

Parallelization of a Non-Hydrostatic Shallow Water Model in Sam(oa)²

Guided Research Project

Raphael Schaller

Fakultät für Informatik

Technische Universität München

Supervisor: Univ.-Prof. Dr. Michael Bader

Advisor: Dipl.-Inf. Oliver Meister

October 26, 2015

This paper discusses extension of the non-hydrostatic shallow water equations towards solving large problems on HPC architectures. We describe how the previous implementation of the non-hydrostatic shallow water equations in the PDE framework sam(oa)² is adapted in order to support parallel execution. Strong scaling as well as weak scaling results are presented, which show a parallel efficiency of about 96.8% for strong scaling on 512 cores and 90.1% for weak scaling on 8192 cores. To further improve performance, we examine and successfully demonstrate the application of the Conjugate Gradient Method—instead of the previously used Jacobi method—for solving the system of linear equations.

1 Introduction

Tsunami simulation is a frequently researched topic in computational science. Usually, the 2-dimensional hydrostatic shallow water equations (SWE) are employed for simulation of wave propagation. However, because of being hydrostatic, they neglect the vertical momentum equation which is feasible only for waves for which $h/\lambda \ll 1$ applies, where λ is the wave length and h

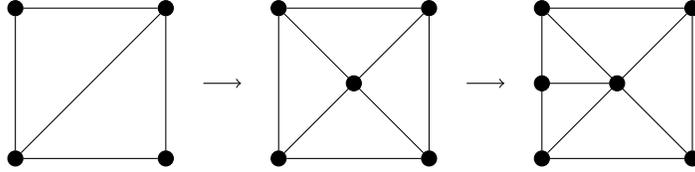


Figure 1: Refinement process of the triangular grid in $\text{sam}(\text{oa})^2$.

the height of the water column [1]. While this is true for waves in propagation phase, at coastal regions, this assumption does not apply anymore. In order to also consider effects influencing waves at coastal regions, the *non-hydrostatic* SWE are utilized, which regard the vertical momentum equation while maintaining a 2D-model.

In [2], the non-hydrostatic SWE were implemented as an extension to the hydrostatic SWE using the PDE framework $\text{sam}(\text{oa})^2$ ([3, 4]). While $\text{sam}(\text{oa})^2$ is targeted at HPC environments, the previous implementation of the extension did not support parallelism. In order to extend the non-hydrostatic SWE extension towards solving large problems, the goal of this paper is the parallelization of it. In addition, to further improve performance, we examine the use of the Conjugate Gradient Method for solving the system of linear equations.

This paper is structured as follows: First, we will briefly introduce $\text{sam}(\text{oa})^2$ in Section 2 and subsequently show how the non-hydrostatic extension is realized (Section 3). In Section 4, the parallelization is discussed. Following this, we examine the Conjugate Gradient Method in Section 5. The results of our work are finally presented in Section 6.

2 $\text{Sam}(\text{oa})^2$

$\text{Sam}(\text{oa})^2$ (Space Filling Curves and Adaptive Meshes for Oceanic and Other Applications)¹ [3, 4] is a parallel framework for solving partial differential equations. In the following, features of $\text{sam}(\text{oa})^2$ which are essential to this paper will be briefly introduced. For a more elaborate explanation, please refer to [3, 4].

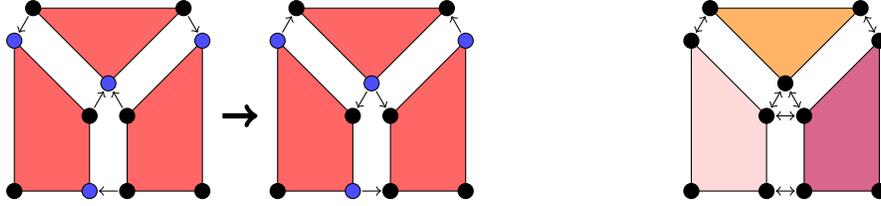


Figure 2: Merging process. Left two pictures: Merging sections owned by one process. Rightmost picture: Merging sections owned by multiple processes. Color of sections denote their process affiliation. The blue marked nodes are master nodes. The figure is based on [4].

2.1 Grid

The dynamically adaptive triangular grid is an integral part of $\text{sam}(\text{oa})^2$. The initial setup are two triangles next to each other (see left picture of Figure 1). The triangles are then split recursively via newest vertex bisection [5], thus fulfilling the constraint that each triangle is adjacent to exactly three nodes; therefore, in the first refinement step, both triangles have to be split. At runtime, a minimum and maximum refinement depth (d_{min} and d_{max} , respectively) can be defined. Between these limits, a refinement criterion determines whether a cell has to be further refined or coarsened. The refinement and coarsening process is repeated for every time step.

$\text{Sam}(\text{oa})^2$ uses the Sierpiński space filling curve (SFC) to define an order, in which the cells are traversed, enabling a cache efficient traversal. In order to support parallelism, the SFC is split into so-called Sierpiński sections which form atomic workloads for each thread.

