SIAM J. SCI. COMPUT.
Vol. 38, No. 6, pp. C678–C712

EVALUATION OF AN EFFICIENT STACK-RLE CLUSTERING CONCEPT FOR DYNAMICALLY ADAPTIVE GRIDS*

MARTIN SCHREIBER†, TOBIAS NECKEL‡, AND HANS-JOACHIM BUNGARTZ‡

Abstract. One approach for tackling the challenge of efficient implementations for parallel PDE simulations on dynamically changing grids is the usage of space-filling curves (SFCs). While SFC algorithms possess advantageous properties such as low memory requirements and close-to-optimal partitioning approaches with linear complexity, they require efficient communication strategies for keeping and utilizing the connectivity information, in particular for dynamically changing grids. Our approach is to use a sparse communication graph to store the connectivity information and to transfer data blockwise. This permits efficient generation of multiple partitions per memory context (denoted by *clustering*), which—in combination with a run-length encoding (RLE)—directly leads to elegant solutions for shared, distributed, and hybrid parallelization and allows cluster-based optimizations. While previous work focused on specific aspects, we present in this paper an overall compact summary of the stack-RLE clustering approach complete with aspects of the vertex-based communication that facilitate understanding the approach. The central contribution of this work is the proof of suitability of the stack-RLE clustering approach for an efficient realization of different, relevant building blocks of scientific computing methodology and real-life computer science and engineering (CSE) applications: We show 95% strong scalability for small-scale scalability benchmarks on 512 cores and weak scalability of over 90% on 8192 cores for finite-volume solvers and changing grid structure in every time step; optimizations of simulation data backends by writer tasks; comparisons of analytical benchmarks to analyze the adaptivity criteria; and a tsunami simulation as a representative real-world showcase of a wave propagation for our approach which reduces the overall workload by 95% for parallel fully adaptive mesh refinement and, based on a comparison with SFC-ordered regular grid cells, reduces the computation time by a factor of 7.6 with improved results and a factor of 62.2 with results of similar accuracy of buoy station data.

Key words. adaptive mesh refinement, space-filling curves, parallel simulation, MPI+X parallelization, shallow water, tsunami

AMS subject classifications. 65Y05, 65M08, 68W10, 68W15, 76N99

DOI. 10.1137/15M1027711

1. Introduction and related work.

Solving partial differential equations (PDEs) numerically is a computationally intensive task and frequently involves adaptive meshes to invest degrees of freedom (DoF) more in parts of the domain that strongly contribute to the accuracy of the approximate solution. The simulations in which we are interested in the context of this work are based on hyperbolic equations which stem from wave-propagation phenomena. Here, the wave movements demand frequent refinement and coarsening operations of the grid at runtime of an application (see Figure 1 for an example in the context of tsunami simulation).

For a parallelization of corresponding adaptive solvers, one challenge is an efficient concept and implementation of dynamic migration of both DoF and metadata at runtime. To tackle this challenge, one can use graph-based partitioners which typically focus on optimizing the interfaces shared among partitions, e.g., by reducing the

*Submitted to the journal's Software and High-Performance Computing section June 24, 2015; accepted for publication (in revised form) September 29, 2016; published electronically November 29, 2016.

http://www.siam.org/journals/sisc/38-6/M102771.html

Funding: This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89).

†CEMPS, University of Exeter, Exeter, EX4 4QF, UK (M.Schreiber@exeter.ac.uk).

‡Institut für Informatik, Technische Universität München, Garching, D-85748, Germany (neckel@in.tum.de, bungartz@in.tum.de).

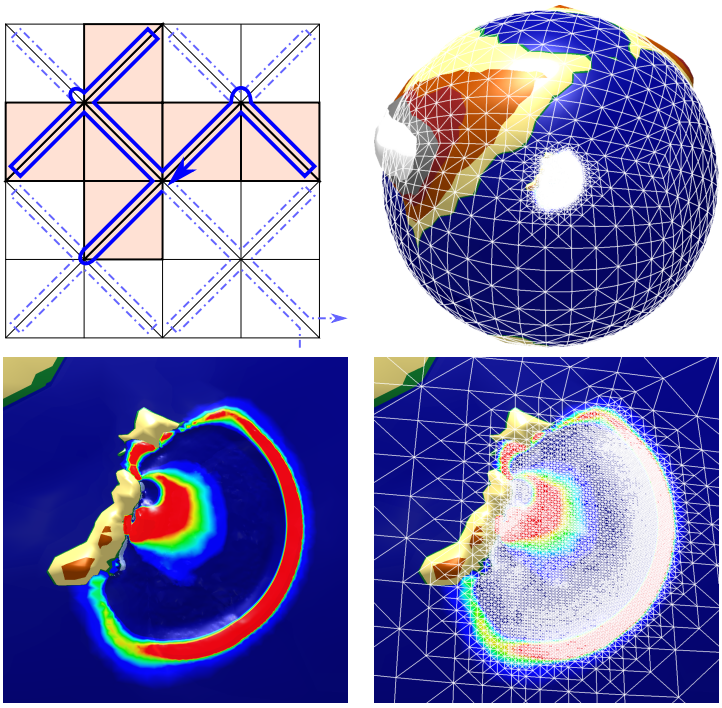

FIG. 1. *Top left image: Embedding of a cubed-sphere domain triangulation into a regularly refined Sierpiński grid to achieve consistent connectivity information with the standard data-exchange method. Top right image: Experimental finite-volume simulation* [13] *of the Tohoku tsunami, a triangulated cubed sphere (see section* 4.4 *for information on data sources and used solvers). Bottom images: Zoomed view on the coastlines of Japan and the propagating wave fronts. The bathymetry is refined in correspondence with the simulation grid. Our development offers an interactive OpenGL visualization of the surface, bathymetry, and grid structure based on the underlying dynamically adaptive triangular grid. The closed surface is generated based on the RLE vertex connectivity information as discussed in section* 3.2. *(Colored figures available in electronic version.)*

