




Massively Parallel Large Scale Inundation Modelling

A. Rak¹ , S. Guthe^{1,2,3} , P. Mewis¹ 

¹Technical University of Darmstadt, Germany

²Fraunhofer IGD, Germany

³Deutsche Lebens-Rettungs-Gesellschaft e.V., Germany

Abstract

Over the last 20 years, flooding has been the most common natural disaster, accounting for 44.7% of all disasters, affecting about 1.65 billion people worldwide and causing roughly 105 thousand deaths[†]. In contrast to other natural disasters, the impact of floods is preventable through affordable structures such as dams, dykes and drainage systems. To be most effective, however, these structures have to be planned and evaluated using the highest precision data of the underlying terrain and current weather conditions. Modern laser scanning techniques provide very detailed and reliable terrain information that may be used for flood inundation modelling in planning and hazard warning systems. These warning systems become more important since flood hazards increase in recent years due to ongoing climate change. In contrast to simulations in planning, simulations in hazard warning systems are time critical due to potentially fast changing weather conditions and limited accuracy in forecasts. In this paper we present a highly optimized CUDA implementation of a numerical solver for the hydraulic equations. Our implementation maximizes the GPU's memory throughput, achieving up to 80% utilization. A speedup of a factor of three is observed in comparison to previous work. Furthermore, we present a low-overhead, in-situ visualization of the simulated data running entirely on the GPU. With this, an area of 15 km² with a resolution of 1 m can be visualized hundreds of times faster than real time on consumer grade hardware. Furthermore, the flow settings can be changed interactively during computation.

CCS Concepts

• **Human-centered computing** → **Scientific visualization**; **Geographic visualization**; • **Computing methodologies** → **Real-time simulation**; **Massively parallel and high-performance simulations**; **Massively parallel algorithms**;

1. Introduction

Flood hazard and flood inundation protection are governmental tasks. Agencies exploit flood hazard warning systems based on gauges for precipitation and discharge. Warning systems leverage computer software to forecast the runoff in rivers and creeks. In recent years flood hazard has received more attention due to ongoing climate change. Flash flood hazard is one new parameter estimated permanently and entering routine computations. A well known example in Germany for its unexpectedness and severeness is the event at the river Ahr with over 170 victims [SMD*21].

Improvements in the field of remote sensing provide very detailed and reliable terrain information in the form of digital elevation maps (DEM). For flood routing purposes a detailed description of the terrain is essential when the flow depth is small. Every small obstacle may reroute the water flow at every stage of the flooding that may result in very different scenarios. In this respect, the high resolution of the laser scan data may significantly improve the accuracy of forecast results. As shown in Figure 1 the flow changes

its state to hypercritical in the field at many places. This is a local energy dissipation that would otherwise need to be modelled by the bed friction term. A high resolution thus significantly improves the water level results.

In the presented software application, the hydraulic inundation problem is solved on the basis of the full unsteady two-dimensional Saint-Venant equations including wetting and drying. The two-dimensional domain is discretized on an equidistant orthogonal grid. The grid structure improves the speedup of using highly parallel computing devices [MHSG*20]. With the massive computa-



Figure 1: Inundation at the Kinzig in 2021. A small paved road is overflowed from right to left.

[†] Statistics based on annual disaster report [CRE21] and summary report over the years 2000 – 2019 [CU20].

tional power of GPUs, it is possible to calculate complete catchments of small rivers. On a 12GB card an area of 35 km by 10 km can be kept in the card memory at resolution of 1 m². With a resolution of 10 m², the area is 100 times larger. The high computational speed let the engineers watch the flood development in situ. In our current implementation, the simulation parameters can be changed interactively but modifications to the terrain, such as interventions or catastrophic dam failures, could easily be implemented as well.

