Solving the kinematic wave equation on drain direction maps
Internship Report

Dock Staal

November 14, 2022

1 Introduction

The Lue software package [1], which is being developed at the Department of Physical Geography of the university of Utrecht, is a parallel software package that provides a framework for large-scale spatiotemporal (in space and time) simulation. One of the current development pursuits is to (partially) provide the functionality of the PCRaster software package [2] that has been developed by this university in the 80s. The goal of my internship was to help with this process and to assist in the parallelization of a method that requires the expertise of numerical schemes for PDEs.

More specifically, the objective was to create an implementation of water propagation using the kinematic wave equation (a 1D PDE to model water flow) in a drain direction map (a grid-based map of a landscape which records which cell flows to which cell) using the HPX parallel software package. Moreover, given that PCRaster has been developed in the 80s, I was also asked to figure out whether there are numerical algorithms to solve the kinematic wave equation on a drain direction map that better suit their needs. This then leads to the following internship goal:

Create a standalone parallel implementation of the kinematic wave equation on a drain direction map.

This objective is then accompanied by the following two research questions.

- What is in the context of LUE, currently the best numerical scheme to create a parallel implementation of the kinematic wave equation on a drain direction map?
- What is a scalable way to implement this numerical scheme?

To answer these questions and to end up with a working implementation I decided to take the following steps.

1. Learn the basics of the asynchronous many-task approach, which is a new parallel programming approach and learn the basics of HPX, which provides the functionality to solve problems using asynchronous many-task.
2. Understand the kinematic wave equation and search the literature for currently used numerical schemes to solve this equation.
3. Write a serial framework in C++ to solve the kinematic wave equation on a drain direction map and provide a testing ground for multiple numerical schemes.
4. Finally, create the parallel implementation of the framework and complete the main goal of the internship.

The reason I wanted to do this internship at the University of Utrecht was to learn the following things.

- Learn enough C++ and all the tools that surround it to be able to write medium-scale programming projects in C++ on my own. One of the barriers to using C++ in my daily project was not the programming language itself. It was the fact that you have to know way more to write C++ projects, such as how to use debuggers, testing software, build from source, use generator tools, Cmake etc. Given that when I am studying I am busy enough with just the projects I have to do for the university, I always resort to python, because I don't have the time to learn all these tools while also solving my problems. Hence, doing such an internship allowed me the space to explore all these tools and to become self-sufficient when programming in C++.
- I wanted to learn how to write parallel software on my own.
- I wanted to work in an interdisciplinary team and see what my role would be as a mathematician with expertise in numerical simulation.

2 Theory

As said in the introduction, the main goal for this internship was to create a parallel implementation of a kinematic wave equation solver that calculates how the water levels evolve in a drain direction map. To that extend it is important to discuss some of these concepts in more detail.

2.1 Drain direction map

Given a landscape, one can look under or above ground to see all kinds of streams flowing. Those streams eventually all come together and flow into an output area. A good example would be a river that ends up in the ocean. A drain direction map or a flow direction map is then a 2d grid-based map that records the direction of flow. More precisely, a drain direction map has the following properties.

- Every cell has a non-negative water level.
- A cell flows into exactly one of its eight (thus including the diagonals) neighbouring cells unless it is a drain. A drain is a boundary point and the outflow is determined via boundary conditions.
- Multiple cells are allowed to flow in one cell.

An example of a drain direction map can be seen in figure 2. These drain direction maps are usually derived from height maps; in the case of figure 2, the map has been derived from the height map that can be seen in figure 1.