number of so-called edge-cuts of the dual graph [33]. The load balancing then migrates data to compute nodes by including assumptions about communication overheads. However, complex operations [5] and typically NP-hard algorithms [40] have to be executed after each adaptivity step which are obviously not close to linear complexity. Therefore, for the partitioning of frequently changing grids, space-filling curves (SFCs) [25, 15, 1] have been widely used in scientific computing (cf. [5, 8, 9, 19, 24, 37]). Using the topological order of grids generated by SFCs results in a spacetree [12, 22, 23, 37]; the leaves then represent the grid cells. These cells can then be traversed using recursive algorithms ([37, 38], e.g.) or in a linearized way (cf. [9]) by traversing only the leaf nodes (see [17]). In this work, we focus on simulation data stored and processed on leaf cells only. Here, the hyperfaces at the leaves (cells, edges, faces, etc.) of a simulation domain can then be uniquely enumerated along the SFC. For parallelization, the unique ordering and the inherent property of a particular SFC can be used to traverse the next cells in their proximity. After enumerating all cells, this range can then be used to cut the grid into partitions of balanced numbers of cells (cf. [5]). This reduces the NP-hard complexity of graph-based partitioners [40].

Solving PDEs on such SFC-induced grids requires not only efficient storage of data on hyperfaces but also efficient data transfer between the grid cells for a parallel

execution. Such a data transfer relies on consistent connectivity information for dynamically changing grids. In the literature, two categories of methods to realize the underlying connectivity information can be observed; we denote these as *explicit* and *implicit*.

With *explicit connectivity*, we refer to methods which explicitly communicate the adjacency information, e.g., about the new location of the adjacent cell and the new compute rank[1] of the adjacent cell via the *face* shared by both cells. In case of a changing grid structure, connectivity information is updated based on new adjacency information, which is then explicitly transferred in a per-face manner. The new compute ranks to which the cell is migrated to are then sent, received, and stored involving edge-adjacent cells. After the data migration, the new connectivity information (based on the explicitly transferred new compute ranks) is then stored per-cell (see [5]) or can be stored only for interpartition shared hyperfaces (see scalability plots in [21], based on the parallelization approach in [36], again based on [5]). This obviously allows only a single partition per memory context, since only message-passing interface (MPI) ranks are communicated. Extending this to additionally communicate local thread IDs to support OpenMP (OMP) parallelization would require additional algorithms for efficiency reasons. To support the communication via vertices, direct storage of connectivity information leads to further drawbacks of high memory consumption by storing adjacency information for all possible vertex-adjacent cells.

With an *implicit connectivity* approach, the connectivity information is neither stored nor updated. It is inferred by traversing down the spacetree after the grid structure has changed (cf. [9], e.g.). Such approaches not only involve computation but also require random-like memory access operations and hence suboptimal awareness of the memory hierarchy.

Our approach—which will be called stack-RLE clustering in the following—represents a third variant for keeping a consistent connectivity. *Explicit* run-length encoded (RLE) graph information is combined with implicit stack-based communication via spacetree properties. This approach relies on grids generated with a particular set of SFCs supporting the so-called stack-based communication approach [15, 4, 3] via an annotation of SFC grammar elements. See [38] for an introduction to this. The RLE used for stack-RLE clustering initially was used to store the connectivity information for edges shared among partitions [28]. Updating this RLE connectivity information is based on adaptivity markers which are used for shrinking and growing the RLE-stored number of edges to account for inserting or removing edges shared with adjacent partitions. For a repartitioning, a bisection of the spacetree of partitions or a merging of two partitions was accomplished without communication of any compute ranks via edges due to bisection properties. Finally, the third ingredient of the stack-RLE clustering approach is the *clustering*, by which we denote the ability to efficiently handle multiple partitions in a memory context. Previous work has shown that the RLE data exchange, in combination with using multiple partitions per shared-memory context, resulted in an efficient execution of problems with significant problem sizes on shared-memory systems [30]. Since the RLE enables a blockwise and efficient communication on shared-memory systems, this allowed for a direct extension to MPI (see [31]). By extending the RLE clustering concept with 0-length encoding for storing the connectivity of partitions which have shared vertices, a sparse

[1] We use this term as a generalization for an MPI-rank for distributed memory systems, as well as thread IDs for shared-memory systems or a combination of MPI-rank and thread ID for hybrid parallelization.

connectivity graph has been developed involving all possible two-dimensional hyperface elements, also including communication via vertices (see [31] for the basic idea). Given the sparse connectivity graph, data migration has been implemented efficiently in a cluster-oriented manner: The raw cluster data (graph nodes) are migrated, and only updating of the intercluster connectivity (graph edges) [27] is required.

2. Contribution. In this work, we summarize and integrate different aspects of the stack-RLE clustering approach and—for the first time—evaluate its applicability to various important building blocks of the scientific computing methodology and computer science and engineering (CSE) applications.

In section 3, we present a summary of the relevant aspects and properties of the stack-RLE clustering and parallelization approach, relying on previous work described above, particularly [15, 3, 28, 30, 31, 27]. A generation of consistent vertex connectivity, without collective operations for dynamic clustering (section 3.3.5), has been used in [31] in a black-box manner but is now described in detail in this paper. The two main goals of this summary are (a) to present all aspects (and, in particular, implementation details) of the the stack-RLE clustering approach in a consistent way in one place and (b) to have a self-contained description of the approach at hand to allow for a discussion of new results.

In section 4, we show the potential of the stack-RLE clustering method and the underlying dynamically changing grids in a realistic context by providing relevant application scenarios. We used small-scale scalability benchmarks of shallow-water simulations (similar to [27]) to analyze the performance of the algorithmic components of our RLE clustering approach in section 4.1.1 and conducted simulations on large-scale systems in section 4.1.2 showing also restrictions of a clustering based on tree splits. In section 4.2, we describe newly developed simulation-data backends for improved performance in writing simulation-data to persistent memory and for generating a closed surface to visualize shallow water equation (SWE) simulations. Section 4.3 presents comparisons with analytical benchmarks showing the potential of the dynamically adaptive grids per-se. Finally, in section 4.4 we provide results for a realistic tsunami scenario application. We use this tsunami scenario as an application which is representative for wave-propagation dominated phenomena to evaluate the RLE clustering beyond a purely artificial benchmark.

