A GPU-based Flood Simulation Framework

Siddharth Shankar*, Alfred J. Kalyanapu†, Charles D. Hansen*, Steven J. Burian†
*Scientific Computing & Imaging Institute, University of Utah, Salt Lake City, Utah - 84112
†Department of Civil and Environmental Engineering, University of Utah, Salt Lake City, Utah - 84112

***Abstract*—We present a multi-core, GPU-based framework for simulation and visualization of two-dimensional floods, based on the full implementation of Saint Venant equations. A validated CPU-based flood model was converted to NVIDIA's CUDA architecture. The model was run on two different NVIDIA graphics cards, a GeForce 8400 GS and a Tesla T10. The model was tested using two case study applications. Implementing the GPU version increased the model performance with speedups ranging from 3X to 6X on GeForce 8400GS, and 50X to 135X on Tesla T10. The GPU version resulted in scalable performance on both the cards compared to the CPU version. The dam break flood event was reproduced and the simulation run time reduced from 2.9 hours to 2 minutes.**

***Index Terms*—Multi-core; CUDA; 2D flood simulation;**

I. Introduction

Computer models have been applied to simulate floods since a long time ([1]). Floods are typically modeled as free surface flows, using one-dimensional (1D) dynamic wave simulations, also known as *Saint Venant* equations ([2]). Even though 1D models are common in flood modeling, the hypothesis of 1D models is not always valid, especially when applied to floodplain flows. They are unable to simulate the lateral diffusion of flood wave, and the discretization of the topography as cross-sections, as opposed to a surface, causing uncertainties for sinuous channels. Moreover, the flood inundation extraction involves the interpolation of water surface elevations to an inundation surface, which is subjective ([3]). These limitations can be overcome with two-dimensional (2D) models, and various numerical schemes have been developed in response ([4], [2], [5], [6]). Applying 2D models enables higher order topographic representation in the simulations. Moreover, with the increasing availability of high resolution and high accuracy Digital Elevation Models (DEM) for floodplains areas, 2D models can be readily integrated with such data sources. However, 2D models are computationally intensive, and need longer runtime to yield significant results.

These computational requirements call for the application of multi-core GPU computing to flood inundation modeling, as the problem fits into the category of finite differencing and would benefit from parallel computation on the GPU. Thus the main objective of this study is a multi-core GPU implementation of the 2D flood model, using the complete set of *Saint Venant* equations. A faster GPU-based model would speed up the model execution for a fixed domain size, or simulate large flood areas in real time.

II. Methodology

The 2D model uses non-linear hyperbolic [7], shallow wave equations (synonymous with *Saint Venant* equations), which are derived from the Navier-Stokes equations ([8]). This is achieved by integrating the continuity and the momentum equations over depth. A staggered grid stencil is used to define the computational domain, and boundary conditions are handled using ghost/ extra cells. The continuity and momentum equations are discretized explicitly to march ahead in time, however the finite difference approach may become unstable for larger increments of Δt. A necessary condition for stability of the explicit scheme, called $Courant$ condition, is applied and ensures numerical convergence ([9]).

III. Implementation

The computational domain is divided into a set of blocks, and the blocks are scheduled onto the Streaming Multiprocessors (SMP) cores. The division of the domain only considers the interior cells, as the exterior cells replicate interior cell values. A 2D shared memory array is allocated corresponding to every block in the domain, and has a layer of extra cells in both dimensions to account for ghosting. The cells in a shared memory block are populated using a method in [10], and ensures least divergence. At the end of a computational iteration, the updated cell values are written back into the global memory.

Updating the extra cells on the edges of the computational domain is carried out as a two-phase process. The first phase involves updating the two rows and then the next phase updates the two columns (or vice versa). The two phases are executed as two different kernel calls, and the block sizes for the kernels is equal to the size of the extra rows/ columns. This method of updating ensures that extra threads are not created, on creating larger blocks, and is illustrated in Figure 1.

Once the cell values are updated, a parallel reduction kernel is executed, which determines the maximum height and velocity values. The sorting method used has been taken from [11]. The maximum values are used by $Courant$ condition as inputs to find the next stable time increment, Δt. The size of the operational domain decreases with subsequent iterations, until it has reached a size of 1x1. This is illustrated in Figure 2.

IV. GPU Enhancements

Two GPU enhancements have been implemented. The first enhancement is a *16 cell padding* scheme. The computational domain is padded with a layer of 16 extra cells, in all four directions, thereby ensuring memory alignment, and coalesced

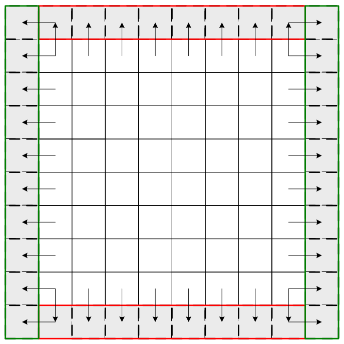

Fig. 1. Updating the extra rows/ columns, scheduled as two blocks onto the SMP cores.

