EPiC Series in Engineering
Volume 3, 2018, Pages 788–799
HIC 2018. 13th International Conference on Hydroinformatics

Parallel simulations of shallow water solvers for modelling overland flows

Bobby Minola Ginting[1]*, Ralf-Peter Mundani[2], and Ernst Rank[3][4]

[1] Chair for Computation in Engineering, Technical University of Munich, Arcisstr. 21 80333 Munich, Germany bobbyminola.ginting@tum.de
[2] Chair for Computation in Engineering, Technical University of Munich, Arcisstr. 21 80333 Munich, Germany mundani@tum.de
[3] Chair for Computation in Engineering, Technical University of Munich, Arcisstr. 21 80333 Munich, Germany ernst.rank@tum.de
[4] Institute for Advanced Study, Technical University of Munich, Lichtenbergstr. 2a, 85748 Garching, Germany

Abstract

In this paper, we simulate overland flows using a cell-centred finite volume (CCFV) method with three solvers – HLLC, central-upwind, and artificial viscosity schemes – and present their parallel efficiency within the framework of our in-house code called NUF-SAW2D. The parallel efficiency is investigated through four explicit time stepping schemes – the first-order Euler and the second, third, and fourth-order Runge Kutta methods. The bed slope terms are solved using a Riemann-solver-free technique and the friction terms are treated semi-implicitly for all time stepping schemes, allowing our model to simulate wet-dry problems with very low water depths on very rough beds. A second-order spatial accuracy is achieved with the Monotonic Upstream-Centred Scheme for Conservation Laws (MUSCL) linear interpolation technique. Since our code is based on an edge-based data structure, the edge-based MinMod limiter function is used to enforce the monotonicity of reconstructed variables. Our code shows accurate results and achieves a good scalability for up to 3.2 million cells or 6.4 million edges using OpenMP parallelisation technique.

1 Introduction

Numerical models based on the shallow water equations (SWEs) have been extensively used as tools to study the flow characteristics such as in rivers, channels, coastal regions, and floodplains. In simulating floodplains, one has to deal with wet-dry problems. The source of flows usually comes from rainfall or point source inflow treated as a source term in the SWEs. In reality, although heavy rainfall occurs, the changes of water depth over a catchment area do not occur instantaneously, thus allowing very low water to flow on very rough beds. This task is very challenging to be simulated since excessive drag forces caused by such rough beds may often occur on the very low water, which can produce negative depths for the models. Therefore, a

*corresponding author

proper solver is required to calculate the convective fluxes of the SWEs, which can take wet-dry interfaces into account accurately. In addition, the bed slope and the friction source terms must also be calculated with the methods, which are able to balance the fluxes and are capable of preventing the excessive drag forces for such wet-dry interfaces. Another problem that might be found for modelling such overland flows in real life applications is that one has to deal with very large domains supporting the high resolution of small-scale flow patterns, where the areas that will be inundated or will remain dry cannot be exactly predicted at the first step of simulation process. For this, a parallel simulation becomes inevitable to save computational time.

In this paper, we investigate three shallow water solvers to deal with overland flows. One of them is a Riemann solver: the HLLC method. Successful applications for overland flows using Riemann solvers were given in [2, 10, 11]. The other one is the central-upwind (CU) method, which is categorised as a Riemann-solver-free scheme and has been proven to be robust [8]. Another solver that we have recently developed in [4, 6, 5] uses an artificial viscosity (AV) technique, which can also be considered as a Riemann-solver-free scheme and has been proven to be accurate to solve the SWEs including the most complex flow phenomena, such as shock waves, transcritical flows, and wet-dry problems. As we pointed out in [6], the AV technique entails a much less computational complexity and was 1.216-times cheaper than the HLLC method but 1.064-times slightly more expensive than the CU scheme for the uses of the first-order (spatial accuracy) method and the Runge-Kutta fourth-order (RKFO) scheme. In our recent work [5], for the uses of the MUSCL technique and the RKFO scheme, it was found that the AV technique became 1.37-times, 1.34-times, and 1.1-times faster than the HLLC, Roe, and CU schemes, respectively. Both the results in[6, 5] were achieved using a single node. The comparisons in a framework for parallel simulation have not been done yet and are thus worth to be investigated particularly for millions of cells.

Another novelty aspect of this study is that we would like to investigate the parallel efficiency of the aforementioned schemes for four explicit time stepping methods – the Euler first-order (EFO) and the Runge-Kutta second (RKSO), third (RKTO), and fourth-order (RKFO) methods – using OpenMP parallelisation technique. An EFO scheme is undoubtedly simple to implement but it might have stability issues for wet-dry problems. To avoid such stability issues, one has typically to use a low Courant-Friedrichs-Lewy (CFL) number, e.g. 0.5, resulting in small time steps that obviously increase the duration of computational time. Conversely, the high-order methods are relatively more complex but can ensure stability using a higher CFL number. This CFL-number problem becomes a phenomenon as though it is a trade-off between the easiness in writing a code, the stability, and the duration of simulation process. In a parallel simulation, this phenomenon might have different characteristics thus a low-order time stepping scheme can computationally outperform a high-order one.