3. Summary of stack-RLE clustering. In this section, we give a brief review of the stack-RLE clustering method [28, 31, 27]. We start with our definition of clustering and then briefly explain why our stack-RLE clustering approach fulfills the corresponding clustering properties. We list different important properties of the approach with respect to relevant aspects of scientific computing applications.

3.1. Definition of clustering. A cluster (the term "clustering" is inspired by the work on cluster-based local time stepping [10] with an efficient implementation becoming feasible with our approach for dynamically adaptive grids) is a partition of grid cells with the following four additional properties:

(1) *Bulk of connected cell data:* A compact bulk of cells connected via hyperfaces is associated uniquely to a partition; hence no cell is shared between two partitions. There is a path from each cell to all other cells inside the partition via shared hyperfaces of cells in the partition.
(2) *Independent memory areas and data access during grid traversals:* The simulation data and the metadata which are associated to each partition are stored in buffers which are nonoverlapping with other buffers. Hence, no data stored at edges and

nodes are shared among partitions. They are replicated and require additional synchronization to join the replicated hyperface data.

(3) *Communication information:* Multiple partitions per thread and hence *efficient* interpartition communication schemes for them are supported for shared- and distributed-memory parallelization (MPI+X) on cluster granularity. For efficiency reasons on shared-memory systems, the connectivity information has to be stored RLE—either by updating it or by recreating it.

(4) *Traversal meta information and user-specified data:* Each partition also has to store traversal metadata, e.g., initial vertex coordinates to start the grid traversal, minimum and maximum adaptive refinement depth limiters, and also possibly required user-specified data. The traversal metadata are required to know where to start traversing the spacetree. The user-specified (optional) data can consist of, e.g., parameters for kernels or the local-grid identifier of the underlying global mesh (e.g., cubed sphere; see Figure 1).

We call a parallelization concept that allows for efficient handling of multiple partitions in each program context a *clustering* and extend this definition to *dynamic clustering* in the case where the above-mentioned properties hold for dynamically changing grids. Several advantages arise with such a concept, and some of them are discussed in this work. Next, we show that our stack-RLE clustering approach fulfills the four clustering properties.

3.2. Realization of clustering with stack-RLE clustering. With SFCs such as those of Hilbert, Peano, Gosper, and Sierpiński the grid can be generated from the leaves of the resulting spacetree. This spacetree can be generated either by refining the cells during the grid traversals and processing only the leaf cells or by a linearization of the leaves, resulting in an ordering of the underlying grid cells C_i along the SFC. Then, each cell C_{i+1} shares exactly one hyperface with C_i. *Cluster property* (1) is fulfilled for the Sierpiński curve (which we use in this work) since it generates partitions based on a subinterval of the underlying mapping of the SFC-enumerated cells. Note that this concept can also be applied to the the Peano, Hilbert, and Gosper curves.

In our implementation, the data for each partition is allocated in a nonoverlapping way, fulfilling *cluster property* (2). With separate heap memory areas for each stream, this allows for shrinking and growing of the number of grid cells in each cluster without forcing reallocations of data areas of other clusters.

Thus the heap allocation results in a gain in flexibility, but this can also involve additional costs for memory allocation. Alternative approaches store all grid data on a single stream, which requires reallocation of the entire stream in the case of a grid-local change of the grid structure; see, e.g., [30] for Cartesian grids.

Regarding *cluster property* (4), the metainformation to start the traversal is stored separately for each partition. This allows directly starting the traversal on a partition.

The partitions which we use in this work are given by splits of the underlying spacetree. An example of such a partitioning is given in Figure 2 (left). While the intrapartition connectivity is given by the stack-based communication, special attention has to be paid to the interpartition communication when searching for matching communication data on adjacent clusters in the same memory context. Based on the ordering of the communicated data, two important properties concerning RLE clustering have been initially used in [28]:

(a) All data associated to one set of shared interfaces (edges and nodes) between adjacent partitions are *ordered by cell index i*.

(b) For each adjacent partition, there is only a *unique single consecutive block of data*.

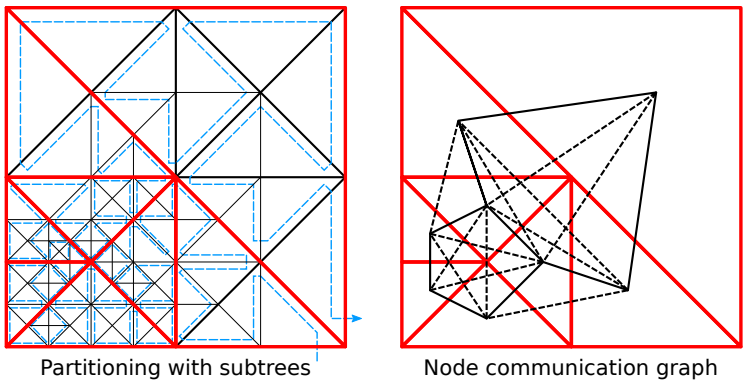

FIG. 2. *Left: Triangulated grid generated by a recursive bisection of the triangle cells. The clusters (bold red bordered) are generated by subtree-oriented intervals of the underlying bisective Sierpiński SFC (blue-dashed line). Right: Sparse communication graph for clusters with connectivity information for edges and vertices shared by multiple clusters. Each red bordered area represents a partition, and the solid lines the edge RLE communication. The dashed lines represent the additionally required zero-length encoded RLE tuples for vertices. (Colored figures available in electronic version.)*

These properties[2] hold for edges and nodes for certain two-dimensional SFCs (Hilbert, Peano, Sierpiński, and Gosper) and allow us to use an RLE for storing the connectivity information; see Figure 3. Here, an arbitrary number of edges takes only one RLE entry in the connectivity information, and only intercluster shared vertices can be stored with an RLE of length 0. However, since most of these vertices are already represented by their higher-dimensional correspondence (edges), they do not have to be stored explicitly. This results in a sparse connectivity graph, as sketched in Figure 2 (right), where edge-based communication is depicted with solid lines and vertex-based communication with dashed lines.

