenarios, the simulation of very large domains is common. In previous emergency-response practice scenarios, domains were reduced in size or resolution was decreased to adhere to memory limitations of available GPUs.

By increasing the number of GPUs used for simulation from one to *N*, the domain size and execution time may be theoretically increased or decreased by factors of *N*, respectively. To implement this feature, a procedure similar to Sætra and Brodtkorb was followed. The domain is first decomposed into *N* equally sized portions. *N* CPU threads are assigned to control each GPU, and the model is run on each portion of the domain in parallel. Near the center of domain, four rows of data are overlapped for each GPU and subdomain. This overlap allows each GPU to iterate over its data twice before swapping interfacial data with the other GPU (Sætra and Brodtkorb 2012).

The current implementation has only been tested on dual-GPU instances. To benchmark single and dual-GPU instances, the hypothetical circular dam break was again used. These performance comparisons are displayed graphically in Figure 5.

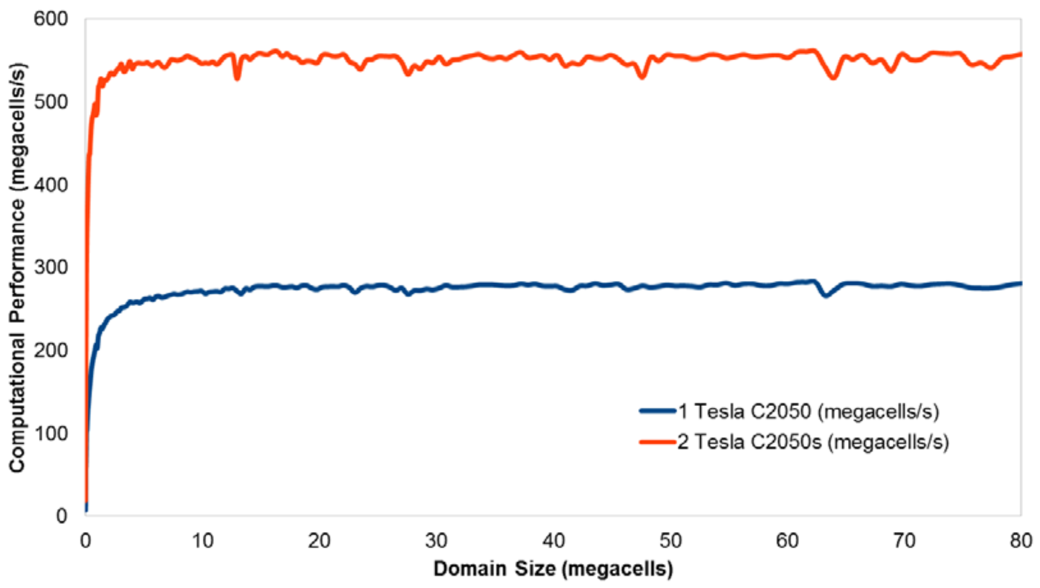

Figure 5. Single and dual-GPU computational performance benchmark for the hypothetical circular dam break.

In many cases, super linear speedup was observed while benchmarking the dual-GPU version. This may be a result of inefficient memory caching in GPU implementations. In the single-GPU case, the entire system must fit within one GPU; when the problem is split between two GPUs, it is possible that memory caching is handled more efficiently (or plays a lesser role) during execution. As these inefficiencies are further investigated and corrected, single and dual-GPU versions should both increase in performance, and dual- versus single-GPU speedups should approach a value near 2.

Enhanced Visualization

Real-time visualization has been incorporated into the GPU implementation of flood scenarios as the GPU model executes. In this instance, real-time is defined as instantaneous visualization of results as it is simulated. This visualization component is founded upon the work of Brodtkorb et al (Brodtkorb, Sætra and Altinakar 2012). Because model and visualization data both reside on the GPU, data are easily accessed by both components, allowing for fast simulation and immersive visualization. Terrains may also be draped with user-defined textures, such as high-resolution satellite imagery. Water surfaces are realistically rendered using the Fresnel equations. Extending the work of Brodtkorb et al., Lotic allows for interactive manipulation of the domain. Presently, analysts can alter the topography and add/remove water as the simulation runs. The ability to manipulate topography may be especially useful to emergency managers, who could use simulations to prepare for future flood events. Figure 6 shows an example of textures, rendering, and topography manipulation on a monitor.

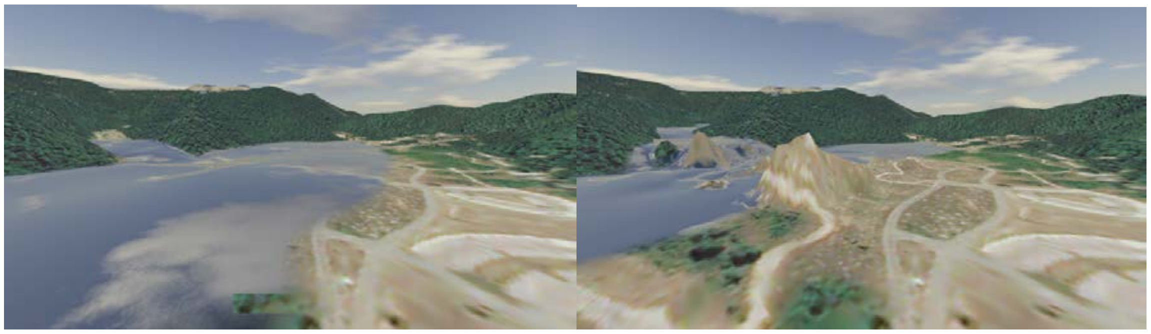

Figure 6. Real-time visualization of the historic Taum Sauk dam break before (left) and after (right) analysts' topography manipulation.

References

Alcrudo, Francisco, and Pilar Garcia-Navarro. "A high-resolution Godunov-type scheme in finite volumes for the 2D shallow-water equations." *International Journal for Numerical Methods in Fluids* 16, no. 6 (1993): 489–505.

Brodtkorb, Andre, Martin Sætra, and Mustafa Altinakar. "Efficient shallow water simulations of GPUs: Implementation, visualization, verification, and validation." *Computers and Fluides* 55 (2012): 1–12.

Chertock, Alina, Shumo Cui, Alexander Kurganov, and Tong Wu. "Well-balanced Positivity Preserving Central-Upwind Scheme for the Shallow Water System with Friction Terms." *J. Hydro* 382 (2010): 88–102.

Judi, David, S Burian, and T McPherson. "Two-Dimensional Fast-Response Flood Modeling: Desktop Parallel Computing and Domain Tracking." *Journal of Computing in Civil Engineering* 25 (2011): 184–191.

Judi, David, Steve Burian, and Timothy McPherson. *Development and validation of a two-dimensional fast-response flood estimation model.* LA-UR-09-01174, Los Alamos, NM: Los Alamos National Laboratory, 2009.

Kurganov, Alexander, and Guergana Petrova. "A second-order well-balanced positivity preservine central-upwind scheme for Saint-Venant system." *Communications in Mathematical Sciences* 5, no. 1 (2007): 133–160.

NVIDA. "CUDA C Programming Guide." 2014.

Sætra, Martin. "Shallow water simulation on GPUs for sparse domains." *Numerical Mathematics and Advanced Applications 2011* (Springer), 2013: 673–680.

Sætra, Martin, and Andre Brodtkolb. "Shallow water simulations on multiple GPUs." *Applied Parallel and Scientific Computing* (Springer), 2012: 56–66.