We use in this paper our in-house code called NUFSAW2D (Numerical simUlation of Free surface ShAllow Water 2D) – which is currently using OpenMP – to simulate overland flows over 0.4–3.2 million cells or 0.8–6.4 million edges, showing the exploitation of the advantage of using a parallel code for a large event compared to a sequential one. The scalability of our code is investigated on a 16-core machine with respect to weak scaling. We organise this paper as follows. In Sect. 2 the methods are concisely discussed and in Sect. 3 we explain the parallelisation strategy of our code. In Sect. 4 the model results are presented, for which our code is tested to simulate a real flood case in Glasgow, UK due to rainfall and surcharge culvert. Finally, conclusions are given in Sect. 5.

2 Mathematical formulation

Integrating spatially over a control cell Ω and estimating the line integral based on the Gauss divergence theorem, the SWEs are written as [4, 6, 5]

$$\frac{\partial}{\partial t}\iint_{\Omega} \mathbf{W} d\Omega + \sum_{i=1}^{N} \left(\mathbf{F}\, n_x + \mathbf{G}\, n_y\right)_i \Delta L_i = \iint_{\Omega} \left(R - I + \mathbf{S}_b + \mathbf{S}_f\right) d\Omega\ , \tag{1}$$

where N is the total number of edges for a cell, ΔL is the length of edge, and n_x and n_y denote the normal vectors outward from the edge in x and y directions, respectively. Since rectangular grids are used in our computations, $N = 4$. All the matrices in Eq. 1 are given by Eq. 2. The variables h, u, v, g, z, and n_m denote respectively the water depth, velocity in x and y directions, gravity acceleration, bed contour, and Manning coefficient, where $c_f = g\ n_m^2\ h^{-\frac{1}{3}}$ denotes the friction factor. Meanwhile, R and I are the rainfall and infiltration, respectively. We also define the water elevation as $\eta = h + z$. Since a CCFV method is used as spatial discretisation scheme, all the aforementioned variables are defined at the centre of each cell.

$$\begin{gathered}\mathbf{W} = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix}\ ,\ \mathbf{F} = \begin{bmatrix} hu \\ huu + \dfrac{gh^2}{2} \\ hvu \end{bmatrix}\ ,\ \mathbf{G} = \begin{bmatrix} hv \\ huv \\ hvv + \dfrac{gh^2}{2} \end{bmatrix}\ , \\ \mathbf{S}_b = \begin{bmatrix} 0 \\ -gh\dfrac{\partial z}{\partial x} \\ -gh\dfrac{\partial z}{\partial y} \end{bmatrix}\ ,\ \mathbf{S}_f = \begin{bmatrix} 0 \\ -c_f u\sqrt{u^2+v^2} \\ -c_f v\sqrt{u^2+v^2} \end{bmatrix}\end{gathered} \tag{2}$$

The convective fluxes $\mathbf{F}$ and $\mathbf{G}$ in Eq. 1 come from the line integral of the boundary of a control cell Ω, creating a local Riemann problem so that a robust solver is required to compute the fluxes at the centre of the edge. For this, three solvers are investigated and we refer to [2, 10, 11, 8, 4, 6, 5] and literature cited therein. Prior to applying such solvers, we reconstruct the left and right Riemann states at edges using the MUSCL technique with the edge-based MinMod limiter function following [3]. The bed slope source terms $\mathbf{S}_b$ are solved using a Riemann-solver-free technique utilising a partial derivative computation, which is neither an upwinding scheme nor a Riemann solution. The friction source terms $\mathbf{S}_f$ are calculated semi-implicitly, which is very simple to be applied but has proven to be robust to avoid stability issues for very low water depths on very rough beds. We refer to [6] for both these source terms. To obtain $\mathbf{W}$ for the subsequent time level, four explicit time stepping schemes are compared, in which the computations of both $\mathbf{S}_b$ and $\mathbf{S}_f$ are similar for all time stepping schemes following their own calculation steps. We refer to [4, 6, 3].

3 Parallelisation strategy

To explain the parallelisation concept of our code, let us now consider a computational domain with 4×5 segments consisting of 20 cells and 49 edges (24 and 25 edges in x and y directions, respectively) shown in Fig. 1. Within a CCFV's framework, two approaches can in general be applied to solve Eq. 1: cell-based and edge-based manner. In a cell-based manner, the loops for calculating $\mathbf{W}$ at the subsequent time level are performed directly for all 20 cells. The

OpenMP parallelisation technique is thus easy to be implemented in this approach, e.g. if one uses 4 threads, each thread is assigned the same amount of cells (5 cells). However, about half of the computations of fluxes $\mathbf{F}$, $\mathbf{G}$, and $\mathbf{S}_b$ at shared edges are useless since those calculations are performed twice – with the same value but different algebraic sign – for two adjacent cells corresponding to the same edge. The computational time thus increases. Another issue might deal with a lower parallel efficiency. This is because the high aspect ratio between the total number of boundary cells (e.g. cells 1–5, 6, 10, 11, 15, 16–20) and the internal cells (e.g. cells 7–9, 12–14) influences the parallel efficiency significantly. For instance, cell 7 requires more CPU time than does cell 6 since all edges corresponding to cell 7 must be calculated using the (aforementioned) solvers, whereas no solver is required to compute edge 18 of cell 6 as this edge is categorised as wall boundary. This obviously results in a load imbalance for a parallel simulation due to a different complexity level of algorithm.