2. Related Work

Efforts to accelerate solvers for the shallow water equations on massively parallel devices have been made since the mid 2000s [HHL*05]. In recent years new solvers and more efficient implementations have emerged to accommodate for the evolving GPU hardware. Multiple second-order shallow water schemes have been implemented on GPUs for flood risk assessment [HPW*16] [SKN*21] [CCSL21]. The same equations are also used for tsunami modelling [GKS*20]. Horváth et al. [HBKK*20] have shown that the results produced by first-order schemes are comparable in accuracy, while reducing computation duration significantly. Sharif et al. [SGH*20] ran a GPU accelerated implementation of an explicit first-order upwind scheme on a multi-node GPU computer cluster. While the aforementioned schemes work on structured cartesian grids, efforts are also made to accelerate solvers working on non-uniform grids. Liu et al. [LQL18] have shown GPU acceleration of a shallow water model on an unstructured mesh. Vacondio et al. [VPF*17] propose a non uniform structured grid based on the Quadtree data structure to provide speedup in areas where high resolution is not necessary. Visualization is predominantly achieved *after* computation using intermediate or final results saved to the disk during simulation. Cornel et al. [CKS*15] visualize damage inflicted at buildings and their vicinity from pre-computed flooding events.

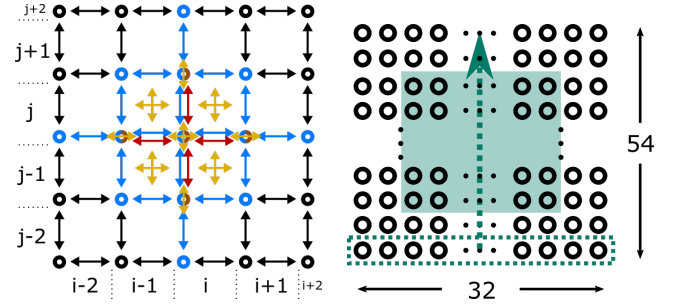
Most recent schemes described here make use of characteristic speeds (Riemann solver). Upwind schemes however (van Leer [vL92]; Kurganov, Noelle and Petrova [KNP01]; Liu et al. [LQL18]; Ginting and Mundani [GM19]) are Riemann-solver-free and may have all the needed properties.

3. Shallow Water Equations

The shallow water wave equations or Saint-Venant equations [Vre94] are the vertically integrated momentum and continuity equations that describe the motion of a thin film of water neglecting the vertical momentum conservation [Mew13]. Written in the discharge form the momentum equations resemble the exact supporting force equations.

$$\begin{aligned} \frac{\partial qx}{\partial t} + \frac{\partial(u \cdot qx + 0.5 \cdot g \cdot h^2)}{\partial x} + \frac{\partial(v \cdot qx)}{\partial y} + g \cdot h \cdot \frac{\partial B}{\partial x} + \frac{g}{h^{4/3} \cdot k_{St}^2} \cdot |\vec{v}| \cdot qx - f \cdot qy &= 0 \\ \frac{\partial qy}{\partial t} + \frac{\partial(u \cdot qy)}{\partial x} + \frac{\partial(v \cdot qy + 0.5 \cdot g \cdot h^2)}{\partial y} + g \cdot h \cdot \frac{\partial B}{\partial y} + \frac{g}{h^{4/3} \cdot k_{St}^2} \cdot |\vec{v}| \cdot qy + f \cdot qx &= 0 \\ \frac{\partial h}{\partial t} + \frac{\partial qx}{\partial x} + \frac{\partial qy}{\partial y} &= 0 \end{aligned}$$

Here qx , qy are the specific discharges in x and y direction. h is the water depth and B is the elevation of the terrain or river bed above a horizontal datum. The position of the free water surface is $\zeta = B + h$. f is the Coriolis parameter and k_{St} is the Manning-Strickler coefficient for the bed friction.



(a) Computational stencil for $\zeta_{i,j}^{t+\Delta t}$. Blue symbols represent water levels and discharges at current timestep. Red symbols at next timestep. Yellow arrows represent impulse fluxes. (b) A warp comprising 32 threads iterates over 54 rows. Only the highlighted cells are written out. The outer cells are not computed, as some of the required data is missing.

Figure 2: (a) Computational stencil of our numerical solver and (b) cells processed by a single warp.