2.2 Framework Design

The framework is traversal based, meaning that each step of a simulation algorithm is processed in its own traversal. Within the traversals, an element-wise event approach is utilized: for each traversal, event handlers can be registered. The event handlers relevant to this paper are the `CELL_ELEMENT_OP`, `MERGE_OP` and `NODE_LAST_TOUCH_OP`. The `CELL_ELEMENT_OP` is called one per cell and is usually used to perform the main calculations. The `NODE_LAST_TOUCH_OP` is called once for each node at the very end of the traversal.

The `MERGE_OP` is central when it comes to parallelism. As described

¹<https://github.com/meistero/Samoa>

in the previous section, `sam(oa)`² splits the grid into atomic sections, each of which is processed by one thread. A section’s border nodes and edges which belong to multiple sections are duplicated (see Figure 2). Therefore, the partial results stored in each duplicated node (edges are not relevant in our case) need to be merged and distributed so that all duplicates stay consistent.

The `MERGE_OP` is executed after the `CELL_ELEMENT_OP` but before the `NODE_LAST_TOUCH_OP`. The arguments of the operator are two border nodes, the local node and the neighbor node. Within the operator, the two nodes are merged and the result is stored in the local node whereas the neighbor node is read-only.

As depicted in Figure 2, the merging itself is performed differently depending on whether the sections sharing a border node are owned by only one or multiple processes. If they are owned by one process (left two pictures), associated border nodes are merged in a master/slave fashion: First, all slave nodes are merged into the master node (marked blue), yielding the final result which is then copied back to all slave nodes. If the sections are owned by multiple processes (rightmost picture), the merging operator is executed in a peer-to-peer fashion, so for all combinations of associated nodes. Both ways are performed automatically by `sam(oa)`².

3 The Non-Hydrostatic Model

In this section, the non-hydrostatic model will be introduced briefly. See [2, 6] for the complete derivation and a detailed discussion. This section follows [2] closely.

Before introducing the non-hydrostatic model, we shall shortly review the hydrostatic SWE implementation in `sam(oa)`². The SWE subpackage aims at solving the 2D-hydrostatic shallow water equations by using the finite volume method, as for instance described in [7]. A state vector $Q_i^n = (h_i^n, (hU)_i^n, (hV)_i^n)$ is placed in every cell i . For each time step $Q_i^n \rightarrow Q_i^{n+1}$, the Riemann problems on the cell edges are solved and the state vector is updated.

3.1 Numerical Model

For the non-hydrostatic model, the hydrostatic shallow water equations have been generalized, resulting in the non-hydrostatic pressure approximated

shallow water equations in conservation law form:

$$\begin{bmatrix} h \\ hU \\ hV \\ hW \end{bmatrix}_t + \begin{bmatrix} hU \\ hU^2 + \frac{1}{2}gh^2 \\ hUV \\ hUW \end{bmatrix}_x + \begin{bmatrix} hV \\ hUV \\ hV^2 + \frac{1}{2}gh^2 \\ hVW \end{bmatrix}_y = \begin{bmatrix} 0 \\ -ghb_x - \left(\left[\frac{h\hat{q}}{2} \right]_x + \hat{q}b_x \right) \\ -ghb_y - \left(\left[\frac{h\hat{q}}{2} \right]_y + \hat{q}b_y \right) \\ \hat{q} \end{bmatrix} \quad (1)$$

Here, (U, V, W) represents the depth averaged velocity vector, $h(x, y) = \eta(x, y) - b(x, y)$ the height of the water column, $\eta(x, y)$ the water elevation above the mean sea level and $b(x, y)$ the bathymetry. Subscripts stand for the partial derivative in the respective direction. g represents the gravitational constant and \hat{q} the non-hydrostatic pressure at the sea bottom.

3.2 Temporal Discretization

The non-hydrostatic SWE extension applies a fraction step scheme. Firstly, the unaltered hydrostatic SWE solver is performed which is subsequently corrected by the non-hydrostatic pressure \hat{q} . The correction parameter \hat{q} is calculated by solving a system of linear equations, resulting from a pressure decomposition as shown in [8]. The correction formulas follow from an Euler time-discretization of Equation (1):

$$(hU)^{n+1} = (\widetilde{hU})^{n+1} - \Delta t \left(\left[\frac{1}{2}h^n \hat{q}^{n+1} \right]_x + \hat{q}^{n+1} b_x \right) \quad (2)$$

$$(hV)^{n+1} = (\widetilde{hV})^{n+1} - \Delta t \left(\left[\frac{1}{2}h^n \hat{q}^{n+1} \right]_y + \hat{q}^{n+1} b_y \right) \quad (3)$$

$$w^{n+1} = w^n + 2\Delta t \frac{\hat{q}^{n+1}}{h^{n+1}}. \quad (4)$$

Here, w is the vertical velocity at the surface of the sea. The tilde indicates the intermediate values which are computed by the hydrostatic SWE solver in the same time step.