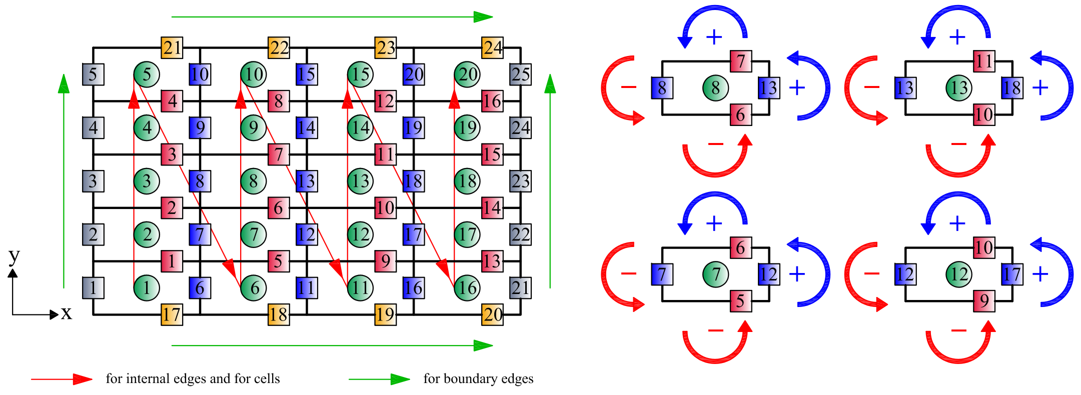


Figure 1: The concept of cell-edge reordering strategy in NUFSAW2D

In an edge-based manner, however, the loops for the fluxes $\mathbf{F}$ and $\mathbf{G}$ as well as $\mathbf{S}_b$ at edges are firstly employed for 49 edges. We call this stage as edge-level. This means, all computations of such fluxes are only performed once and later are used for calculating $\mathbf{W}$ at cells by summing them with the proper algebraic signs (cell-level). Since the redundant computations are avoided, this edge-based manner requires definitely less CPU time than does the cell-based one. It is yet a challenging task that must be revealed in providing an index relationship between cells and edges to support the efficiency of the edge-and-cell-levels-driven computations. This is how the idea of our cell-edge reordering strategy emerged in this paper as well as in [7]: to allow better memory access, to ensure that all threads work efficiently, and to avoid unnecessary waiting times due to data-dependency problems.

We firstly arrange the cell numbering following the Z-pattern as shown in Fig. 1 This numbering is addressed for the cell-level-driven computations. Since rectangular grids are used, edges can simply be classified into two types: the x and y directions. To deal with the high aspect ratio between the total number of boundary and internal cells, we distinguish the internal edges (which have two adjacent cells, e.g. edges 1–16 and 6–20 in x and y directions, respectively) from the boundary edges (which only have one adjacent cell, e.g. edges 17–24 in x direction and 1–5, 21–25 in y direction). This diversification aims to classify the computation procedures based on their own complexity levels. For example, the MUSCL reconstruction technique is applied for such internal edges (to achieve a second-order accuracy), but not for wall boundary edges (as at these edges, the normal fluxes must remain zero and the depth is simply taken similar to the corresponding cell value, thus only achieving a first-order accuracy). The procedures for the internal edges require therefore more CPU time than those for the wall boundary edges. Without this classification, the load imbalance would be more vulnerable to occur among threads. It is now clearly shown that each edge has a certain pattern concerning its index-relationship to cell as well as the index-relationship of cell to edge. These certain indexes will obviously help ease the compiler to exploit the instruction pipelining (for vectorisation) on node-level. Regarding this indexing system, the fluxes $\mathbf{F}$, $\mathbf{G}$, and $\mathbf{S}_b$ in Eq. 1 are calculated according to edges 1–24 in x direction and 1–25 in y direction, whereas the terms R, I, and $\mathbf{S}_f$ are computed according to cells 1–20.