To keep the resolution as large as possible a very slim numerical solver is implemented. This solver uses a very simple first order upwind scheme on a staggered mesh with explicit time integration. It is robust, positivity preserving, well-balanced and reproduces hydraulic jumps well. The verification of the scheme for several test cases is described in more detail in a paper in preparation. The computational stencil of the solver is shown in Figure 2a.

4. CUDA Implementation

In this section we introduce our implementation of the proposed time integration scheme in CUDA. The shallow water simulation is implemented from scratch in C++ and CUDA Toolkit 11.4. It is a standalone application that simulates a flooding event on the GPU while optionally visualizing the terrain and computed water levels simultaneously using OpenGL.

4.1. Implementation Details

For the simulation, eight buffers with the size of the grid are required. The buffers store the water level ζ and fluxes qx , qy at the current and next timestep and the terrain height and roughness, respectively. These buffers are initialized on the CPU as linear float arrays in row-major order and transferred to the GPU's device memory at the beginning of the execution. Further transfers between the host and device are avoided by keeping the buffers in device memory and swapping the pointers to old and new buffers between consecutive kernel launches.

Our implementation comprises two kernels that are launched successively. The first kernel implements constraints on ζ^t , $qx^{t-\Delta t/2}$ and $qy^{t-\Delta t/2}$ at the grid's boundaries. The second kernel is launched afterwards and performs the actual time integration computation. This separation allows for flexibility, since the same compute kernel can be used in different scenarios where other boundary constraints are needed.

To make efficient use of the GPU's resources the underlying architecture must be taken into account. The threads running on the GPU are grouped into units of 32 threads named warps. Given that the threads in a warp agree on their execution path, all threads execute

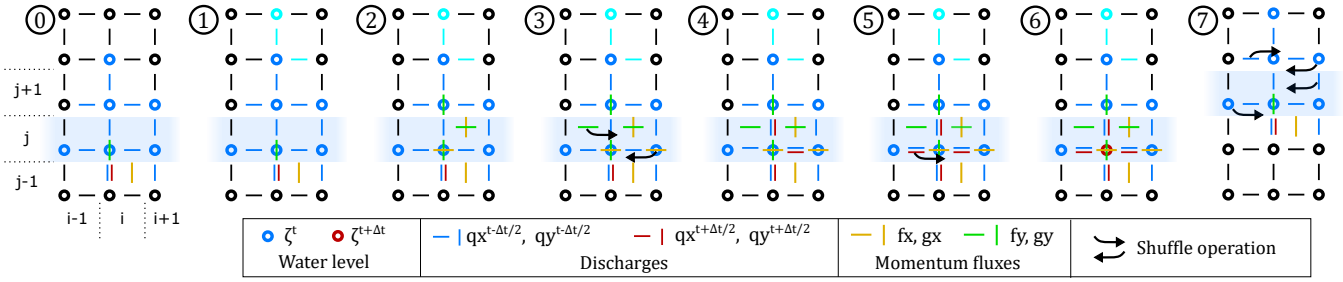


Figure 3: Simulation steps involved in computing $\zeta_{i,j}^{t+\Delta t}$ visualized for a single thread. The highlighted row illustrates which nodes the threads participating in the corresponding warp are currently computing. The cyan symbols in steps 1-6 represent values that are not currently used for computation, but prefetched from global memory simultaneously for the next row iteration beginning with step 7.

instructions simultaneously [NVI22]. Memory accesses of a warp are coalesced into as few transactions as needed, depending on the memory address distribution of the request. Our structured grid facilitates this, as successive threads access successive memory addresses. Up to 32 warps are further grouped into thread blocks. Warps in a thread block can communicate data via on-chip shared memory, however this requires thread block synchronization as the warps do not execute concurrently. This warp-cooperative approach is favored in related work to subdivide the domain into blocks with thread block dimensions [HPW*16] [VPF*17] [SKN*21]. However, these schemes tend to be latency bound.