3.3 Spacial Discretization

Due to the different sized triangular grid cells in $\text{sam}(\text{oa})^2$, it would not be clearly defined how to determine the mesh-width at a discontinuity between two different sized triangular cells. Thus, in order to avoid complications,

Algorithm 1: Non-hydrostatic SWE algorithm.

```
for( all time steps ) {
    adaptivity_traversal();
    euler_traversal();
    lse_traversal();
    solver();
    correction_traversal();
}
```

the two additional degrees of freedom (\hat{q} and w) are not placed in the cell center—in contrast to the non-hydrostatic SWE implementation in [6]—but in the cell nodes. These nodes then span a so called *dual cell* (see Figure 3). This enables a matrix-free solution of the system of linear equations. [2]

Finally, the algorithm is as follows (see Algorithm 1): First of all, the grid is refined/coarsened as explained in Section 2.1 (`adaptivity_traversal`). Following this, the regular Euler time step for the hydrostatic SWE is performed (`euler_traversal`). Then, the system of linear equations is set up (`lse_traversal`) which is solved afterwards (`solver`). Finally, the hydrostatic solution is corrected and w is updated (`correction_traversal`) (see Equations (2) to (4)).

4 Parallelization

Even though `sam(oa)`² greatly supports parallel execution of algorithms, the previous implementation of the non-hydrostatic SWE could only be executed sequentially [2]. In order to enhance performance and to enable simulation of huge scenarios which would not fit into a single node’s memory, we have adapted the implementation to support shared as well as distributed memory parallelism. In the following, the adaptations which were required will be explained.

4.1 Merge Operators

As described in Section 2.2, the `MERGE_OP` is an integral part for the parallelism of `sam(oa)`². For the non-hydrostatic SWE, we need to define two `MERGE_OP`s: one after setting up the equations and one after the correction step.

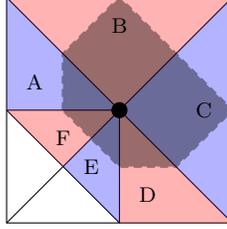


Figure 3: Dual cell. The colored regions are the regular triangular grid cells of $\text{sam}(\text{oa})^2$. The black shadow indicates the dual cell spanned by the center node. This figure is based on [2].

When setting up the system of linear equations, we traverse all cells of the domain and calculate—within the `CELL_ELEMENT_OP`—the 3×3 -matrix A and the right hand side vector p , yielding the system of equations $Ax = p$. While A is placed in the cell center and thus is not of concern here, p is distributed to the three adjacent nodes of each cell. Let M_n be the set of cells adjacent to a node n (for instance, in Figure 3, $M_n = \{A, B, C, D, E, F\}$ for the center node) and p_m the right hand side vector calculated within the `CELL_ELEMENT_OP` such a cell $m \in M_n$. Then, the resulting right hand side \hat{p}_n stored on the node n is calculated as follows:

$$\hat{p}_n = \sum_{\forall m \in M_n} p_{m;n} \quad (5)$$

where $p_{m;i}$ is the i th vector element of p_m .

Now, suppose the adjacent cells of node n are placed in different sections s_0, s_1, \dots, s_k . Let $\hat{M}_{n,i}$ be the set of adjacent cells of node n placed in section s_i , then $M_n = \hat{M}_{n,0} \cup \hat{M}_{n,1} \cup \dots \cup \hat{M}_{n,k}$. Hence,

$$\hat{p}_n = \sum_{\forall m \in M_{n,0}} p_{m;n} + \sum_{\forall m \in M_{n,1}} p_{m;n} + \dots + \sum_{\forall m \in M_{n,k}} p_{m;n}. \quad (6)$$

Therefore, the `MERGE_OP` has to sum up the partial right hand sides stored on each duplicated node:

```
local_node.p := local_node.p + neighbor_node.p
```

During the correction step, again, all cells are traversed. In the `CELL_ELEMENT_OP`, on the one hand, the discharges hU and hV are corrected, which are, however, stored in the cell itself. On the other hand, two helper variables are calculated which are later used in the `NODE_LAST_TOUCH_OP`

to update the vertical velocity w . These helper variables are stored on the nodes and therefore need to be merged.

The first variable is the sum of the inverse of the water height h_m of all cells m adjacent to a node weighted by the area $a_{n,m}$, which the cell contributes to the dual cell n :

$$h_{inv;n} = \sum_{\forall m \in M_n} (1/h_m) a_{n,m} \quad (7)$$

To illustrate, in Figure 3, we would sum up the inverse of the water height in each grid cell (colored regions) weighted by the area of the shadowed region of each cell.

The second variable denotes the total area of the dual cell n :

$$a_n = \sum_{\forall m \in M_n} a_{n,m} \quad (8)$$

Thus, in Figure 3, it would be the sum of the area of all shadowed regions.

Thus, both variables can again (like in Equation (6)) be merged by summing up the partial results stored on each duplicated node:

```
local_node.h_inv := local_node.h_inv + neighbor_node.h_inv
local_node.a := local_node.a + neighbor_node.a
```

4.2 Inconsistent Solutions

Even if the two aforementioned MERGE_OPs are actually enough for parallelization of the non-hydrostatic SWE in sam(oa)², we observed an unexpected behavior during test runs. It was found that, after solving the system of linear equations, the solution x and the residuum r sometimes would be inconsistent between associated border nodes. Since it only occurred when using MPI parallelization, it seems apparent that it was related to the peer-to-peer fashion with which border nodes are merged if sections are owned by multiple processes (see Section 2.2).