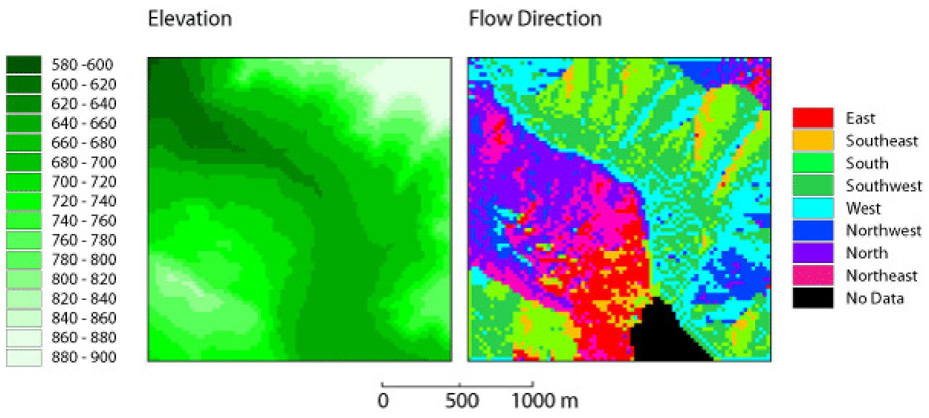


Figure 1: An example of a height map. This figure is taken from [3].

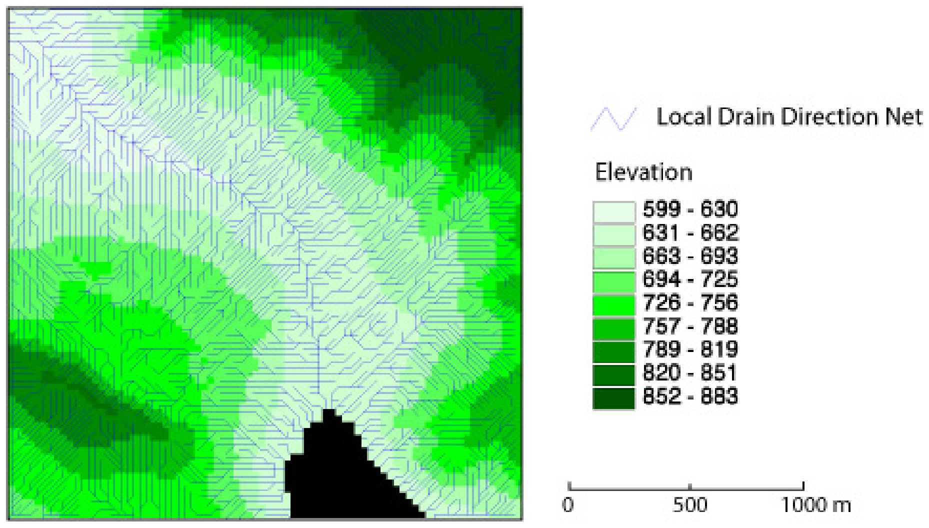


Figure 2: Drain direction map associated with the height map of figure 1. This figure is taken from [3].

2.2 LUE and PC raster

LUE [1] and PCRaster [2] are both software packages targeted at domain experts to provide functionality to work with map algebras. In other words, both of the software packages provide functionality to perform operations on 2d grids. PCRaster, developed in the 80s by the university of Utrecht, has specific functionality to implement and create 2d spatiotemporal environment models. The idea is that a domain expert can create a python script that calls those functions. After that it is the job of PCRaster to perform these operations on the map algebra.

LUE is a parallel software package that aims to provide functions to work with and store large amounts of data. One of main functionalities it provides is to work with map algebras. One of the purposes of LUE is to provide parallel implementations for some of the PCRaster functions.

2.3 Kinematic wave equation

Given a drain direction map, we would like to model the evolution of water levels. Naturally, given the vastness of the hydrology literature, there is a great choice in the models we could use. However, my task was to look at the kinematic wave equation as a model for this process. These equations are derived from the one-dimensional Saint-Ventant equations under the assumption that the momentum equation is dominated by friction and gravity forces [4]. The mathematical formulation of the equations is

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = q \tag{1}$$

$$Q(t) = \alpha A(t)^{\beta}, \tag{2}$$

where A is the average crossectional area of the water flow in a stream, Q is the average discharge, q is a source term, (for instance to incorporate rainfall) and α and β are just some coefficients. In most literature, these last two constants will be provided by Manning's equation. That means that α is a function of slope and β will be taken to be $\beta = \frac{5}{3}$.