It is important to note here that the arrays of $\mathbf{S}_b$ are not only saved according to the aforementioned 49 edges, since the bed slope fluxes may have two different values for each edge especially when dealing with wet-dry interfaces (for calculation details refer to [6]). At edge 12 (internal edge in y direction) – due to the hydrostatic and topography reconstructions in [6] – those bed slope fluxes must be saved twice to include the effects of cells 7 and 12. However, at edge 18 (boundary edge in x direction), one only needs to save those fluxes once as such an edge only has one corresponding cell (cell 6). Therefore, according to Fig. 1, one has to serve arrays for $\mathbf{S}_b$ in the edge-level-driven computations according to 49 edges + 31 (additional) edges. The next step is to do the cell-level-driven calculations after the edge-level-driven computations are totally finished. To this end, a synchronisation is required among threads. The procedures in the cell-level are simpler than those of the edge-level, since this is basically achieved by summing all the corresponding fluxes $\mathbf{F}$, $\mathbf{G}$, and $\mathbf{S}_b$ with the proper signs and using the values of R, I, and $\mathbf{S}_f$ from the previous time level. This is illustrated in Fig. 1.

We illustrate in Fig. 2 the load distribution of the computational domain from Fig. 1 using 4 threads based on the parallel algorithm in our code NUFSAW2D. Indeed, some load imbalance issues still appear in the edge-level-driven computations. With a static load balancing (default), thread 0 gets the most amount of loads (9 internal + 6 boundary edges), whereas thread 3 receives in contrast the least amount of loads (4 internal + 4 boundary edges). This leads to an unbalanced load distribution and may decrease the parallel efficiency. This is the first reason of the load imbalance issues in our implementation. However, as the numbers of edges and cells increase, e.g. to millions of cells, the effects of these unbalanced load distributions become less and less significant. It is also possible to apply another load balancing directive such as dynamic or guided. These directives can enhance the performance of static at a cost of more overheads due to the distributions of the different load packages during runtime.

It should be noted for wet-dry problems, although one could precisely allocate the same load distributions to all threads in the edge-level, load imbalance issues may still exist. This is because for wet-dry and dry-dry interfaces, the scheme turns to a first-order scheme – so that no MUSCL reconstruction technique is required – in contrast to wet-wet interfaces which could either be a second-order scheme if no discontinuous flow exists or be a first-order scheme when dealing with discontinuity. Typically, the second-order scheme is 1.7–2-times more expensive per time-step than the first-order one. To accomplish the algorithm for such wet-wet, wet-dry, and dry-dry interfaces, which can only be known throughout simulation time, the nested branch statements (if-then-else) shown in Alg. 1 become inevitable. This is the second reason of the load imbalance issues occurring in our study.

In Fig. 2, it has been shown that our cell-edge reordering strategy can ensure a balanced load distribution in the cell-level-driven computations. However, a load imbalance may still emerge in this level due to wet-dry problems. As previously explained, since the friction terms $\mathbf{S}_f$ are treated semi-implicitly following [6], it is required to transform the unit discharges back

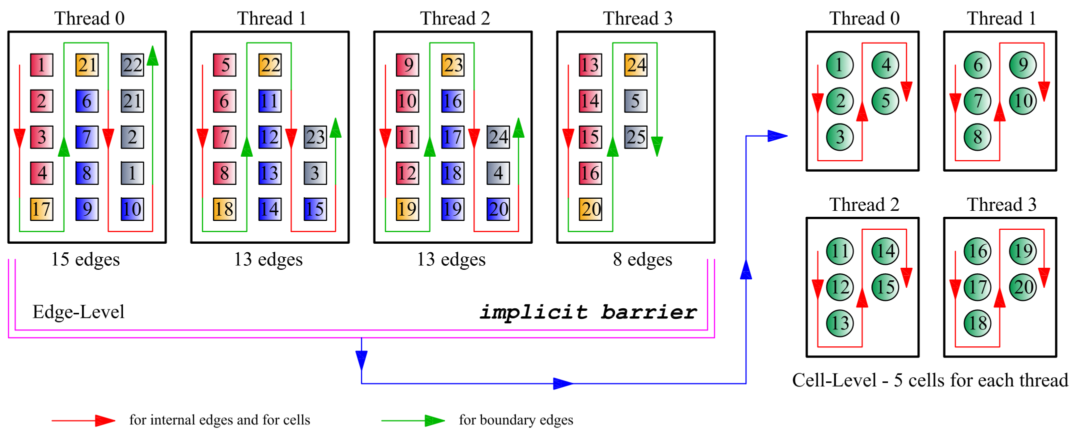


Figure 2: Load distributions among threads with a static load balancing in NUFSAW2D

Algorithm 1 Classification of wet-wet, wet-dry, and dry-dry interfaces in the edge-level

```
1: if wet-dry or dry-dry interfaces at edges then
2:     calculate first-order scheme
3: else
4:     calculate second-order scheme
5:     if reconstructed variables are not monotone then
6:         calculate first-order scheme
7:     end if
8: end if
```

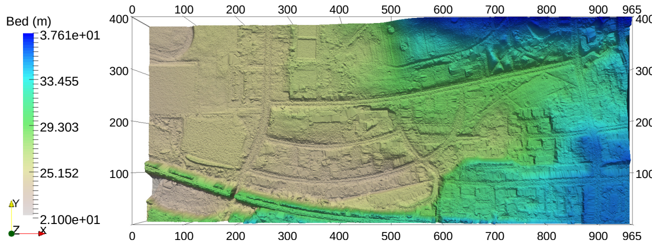
to the velocities at each calculation step of the EFO, RKSO, RKTO, or RKFO scheme. For a division by a very low depth especially for dry or almost-dry cells, some numerical oscillations may arise. To anticipate this, a simple treatment is devised in Alg. 2. Consequently, the model may again suffer from a load imbalance issue due to the uncertainty of the total number of wet and dry cells throughout simulation time. This becomes the third reason of why our model may suffer from load imbalances. Yet, we will show in Sect. 4 that we can still achieve good scalability.