Sam(oa)² does not specify a fixed order in which nodes are merged, so it may be different for each of the border nodes. Let x_n be a variable stored on node n which is to be merged by addition. If $|x_u| \ll |x_v|$ holds for some combinations of associated nodes u and v , absorption may occur depending on the order in which the nodes are merged. Thus, two associated nodes may each hold a different x even though the same nodes were merged. In a hypothetical scenario where we have three nodes n_0, n_1, n_2 , each holding a

Algorithm 2: MERGE_OP for adjusting inconsistent solutions.

```

function merge_op(local_node, neighbor_node) {
    if( neighbor_node is master ) {
        local_node.x := neighbor_node.x;
    }
}

```

variable x_i , there could be one node n_0 holding a very large x_0 (e.g. 10^{10}), one node n_1 holding a very small x_1 (e.g. -10^{10}) and one node n_2 holding an x_2 which is almost 0 (e.g. 10^{-10}). Let \hat{x}_i be the final result on node n_i after merging and ϵ_j the round-off error of the j th addition, then e.g.:

$$\hat{x}_0 = \underbrace{(x_0 + x_1)}_{=0+\epsilon_0} + x_2 = x_2 + \epsilon_0 + \epsilon_1$$

$$\hat{x}_1 = \underbrace{(x_1 + x_2)}_{=x_1+\epsilon_2} + x_0 = 0 + \epsilon_2 + \epsilon_3$$

$$\hat{x}_2 = \underbrace{(x_2 + x_0)}_{=x_0+\epsilon_4} + x_1 = 0 + \epsilon_4 + \epsilon_5$$

Thus, $\hat{x}_0, \hat{x}_1, \hat{x}_2$ are inconsistent.

This issue was solved by adding two more traversals, which are solely targeted at merging border nodes again, one before and one after the solver. The first one is used to merge the right hand side p of the equations, the second one to merge the solution x and the residuum r . The adapted Algorithm 1, extended by the two additional traversals denoted as `adjustment_traversal`, is depicted in Algorithm 3.

The MERGE_OP of these traversals works as follows: Sam(oa)² always tags one node of a set of associated nodes as master node. The MERGE_OP simply overwrites the local node's result if the neighbor node is tagged as master node. Because the MERGE_OP is executed for all combinations of associated nodes, the master's result will be copied to all others, leading to consistent results. The pseudocode for the x -MERGE_OP is shown in Algorithm 2.

5 Conjugate Gradient

Since not only parallelism is able to speed up computation, we also examined another way to optimize the non-hydrostatic SWE implementation. In

Algorithm 3: Non-hydrostatic SWE algorithm with adjustment traversals.

```
for( all time steps ) {  
    adaptivity_traversal();  
    euler_traversal();  
    lse_traversal();  
    adjustment_traversal();  
    solver();  
    adjustment_traversal();  
    correction_traversal();  
}
```

the previous version, the Jacobi method was used to solve the system of linear equations. This was due to the fact that Jacobi had already been implemented in `sam(oa)`². Furthermore, it was important to guarantee that the solver would always be able to solve the system—regardless of the kind of system matrix. Finally, performance was irrelevant.

Because the Conjugate Gradient Method (CG) had also already been implemented in `sam(oa)`² and promised a much faster convergence than the Jacobi method, we examined its use for our purposes. CG is only defined for symmetric positive definite system matrices [9]. For a matrix to be positive definite it is sufficient that the matrix is strictly diagonally dominant and all diagonal elements are positive [10].

According to investigations performed in [2], the matrix is strictly diagonally dominant. Our own tests—the test cases described in Section 6.1—confirm these results. Moreover, they indicate that all diagonal elements are positive; thus, we consider the matrix positive definite. As stated in [6], however, the matrix can not be considered symmetric. Our own investigations show the converse, though: We only observed symmetric matrices in all our test cases. The practical application of CG was also successful. In order to double-check that CG in fact found the correct solution, we executed Jacobi after convergence of CG: For all test cases, CG found the correct solution.

As depicted in Figure 4, CG converges much faster than Jacobi. Here, the standing wave scenario was executed for different grid resolutions without adaptivity. The number of iterations in each time step was then averaged over the first 0.1 seconds of simulation time.

To sum up, CG provides a very performant way to solve the system of linear equations. Even though many tests have been conducted to ensure the characteristics of the system matrix, this is certainly not proof enough

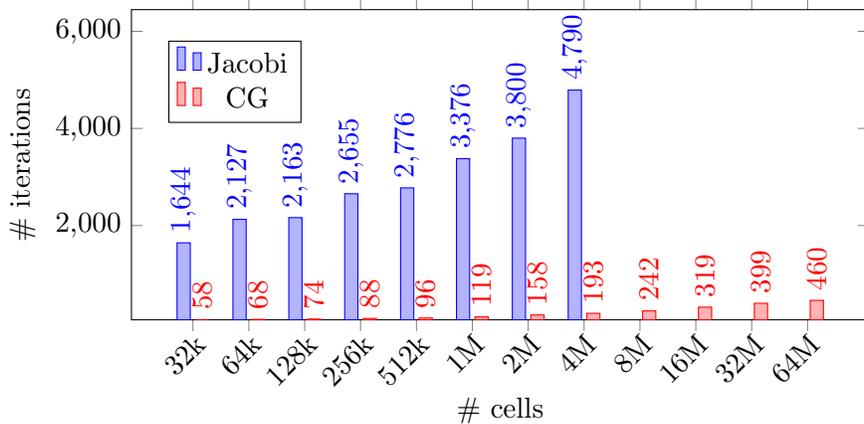


Figure 4: Rounded number of iterations until convergence ($\epsilon = 10^{-5}$) of Jacobi vs. CG for different grid resolutions of the standing wave test case. The number of iterations is averaged over the first 0.1 seconds of simulation time. For Jacobi, the measurements were stopped after 4M cells since it took too much time.