2.4 Finite Volume Methods

Finite volume methods are mostly used for hyperbolic problems where we consider a quantity that moves with finite speed from one place to another [5]. Examples of this are the distributions of heat or the movement of a fluid in a river. The idea of finite volume is to work with the problem. Instead of finding a smart way to discretize the PDE, as we do in finite difference. We discretize the problem into a grid and estimate the amount of quantity that has been moved through the boundaries.

In more mathematical terms, the first step when we take when applying finite volume methods to a PDE problem is to discretize space and create cell averages

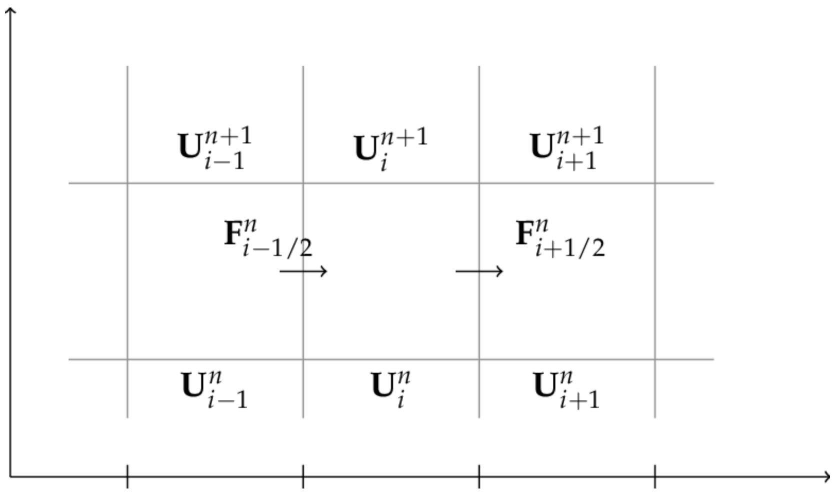


Figure 3: Example of a process to calculate the next time step U_i^{n+1} using finite volume. First we have the quantities U_{i-1}^n, U_i^n and U_{i+1}^n. After that we calculate the fluxes $F_{i-1/2}^n$ using U_{i-1}^n and U_i^n and $F_{i+1/2}^n$ using U_i^n and U_{i+1}^n. Using these fluxes we calculate the the next cell average U_i^n using (3). This figure is taken from [6].

of the quantity we are modelling. See figure 3. Then, in every timestep, we update the cell averages U by estimating the amount of quantity that entered and left the cell. We do this by using a flux estimation F on the boundary. Hence, if we think about a one-dimensional system we get the following mathematical equation.

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta x}(F^n(U_{i-1}^n - U_i^n) - F^n(U_i^n - U_{i+1}^n), \tag{3}$$

where Δx is the cell length, Δt the time step lenght and $F(U_{i-1}^n, U_i^n)$ is the flux between cell $i-1$ and i at timestep n usually denoted by $F_{i-1/2}^n$.

2.5 HPX and AMT

Asynchronous many tasks (AMT) [7] is a relatively new parallel programming model. In the AMT approach, the programmer splits the program up into smaller tasks. Moreover, the programmer also indicates the dependencies between the tasks. For instance, the programmer can tell the program that task 3 is only able to be run if 1 and 2 have run and what data these tasks need to provide to task 3. The fundamental idea of AMT is then to let the computer figure out what tasks to run and when. Consequently, an AMT program consists of four fundamental components: a list of tasks, a list of runnable tasks, a scheduler that schedules the runnable tasks and processes that can run these tasks.

To facilitate the AMT programming model we need libraries with implementations of the needed functionalities. The standard library for AMT in C++ is called HPX [8]. For a user, HPX consists of two main features. First of all, we have futures. These are objects that represent to be executed tasks. Secondly, we have dataflows, which are functions used to tell the program what the dependencies are between tasks.