Algorithm 2 Wet-dry treatment in the cell-level

```
1: if depth at the subsequent (time or calculation) level $\leq 10^{-6}$ m then
2:     unit discharges and velocities are simply set to zero
3: else
4:     transform unit discharges back to velocities for the friction terms
5: end if
```

4 Results

We present here results of a flood simulation for Glasgow, UK with 0.4–3.2 million cells (0.8–6.4 million edges). A Linux computing cluster operated at the chair for Computation in Engineering, Technical University of Munich was used. More detailed information on the machine can be found in [1]. Our code was written in Fortran. We compiled our code using Intel Fortran 17.04 on the Ubuntu 12.05.5 LTS operating system with –O3 optimisation flag and used double-precision arithmetic. The data was obtained from [9] and provided in the Digital Elevation Model (DEM) format, of which the size is 965×401 m and the ground elevations vary within the range of $+21$ m to $+37.61$ m as given in Fig. 3. This shows a very complex topography shape. Dry bed condition is set initially at all cells, for which all boundaries of the simulated area are set as wall boundaries. The Manning coefficients range between 0.02–0.05 s $\mathrm{m}^{-1/3}$ showing that this case deals with the very rough bed. The flow sources are generated by a rainfall hydrograph, uniformly distributed to all parts of the domain without infiltration and a surcharge culvert flow at the coordinate of (919.75, 337.75) m, both of which are shown in Fig. 3. No observation data was provided; a comparison of several 1D, 2D, and 3D models was given instead. In this paper, we select the results of those 2D models: ISIS 2D, MIKE FLOOD, SOBEK, TUFLOW, and XPSTORM and compare them with our model. For more detailed information, see [9].



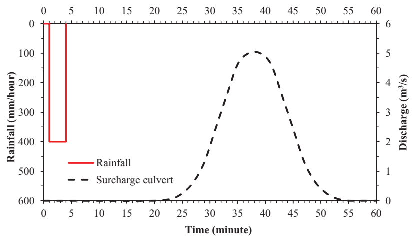


Figure 3: Topography data (left) and flow sources hydrographs (right)

At points P2 (561.75, 255.75) m and P6 (595.75, 145.75) m, we compare our results with those given in [9]; it should be noted that the simulation results in [9] are only available using 98,000 cells. The adaptive time step Δt is calculated as Eq. 3, where Δx and Δy are the cell size. The time step Δt is limited by the CFL numbers of 0.5–0.9. To ensure stability, we also use a (quite high) value Δt_{dry} to limit Δt not being too high especially at the beginning of the simulation time, where the completely dry bed must be simulated without any input flow (see the flow source hydrographs in Fig. 3).

$$\Delta t = \min\left(\Delta t_{\mathrm{dry}}, \mathrm{CFL}\frac{\min(\Delta x, \Delta y)}{\sqrt{u^2+v^2}+\sqrt{gh}}\right), \tag{3}$$

If Δt_{dry} is not applied, some significant oscillations may appear at the beginning of the simulation, which can afterwards affect the accuracy, although the computations might still be stable. We found out the proper value for Δt_{dry} is 1 s considering the largest total number of cells used in this study (3.2 million cells).

The total simulation time is set to 5 hours. For the sake of simplicity, we present our results here only with the RKSO method using 3.2 million cells and CFL = 0.5 as presented in Fig. 4.