to guarantee that there will always only be symmetric and positive definite system matrices. Therefore, the use of CG should be treated with caution and its validity should be verified for each scenario.

6 Results

6.1 Test Cases

For speedup and conformity tests, mainly two test cases were performed. The *standing wave* and *dam break* scenario. While the former is basically a 1D scenario, the latter is a 2D scenario. Furthermore, for matrix examination, the *wave run-up on a beach* and *wave propagation over a submerged bar* scenarios were conducted. The standing wave, wave run-up on a beach and wave propagation over a submerged bar scenarios are described in detail in [2].

The initial structure of the dam break scenario is depicted in Figure 5. It consists of a circle of elevated bathymetry of radius β and height b_1 , whose transition to the normal bathymetry (height b_0) is linear. γ denotes the

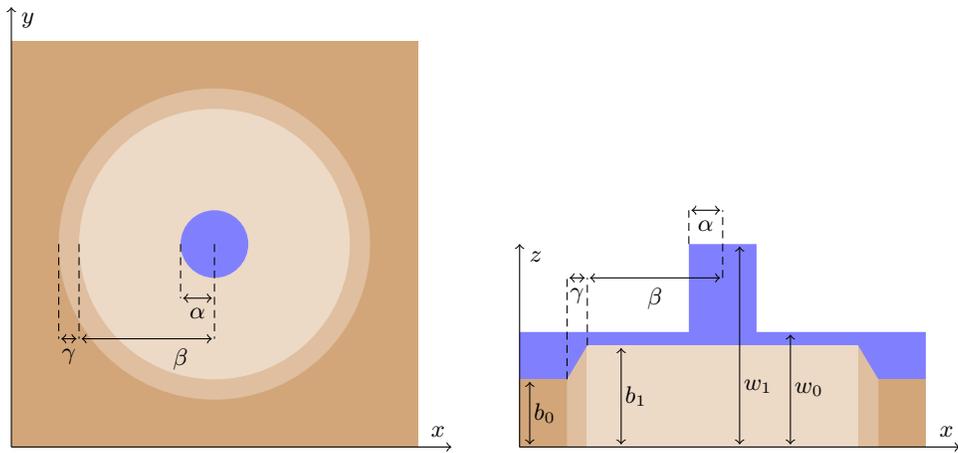


Figure 5: Initial structure of the dam break scenario, depicted from above (left picture) and sideways (right picture). The pictures are not drawn to scale. α and β are the radii of the elevated water and the elevated bathymetry circle, respectively. γ is the width of the transition between normal and elevated bathymetry. b_0 and b_1 are the height of the normal and the elevated bathymetry, respectively. w_0 and w_1 are the height of the water at rest and the elevated water, respectively.

width of this transition. In the middle, within a circle of radius α , the water is elevated to a height of w_1 , whereas w_0 denotes the normal water height.

6.2 Numerical Issues

For the dam break scenario we observed numerical instabilities, as depicted in Figure 6. In this case, the dam break parameters were set as follows: $\alpha = 0.02$, $\beta = 0.1$, $\gamma = 0$, $b_0 = -100$, $b_1 = -5$, $w_0 = 0$, $w_1 = 10$.

We first spotted the error when simulating a dam break scenario with $\gamma = 0$, so an infinitely steep bathymetry gradient. Since the error occurred not for all grid resolutions (e.g. for $d_{\max}=15$ in Figure 6) we concluded that it may be related to that steep bathymetry gradient. However, setting γ to higher values and hence explicitly flattening the gradient did not enhance the solution.

Furthermore, we discovered that it is irrelevant whether the simulation is executed singlethreaded or multithreaded, with one or multiple sections, which is why parallelization can be excluded as source of the problem. Specifying a lower linear solver error bound ϵ (default is $\epsilon = 10^{-5}$) did also not improve the solution. A lower courant number—which denotes the time step size relative to the CFL condition (default: 0.450)—was likewise not beneficial. Additionally, we tried to turn off adaptivity which also did not help.

Therefore, we neither are able to state the origin of the issue nor to name countermeasures. It may, however, result from the fact that during discretization of the non-hydrostatic correction formulas, a surface integral was neglected, influencing the behavior at abrupt bathymetry changes [2].

6.3 Performance

All measurements were conducted on the SuperMUC Thin Nodes², operated by the Leibniz Supercomputing Centre. Each node consists of two Xeon E5-2680 with 8 physical cores each. 512 nodes are combined to one island, therefore containing 8192 cores. The nodes are connected via Infini-band FDR10. We performed all measurements using *PipeCG* as solver, an optimized version of the regular CG.