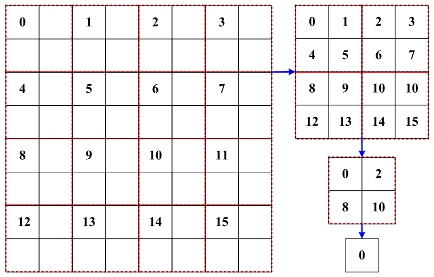

Fig. 2. Parallel reduction with a block size of 2x2.

memory calls by threads in a half-warp (32 threads make up a warp, 16 are executed as an atomic unit). On a device with low compute capability ([12]), as on the 8400 GS - which supports compute capability 1.1, this scheme ensures an atomic memory fetchup operation, thereby saving time on reading data from the global memory. In absence of such a scheme, all threads are executed in serial, and add to the execution time. The effect of this enhancement is highlighted in Figure 3.

The second enhancement comes into play during the execution of the parallel reduction method, and cause execution times for subsequent iterations to fall down drastically. In order to ensure that the block size does not exceed the new sub-array size, the block size is altered dynamically. The following two equations involve calculating the new sub-array and the block-size,

$$arrW = (arrW + blkW - 1)/blkW, \tag{1}$$

The second equation is solved, only if the sub-array width is lesser than the block width,

$$blkW = 2^{\log_2 (arrW\%blkW)} \tag{2}$$

where, $arrW$ and $blkW$ are the sub-array and the block size at any given iteration. The same set of equations are used for calculating the new height.

V. Results

For the purpose of testing and validating the GPU model, two real life case studies have been used. The first case study is of a flooding scenario due to rainfall, while the other one is a dam break event. Table I presents a comparative view of the spatial features of the two case studies.

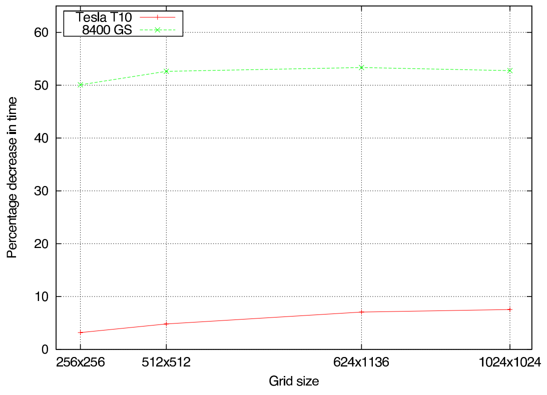

Fig. 3. Percentage decrease in execution times on the two GPUs from 1 cell to 16 cell padding.

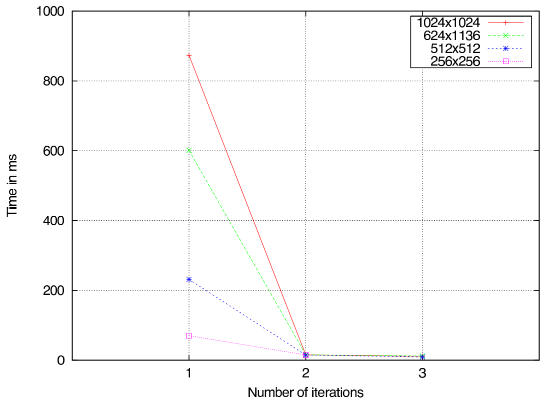

Fig. 4. Drop in execution time with subsequent iterations.

All the GPU-based tests were run on two different machine configurations. Table II presents these configurations. Both these machines use Linux 64-bit operating system. All our CPU tests ran on a 3.2GHz AMD Desktop, with 4 GB of memory. The CPU model uses a single core, and forms the baseline for measuring speedups.

For the purpose of benchmarking, the dataset corresponding to the first case study (256x256) was resampled, and 512x512 and 1024x1024 datasets were created. The dataset from the second case study, 624x1136, was used as it is. The simulations were run for 50,000, 100,000, and 150,000 numerical iterations.

Case Study	Greens Bayou	Taum Sauk
Spatial Res	25 metre(s)	10 metre(s)
Grid Size	256x256	624x1136
Area	41 sq. km(s)	71 sq. km(s)

TABLE I
A COMPARISON OF THE SPATIAL FEATURES OF THE TWO CASE STUDIES.

Figure 5 presents a comparison of the speedups from: 1. the CPU to the GPUs and, 2. between the GPUs. The gains obtained are indicative of the overall application, and not of a specific function or kernel. The speedup from the CPU to the 8400 GS ranges from 3X to 6X, and that from the CPU to

CPU Frequency	2.33 GHz	2.67 GHz
Memory	2 GB	24 GB
Graphics Card	8400 GS	Tesla T10
CUDA Cores	16	240

TABLE II
A COMPARISON OF THE CONFIGURATION OF THE MACHINES USED.

the Tesla is in the range of 50X to 135X. The GPU to GPU speedups are in the range of 12X to 25X, which lines well with the scaling up of the cores (15X). For the dam break event, the GPU model was able to simulate 25 minutes of flood simulation in 120 seconds on the Tesla, compared to 2 hours on the CPU. This makes the GPU model better than real time, and serves the needs of flood evacuation and emergency planning.