In contrast, our implementation does not rely on shared memory and therefore does not require thread block synchronization. This lack of memory barriers allows us to fully overlap computation with data fetching. This is essential in achieving high utilization of the GPU's resources, since memory operations, particularly when accessing device memory, introduce high latencies. These latencies must be hidden by executing instructions independent of the accessed data [NVI22]. For this purpose, we subdivide the domain into blocks of 28×50 cells, that are processed by one warp each, as illustrated in Figure 2b. The width of these blocks is limited by the warp size of 32, while the height can be chosen arbitrarily. Two ghost cells are required on each side of the block, due to the data dependencies of the stencil shown in Figure 2a. A warp computes one row at a time while prefetching the data required for the next one. This happens concurrently, since the computation is independent of the prefetched data.

Similar to related schemes, a dry-wet mask is used to further reduce computation times [HPW*16]. The compute kernel leverages this mask to determine the first and last wet row in the current block. Only those and the rows in-between are fetched and computed.

4.2. Simulation Steps

In this section the steps involved in the computation kernel are outlined. Figure 3 gives an overview of the data dependencies and operations in each step in the context of a single CUDA thread. The kernel's row loop described in the previous section is shown in steps ① - ⑦. They are preceded by step ① in which initial fields are fetched from device memory and shuffled between threads. The following describes the steps necessary to compute $\zeta_{i,j}^{t+\Delta t}$ in the responsible thread.

0. Load all required fields from previous time step into registers.

These are shown in blue in Figure 3.0. The values in column i are loaded from device memory. Required values in columns $i-1$ and $i+1$ are then shuffled up and down, respectively.

1. Start prefetching the required values for the subsequent warp iteration $\zeta_{i,j+3}^t, q_{i,j+2}^{t-\Delta t/2}, q_{i,j+2}^{t-\Delta t/2}, B_{i,j+2}$ and $R_{i,j+1}$. These are partially shown in cyan in Figure 3.1. This does not result in a stall in the following computations, since these do not depend on the registers used to store the prefetched values.
2. Compute the momentum fluxes $f_{x,i,j}, f_{y,i,j}, g_{x,i,j}$ and $g_{y,i,j+1}$. They are shown in yellow and green in Figure 3.2.
3. Shuffle up momentum flux $f_{y,i-1,j}$ and shuffle down $f_{x,i+1,j}$. The momentum fluxes $g_{x,i,j-1}$ and $g_{y,i,j}$ were computed in the previous warp iteration and are available in their respective registers.
4. Compute the fluxes $q_{i,j}^{t+\Delta t/2}$ and $q_{i,j}^{t+\Delta t/2}$ shown in red in Figure 3.4. The results are written to global memory.
5. Flux $q_{i-1,j}^{t+\Delta t/2}$ is shuffled up from west neighbor thread. Flux $q_{i,j-1}^{t+\Delta t/2}$ is available from previous warp iteration.
6. With all fluxes neighbors at $t + \Delta t/2$ now available, the water level $\zeta_{i,j}^{t+\Delta t}$ is computed. The result is written to global memory.
7. The warp now advances north by one row by utilizing the values prefetched in step 1. All data required for this operation is available in the registers of the warp. Values for which a northern neighbor is available in registers of the same thread adopt their value and are shown in solid blue in Figure 3.7. The remaining values are then shuffled up and down. Finally, the thread continues with step 1 or exits when the desired number of rows have been computed.

4.3. In Situ Visualization

The computed results for the water level ζ , the fluxes qx, qy and the terrain B can optionally be visualized in 3D during simulation. For this, CUDA interoperability functions for OpenGL are used to keep the data on the GPU. This avoids latency-heavy GPU-CPU communication and allows for an efficient implementation. We achieve this by rendering an equidistant orthogonal mesh with the size of the simulated area. Each device buffer of the simulation data is treated as a texture which are sampled in the GLSL shaders to determine vertex positions and fragment colors. An example of the in situ visualization of the Kinzig dataset is shown in Figure 4. Wet cells are rendered in different colors, for which the user can specify