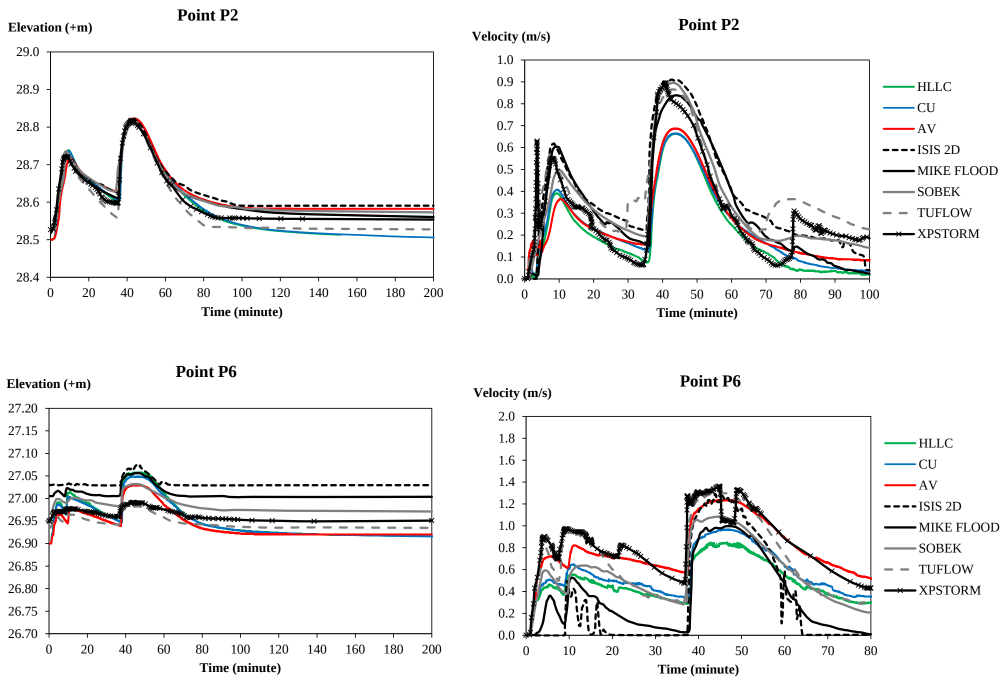


Figure 4: Results at points P2 and P6 using 3.2 million cells with the RKSO method

One can see that a double-peaked shape hydrograph is shown by our results at both points P2 and P6 similar to those given in [9]. This is caused by the very intense rainfall during minutes 1–4. At point P2, all the schemes in our model predict the first and second peak values similarly for both the depth and the velocity. After minute 65, the AV scheme computes a slightly higher water depth of approximately 4 cm from those of the HLLC and CU schemes. For the velocity, the HLLC scheme estimates the lowest velocities during minutes 15–30 and minutes 65–100, whereas the AV scheme yields the highest velocities after minute 65. At point P6, no significant differences are shown between the HLLC and CU schemes for water depth. Meanwhile, the AV scheme predicts the first and second peaks of the hydrograph with the lower values of approximately 2 cm and 3 cm, respectively. The quite significant differences are however shown for the velocity, in which the AV scheme calculates the second peak value 0.4 m/s higher than does the HLLC scheme. Unfortunately, with regard to accuracy, we cannot decide which one of the three schemes is the most accurate one, since no observed result was provided. However, we have presented that our model is in agreement to the other models given in [9]. In Fig. 5, the flood inundation areas at minutes 15 and 300 using the CU scheme are highlighted.

Regarding the parallelisation, we perform a weak scaling, where the number of cells increases linearly with the number of threads. For 2, 4, 8, 12, and 16 threads, the segment configurations used are 982×408, 1390×576, 1966×814, 2403×999, and 2779×1152, respectively. For the EFO method, we limit CFL = 0.5 (as some oscillations appeared when using CFL = 0.9, although the computation was still stable), whereas for the RKSO, RKTO, and RKFO methods, CFL = 0.9 is used. To become independent from a pure computational time, we base our parallel study on a performance metric (PM) expressed as Mcell/s/core (million cells per second per core)

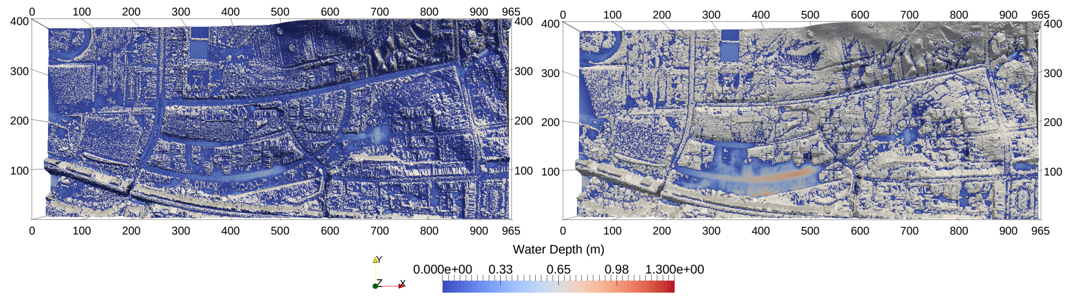


Figure 5: Inundation area using 3.2 million cells at minute 15 (left) and minute 300 (right)

which is a comparison between the number of cells for a total number of calculation steps that can be processed per unit of time using one core; if this metric remains constant for different numbers of threads, the code scales well. We calculate the PM value for the first 60 minutes, meaning that each combination may have a different number of time/calculation levels to reach the time target – and evaluate it five times. The truncated mean procedure is thus used, so that an average CPU time value for each combination can be obtained. To get an unbiased percentage of parallel efficiency for each time stepping method, we use its highest PM value with 2 threads among the three solvers as an absolute parameter (in this case, since the CU scheme always achieves the best performance, its PM value with 2 threads in each time stepping method becomes the absolute parameter). The dynamic directive is used for load balancing purpose.