When programming using older HPC libraries such as MPI the programmer needs to manually sync up processes to transfer data. This not only makes life considerably more difficult in comparison with using the AMT model but when not properly optimised the synchronization that needs to be done to facilitate the sending over of data can cause considerable overhead. So the main goal of AMT is to make HPC easier to do by taking the largest bottleneck out of programming in parallel i.e. scheduling tasks. However, even though it may be easier to write a good scalable program using the AMT model there are still a lot of variables to keep track of. For instance, when using HPX one needs to make tasks that are not too long and not too short for optimal performance. Namely, if tasks are too short, the overhead of making future objects becomes very large and slows down the program. However, if the tasks are too long, it may happen that all process but one is running, because the data of that one process is used for all upcoming tasks.

3 Methods

Solving the kinematic wave equation on a parallel machine can be decomposed into two steps. First of all, the numerical problem must be considered and a method of discretization has to be chosen. Secondly, a scalable HPX implementation must be made. In this section, I will discuss both aspects and the way they were tackled.

3.1 Solvers

From a mathematical perspective, the kinematic wave equation is a non-linear, scalar convection equation. Thus, luckily, nowadays there is an abundance of good numerical schemes to choose from to solve this problem. In the end, I decided to implement a finite volume scheme for a couple of reasons. First and foremost, it is relatively easy to think about ways to extend the numerical scheme of the one-dimensional kinematic wave equations on a drain direction map. Namely, the problem arises when we consider a cell where two or more streams flow into the cell. This problem is in essence not one-dimensional anymore. However, if we think in terms of finite volume methods the extension becomes completely trivial that is, we have just multiple inward fluxes and only one outward flux. Consequently, there was no need to play with the equations themselves to incorporate this effect and to discretize those new equations. Secondly, finite volume methods have a whole range of nice properties for hyperbolic problems, such as being conservative. Thirdly, it is easy to experiment with lots of different schemes as we only have to change the flux function if we want to try out a different schema. Finally, there are a lot of different flux functions to choose from so that a modeller using LUE can decide for themselves if they rather have more speed or accuracy.

3.2 Parallelization

Finding a parallel implementation of the solver ended up being a trivial task. Namely, because a simple finite method is a focal operation in map algebra terms. That is, it uses only the information from the neighbouring cells. Fortunately, the developers of LUE already created a framework for focal operations and thus the solver already fits perfectly in the LUE framework.

When solving hyperbolic problems numerically, one should always satisfy the CFL condition. Informally, this means that the integration steps should be so small that information from one cell is only able to travel to its neighbouring cells. For this particular problem, it means that

$$\frac{\alpha\beta A^{\beta-1}\Delta t}{\Delta x} \leq 1 \tag{4}$$

should be satisfied for all the cells, where α and β come from (2)

There are multiple ways to go about finding an appropriate time step. The simplest method, and the one we ended up implementing, is to choose a time step for which the water flow should reach unphysical speeds to not satisfy the CFL condition. For example, the LUE team works mostly with cells with a width and height of 100 meters. Hence, if we take Δt to be equal to one, then this would imply that the water should flow a hundred meters per second to not satisfy the CFL condition. This simply doesn't happen in nature and hence one is a safe choice for Δt.