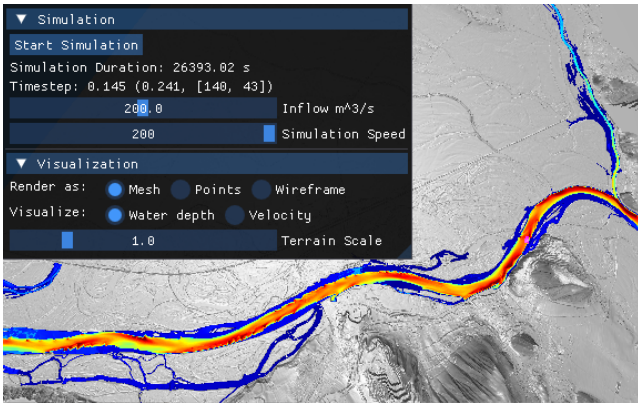


Figure 4: Screenshot of user interface for in situ visualization.

whether these are computed from the water level or velocity. The inflow parameter manipulates the amount of water flowing into the river at the border of the grid, allowing observation of the flooding behaviour under different conditions. The application can also be used to demonstrate historical flooding events, such as the collapse of the Malpasset Dam.

5. Results

	RTX 2060M	GTX 1080TI	RTX 3080	RTX 3090
Mem BW	336 GB/s	484.4 GB/s	760.3 GB/s	936.2 GB/s
Cores	1920	3584	8704	10496
Kinzig	1609 s	1242 s	636 s	512 s
Malpasset	1133 s	840 s	410 s	340 s

Table 1: Performance measurements for different GPUs. Mem BW is the theoretically achievable memory bandwidth of the device. Cores is the number of CUDA Cores.

Table 1 shows the performance measurements of different data sets on multiple consumer-grade GPUs. The Kinzig is a river in southwestern Germany. Our terrain dataset covers an area of 4800×3000 meters with a resolution of 1 m [Mew21]. The simulation runs with a time step of $\Delta t = 0.1$ s. In this area the Kinzig river flows from the eastern to the western edge. In our Kinzig test case the terrain starts entirely dry while a constant inflow is applied to the influx area. Within three simulated days the water levels reach a stationary state, which is shown in Figure 5. This time period is computed within 512 s on a RTX 3090, which is 506 times faster than real time.

In the Malpasset test case the Malpasset Dam collapse of 1959 is simulated for one hour. The terrain data comprises an area of 17.5×8.5 km at a resolution of 1 m. The simulation runs with a time step of $\Delta t = 0.01$ s. The simulated time period is computed within 340 s on an RTX 3090.

Table 1 shows the relationship between memory bandwidth and computation duration. The memory bandwidth of the RTX 3090 is roughly three times higher than the RTX 2060M. Likewise, the computation is roughly three times faster. This is due to a memory bandwidth-bound kernel with a DRAM utilization of up to 80%.

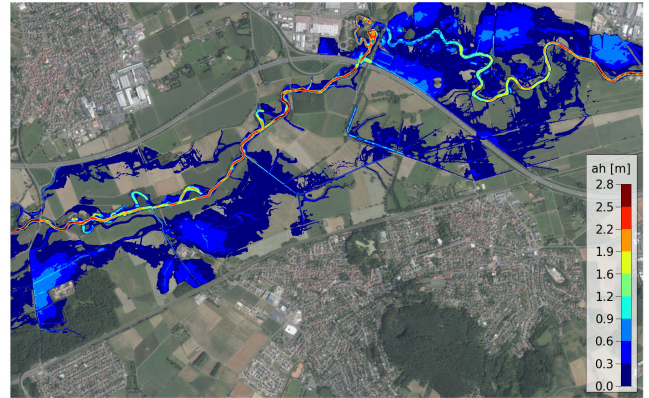


Figure 5: Water levels of the Kinzig river after a simulated period of three days or 2592000 iterations.