Figure 7 shows the MPI-only strong scaling of the standing wave scenario. The simulation ran for 100 time steps with a fixed grid resolution of 19, 21 and 23, so with 2M, 4M or 8M cells, respectively. Each MPI process incorporated only one section to minimize overhead. Sections splitting was

²<http://www.lrz.de/services/compute/super Tuc/systemdescription/>

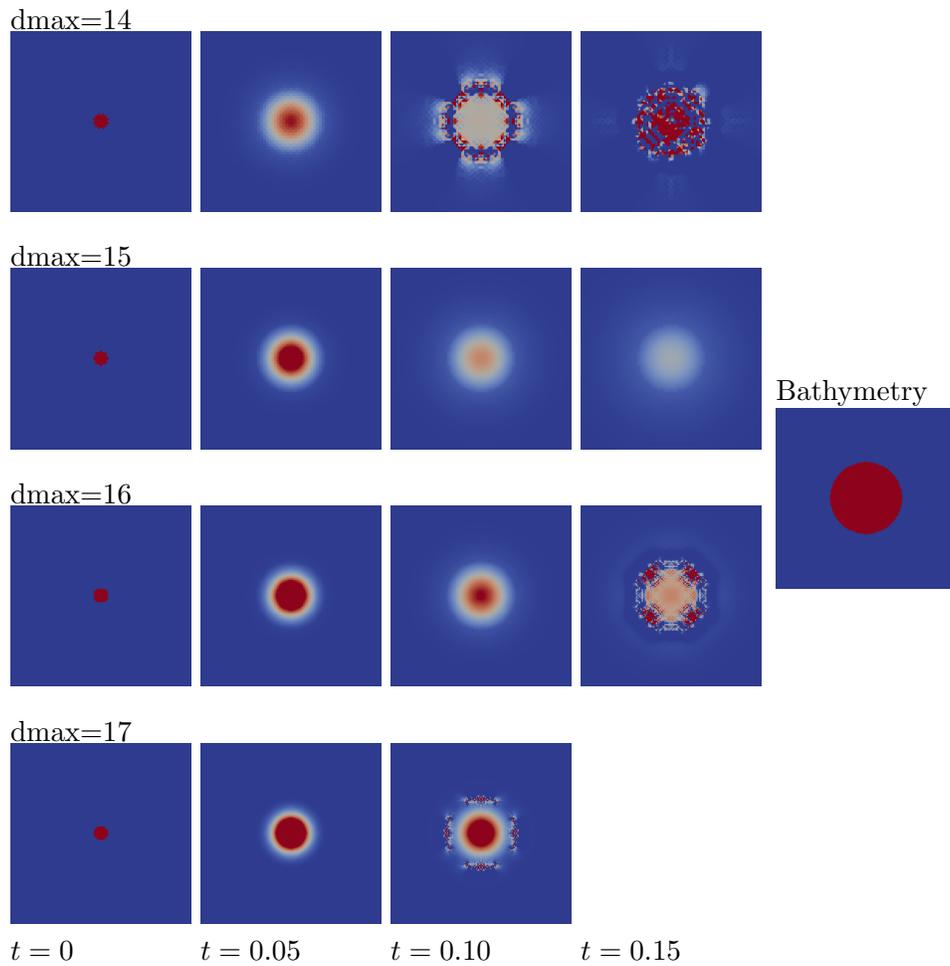


Figure 6: Numerical instability when simulating the dam break scenario. Different grid resolutions ($d_{\min}=4$ always) were compared at different time steps. The final picture of $d_{\max}=17$ is not included since the solver did not converge anymore. The rightmost picture shows the bathymetry.

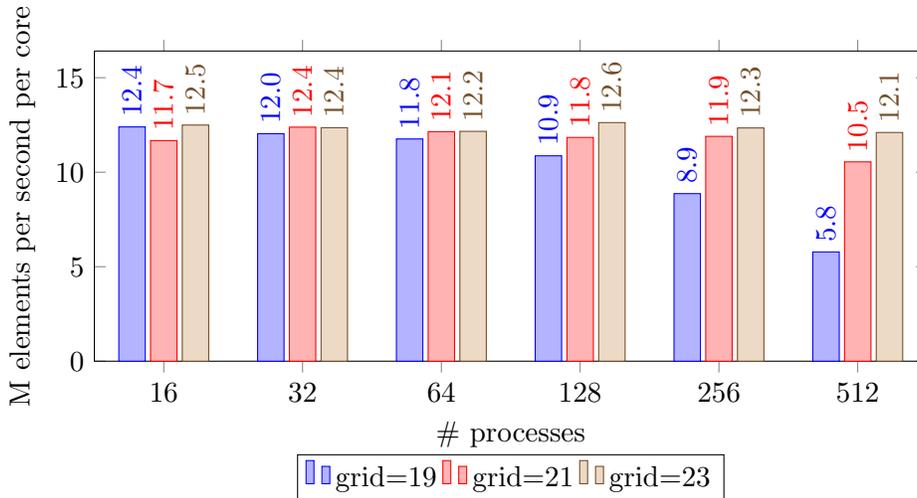


Figure 7: MPI-only strong scaling of the standing wave scenario. The y-axis denotes the number of elements each core processed per second in the simulation phase, thus giving a measure of parallel efficiency.

turned on; therefore, during load balancing, the sections were not treated as atomic work loads anymore but may have been split and distributed to other processes [4]. For 512 processes and a resolution of 23, a parallel efficiency of 96.8% was accomplished. For 128 processes, there even was a super linear speedup reaching 101.0% of the element throughput with 16 processes. A lower grid resolution, however, drastically diminished the parallel efficiency since each process held only a low number of cells—e.g. for a resolution of 19 and 512 processes 2048 cells. Therefore, this resulted in an increased overhead.