4 Results

The first month of the internship was devoted to learning how HPX works and finding out what numerical method we should use for the kinematic wave equation. As discussed in subsection 3.1, I decided to work with finite volume methods. To test whether this was a good idea at all, I devoted the second month of my internship to creating a framework in C++ to test these methods (see [9] for the code). In this framework, I worked with the upwind finite volume method. In terms of equations (1) and (3) this means that

$$F^n_{i-1/2} = Q^n_i. \tag{5}$$

When creating the framework to test the finite volume method the biggest struggle was to create a realistic drain direction map object. It turned out that given the time constraints a realistic map was too big of a challenge. Hence, I ended up using an algorithm called recursive backtracking to generate drain direction maps. This algorithm is normally used to create mazes and thus, the generated drain direction maps look like mazes, see figure 4. Consequently, the flow direction does not at all look natural in the simulations. Nevertheless, we can still see if the solutions created by the upwind method make sense by looking at the following points. First of all, given the nature of the kinematic wave equations, the larger the quantity A is at a point, the larger the discharge, so the faster the stream should flow. Secondly, the solution should propagate to the drain and finally, we can look for instabilities.

In figure 4, we see a simulation using the upwind method on a maze-like drain direction map. The simulation was set up in such a way that uniformly over the whole map a slight rain was falling up to $t = 3$. What we can see from the simulation is that the water starts collecting in the main stream that leads to the drain in the upper left corner.

In conclusion, we can see from this simulation, and others that have been done using the framework [9], that the upwind method looks very promising. First of all, we see that the yellow parts, the parts that have the largest A, move the quickest. Secondly, the stream goes to the drain and finally no instabilities occur as long as the CFL condition is satisfied.

Conclusion

In the introduction, we stated the main goal of this internship.

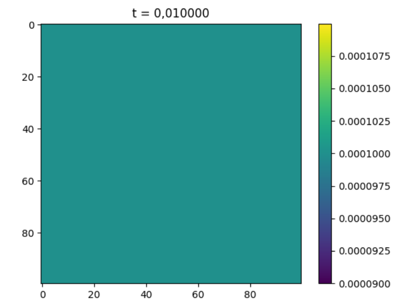
Create a standalone parallel implementation of the kinematic wave equation on a drain direction map.

This goal is then accompanied by the following two research questions.

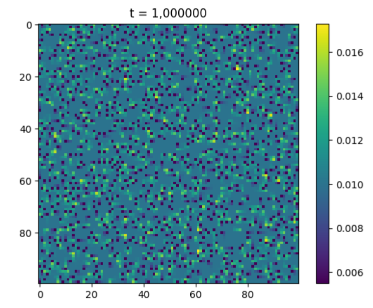
- What is in the context of LUE, currently the best numerical scheme to create a parallel implementation of the kinematic wave equation on a drain direction map?
- What is a scalable way to implement this numerical scheme?

As one might notice, I didn't mention any standalone parallel implementations of the kinematic wave equation on a drain direction map. The reason for this is quite simple, the internship lasted two months and in the end, I didn't have enough time to create such a program. A conclusion might be then that the internship failed. However, I would like to argue the difference.

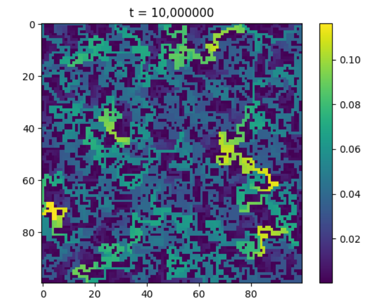
First of all, both the accompanying research questions have been addressed in the methods section. The best numerical scheme to use in my option is the finite volume method, see subsection 3.1, because the extension of the kinematic wave equations to drain direction maps is the easiest, the nice properties these methods have for hyperbolic problems and the plethora of variations in these