We emphasize that the stack-based communication approach does not need any explicitly stored connectivity information for intrapartition communication. Hence, the intrapartition communication is represented by a node of the connectivity graph. With the two communication properties above, the interpartition communication is then given by an edge of the connectivity graph, representing an RLE entry for the intercluster shared hyperfaces. For the implementation of the communication, we use a replicated data scheme on shared- and distributed-memory systems [31] which allows independent writing and accessing data on hyperfaces and hence fulfilling the MPI+X cluster property (3). Hence, we see all cluster properties (1)–(4) fulfilled for our stack-RLE clustering concept.

3.3. Properties of stack-RLE clustering. The stack-RLE clustering approach has different additional properties that are described below and come in handy for (large-scale) applications in scientific computing; specific examples in the context of hyperbolic PDEs are described in section 4.

3.3.1. Shared- and distributed-memory support. The MPI+X cluster property (3) of our approach realized via the sparse-graph connectivity described above directly yields the applicability to shared- and distributed-memory systems: If

[2]This was independently discussed and proven in the context of an MPI-only parallelization and a stack-based communication in [2, 1].

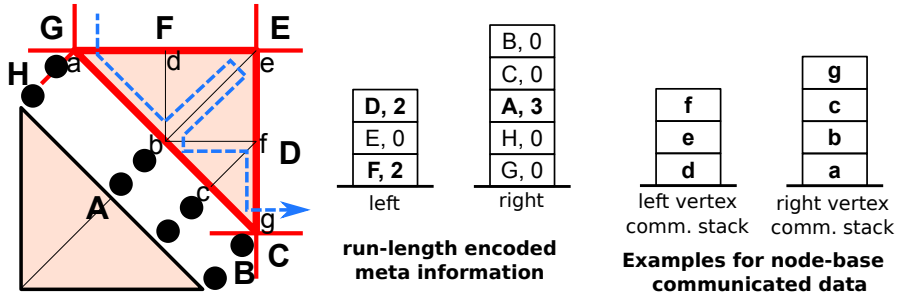

FIG. 3. *Example for run-length encoding of the connectivity information. All meta information given for the reference cluster is highlighted with the thick red border. Left: Clusters adjacent to the reference cluster are denoted by A–H, and explanatory communication data is denoted by a–g. Right: The RLE connectivity information for the data stored on the left and right communication stacks. Here, we focus on the communication of vertex data stored on the right communication stack (for edge-based communication data, see* [28]*): The first two entries* $(G, 0)$ *and* $(H, 0)$ *account for exchanging the first vertex data a to clusters G and H. The RLE* $(A, 3)$ *first of all represents three edges. Since four vertices are associated to three edges, we conclude that the four vertices* (a, b, c, g) *are exchanged with cluster A. The read-position of the vertex communication data is then increased by the number of edges. Then, vertex data* (g) *is exchanged with clusters B and C. Special attention has to be paid to the first and last vertices for generic cases; see* [26, sec. 5.2.4]. *(Colored figures available in electronic version.)*

a cluster is stored in the same memory context, a reference to the cluster is stored in the RLE and the communication data is transferred read-only from the adjacent cluster. For clusters stored in a different memory context, the MPI rank is stored in the RLE connectivity information. Data to be transferred is then sent and received via blockwise operations. Hence, we see the efficiency aspect of *cluster property* (3) now fulfilled. In both the shared- and the distributed-memory parallelization cases, we can efficiently use the RLE for these operations since the full bandwidth is only achieved with packages of sufficient size.

3.3.2. Forest of spacetrees. Assembling several spacetrees together generates a forest of spacetrees [9]. For valid connectivity, the simplex of a spacetree should be embedded into a regularly refined grid of a spacetree (see left image in Figure 1). For the Sierpiński SFC, this ensures that the communicated data are generated in the correct order. Since we can also invert the processing of the communicated data depending on cluster-local metadata (property (4)), we support arbitrarily shaped initial triangulations (see Figure 4 for examples). The real flexibility is demonstrated in the leftmost image with the initial domain triangulation based on a scalable vector graphics (svg) file import with automatic connectivity detection and initialization of the base triangles. In our implementation, we store all clusters in a (set-like) binary tree structure which allows fast access and execution of operations on clusters. Note that only branches to rank-local existing leaves are stored; hence the *forest of spacetrees is not replicated* on compute ranks.

3.3.3. Dynamic adaptivity and updating connectivity information for changing grids. The stack-RLE clustering obviously supports dynamic adaptive mesh refinement by construction. To realize dynamic adaptivity efficiently, fast algorithms to update the connectivity information of the RLE are required when the grid changes by insertion or deletion of vertices and edges. Therefore, adaptivity markers on communication stacks are used in our implementation [28]; see Figure 5. Here, adaptivity markers M_R and M_C are communicated via the edges. One single refine-

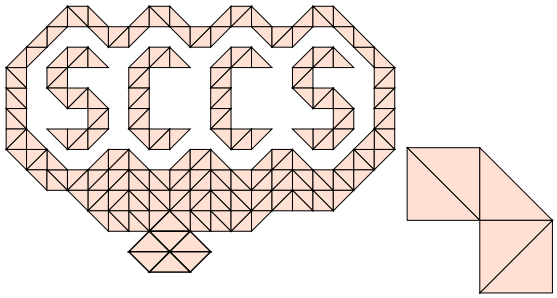

FIG. 4. *Initial domain triangulations which would violate the embedding into an existing regularly refined spacetree (see left image in Figure 1). By inverting the processing order of the inter-partition exchanged data, this still results in a valid connectivity. Note that by additional tree split operations, embedding of such domains becomes feasible. (Colored figures available in electronic version.)*

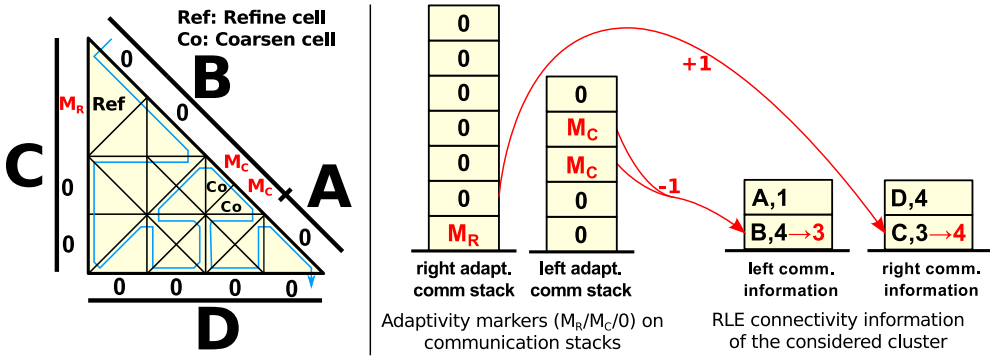

FIG. 5. *Schematic depiction of RLE connectivity update: A–D denote adjacent clusters. At the last adaptivity traversal, refinement markers M_R are transferred via edges which are split for refining the edge, and coarsening markers M_C are communicated along the cathetii of cells which should be coarsened (left image). The stack is analyzed after this traversal (middle image), and based on this stored information, refinement markers M_R increase the RLE, and coarsening markers M_C decrease the RLE-stored connectivity information. (Colored figures available in electronic version.)*

ment marker M_R accounts for an edge which is split by refining the cell. Therefore, the associated RLE has to be increased by one. Coarsening markers are sent along the cathetus involved for coarsening, and two of these markers (for consistency reasons they always have to appear pairwise) account for decreasing the corresponding RLE element by one. This accounts for removing one vertex by merging two cells. This avoids computation and communication of destination compute ranks as typically used in approaches with explicitly updated connectivity information, resulting in an implicit update mechanism with location-independent adaptivity markers. Instead of adaptivity markers, one can also communicate unique identifiers for each cluster (e.g., cluster-ID and rank-related information) via the edges to reconstruct the RLE connectivity information.[3]

Changing the number of edges in the RLE entries also automatically updates the number of *vertices*. A zero-encoded RLE does not require any special handling since

[3]Our RLE concept was inspired by [36] (ver. 2011; based on SFC-cuts in [5]), which explicitly communicated MPI ranks via edges, supporting only MPI and edge communication.

it is associated to an element on the communication stack which is positioned relative to the updated edge-communication RLE information.

3.3.4. Cluster migration. Another property of stack-RLE clustering is the possibility of easily migrating clusters to allow for load balancing. To realize the cluster-based data migration, we use the property of location-independent intracluster communication and the intercluster communication represented by the edges of the connectivity graph; see Figure 3. First, we compute the global enumeration of all cells based on the number of cells in each cluster with a parallel prefix sum; see, e.g., [16]. Then, the cluster is annotated with the MPI rank to which the cell index of the cluster's SFC-enumerated middle cell has to be migrated for optimized load balancing. So far, this follows well-known load balancing approaches and data migration strategies.

The data migration itself is then accomplished in a cluster-oriented manner by migrating the raw cluster data. This is highly efficient since most of the data are stored in streams, and grid metainformation is inferred during grid traversal without explicitly storing it. Only the intercluster connectivity via edges requires further updates, which is now based on *explicitly* transferred MPI ranks. We emphasize that updating the connectivity information has been changed from a per-hyperface way to the cluster approach for a distributed memory parallelization. This is the main difference from all previously used approaches for stack-based communication and in particular the only algorithmic part of communicating MPI ranks for keeping consistent connectivity information.

3.3.5. Dynamic cluster generation. Several cluster generation strategies can be implemented with our parallelization approach. The threshold-based and scan-based approaches were evaluated in detail on shared-memory systems; see [30]. In this paper we evaluate both methods for distributed-memory systems. For a threshold-based method, splitting a cluster is requested as soon as the number of cells exceeds the threshold T_s. When the number of cells of two clusters sharing the same parent in the spacetree lies below the threshold T_j, we join these clusters.[4] These thresholds are sensitive to simulation characteristics such as the number of cores and the problem size; see [28]. For a scan-based method, we split the clusters which would overlap with an optimal SFC-partitioning approach (see [30]). After splitting or joining clusters, the RLE connectivity information has to be reconstructed; here we present two approaches which only rely on communication on the adjacent clusters and hence are without collective global operations.

Edge-connectivity information. This type of connectivity is important for communication data which are exchanged among the edges. Such communication data are relevant for, e.g., edge DoF of discontinuous Galerkin solvers and its limiters, DoF for finite-volume solvers, adaptivity markers to generate a consistent grid, and markers to update the RLE connectivity information. To split a cluster, we need information about (a) the number of shared hyperfaces along the separating hyperfaces (connectivity data) and (b) the association of persistent data to both child clusters.

With the ordering of the elements on the communication stacks, the new connectivity data (requirement (a)) during the split operation can be inferred [28]; see Figure 6. Here, we *stop our grid traversal* at certain positions in the grid and then *implicitly* derive the required quantities on the number of hyperfaces and cells for the

[4]For terminology reasons, we mention that "splitting" and "joining" refer to repartitioning of dynamically adaptive grids and not to refining and coarsening operations on cells.