Horváth et al. [HBKK*20] implemented the Chen-Noelle scheme (CN) [CN17], a first-order upwind shallow-water scheme, in CUDA on a cartesian grid. They simulated a flooding event of the Danube river in the Lobau area with a size of $8.1 \text{ km} \times 5 \text{ km}$ at a resolution of 3 m. This amounts to 4.5 million cells in contrast to the 14.4 million cells of our Kinzig domain with a similar percentage of wet cells. Their simulation was run on a GTX 1080, the specifications of which are comparable to our RTX 2060 Mobile. Their implementation computes one simulated second in 6.2 ms. In contrast, one simulated second in our Kinzig case study takes $\frac{1608.7s}{2592000 \cdot \Delta t} \approx 6.2 \text{ ms}$ to compute on the RTX 2060M for a domain with three times the amount of cells.

This result suggests a speedup of three compared to Horváth et al. [HBKK*20]. We confirmed this by recreating their Marchfeld test case, in which an area of $17 \text{ km} \times 19.5 \text{ km}$ with a resolution of 3 m is simulated for twelve days, and running it on a GTX 1080. While their CN implementation took 27372 s to compute this test case, ours took 10096 s, resulting in a speedup of 2.71.

6. Conclusion

We presented a system to forecast flood inundation problems. Extensive testing showed that it produces accurate results of the hydraulic problem. Furthermore, the software is tested against historic flood events and showed good results in these tests. We showed that our system is very efficient and faster than the available state-of-the-art. With this forecasts are even possible on consumer grade hardware within a few minutes. A three day forecast for a medium sized river with 15 million cells takes just 8 minutes on an RTX3090 GPU. For demonstration purposes or studies of different building designs the software can be applied interactively and the parameters can be changed on the fly. This way our system may contribute to counteract one of the most severe natural disasters.

7. Acknowledgements

This research is closely related to a project funded by the German Federal Ministry for Economic Affairs and Energy in the ZIM initiative for “Development of new techniques in computer modeling for flood prevention”.