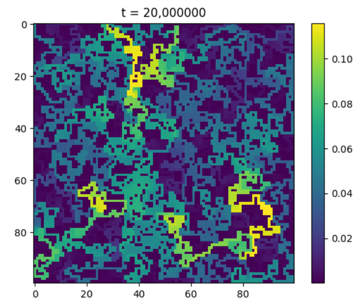
(a) $t = 0.01$



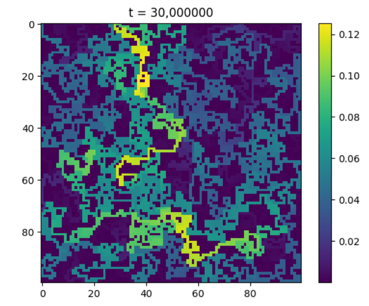
(b) $t = 1$



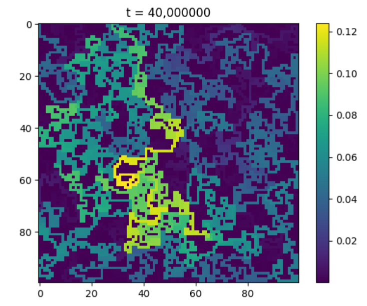
(c) $t = 10$



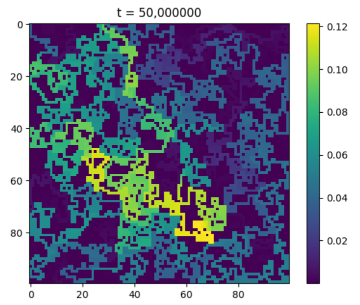
(d) $t = 20$



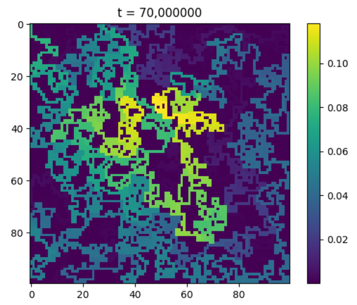
(e) $t = 30$



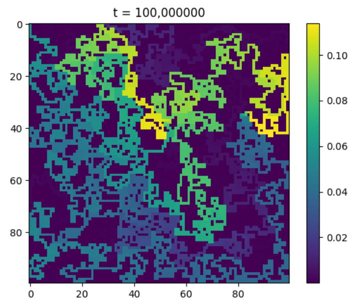
(f) $t = 40$



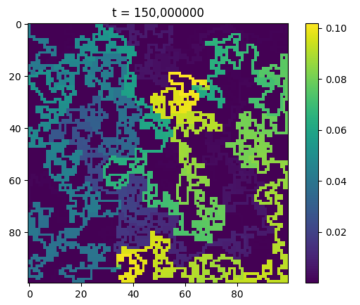
(g) $t = 50$



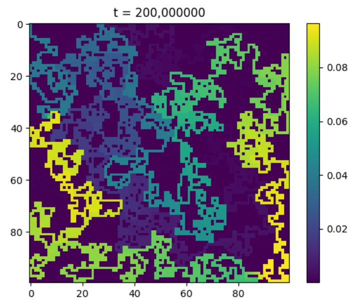
(h) $t = 70$



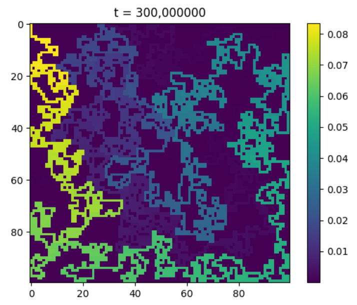
(i) $t = 100$



(j) $t = 150$



(k) $t = 200$



(l) $t = 300$

Figure 4: In this plot we see a simulation using the upwind method on a maze-like drain direction map. The simulation was set up in such a way that uniformly over the whole map a slight rain was falling up to $t = 3$. What we can see from the simulation is that the water starts collecting in the main stream that leads to the drain in the upper left corner.

methods that allow for a trade-off in computational precision and speed. The scalability question has also been handled, see subsection 3.2. Namely, since the finite volume method leads to a focal operation, the operation fits perfectly in the already set-up framework of LUE. Therefore, the simplest scalable way to implement the solver, at least for the people of LUE, is to just fit the operation into their framework.

Secondly, in the end, it would have only added a bit more value if I made the parallel implementation of the serial program that I created in this internship. To be specific, because I was able to find a solver that was a focal operation in map algebra terms, it fits right into the already set up parallel framework of LUE. Hence, how to parallelize the problem has already been addressed by the LUE team. Moreover, the serial implementation is more than enough to see whether the finite volume method makes sense.