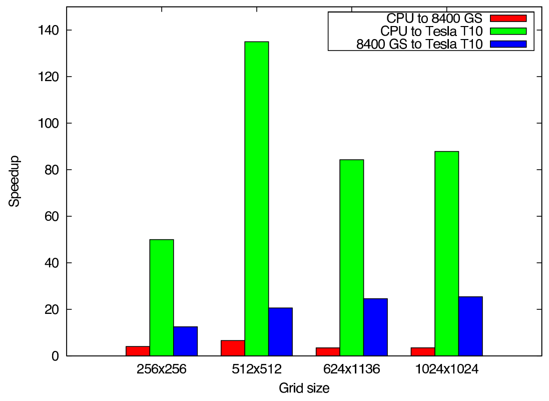

Fig. 5. Comparison of the speedups for different grid sizes.

Figure 6 illustrates the scalability of the GPU model, which follows a nearly linear fit. This is a useful deduction, as it helps us to speculate the execution times for much larger (or smaller) grids. It also helps to decide whether to go for a coarser resolution with longer simulation times vs. finer resolution and shorter simulation times.

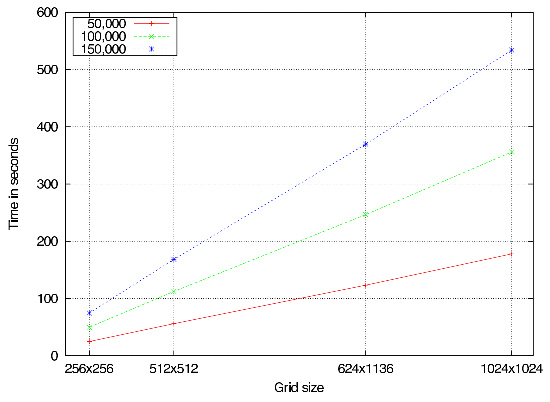

Fig. 6. Scalability of the GPU model with different grid sizes and iterations.

Table III finally highlights the global memory footprint on the GPU, for different grid sizes. These figures are inclusive of the *16 cell padding* scheme, as mentioned in the Section IV. A ”ping-ponging” scheme allows for easy transition of data between successive iterations, and requires two set of buffers two be created.

Dataset Size	Memory Footprint
256x256	2.2 MB
512x512	7.9 MB
624x1136	20.5 MB
1024x1024	29.8 MB

TABLE III
THE GPU GLOBAL MEMORY FOOTPRINT FOR DIFFERENT DATASETS.

REFERENCES

[1] N. H. Crawford and R. K. Linsley, “Digital simulation in hydrology: Stanford watershed model iv,” Stanford University, Tech. Rep., 1966.
[2] G. Tayfur, M. L. Kavvas, R. S. Govindaraju, and D. E. Storm, “Applicability of st. venant equations for two-dimensional overland flows over rough infiltrating surfaces,” *Journal of Hydraulic Engineering*, vol. 119, no. 1, pp. 51–63, 1993.
[3] P. D. Bates and A. P. J. De Roo, “A simple raster-based model for flood inundation simulation,” *Journal of Hydrology*, vol. 236, no. 1-2, pp. 54–77, 2000.
[4] W. Zhang and T. Cundy, “Modeling of two-dimensional overland flow,” *Water Resources Research*, vol. 25, no. 9, pp. 2019–2035, 1989.
[5] R. Lamb, A. Crossley, and S. Waller, “A fast two-dimensional floodplain inundation model,” *Proceedings of the Institution of Civil Engineers - Water Management*, vol. 162, no. 6, pp. 363–370, 2009.
[6] D. R. Judi, “Advances to fast-response two-dimensional flood modeling,” Ph.D. dissertation, University of Utah, 2009.
[7] P. G. Navarro, P. Brufau, J. Burguete, and J. Murillo, “The shallow water equations: An example of hyperbolic system,” *Monografias de la Real Academia de Ciencias de Zaragoza*, vol. 31, pp. 89–119, 2008.
[8] J. H. Ferziger and M. Peric, *Computational Methods for Fluid Dynamics*, 3rd ed. Springer, 2002.
[9] J. A. Liggett and D. A. Woolhiser, “Difference solutions of the shallow-water equations,” *Journal of the Engineering Mechanics Division*, vol. ASCE 93, no. EM2, pp. 39–71, 1967.
[10] P. Micikevicius, “3d finite difference computation on gpus using cuda,” in *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, GPGPU-2*, Washington, D.C., United States, 2009, pp. 79–84.
[11] M. Harris, “Optimizing parallel reduction in cuda,” 2007. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf
[12] NVIDIA, “Cuda programming guide, 2.3,” 2009. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf