ASAGI – A Parallel Server for Adaptive Geoinformation

Sebastian Rettenberger[*]
rettenbs@in.tum.de

Oliver Meister[*]
meistero@in.tum.de

Michael Bader[*]
bader@in.tum.de

Alice-Agnes Gabriel[†]
gabriel@geophysik.uni-muenchen.de

[*]Technical University of Munich, Department of Informatics
Bolzmannstr. 3, 85748 Garching, Germany
[†]Munich University, Department of Earth and Environmental Sciences, Geophysics
Theresienstr. 41 80333 Munich, Germany

ABSTRACT

We present ASAGI, an open-source library with a simple interface to access Cartesian material and geographic datasets in massively parallel simulations with dynamically adaptive mesh refinement (AMR). ASAGI distributes geographic datasets over all compute nodes storing only a portion of the dataset on each node. An automatic replication mechanism copies the data between nodes to assure fast local access even after load migration in the application. We demonstrate ASAGI's preparedness for up-to-petascale simulations in three use cases. We simulate a Tsunami on 512 cores and a porous media flow on up to 8,192 cores of SuperMUC with the AMR framework sam(oa)^2. We also run an earthquake simulation with SeiSol on 65,536 cores. For all applications, ASAGI provides large complex 3D material datasets required for the realistic scenarios. The NUMA-awareness of ASAGI turned out to be especially useful for the hybrid MPI+OpenMP parallelization of both codes.

CCS Concepts

•Computing methodologies → Massively parallel and high-performance simulations; •Applied computing → Earth and atmospheric sciences; •Theory of computation → *Massively parallel algorithms;*

Keywords

Adaptive mesh refinement; Geoinformation; Large scale applications; Realistic simulations

1. INTRODUCTION

Adaptive mesh refinement (AMR) is a key ingredient in many application domains [6, 8, 10, 12, 15, 16, 22]. It allows to resolve "areas of interest", such as wave fronts, shocks or (moving) boundaries with a fine mesh while allowing a coarser resolution for the rest of the domain. This can lead to a much shorter time-to-solution for many applications [15]. Recent advances have also demonstrated scalability of AMR on up to 1.6 million cores [16] making it interesting for current and next generation supercomputers.

Scaling AMR to such a large number of compute cores requires complex load balancing steps to adapt to the growing and shrinking number of cells in mesh partitions. These load balancing steps migrate mesh elements (cells/vertices) and the corresponding unknowns between nodes to assure best hardware utilization. An additional issue arises, however, if the application requires fast access to large space- and time-dependent material or geographic datasets. In this paper geographic datasets are the permeability and porosity of underground oil reservoirs in porous media flow, the bathymetry and sea floor displacement for tsunami simulation, and material velocity properties for seismic wave propagation. We will refer to such datasets as *geoinformation*.

Large geoinformation datasets are required for many realistic simulations. Due to their size, it is not practical to replicate the datasets on every node in a distributed memory parallelization. This is especially applicable for time-dependent datasets. However, distributing the data means that it must be migrated along with the cells during load balancing to ensure fast local access. Since cells can be, depending on the input data, refined, coarsened and migrated at any point in the simulation, it is hard to predict which data is required after migration. In addition, there might exist different resolutions for a single dataset further complicating the decision.

In this paper, we present ASAGI[1], an open-source library that provides a simple and easy-to-use interface for applications. It takes over the migration of geoinformation without explicit migration calls from the application. Instead, it automatically replicates necessary parts of the dataset on different compute nodes and deletes data that is no longer required. ASAGI replicates data only on-demand and implements a block-caching strategy that exploits the temporal and spatial locality of geoinformation and minimizes the amortized latency. The hybrid MPI+PThread parallelization in ASAGI supports hybrid parallelization (MPI+X) in applications and can detect and optimize accesses from different NUMA domains. We use ASAGI in the AMR framework sam(oa)^2 for flows in porous media and tsunami sim-

EASC '16, April 26 - 29, 2016, Stockholm, Sweden
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4122-6/16/04...$15.00
DOI: http://dx.doi.org/10.1145/2938615.2938618

[1]https://github.com/TUM-I5/ASAGI

ulations and show scalability to 8,192 cores. ASAGI also simplifies the initialization of codes based on static unstructured meshes, such as the earthquake simulation code SeisSol. For both applications, ASAGI provides complex 3D material parameters and bathymetry/displacement data to simulate realistic scenarios.

In the next section we discuss alternative approaches for geoinformation in AMR. In Secs. 3 and 4 we present the high-level design decisions of ASAGI and a detailed description of its implementation. Sec. 5 contains the results of our experiments with sam(oa)^2 and SeisSol. A conclusion of our work is provided in Sec. 6.

2. RELATED WORK

Geoinformation for AMR is often neglected as it is only an issue for very large datasets and for large simulations on many compute nodes. For smaller setups, it is usually sufficient to hold the complete geoinformation in memory on every node. This approach allows very fast access to geoinformation and scales well, especially in strong scaling setups. Our results show, nevertheless, that this approach is only optimal if NUMA domains are considered in the placement of geoinformation in memory.

One approach to handle datasets that do not fit into the memory of a single node is to use database-like services. In [12], Patra et al. connect their application to GRASS GIS, a geographic information system, designed to handle very large datasets. However, this results in a 1:n communication pattern which may create a bottleneck in massively parallel simulations. Besides, the batch systems of modern supercomputers are usually not very well suited for running database-like services.

A second approach, for tsunami simulation, is described in [14,15]. Here, the idea is to use a 2D tree structure to provide multiple resolutions of the bathymetry dataset. To avoid the limitations of main memory, only the index structure (i.e. the inner nodes) of the tree is stored in memory. The leaves of the tree are kept on disk. Although this approach allows handling datasets in the range of terabytes, it does not completely avoid disk accesses during the simulation.

Since refinement, coarsening and load balancing of cells is hard to predict in AMR, it is not clear which data points will be required after a load balancing step (Fig. 1). Thus, migrating geoinformation together with mesh elements is not practical. To avoid the migration as well as disk accesses, ASAGI implements a lazy replication strategy described in the next section. To the best of our knowledge, ASAGI is the only available system for geoinformation limited only by the combined memory of all nodes and completely obviates the need for disk accesses during the simulation.

3. DESIGN

One of the main design goals of ASAGI is to provide an interface for applications that is as simple as possible and allows easy integration into existing codes. ASAGI calls must be added at three different points in the application as shown in Listing 1: At the beginning to synchronize processes and load the datasets, during the simulation to access the data and at the end to free resources. Most important, ASAGI does not require explicit information about load balancing and migration of cells in the application. This information is derived from the access pattern of the application. Despite the simple interface, ASAGI supports datasets with multiple resolutions and can manage multiple independent datasets.

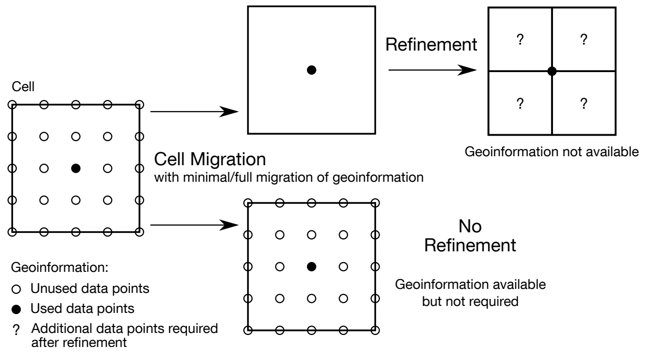
Cell
Refinement
?
Geoinformation not available
Cell Migration
with minimal/full migration of geoinformation
No Refinement
Geoinformation available but not required
Geoinformation:
○ Unused data points
● Used data points
? Additional data points required after refinement

Figure 1: An optimal migration strategy for geoinformation would need to know whether a cell is further refined after migration.

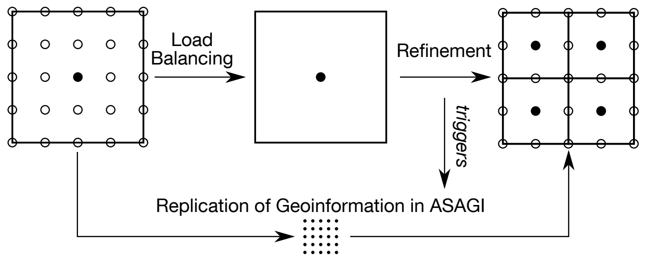

Figure 2: Data is only replicated on-demand in ASAGI.

Accesses to geoinformation typically have a spatial and temporal locality in the application. If a grid point is accessed, there is a very high probability that many of the neighboring grid points will be required as well. Although the exact access pattern of adaptive simulations is not easily predictable, this assumption holds for most applications.

As a further simplification, ASAGI stores datasets only as Cartesian grids and does not use the mesh structure of the application. Since geoinformation often comes from sampled physical properties, many datasets are available as Cartesian grids anyway. The netCDF library is used as an I/O backend and accepts netCDF files that respect the COARDS (Cooperative Ocean/Atmosphere Research Data Service) conventions [1], an established standard for geographic data. Nearest-neighbor interpolation is used in ASAGI to provide valid data between grid points. The interface is sufficiently flexible to support more sophisticated interpolation schemes in the application as shown in Sec. 5.2.

Instead of predicting possible cell refinements, ASAGI replicates data points only on-demand (Fig. 2). Whenever the application requires new information not already available on the current node, a replication process is triggered. The replication is based on point-to-point communication and is completely decentralized. Disk access during the simulation is only required for datasets that do not fit into the combined memory of all nodes. Since we do not migrate but rather replicate data points, we also track the least recently used points and automatically evict them.

To reduce management overhead, ASAGI divides the dataset into equal-sized rectangular *chunks*, similar to the net-

```
#include <asagi.h>
int main(int argc, char** argv) {
  // Initialization ...
  // Initialize ASAGI
  asagi::Grid geoData = asagi::Grid::create();
  geoData->setComm(MPI_COMM_WORLD);
  geoData->setThreads(omp_get_max_threads());
  // Set other parameters (optional):
  geoData->setParam("CACHE_SIZE", "64");
  // Open the dataset in parallel
  #pragma omp parallel
  { geoData->open("material_data.nc"); }

  // Do the simulation
  while (t < end_time) {
    // Loop over all cells
    #pragma omp parallel for
    for (int i = 0; i < cells.size(); i++) {
      float par = geoData->getFloat(cells[i].coord);
      // Compute 'cells[i]' with parameter 'par'
    }
    // MPI Communication/Load Balancing ...
  }

  delete geoData;
  // Free other resources
}
```

Listing 1: Integration of ASAGI into a hybrid MPI+OpenMP parallelized code written in C++. ASAGI provides similar interfaces for C and Fortran.

CDF library, and does not manage single grid points or arbitrarily shaped regions. These chunks group (spatially and temporally) adjacent grid points together and are used as cache lines. Thus, whenever a task of the application requests a grid point not stored locally, the whole chunk is transfered to a local cache. Every subsequent access to grid points in the same chunk can then be handled without further communication. Since we assume that the application often requests adjacent grid points, most of the requests can be satisfied by the cache which amortizes the cost of replicating chunks from other nodes or loading them from the file system.

4. IMPLEMENTATION

To scale to a large number of nodes, ASAGI distributes the datasets to all processes and does not use a centralized server to access them, thus, avoiding the bottleneck of typical 1:n communication patterns. In addition, the server part is directly integrated into the library and is therefore automatically started with the application making it easy to use with batch systems.

When a task of the application requests a grid point, ASAGI executes the following steps:

1. Determine the chunk containing the grid point.

2. If the chunk is not in the local cache, look for it in other NUMA domains and/or on other MPI ranks and copy it to the local cache.

3. If the chunk is not stored on any node, load it from the file.

4. At this point the chunk is in the local cache and the requested value is returned.

A detailed description of the implementation and available options for steps 2 and 3 is given in the following subsections.

4.1 Local chunk storage

ASAGI supports three different methods to handle chunks. In *full* mode, ASAGI loads the whole dataset during the initialization. If MPI communication is enabled, the dataset gets distributed over all MPI ranks, such that each rank has to store only a portion of the whole set. Instead of reading the dataset at initialization, ASAGI can also be used in *cache* mode. In this mode, chunks are only loaded on-demand when they are required by the application. While this mode reduces the initialization time, it usually increases the total execution time since it complicates the replication of chunks between nodes. A global dictionary needs to be maintained with all current locations of chunks. This dictionary needs to be updated with every replication, which is not necessary in full mode since the initial position of a chunk is fixed and chunks are always copied from there. Therefore, the cache mode is primarily useful if the dataset is very large and does not fit into the total memory of all nodes or only a very small part of the dataset is actually processed by the application. Finally, ASAGI has a *pass-through* mode where accesses are not optimized with the chunk cache but directly passed to the underlying I/O library. Since this mode is only useful for testing, it is not discussed further in this paper.

To automatically evict unused chunks from the cache, ASAGI implements the two-handed clock algorithm originally used as a page replacement algorithm in operating systems [21]. The algorithm provides a reasonable approximation for the least-recently-used (LRU) strategy with a very low overhead. The actual size of the cache can be configured through ASAGI's interface.

4.2 Chunk replication

ASAGI has three different ways to handle multiple MPI processes for full and cache mode. The simplest way is to turn MPI communication completely off. For the full mode, this means that the whole grid is stored in each process. Although only possible for smaller datasets, this combination can be used as a performance baseline to which other modes can be compared. If MPI communication is turned off in cache mode, ASAGI only looks for the chunk in the local cache or on the local node and, if the chunk is not found there, directly reads it from the file.

ASAGI can switch between remote memory access (RMA) and an explicit communication thread when MPI communication enabled. The RMA implementation is based on MPI windows with passive target communication from the MPI-2 standard [11]. In full mode, the MPI windows are set up after loading the data from the file and are configured such that each process has direct access to all chunks of the dataset. In cache mode, the local caches from each node are accessible via RMA to replicate chunks. This is in contrast to the full mode where caches are completely private and replication is only done from the initially loaded dataset. To maintain the global dictionary with RMA, required for the cache mode, ASAGI uses MPI mutexes based on the MPI-3 functions `MPI_Fetch_and_op` and `MPI_Compare_and_swap`. The implementation is similar to [3] with the following slight modification. Instead of updating the next pointer and waiting in an `MPI_Recv` when the mutex is locked, we use an `MPI_Ssend` which stores an implicit previous pointer. To forward the lock to the next process in the queue, an `MPI_Recv` with `MPI_ANY_SOURCE` is used. This optimization avoids the update of the next pointer as described in [3].

Instead of using RMA, ASAGI can start an explicit communication thread using the pthread library. This thread waits for queries from other processes and sends back the requested chunk. In cache mode, the thread also manages the dictionary updates. Although only one communication thread is started independent of the number of datasets, ASAGI still requires one core which cannot be used by the application. The communication thread is therefore only useful for codes with a hybrid parallelization.

For hybrid codes, ASAGI can activate NUMA detection. If more than one NUMA domain is detected, ASAGI creates a chunk cache for each NUMA domain and caches not only accesses to remote nodes, but also to other NUMA domains. The additional caching together with the increased memory bandwidth can speedup the accesses to the dataset as shown in Sec. 5. In full mode, ASAGI distributes the dataset over all NUMA domains at initialization and provides an option to search for a chunk in caches of all local NUMA domains before using MPI.

5. RESULTS

To test the performance and scalability of ASAGI, we used the AMR framework sam(oa)^2 [10] and the earthquake simulation code SeisSol [7]. All tests were performed on SuperMUC Phase 1 which is ranked 23rd on the Nov'15 TOP500 list [19]. SuperMUC features 9216 dual socket Intel Xeon E5-2680 nodes with 16 cores and 32 GiB memory per node. The nodes are organized in 18 islands connected in a 4:1 pruned tree via Infiniband FDR10. All input files were stored on the GPFS file system with an aggregated bandwidth of 180 GiB/s. We used Intel MPI for sam(oa)^2 but IBM MPI for SeisSol since the respective MPI library has performed best with each code in the past.

For all datasets, we set the chunk size in the netCDF file to exactly match the chunk size and thus the access pattern of ASAGI. Fig. 3 shows the load time of the large setup described in Sec. 5.1.1. The chunked storage in the netCDF file leads to a factor 13 improvement in the initialization time on 512 nodes. Note that finding the optimal chunk size for a netCDF file for ASAGI is trivial since the library loads the data chunk-by-chunk.

5.1 Sam(oa)^2

Sam(oa)^2 is a parallel framework for dynamically adaptive solutions of Partial Differential Equations (PDEs) based on Sierpinski space-filling curve traversal. Currently, two major applications are supported by sam(oa)^2: porous media flow and tsunami wave propagation. For both scenarios, the execution is split into two phases: the initialization phase starts with a grid consisting of only a small number of cells and successively refines and distributes the grid to multiple MPI ranks using appropriate local error indicators. Afterwards, the time stepping phase executes the simulation on the generated grid, allowing further refinement and coarsening in each time step. To model heterogeneities in the computational domain, external files store the cell-local geoinformation managed by ASAGI. Hence, each time a cell is refined, ASAGI is invoked for data access.

In all the experiments with sam(oa)^2 we analyze the two phases separately. During the first phase (the initial mesh

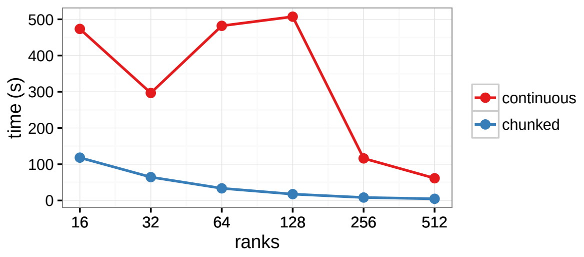

Figure 3: Load time of $4\times$ 4.2 GiB of geoinformation with continuous and chunked netCDF files. ASAGI was configured in full mode with MPI windows and NUMA detection enabled.

adaption), there are very coarse cells which have a suboptimal access pattern for ASAGI since the fine dataset is sampled numerous times to create a cell average. This simplification in sam(oa)^2 avoids creating different resolutions of the same dataset in an additional preprocessing step. However, the averaging for the coarse cells often requires different chunks and therefore multiple chunk replications for a single cell. This effect leads to significantly slower mesh adaption rates in the beginning. We only measure and compare the time for complete mesh adaption steps of sam(oa)^2 since ASAGI calls are tightly integrated into the mesh adaption.

5.1.1 Two-Phase Porous Media Flow

The first scenario is a 2.5D two-phase porous media flow model for oil reservoir simulation [10]. Using a low-order IMPES (Implicit pressure, explicit saturation) scheme, the simulation alternates the solution of a linear system for pressure updates and explicit time steps for phase saturation updates. Cell permeability tensors and cell porosities are provided by ASAGI. AMR is supported in the horizontal dimensions, the vertical dimension is uniformly refined.

We tested the scenario with two different mesh sizes in sam(oa)^2. In the small test case, we used an input file with 71 million grid points ($\sim$1 GiB) and allowed vertical refinement between 8,000 and 2 million cells with 340 layers in the horizontal dimension. For the large setup, we increased the input file by a factor of four in x and y dimension leading to 17 GiB of geoinformation and allowed a vertical refinement from 8,000 to 33 million cells. With this size, we have already reached the memory limit of a SuperMUC node. Storing the whole dataset on all nodes is only possible for 64 resp. 256 and more nodes (without/with NUMA detection). For smaller node counts, the combined memory requirements from sam(oa)^2 and ASAGI exceed the limit. The chunk sizes are fixed to $16\times16\times340$ grid points for the small and $64\times64\times340$ grid points for the large setup. The caches are set to 256 chunks per node (340 MiB resp. 5.3 GiB).

Fig. 4 shows the initial mesh adaptions of sam(oa)^2 with different ASAGI configurations. Sam(oa)^2 is executed with one task per node and 15 (communication thread) resp. 16 (MPI windows) OpenMP threads per node. Since the whole dataset is required to initialize the simulation, the full mode of ASAGI is much faster than the cache mode even if we include the load time in the measurement. Throughout the experiments, the communication thread version performs better than the MPI window implementation despite the fact that only 15 cores are available for the computation. One major disadvantage of MPI windows with passive target

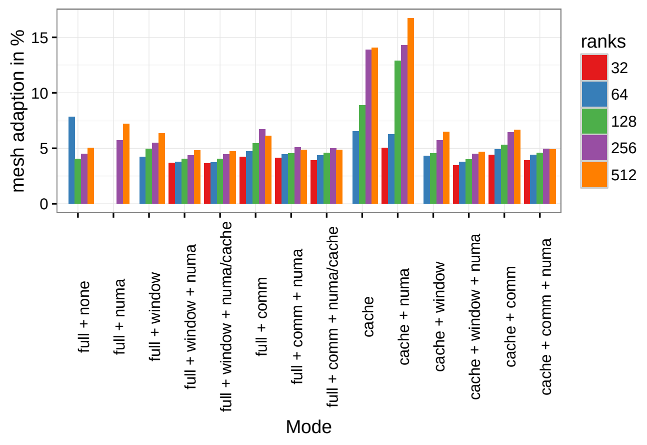

Figure 6: Overhead of the mesh adaption in sam(oa)^2 for the large oil reservoir simulation. An explanation of the modes can be found in Table 1.

communication is that they are not guaranteed to progress if the target does not call any MPI functions. Since we did not start the explicit progress thread of the Intel MPI library, some tasks have to wait until they get their requested chunks which slows down the whole mesh adaption. The results also show that the NUMA-aware storage and communication lead to additional performance improvements.

Since all chunks are stored in the cache of at least one node after the initial mesh adaptions, the cache mode (with MPI enabled) delivers a similar performance to the full mode in the simulation (Fig. 5). Indeed, the NUMA-aware caches have a greater impact on the performance than the MPI communication pattern. In this state, the simulation is dominated mostly by the linear solver and the whole mesh adaption including the ASAGI calls only requires between 4.7% and 4.9% of the total wall time when enabling MPI communication and NUMA detection in full mode on 512 nodes. Storing the whole dataset on every node reduces this only to 3.2% (Fig. 6).

5.1.2 Tsunami Simulation with Time-Dependent Displacements

The second scenario in sam(oa)^2 is a shallow water wave propagation solver used for tsunami simulation. The simulation scheme is fully explicit in this case, hence computation per time step is comparably small and calls to ASAGI have a much larger impact on the performance during the simulation. In contrast to the first setup, the shallow water implementation is purely 2D and based on bathymetry and sea floor displacements. However, sam(oa)^2 has the option to use time-dependent displacements generated by earthquake simulation codes. Such displacements elevate the sea floor and thus the water over time and are considered to be more realistic compared to an instant elevation produced by static displacements.

We decided to use the 2011 Tohoku tsunami (Fig. 8) as an example for the shallow water implementation since it is well documented and there are many datasets available. Thus, our setup consists of a 2D bathymetry dataset obtain from GEBCO [2] with $14,000 \times 8,000$ grid points and a 3D displacement file computed with SpecFEM [5] with 80 time steps and $2572 \times 3621 \times 80$ grid points. This results in 427 MiB of bathymetry data and 2.8 GiB of displacement data. We use a chunk size of 128 grid points in x and y direction and four grid points in time. The chunk caches are configured to hold 512 bathymetry and 512 displacement chunks and require $32 + 128$ MiB of memory.

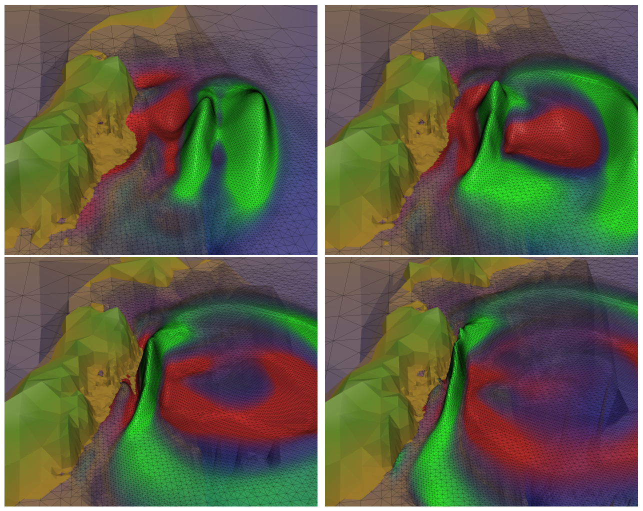

Figure 8: Visualization of the 2011 Tohoku tsunami after 3, 10, 17 and 23 min on a fully adaptive grid simulated with sam(oa)^2.

Fig. 7 shows, similar to the porous media flow, the performance for initial mesh adaption and the mesh adaption during the simulation. For the initial mesh adaption of the time-dependent dataset, the improvement of the NUMA detection is clearly visible. During the actual simulation storing the whole dataset on each node performs best but also the full mode with MPI communication and the cache mode deliver a reasonable performance.

5.2 SeisSol

SeisSol is an earthquake simulation package that couples 3D seismic wave propagation to the simulation of dynamic rupture propagation across earthquake fault zones [18]. In contrast to sam(oa)^2, it does not use AMR but relies on fully unstructured tetrahedral meshes. These meshes are crucial to resolve complex geometries of realistic underground structures and resolve small-scale earthquake dynamics. We integrated ASAGI into the initialization routines of SeisSol to load complex 3D velocity models (material properties describing the seismic wave speeds) required for accurate propagation of seismic waves. Note, that the initialization of element-wise material properties avoids the efforts of specifically meshing the interfaces represented by geological changes in material properties. Since the initialization of the velocity model is completely decoupled from the rest of the code (in contrast to sam(oa)^2), SeisSol can act as benchmark application for ASAGI. In addition, due to the fixed mesh in SeisSol it is also possible to count the exact number of accesses to ASAGI.

Although there are two alternative approaches for loading 3D velocity models, we explicitly decided to use the Cartesian grids in ASAGI as a common interface. One method for static unstructured meshes is to integrate geoinformation di-

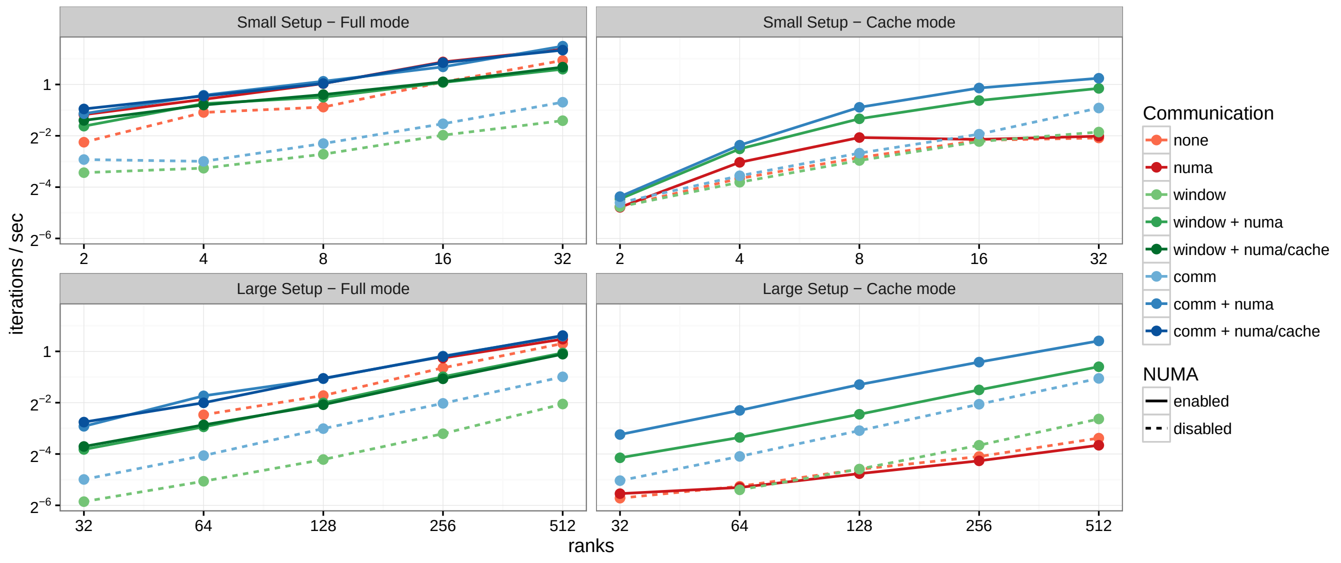

Figure 4: Scaling of the initial mesh adaption with different ASAGI configurations in the porous media flow scenario. An explanation of the communication patterns can be found in Table 1.

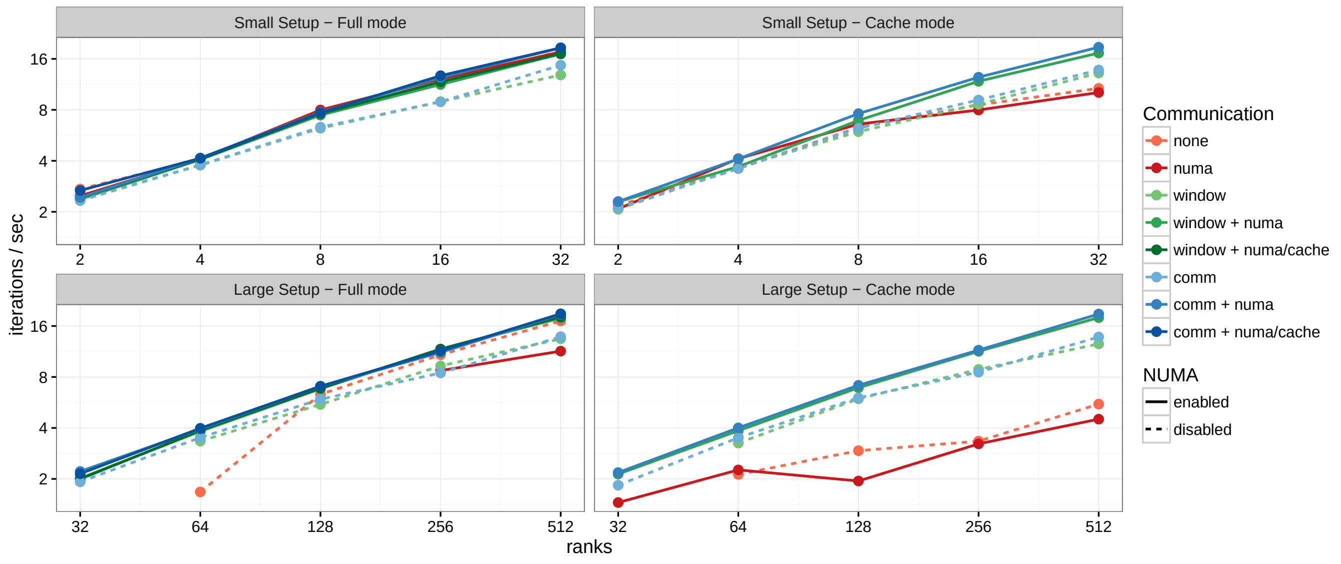

Figure 5: Scaling of the mesh adaption step during the simulation with different ASAGI configurations in the porous media flow scenario. An explanation of the communication patterns can be found in Table 1.

Table 1: Communication patterns in ASAGI

none	No MPI communication and no NUMA detection → In full mode the whole dataset is stored on every node.
numa	No MPI communication but with NUMA detection → In full mode the whole dataset is stored on every node.
window	With MPI communication using RMA (MPI windows), no NUMA detection
window + numa	With MPI communication using RMA and NUMA detection
window + numa/cache	Only avail. in full mode: MPI communication using RMA and NUMA detection; caches from other NUMA domains are searched for the chunk before using MPI
comm	With MPI using a communication thread, no NUMA detection
comm + numa	With MPI using a communication thread and NUMA detection
comm + numa/cache	Only avail. in full mode: MPI using a communication thread and NUMA detection; caches from other NUMA domains are searched for the chunk before using MPI

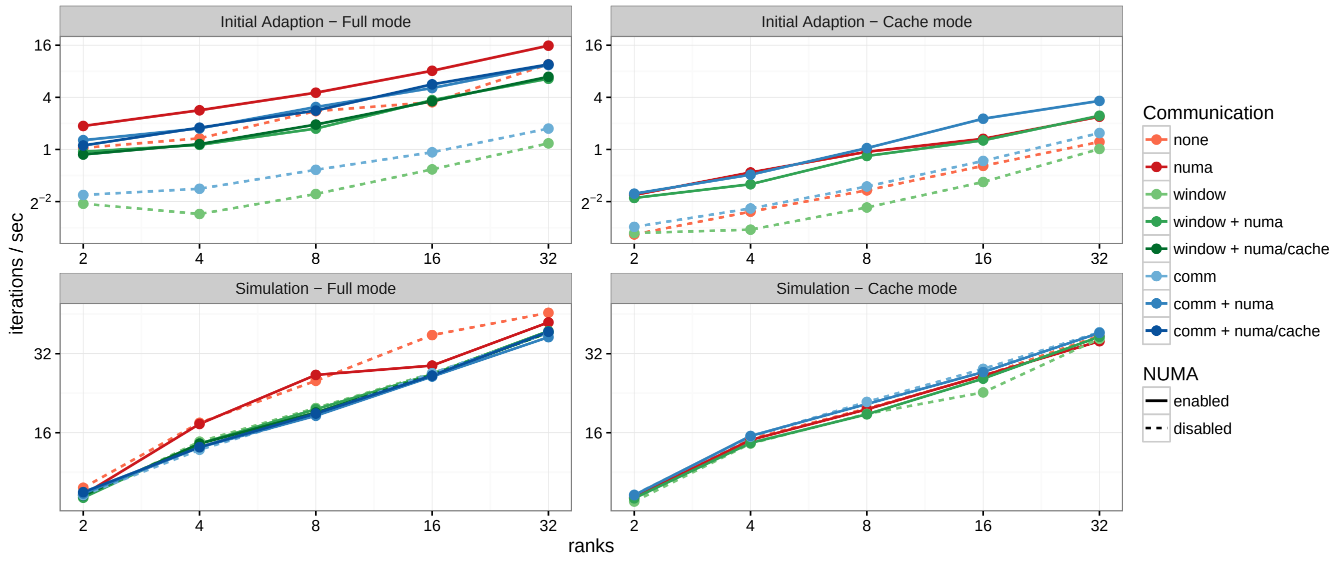

Figure 7: Scaling of the initial mesh adaption steps (top) and the mesh adaption during the simulation (bottom) of the Tohoku scenario with different ASAGI configurations. An explanation of the communication patterns can be found in Table 1.

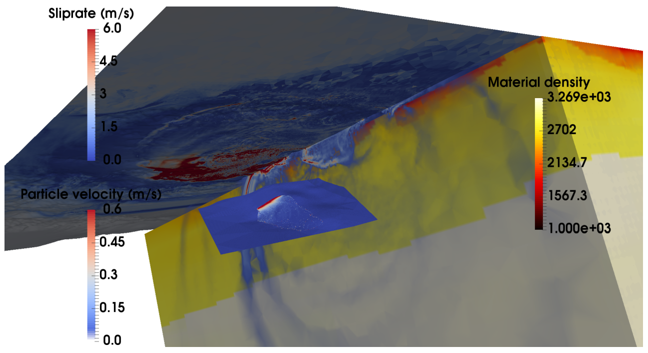

Figure 9: Coupled dynamic rupture and wave propagation simulation based on the 1994 Northridge earthquake with the 3D velocity model CVM-H.

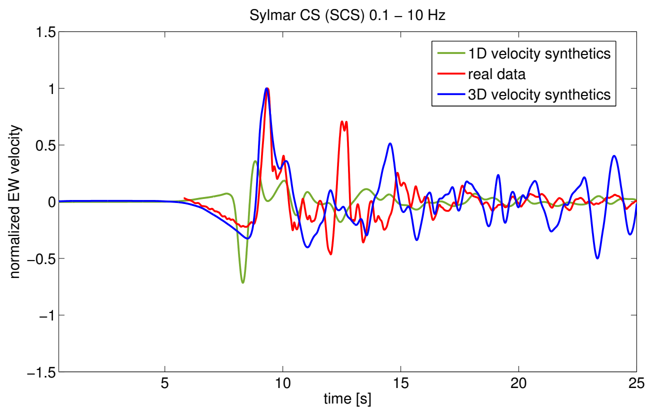

Figure 10: Observed ground motions compared to respective simulations with SeisSol based on the complex 3D model provided by ASAGI and a simplified 1D velocity model at station Sylmar CS (SCS). The 3D velocity model captures the first strong seismic waves much better than the simplified model.

rectly into the mesh structure, as for example done in [17]. While this is very simple for a single simulation, it is less efficient for extended studies since it requires a new dataset for every change of the mesh. Compared to simple Cartesian grids, the integration of geoinformation into the mesh structure also complicates the preprocessing step, especially for very large meshes with 100+ million cells. We also decided against the direct integration of community velocity models (e.g. [9, 20]), which provide explicit code to compute the material properties for any coordinate. Such models would require changes of SeisSol's code base for every new model and would limit us to models providing a library.

To test the ASAGI integration, we simulated an earthquake scenario based on the 1994 Northridge earthquake (Fig. 9) integrating the SCEC Community Velocity Model – Havard (CVM-H) [13, 20]. We used 3D meshes (1.9 million and a 74.8 million elements) constructed from geological constraints such as high-resolution topography data and SCEC Community Fault Model as presented in [4], but replaced the 1D velocity structure with the complex 3D velocity model. This replacement influences the local ground motions and overall high-frequency content of the seismic wave field. For the meshes, we created input datasets with 900 meter resp. 200 meter resolution resulting in 5.7 and 527 million grid points. The datasets consist of three different parameters (density, shear modulus and the first Lamé parameter), resulting in 66 MiB resp. 5.9 GiB of input data. To project the Cartesian grid to the cell centers of the tetrahedra, SeisSol samples the eight surrounding grid points and implements a trilinear interpolation. The chunk size in the netCDF file and in ASAGI is set to 32^3 grid points and the cache is configured to hold 128 chunks per node (48 MiB). Since the whole dataset is required, we only tested the full mode of ASAGI which performs much better in such cases.

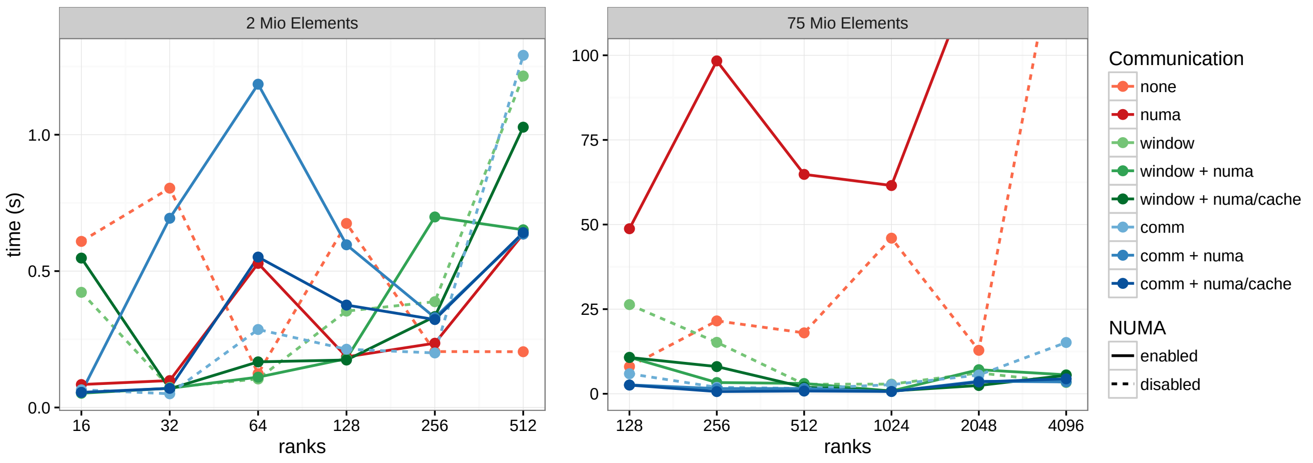

Figure 11: Load time for the 66 MiB (left) and 5.9 GiB (right) velocity model datasets. An explanation of the communication patterns can be found in Table 1.

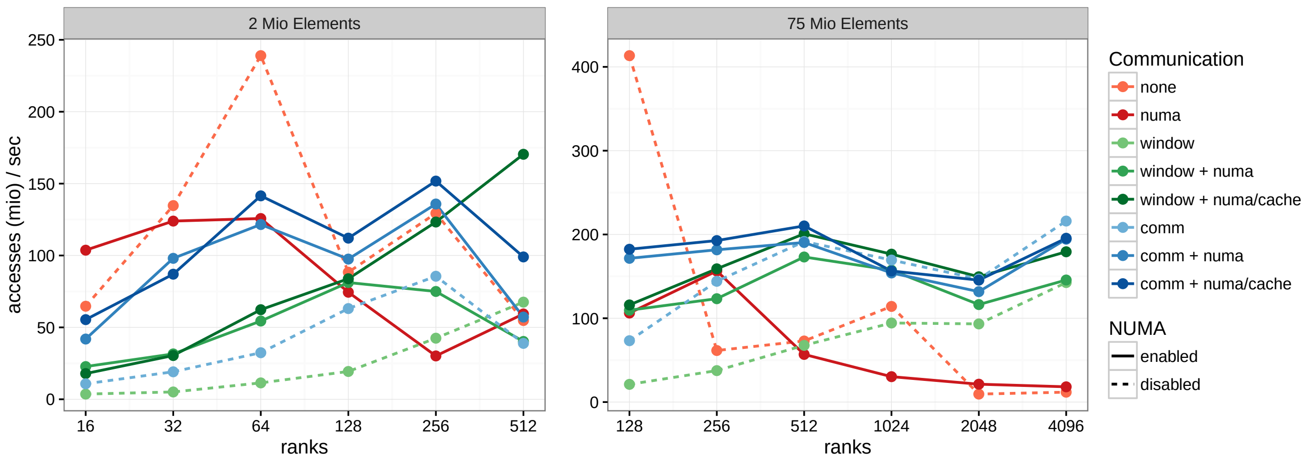

Figure 12: Number of data accesses per second handled by ASAGI in the Northridge setup. An explanation of the communication patterns can be found in Table 1.

In addition to the initialization benchmarks, we also simulated 50 s of the earthquake on the high resolution mesh. For SeisSol, this was the first simulation at that scale that used a complex 3D velocity model. Previous simulations were only based on a few material regions provided via a CAD model or used simple 1D velocity structure. The comparison of synthetic and observed ground motions covering the full frequency band relevant for seismic engineering (up to 10 Hz) at station Sylmar CS (SCS) is shown in Fig. 10. The 3D velocity model captures the strong, first arriving seismic waves much better than the simplified 1D model.

Figs. 11 and 12 present the load and the initialization time in SeisSol for the small and the large datasets. Since the total initialization takes only between 0.1 and 1 seconds for the small dataset, measurement errors are inevitable. However, as can be seen, the NUMA-aware storage has a stronger influence on the performance than the MPI communication mode which confirms our experiments with sam(oa)^2. The long load times we measured for the large setup when disabling MPI communication can be explained by the additional I/O requirements. If MPI is enabled, each point in the dataset is only loaded by one rank. If MPI is disabled, however, each data point is loaded from all ranks, increasing the I/O transfers by a factor equal to the number of ranks. In addition, disabling the MPI communication in ASAGI also removes the implicit barrier between loading the data and the initialization in SeisSol. Thus, and due to measuring only rank 0, imbalances in the load time are only visible in the second phase of the initialization.