The weak scaling results are given in Fig. 6. It is shown that for the EFO method, the CU scheme achieves the highest performance metric among the others (5.7 Mcell/s/core using 2 threads), whereas the HLLC and AV schemes acquire 4.7 and 4.2 Mcell/s/core, respectively. This result is in contrast to that obtained in [5], where the AV scheme could outperform the HLLC and CU schemes for the use of the RKFO method with a single core machine. One can see, as the number of cells becomes 2779×1152 as well as the number of threads increases to 16, the CU scheme loses performance of approximately 25% from 5.7 to 4.2 Mcell/s/core. It is probably due to an exhaustively optimisation for 2 threads by the compiler. This phenomenon can further be explained by the results of 12 and 16 threads which almost remain constant within the range of 74–75%. A similar fact is also shown by the other time stepping schemes RKSO and RKTO methods by acquiring (almost) constant efficiency for the uses of 12 and 16 threads. Using 16 threads, the EFO, RKSO, and RKTO methods can achieve a sufficient parallel efficiency of higher than 70%.

Indeed, the RKSO and RKTO methods require less time levels due to the uses of higher CFL numbers, thus suffering from fewer overheads of the OpenMP synchronisation. However, they still slightly obtain lower parallel efficiency than that of the EFO method. This is because in one time level, these methods require several calculation levels, with which the effects of the three possible aforementioned load imbalance issues may become significant to reduce the parallel efficiency. An interesting exception is shown by the RKFO method which can attain a parallel efficiency of approximately 77% using 16 cores by the CU scheme. This phenomenon might show that less time levels due to higher CFL numbers may sometimes be beneficial for parallel efficiency purpose despite having to deal with more possibilities of load imbalance issues due to a greater number of calculation levels.