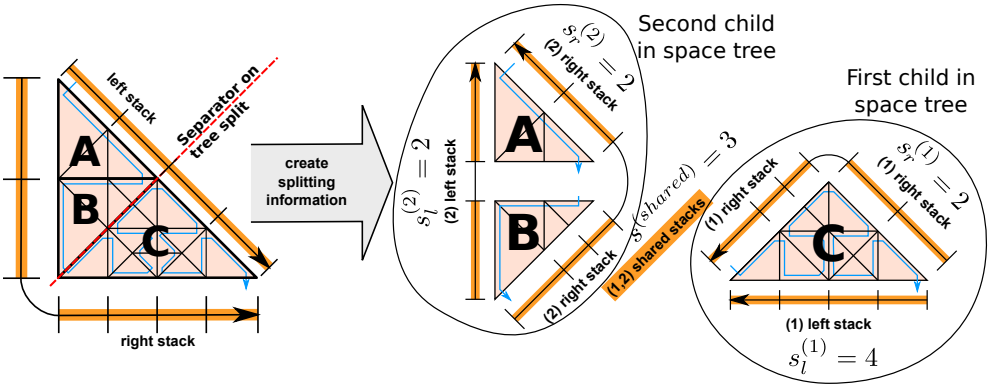

FIG. 6. *Reconstruction of RLE connectivity information during grid traversal for tree-split-based cluster generation.* $s^{\{1,2\}}_{\{l,r,shared\}}$ *represents the number of edges on the left (l) and right (r) communication stacks or the shared edges for the* 1*st and* 2*nd split clusters. The traversal stops at particular steps during the grid traversal to compute the values for s based on the number of elements stored on the communication stacks. The shared number of elements can be inferred by considering the changes on communication stacks before and after traversing partition B. This yields a valid RLE connectivity information without communicating any adjacency related information via edges. (Colored figures available in electronic version.)*

split. We refer to all this information as *split information*.

The derivation of the split information is joined in our implementation with the last backward adaptivity traversal. This avoids an additional grid traversal by joining two traversals (see also [37] for merging traversals). A sketch of this is given in Figure 6. We annotate the spacetree nodes on the 2nd level relative to the cluster to be split and by following the SFC in the reverse direction with (a, b, c, d). Now, we consider partitions (A, B, C) generated by cells of the leaf nodes of subtrees $(a, b, c \cup d)$. We denote the stacks for the left and right adaptivity communication stacks by $\mathbf{S}_{left}$ and $\mathbf{S}_{right}$. $\mathbf{C}$ stands for the cell stack. We determine the number of cells of the parent element from the elements stored on the cell stack: $c^{(parent)} := |\mathbf{C}|$. $s^{(1)}_{\{left,right\}}$ stores the information on the split operation for the first child and $s^{(2)}_{\{left,right\}}$ for the second child on the 1st level. The number of new edges shared among both children of the cluster is computed in $s^{(shared)}$. Furthermore, the number of persistent data (cells) associated to both child clusters is stored in $c^{(1)}$ and $c^{(2)}$.

The following steps allow us to compute the new clusters (note that this information is inferred via a backward traversal):

1. *Partition A*: After the traversal of cells in partition A, the communication edges on the left stack for the second partition B are stored as
$$s^{(2)}_{right} := |\mathbf{S}_{left}| \ .$$
2. *Partition B*: After the traversal of cells in partition B, one computes the shared edges by
$$s^{(shared)} := |\mathbf{S}_{left}| - s^{(2)}_{right},$$
and the number of parent intercluster edges for the 2nd partition on the right stack is saved in
$$s^{(2)}_{left} := |\mathbf{S}_{right}|.$$

The processed number of cells which is then associated to the second subpartition created by the split operation is inferred by

$$c^{(2)} := c^{(parent)} - |\mathbf{C}|,$$

where $|\mathbf{C}|$ is the remaining number of cells to be processed. This also represents the number of cells assigned to the first subpartition C:

$$c^{(1)} := |\mathbf{C}|.$$

3. *Adaptivity*: Due to the adaptivity traversal (cf. section 3.3.3), the information on the shared edges $s^{(shared)}$ only represents the quantity of shared hyperfaces before refining and coarsening cells in the grid. Based on the corresponding refinement and coarsening markers M_R and M_C, the number of hyperfaces $s^{(shared)}$ has to be updated (see section 3.3.3).
4. *Partition C*: After the traversal of cells in C, the information to reconstruct the communication information is then given by

$$s^{(1)}_{right} := |\mathbf{S}_{left}| - s^{(2)}_{right}$$

and

$$s^{(1)}_{left} := |\mathbf{S}_{right}| - s^{(2)}_{left}.$$

If partitions A and B do not exist, clusters with leaf elements stored on level 1 or 2 need to be handled appropriately. In the example case of a cluster consisting only of two cells, we can directly set $s^{(1/2)}_{right} := 1$, $s^{(1/2)}_{left} := 1$, $s^{(shared)} := 1$.

With the derived information, we perform the split operation a posteriori: The *persistent cell data* (cf. requirement (b)) on the stack can be split directly using the split information $c^{(\{1,2\})}$. To account for the new traversal start, the *traversal metainformation* (providing, e.g., the starting point of the grid traversal) has to be updated. The *RLE communication information* of both children can also be determined based on the number of shared edges of both children given in $s^{(\{1,2\})}_{\{left,right\}}$ and $s^{(shared)}$. Note that no global communication or global operation such as a global grid traversal is required to perform these operations for the split.

Joining clusters is rather straightforward and can be implemented with the following steps:

1. Concatenate the cell data storage,
2. join both clusters' traversal metainformation, and
3. join both clusters' communication metainformation and remove the RLE information about the edges shared by both child clusters.

So far, this results in valid connectivity information for the clusters involved in the local split/join process. For adjacent clusters in the same memory context, the information on split connectivity information is inferred by directly accessing the adjacent cluster's connectivity information. We can distinguish between different constellations which are depicted in Figure 7. The clusters generated by split/join operations are accessible by pointers to/from their parents and children. Hence, accessing the split/joined clusters does not require storing the entire spacetree. All possible cases which can occur due to split/join constellations of two adjacent clusters are then processed (see Figure 7). After the split/join operations, an iteration is performed over all RLE communication elements which tests the local cluster and the adjacent cluster states.