In conclusion, I learned a great deal from the internship, see also the appendix. Moreover, both from a theoretical and an experimental standpoint, see figure 4, does the finite volume method look usable. Consequently, I still consider this internship a success and I hope the algorithm will be of use to the LUE software package.

5 Abstract: Personal account of the internship

As stated in the introduction I had a lot of personal education goals when starting the internship. In this abstract, I would like to discuss precisely what I learned.

5.1 C++ proficiency

During the internship, my C++ proficiency increased a lot. When I started, I spend the first few weeks learning and writing a simple parallel application using HPX. During this process, I also learned the basics of CMake. After these weeks, I spend some time understanding the kinematic wave equation and what kind of solvers people have used to model it. In the second month, I ended up writing a medium-scale program, see [9], that implements a finite volume solver. When working on this project, I learned a lot of new C++, more than the basics of CMake, how to use debuggers, work with tools like clang-tidy, separate software in the OS using conda's virtual environments and many more things. Overall I can say that the goal of becoming self-sufficient with C++ has been reached.

5.2 Learning HPX

As already stated in the conclusion I wasn't able to write the HPX implementation of the serial framework mentioned above. However, during the first few weeks, I made an extremely simple program using the HPX library, that multiplies the numbers of a partitioned vector. In the earlier stages of the internship, I also spend a lot of time going through the documentation to understand HPX and AMT from a theoretical standpoint. So I became somewhat familiar with HPX, however, I am not fluent with it (yet).

5.3 My role as a mathematician

One of the most fun things was the fact that I was the Mathematics expert on the team. Hence, I had to stand on my own two legs to figure out what the best methods would be. Moreover, I had to convince the people I was working with that the finite volume method would be very suitable for modelling the kinematic wave equations. In the end, seeing the first test of my solution succeed and talking about the possible adoption of the algorithm into LUE, was very satisfying.

References

[1] K. de Jong, D. Panja, M. van Kreveld, and D. Karssenberg, "An environmental modelling framework based on asynchronous many-tasks: Scalability and usability," *Environmental Modelling & Software*, vol. 139, p. 104998, 2021.

[2] D. Karssenberg, O. Schmitz, P. Salamon, K. de Jong, and M. F. Bierkens, "A software framework for construction of process-based stochastic spatio-temporal models and data assimilation," vol. 25, no. 4, pp. 489–502, 2010.

[3] G. I. T. T. Alliance, "Terrrain analysis (intermediate)." Last accessed 3 September 2022.

[4] V. Te Chow, D. Maidment, and L. Mays, *Applied Hydrology*. McGraw-Hill series in water resources and environmental engineering, Tata McGraw-Hill Education, 2010.

[5] R. J. LeVeque *et al.*, *Finite volume methods for hyperbolic problems*, vol. 31. Cambridge university press, 2002.

[6] H. Lund, "Relaxation models for two-phase flow with applications to co2 transport," 2013.

[7] S. Knight, G. M. Baker, M. Gamell, D. Hollman, G. Sjaardema, H. Kolla, K. Teranishi, J. J. Wilke, N. Slattengren, and J. C. Bennett, "Exploring asynchronous many-task runtime systems toward extreme scales," tech. rep., Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2015.

[8] H. Kaiser, P. Diehl, A. S. Lemoine, B. Lelbach, P. Amini, A. Berge, J. Biddiscombe, S. R. Brandt, N. Gupta, T. Heller, K. A. Huck, Z. Khatami, A. Kheirkhahan, A. Reverdell, S. Shirzad, M. Simberg, B. Wagle, W. Wei, and T. Z. 0009, "Hpx - the c++ standard library for parallelism and concurrency," *J. Open Source Software*, vol. 5, no. 53, p. 2352, 2020.

[9] D. Staal, “Kinematic-wave.” `https://github.com/DockStaal/Kinematic-Wave`, 2022.