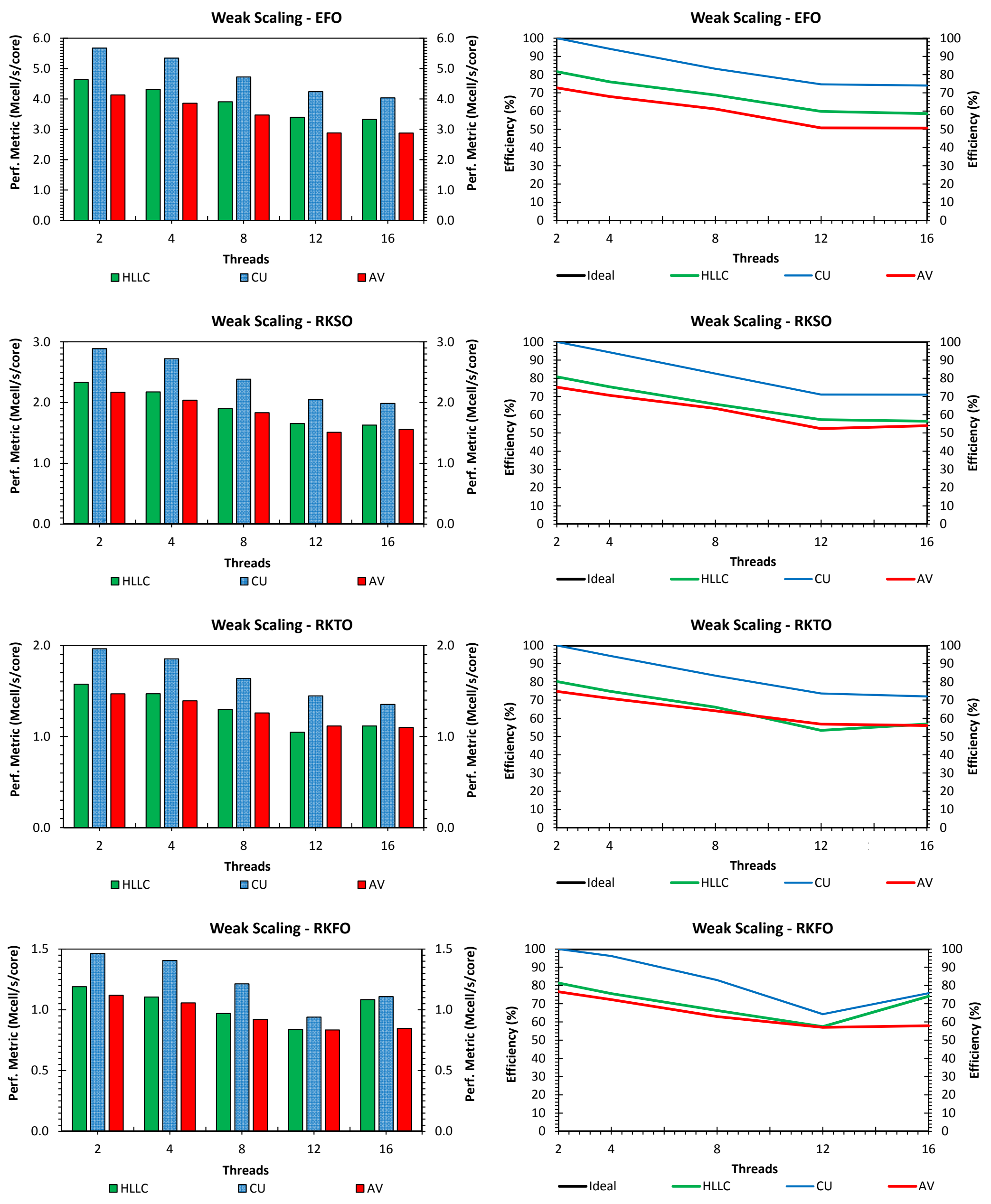


Figure 6: Results of weak scaling of NUFSAW2D

5 Conclusions

A parallel study of three shallow water solvers – HLLC, CU, and AV schemes – has been presented for overland flows using an in-house code NUFSAW2D. The spatial discretisation scheme was a CCFV method, in which the second-order accuracy was achieved with the MUSCL reconstruction technique limited by the edge-based MinMod limiter function. Four time stepping schemes were investigated: the EFO, RKSO, RKTO, and RKFO methods.

We have shown that our code can simulate the overland flows of the real flood case in Glasgow, UK accurately in comparison to other numerical models. No stability problem was found to simulate very low depths on very rough beds due to the semi-implicit formulation of the friction source terms. In addition to accurate results, we were also able to show that using the EFO method our code with the CU scheme scaled well for a parallel efficiency of 86% using 8 threads (for 1.6 million cells/3.2 million edges). Using 12 threads (for 2.4 million cells/4.8 million edges) and 16 threads (for 3.2 million cells/6.4 million edges), the parallel efficiency remained almost constant for 75%, showing that our code is efficiently parallelisable and scales well for millions of cells. This could be achieved due to a better memory access given by the cell-edge reordering strategy devised in our code that also helped ease the compiler to exploit the instruction pipelining and parallelisation. Finally, we conclude that for a parallelisation study, solvers with a first-order time stepping may be more advantageous to achieve a better parallel efficiency with a higher performance metric.

6 Acknowledgements

The first author gratefully acknowledges the DAAD (German Academic Exchange Service) for supporting his research in the scope of the Research Grants – Doctoral Programmes in Germany 2015/16 (57129429). The authors would also like to thank Prof. Michael Manhart and Florian Mintgen from the Chair of Hydromechanics (Technical University of Munich) for invaluable discussions. The authors are also grateful to all the anonymous reviewers who provided many constructive comments and suggestions.

References

[1] https://sandstorm.inf.bauwesen.tu-muenchen.de/wiki/index.php/Sandstorm-Cluster_(En). (accessed on Mar. 01, 2018).

[2] L. Cea and E. Blade. A simple and efficient unstructured finite volume scheme for solving the shallow water equations in overland flow applications. *Water Resour. Res.*, 51:5464–5486, 2015. https://doi.org/10.1002/2014WR016547.

[3] A. I. Delis and I. K. Nikolos. A novel multidimensional solution reconstruction and edge-based limiting procedure for unstructured cell-centered finite volumes with application to shallow water dynamics. *Int. J. Numer. Meth. Fl.*, 71(5):584–633, 2013. https://doi.org/10.1002/fld.3674.

[4] B. M. Ginting. A two-dimensional artificial viscosity technique for modelling discontinuity in shallow water flows. *Appl. Math. Model.*, 45:653–683, 2017. https://doi.org/10.1016/j.apm.2017.01.013.

[5] B. M. Ginting and H. Ginting. Invesigation of an artificial viscosity technique in a second-order cell-centred finite volume model for unsteady shallow water flow simulations. 2018. unpublished.

[6] B. M. Ginting and R.-P. Mundani. Artificial viscosity technique: A riemann-solver-free method for 2d urban flood modelling on complex topography. In P. Gourbesville, J. Cunge, and G. Caignaert, editors, *Advances in Hydroinformatics*. Springer Water, Singapore, 2018. https://doi.org/10.1007/978-981-10-7218-5_4.

[7] B. M. Ginting and R.-P. Mundani. Parallel flood simulations for wet-dry problems using dynamic load balancing concept. *J. Comput. Civil Eng.*, 2018. under review.

[8] A. Kurganov and G. Petrova. A second-order well-balanced positivity preserving central-upwind scheme for the saint-venant system. *Commun. Math. Sci.*, 5(1):133–160, 2007. https://projecteuclid.org:443/euclid.cms/1175797625.

[9] S. Neelz and G. Pender. *Benchmarking the latest generation of 2D hydraulic modelling packages*. Environment Agency, Horison House, Deanery Road, Bristol, BS1 9AH, 2013.

[10] I. Oezgen, J. Zhao, D. Liang, and R. Hinkelmann. Urban flood modeling using shallow water equations with depth-dependent anisotropic porosity. *J. Hydrol.*, 541:1165–1184, 2016. https://doi.org/10.1016/j.jhydrol.2016.08.025.

[11] X. Xia, Q. Liang, X. Ming, and J. Hou. An efficient and stable hydrodynamic model with novel source term discretization schemes for overland flow and flood simulations. *Water Resour. Res.*, 53:3730–3759, 2017. https://doi.org/10.1002/2016WR020055.