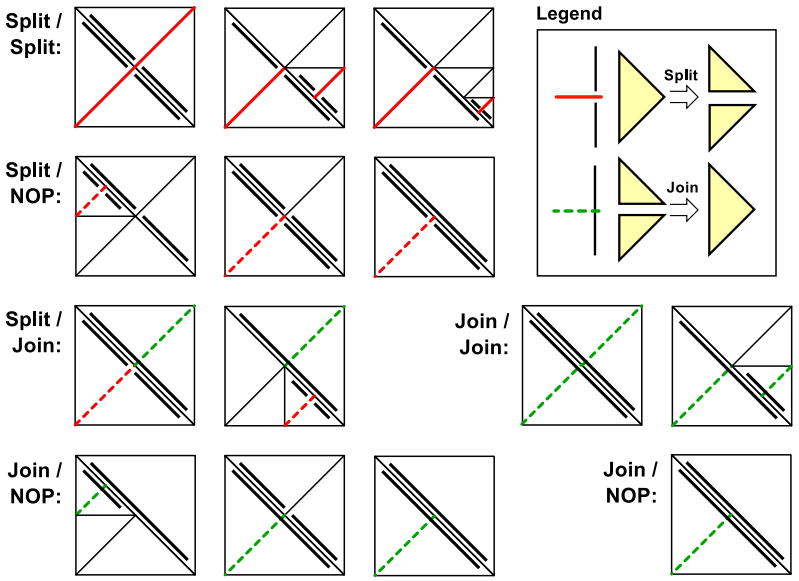

FIG. 7. *Possible constellation edges shared among clusters due to split and join states. (Colored figures available in electronic version.)*

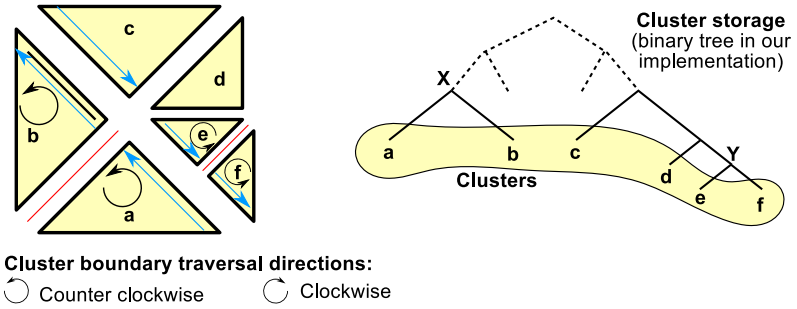

FIG. 8. *Example of split/split state: As split into e and f. The connectivity data has to account for the split states of the adjacent cluster. The traversal directions (blue arrows) are in either the clockwise or the counterclockwise direction. (Colored figures available in electronic version.)*

In the following, we give one example for a split/split case (Figure 8, left) to explain how to reconstruct the connectivity information. It is important whether the first or second adjacent split child node along the SFC is traversed first or second to append additionally required RLE connectivity information in the correct order. Let the direction flag $D = 0$ denote a traversal order to reconstruct the connectivity information *first, then second child* and $D = 1$ denote the case *second, then first child*. Two rules are needed that potentially change the direction flag D:

- *Rule* (1) Traversal order of adjacently split children: If the cluster boundary traversal directions of the local and the adjacent parents' cluster are identical, the cluster traversal order (for updating the RLE) of the adjacent children has to be in the same direction ($D := 0$); otherwise it is reversed ($D := 1$).
- *Rule* (2) Traversal order of locally split children: If the RLE of the second local child is updated, the traversal order is reversed ($D := 1 - D$).

We now apply the two rules to child cluster a in the considered example shown in

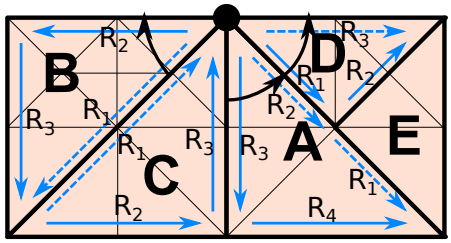

FIG. 9. *Reconstruction of the vertex-connectivity information based on the* edge *RLE connectivity information R_i for cluster C. A trace of the clusters sharing the considered vertex is generated by following the RLE connectivity information R_i associated to the same vertex (see arrows in picture). This allows a reconstruction of consistent intercluster vertex-connectivity information: For both RLE connectivity edges (R_1, R_3) which share the vertex, the adjacent clusters are traversed. First, we follow R_1 in cluster C, leading to cluster B. The next RLE element in cluster C which also shares the same vertex is given by R_2, which is a boundary edge. Hence, the trace generation for this direction is finished. Second, we follow R_3 in cluster C, leading to cluster A. Here, R_2 shared the same vertex, leading to cluster C and finalizing the search since RLE R_3 in cluster D is a boundary edge. Finally, the clusters which were accessed during the trace generation are stored as* 0*-RLE tuples in the connectivity information. (Colored figures available in electronic version.)*

Figure 8. Rule (1): The traversals on the cluster boundary of the shared edges between clusters a, e, and f are all counterclockwise; we set $D := 1$ to account for opposite directions. Rule (2): Since cluster a is the first child's triangle, the adjacent traversal order is not changed. In total, $D = 1$ leads to a reversed, second-first traversal of the adjacent child clusters (i.e., cluster f is searched first for adjacent edge parts, followed by cluster e). Each traversal results in inserting corresponding new RLE entries to the RLE metainformation, replacing the RLE entry associated to the split parent cluster. For these operations, neither is a spacetree traversal required, nor does the structure of the forest of spacetrees have to be stored (see [28, 26] for a detailed description of this algorithm).

For clusters stored in another memory context, the connectivity information is updated based on information about the split or joined connectivity transferred via MPI send/recv. Hence, this involves nonglobal communication operations and a single pass only.