References

- [CCSL21] CARLOTTO T., CHAFFE P., SANTOS C., LEE S.: SW2D-GPU: A two-dimensional shallow water model accelerated by GPGPU. *Environmental Modelling & Software* 145 (09 2021), 105205. doi:10.1016/j.envsoft.2021.105205. 2
- [CKS*15] CORNEL D., KONEV A., SADRANSKY B., HORVÁTH Z., GRÖLLER E., WASER J.: Visualization of Object-Centered Vulnerability to Possible Flood Hazards, 2015. doi:10.1111/cgf.12645. 2
- [CN17] CHEN G., NOELLE S.: A new hydrostatic reconstruction scheme based on subcell reconstructions. *SIAM Journal on Numerical Analysis* 55, 2 (2017), 758–784. doi:10.1137/15M1053074. 4
- [CRE21] CRED—CENTRE FOR RESEARCH ON THE EPIDEMIOLOGY OF DISASTERS: Disaster Year in Review 2020: Global Trends and Perspectives, 2021. 1
- [CU20] CRED—CENTRE FOR RESEARCH ON THE EPIDEMIOLOGY OF DISASTERS, UNDRR—UNITED NATIONS OFFICE FOR DISASTER RISK REDUCTION: The Human Cost of Disasters. An Overview of the Last 20 Years (2000–2019), 2020. 1
- [GKS*20] GILES D., KASHDAN E., SALMANIDOU D. M., GUILLAS S., DIAS F.: Performance analysis of Volna-OP2 – massively parallel code for tsunami modelling. *Computers and Fluids* 209 (9 2020). doi:10.1016/j.compfluid.2020.104649. 2
- [GM19] GINTING B. M., MUNDANI R.-P.: Comparison of shallow water solvers: Applications for dam-break and tsunami cases with reordering strategy for efficient vectorization on modern hardware. *Water* 11 (2019), 639. doi:10.3390/w11040639. 2
- [HBKK*20] HORVÁTH Z., BUTTINGER-KREUZHUBER A., KONEV A., CORNEL D., KOMMA J., BLÖSCHL G., NOELLE S., WASER J.: Comparison of Fast Shallow-Water Schemes on Real-World Floods. *Journal of Hydraulic Engineering* 146 (1 2020), 05019005. doi:10.1061/(asce)hy.1943-7900.0001657. 2, 4
- [HHL*05] HAGEN T., HJELMERVIK J., LIE K.-A., NATVIG J., HENRIKSEN M.: Visual simulation of shallow-water waves. *Simulation Modelling Practice and Theory* 13 (11 2005), 716–726. doi:10.1016/j.simpat.2005.08.006. 2
- [HPW*16] HORVÁTH Z., PERDIGÃO R. A., WASER J., CORNEL D., KONEV A., BLÖSCHL G.: Kepler shuffle for real-world flood simulations on GPUs. *International Journal of High Performance Computing Applications* 30 (11 2016), 379–395. doi:10.1177/1094342016630800. 2, 3
- [KNP01] KURGANOV A., NOELLE S., PETROVA G.: Semidiscrete central-upwind schemes for hyperbolic conservation laws and hamilton-jacobi equations. *SIAM J. Sci. Comput.* 23, 3 (2001), 707–740. 2
- [LQL18] LIU Q., QIN Y., LI G.: Fast simulation of large-scale floods based on gpu parallel computing. *Water (Switzerland)* 10 (5 2018). doi:10.3390/w10050589. 2
- [Mew13] MEWIS P.: Sparse element for shallow water wave equations on triangle meshes. *International Journal for Numerical Methods in Fluids* 72, 8 (2013), 864–882. doi:10.1002/flid.3761. 2
- [Mew21] MEWIS P.: Estimation of vegetation-induced flow resistance for hydraulic computations using airborne laser scanning data. *Water* 13, 13 (2021). doi:10.3390/w13131864. 4
- [MHSG*20] MORALES-HERNÁNDEZ M., SHARIF M. B., GANGRADE S., DULLO T., KAO S.-C., KALYANAPU A., GHAFOR S., EVANS K., MADADI-KANDJANI E., HODGES B.: High-performance computing in water resources hydrodynamics. *Journal of Hydroinformatics* 22 (03 2020). doi:10.2166/hydro.2020.163. 1
- [NVI22] NVIDIA CORPORATION: CUDA Toolkit Documentation - Programming Guide, 2022. Version v11.6.1. 3
- [SGH*20] SHARIF M. B., GHAFOR S. K., HINES T. M., MORALES-HERNÁNDEZ M., EVANS K. J., KAO S. C., KALYANAPU A. J., DULLO T. T., GANGRADE S.: Performance evaluation of a two-dimensional flood model on heterogeneous high-performance computing architectures. Association for Computing Machinery. doi:10.1145/3394277.3401852. 2
- [SKN*21] SHAW J., KESSERWANI G., NEAL J., BATES P., SHARIFIAN M. K.: LISFLOOD-FP 8.0: The new discontinuous Galerkin shallow-water solver for multi-core CPUs and GPUs. *Geoscientific Model Development* 14 (6 2021), 3577–3602. doi:10.5194/gmd-14-3577-2021. 2, 3
- [SMD*21] SCHÄFER A., MÜHR B., DANIELL J., EHRET U., EHMELE F., KÜPFER K., BRAND J., WISOTZKY C., SKAPSKI J., RENTZ L., ET AL.: Hochwasser Mitteleuropa, Juli 2021 (Deutschland): 21. Juli 2021–Bericht Nr. 1 „Nordrhein-Westfalen & Rheinland-Pfalz”. 1
- [vL92] VAN LEER: Progress in multi-dimensional upwind differencing. *Part of the Lecture Notes in Physics book series (LNP, volume 414)* (1992). 2
- [VPF*17] VACONDIO R., PALÙ A. D., FERRARI A., MIGNOSA P., AURELI F., DAZZI S.: A non-uniform efficient grid type for GPU-parallel Shallow Water Equations models. *Environmental Modelling and Software* 88 (2 2017), 119–137. doi:10.1016/j.envsoft.2016.11.012. 2, 3
- [Vre94] VREUGDENHIL C. B.: *Numerical methods for shallow-water flow*. Kluwer Academic Publishers, 1994. 2