6. CONCLUSIONS

We presented ASAGI, an open-source library for geoinformation in adaptive simulations. The library provides a simple interface that can easily be integrated into existing applications. The distributed design of ASAGI allows it to handle very large geoinformation datasets without disk accesses. The replication of data between compute nodes and NUMA domains in ASAGI is automatically triggered by data accesses from the application. ASAGI uses chunks (similar to netCDF) to cache data on the local node and to reduce the number of replications.

Our results show that ASAGI is prepared for petascale simulations on current supercomputers. With the AMR framework sam(oa)^2, ASAGI scaled up to 8,192 cores on SuperMUC. Already on the dual socket Intel Xeon E5-2680 nodes, sam(oa)^2 profits from the NUMA-aware storage optimization in ASAGI. In addition, we were able to initialize a complex 3D velocity model with 5.9 GiB in only a few seconds in SeisSol using ASAGI on 65,536 compute cores.

7. ACKNOWLEDGEMENT

ASAGI was developed as part of the ASCETE (Advanced Simulation of Coupled Earthquake-Tsunami Events) project funded by the Volkswagen Stiftung. Computing resources for this project have been provided by the Leibniz Supercomputing Centre under grant: pr45fi.

8. REFERENCES

[1] Conventions for the standardization of NetCDF files, 1995. http://ferret.wrc.noaa.gov/noaa_coop/coop_cdf_profile.html, Accessed: 2016-02-11.

[2] The GEBCO_08 Grid, version 20100927, 2010. http://www.gebco.net.

[3] J. Dinan, P. Balaji, D. Buntinas, D. Goodell, W. Gropp, and R. Thakur. An implementation and evaluation of the MPI 3.0 one-sided communication interface. *Concurrency and Computation: Practice and Experience*, 2016.

[4] A. Gabriel and C. Pelties. Simulating large-scale earthquake dynamic rupture scenarios on natural fault zones using the ADER-DG method. In *EGU General Assembly Conference Abstracts*, volume 16 of *EGU General Assembly Conference Abstracts*, page 10572, May 2014. poster abstract.

[5] P. Galvez, J.-P. Ampuero, L. A. Dalguer, S. N. Somala, and T. Nissen-Meyer. Dynamic earthquake rupture modelled with an unstructured 3-D spectral element method applied to the 2011 M9 Tohoku earthquake. *Geophysical Journal International*, 198(2):1222–1240, 2014.

[6] D. L. George and R. J. LeVeque. Finite volume methods and adaptive refinement for global tsunami propagation and local inundation. *Science of Tsunami Hazards*, 24:319–328, 2006.

[7] A. Heinecke, A. Breuer, S. Rettenberger, M. Bader, A.-A. Gabriel, C. Pelties, A. Bode, W. Barth, X.-K. Liao, K. Vaidyanathan, M. Smelyanskiy, and P. Dubey. Petascale high order dynamic rupture earthquake simulations on heterogeneous supercomputers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis SC14*, pages 3–14, New Orleans, LA, USA, Nov. 2014. IEEE.

[8] A. Humphrey, D. Sunderland, T. Harman, and M. Berzins. Radiative heat transfer calculation on 16384 GPUs using a reverse Monte Carlo ray tracing approach with adaptive mesh refinement. In *The 17th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC 2016)*, 2016. accepted.

[9] M. D. Kohler, H. Magistrale, and R. W. Clayton. Mantle heterogeneities and the SCEC reference three-dimensional seismic velocity model version 3. *Bulletin of the Seismological Society of America*, 93(2):757–774, 2003.

[10] O. Meister and M. Bader. 2D adaptivity for 3D problems: Parallel SPE10 reservoir simulation on dynamically adaptive prism grids. *Journal of Computational Science*, 9:101–106, 2015.

[11] MPI Forum. MPI: A message passing interface standard. Version 2.0, May 1998.

[12] A. K. Patra, A. Bauer, C. Nichita, E. B. Pitman, M. Sheridan, M. Bursik, B. Rupp, A. Webber, A. Stinton, L. Namikawa, et al. Parallel adaptive numerical simulation of dry avalanches over natural terrain. *Journal of Volcanology and Geothermal Research*, 139(1):1–21, 2005.

[13] A. Plesch, C. Tape, R. Graves, P. Small, G. Ely, and J. Shaw. Updates for the CVM-H including new representations of the offshore Santa Maria and San Bernardino basin and a new Moho surface. In *2011 Southern California Earthquake Center Annual Meeting, Proceedings and Abstracts*, volume 21, page 214, 2011.

[14] S. Popinet. Quadtree-adaptive tsunami modelling. *Ocean Dynamics*, 61(9):1261–1285, 2011.

[15] S. Popinet. Adaptive modelling of long-distance wave propagation and fine-scale flooding during the Tohoku tsunami. *Natural Hazards and Earth System Sciences*, 12(4):1213–1227, 2012.

[16] J. Rudi, A. C. I. Malossi, T. Isaac, G. Stadler, M. Gurnis, P. W. J. Staar, Y. Ineichen, C. Bekas, A. Curioni, and O. Ghattas. An extreme-scale implicit solver for complex PDEs: Highly heterogeneous flow in earth's mantle. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 5:1–5:12, New York, NY, USA, 2015. ACM.

[17] O. Sahni, K. E. Jansen, C. A. Taylor, and M. S. Shephard. Automated adaptive cardiovascular flow simulations. *Engineering with Computers*, 25(1):25–36, 2008.

[18] SeisSol, 2016. http://www.seissol.org/.

[19] E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer. Top500 list, November 2015. http://www.top500.org.

[20] M. P. Süss and J. H. Shaw. P wave seismic velocity structure derived from sonic logs and industry reflection data in the Los Angeles basin, California. *Journal of Geophysical Research: Solid Earth*, 108(B3), 2003. 2170.

[21] A. S. Tanenbaum. *Modern operating systems (2. ed.)*. Prentice Hall, 2001.

[22] K. Unterweger, R. Wittmann, P. Neumann, T. Weinzierl, and H.-J. Bungartz. Integration of FULLSWOF2D and PeanoClaw: adaptivity and local time-stepping for complex overland flows. In *Recent trends in computational engineering - CE2014 : optimization, uncertainty, parallel algorithms, coupled and complex problems.*, pages 181–195. Springer, 2015.