The weak scaling of the same scenario is displayed in Figure 8. Here, the scaling was performed for the first 0.05s of simulation time for one up to 8192 processes, using up to an entire SuperMUC island. The simulation was also executed on 16384 cores (thus, two islands), but due to a time quota on SuperMUC for only 0.0002s of simulation time (marked red in Figure 8). For the same reason, runs on more cores were not possible. The fixed grid resolution was scaled linearly from 14 (32k cells) for one process to 28 (512M cells) for 16384 processes, thus doubling the number of grid cells each time the number of processes was doubled. Furthermore, each MPI process held one section and section splitting was turned on. For 8192 processes, we achieved a parallel efficiency of 90.1%.

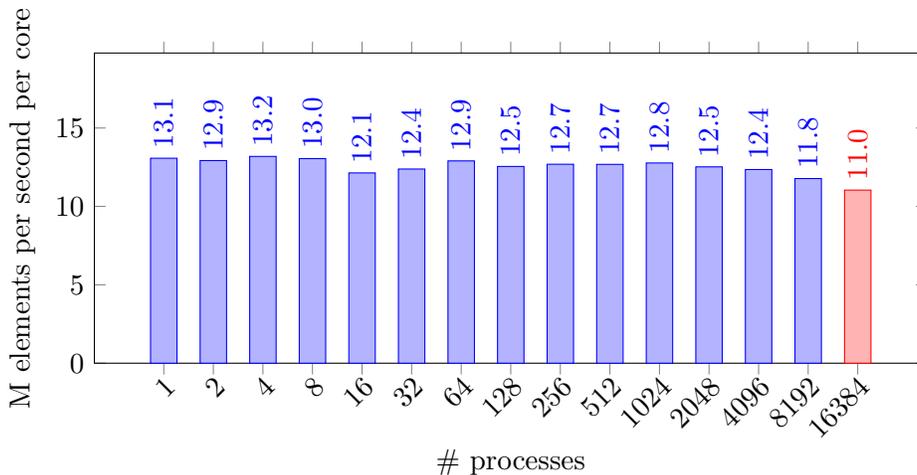


Figure 8: MPI-only weak scaling of the standing wave scenario. The y-axis denotes the throughput of elements each core processed per second in the simulation phase, thus giving a measure of parallel efficiency. Due to a time quota on SuperMUC, the run represented by the red marked bar was performed for only 1/250 of the simulation time of the regular runs.

As shown in Algorithm 3, a single time step consists of several components. Figure 9 indicates the relative time spent in each of those components during the strong scaling measurements (*Adaptivity*: refine/coarsen grid; *Euler*: Euler time step; *LSE*: set up of system of linear equations; *Adjustment*: adjustment of inconsistent nodes (both traversals are combined); *Solver*: solving of system of linear equations; *Correction*: correction of hydrostatic solution). As can be seen, the solver took most with about 80% of the time. The higher the grid resolution, the larger portion of time was spend in the solver, which is due to the fact that the number of iterations of the solver increases with a finer grid (see Figure 4). For grid=21 and grid=23, the solver’s portion stood rather constant. For grid=19, on the other hand, the portion increased for a higher number of processes. This is because the solver is particularly affected by the aforementioned overhead, which can also be observed in Figure 7.

Figure 10 displays the component breakdown for the weak scaling measurements. The chart clearly shows how the number of iterations of the solver affects the component breakdown: The more iterations, the larger is the solver’s portion of the total time. The high number of iterations for the run on 16384 cores results from the fact, that, at the beginning of the simu-

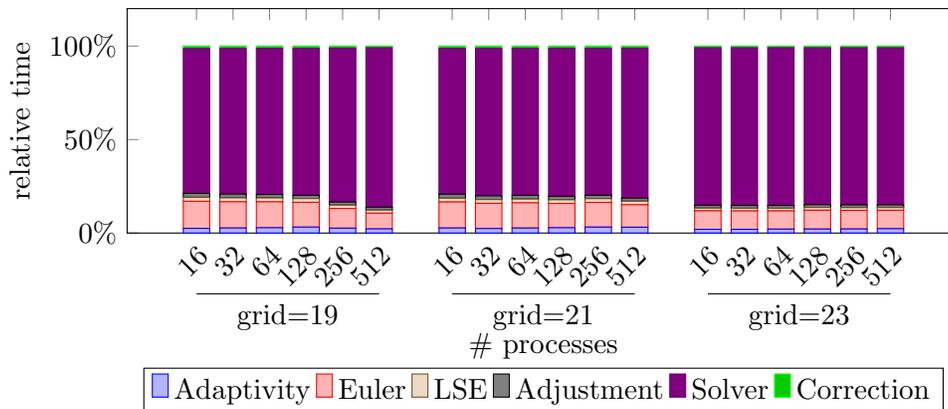


Figure 9: Component breakdown for strong scaling of the standing wave scenario for different grid resolutions. The same parameters as in Figure 7 were used. The time spent in a particular component relative to the total run time is given.

lation of the standing wave scenario, the number of iterations is on average higher than later. Since the simulation time was only 0.0002s for 16384 cores, mostly time steps with high iteration numbers were performed.

7 Conclusion and Outlook

The primary goal of this paper was to realize parallelism for the non-hydrostatic SWE implemented in sam(oa)². As discussed in Section 6, an excellent strong as well as weak scaling was achieved. To further improve performance, we successfully employed the Conjugate Gradient Method for the non-hydrostatic SWE. However, it could not be guaranteed that CG will always work, since we investigated the properties of the matrix only by experiments. Thus, even if we have observed only symmetric and positive definite matrices, this can not be generalized for all scenarios. Hence, further investigations are required.

On the other hand, we discovered numerical instabilities during simulation of the dam break scenario, which are, however, not related to parallelism. The cause could only be guessed since we have not found a countermeasure. Therefore, this issue also demands further examination.

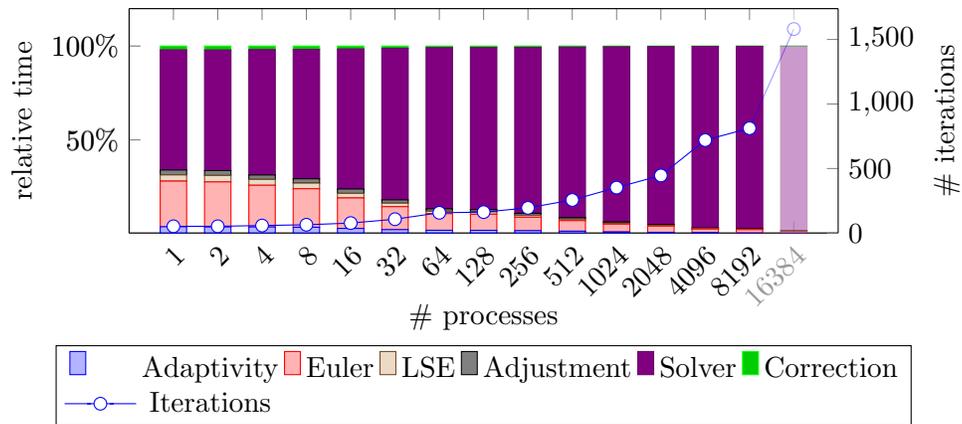


Figure 10: Component breakdown for weak scaling of the standing wave scenario. The same parameters as in Figure 8 were used. The time spend in a particular component relative to the total run time is given. Furthermore, the blue line denotes the number of CG iterations per time step averaged over all time steps. Due to a time quota on SuperMUC, the run represented by the pale bar was performed for only 1/250 of the simulation time of the regular marked runs.

References

- [1] B. Levin and M. Nosov, *Physics of Tsunamis*. Springer Science & Business Media, 2008.
- [2] P. Samfaß, “A Non-Hydrostatic Shallow Water Model on Triangular Meshes in sam(oa)²,” studienarbeit/sep/idp, Institut für Informatik, Technische Universität München, Apr. 2015.
- [3] O. Meister, K. Rahnema, and M. Bader, “A Software Concept for Cache-Efficient Simulation on Dynamically Adaptive Structured Triangular Grids,” vol. 22 of *Advances in Parallel Computing*, pp. 251–260, IOS PRESS, 2012.
- [4] O. Meister and M. Bader, “2D Adaptivity for 3D Problems: Parallel SPE10 Reservoir Simulation on Dynamically Adaptive Prism Grids,” *Journal of Computational Science*, vol. 9, pp. 101–106, May 2015.
- [5] W. F. Mitchell, “Adaptive Refinement for Arbitrary Finite-Element Spaces with Hierarchical Bases,” *Journal of Computational and Applied Mathematics*, vol. 36, no. 1, pp. 65 – 78, 1991.
- [6] P. J. Samfaß, “Extension of the Finite Volume Solver SWE towards the Non-Hydrostatic Shallow Water Equations,” bachelor’s thesis, Institut für Informatik, Technische Universität München, Sept. 2014.
- [7] D. L. George, “Augmented Riemann Solvers for the Shallow Water Equations over Variable Topography with Steady States and Inundation,” *Journal of Computational Physics*, vol. 227, no. 6, pp. 3089–3113, 2008.
- [8] A. J. Chorin, “Numerical Solution of the Navier-Stokes Equations,” *Mathematics of computation*, vol. 22, no. 104, pp. 745–762, 1968.
- [9] M. R. Hestenes and E. Stiefel, “Methods of Conjugate Gradients for Solving Linear Systems,” *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409–436, 1952.
- [10] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge university press, 2012.