Vertex connectivity information. This type of connectivity information can be used, e.g., to implement flux limiters for discontinuous Galerkin simulation. For our case, we use it to construct a continuous surface of the underlying discontinuous approximated solution. The vertex-connectivity information approach was first used in [31]; here we summarize the approach and provide details on the implementation. We can reconstruct the 0-length encoded vertex-connectivity information based on the consistent edge RLE connectivity information; see Figure 9 for an example. Without loss of generality, we only consider periodic boundary conditions. If the cluster also shares the vertex, two edge RLE connectivity elements are associated to a single vertex in each cluster. For shared-memory systems, this allows traversing over all clusters sharing the vertex by following the vertex-associated edge RLE connectivity information. We can generate *a trace of the clusters sharing the vertex.* The limitation of up to eight adjacent cells per vertex (here we assume that angles less than 45 degrees should be avoided since this would result in stronger time step restrictions) results in a *constant runtime* for each shared vertex. Under the assumption that *avgRLE* edge connectivity information entries are stored in average per cluster, this results in a runtime complexity of $O(\#Cluster \cdot avgRLE)$ for the reconstruction, with *avgRLE* being

a relatively low average number of RLE elements for edges per cluster (cf. [31]). Since we are interested in the vertex-based communication for visualization purposes, we focus on shared-memory systems only. Note that the limitation of up to eight vertex-adjacent cells allows for a direct reconstruction of vertex-connectivity information also on distributed-memory systems with only up to six iterative communications via the RLE edge connectivity information. In this way, in a constant number of passes one could also compute consistent connectivity information for vertices without any global communication. Our approach has its origins in a shared-memory implementation; this was also independently developed in the context of petascale simulations for Reactor Hydrodynamics to avoid global collectives with MPI-based distributed-memory parallelization; see [11].

4. Extensive case studies with stack-RLE clustering. After the theoretical discussion and summary of the stack-RLE clustering approach in the previous section, we now apply it to different relevant tasks in the context of scientific computing and perform a detailed evaluation. The following results are based on shallow-water finite volume simulations executed on dynamically changing grids. We extend the first small-scale scalability test from [27] to a large-scale distributed-memory system to show the large-scale applicability of our approach. This is followed by two advantages of our approach regarding the visualization of simulation data: For online visualization (via OpenGL), consistent vertex data for discontinuous discretizations at cluster interfaces is provided. A backend for offline visualization allows us to efficiently write simulation data to persistent memory with the focus on a shared-memory node. An analytical benchmark is then used to validate the implementation of the finite-volume solvers and to show the runtime improvements of the dynamical adaptive grids. Finally, we show the applicability to a realistic scenario: a Tohoku tsunami simulation.

4.1. Scalability studies. For our scalability studies, we use a discontinuous radial dam break. The size of the simulation domain is set to $5000m \times 5000m$, and the radial dam break center is placed at $(2500m, 2000m)$ with a radius of $500m$. We use a finite-volume with local Lax–Friedrichs flux solver and an explicit Euler adaptive time stepping method. These studies were conducted with single precision. Load balancing algorithms are executed after each time step. The adaptivity traversals to refine/coarsen grid cells also involve computing the cell's vertex coordinates.

4.1.1. Small-scale scalability. We conducted several benchmark studies on up to 512 cores which show the influence of several parameters which the clustering offers. These benchmarks were conducted on the CoolMUC2 cluster system (Dual Socket, 14-core Haswell, FDR14 Infiniband interconnect, 64 GB memory per node).[5] The simulation grid is initialized with a regular refinement depth of 22 and 8 additional refinement levels. For the cluster generation, we evaluated two different strategies: The first one is referred as the *threshold* method with two threshold values: A threshold of $T_s := 4096$ for splitting a cluster if the number of cells exceeds this size and $T_j = T_s/2$ for joining clusters if the number of cells of both clusters undershoots this value. The second cluster generation method is referred to as the *scan* method which splits the clusters (note that in this work we only focused on clusters generated by subtrees of the spacetree) to optimize the overall workload balance, and we assume a heterogeneous workload.[6] The simulation for the initial radial dam break is ex-

[5] https://www.lrz.de/services/compute/linux-cluster/overview/

[6] This is not given, e.g., for the augmented Riemann solvers which are used for the experimental tsunami simulations.

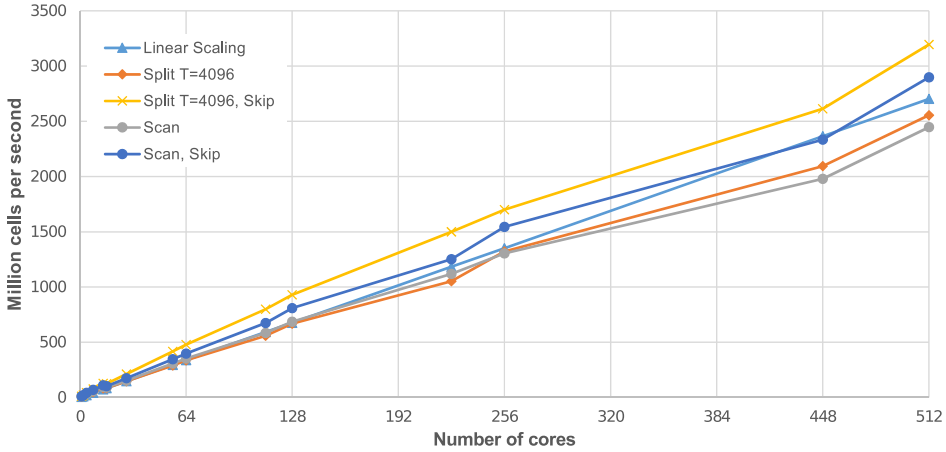

FIG. 10. *Scalability on the CoolMUC2 system. We evaluated several variants of cluster-generation and cluster-based optimizations. The threshold-based cluster generation together with the cluster-based optimization of skipping adaptivity traversals of already conforming clusters results in a robust performance boost also in the context of distributed memory systems. (Colored figures available in electronic version.)*

ecuted for 100 iterations: In each iteration, we successively execute a time step of the simulation-adaptivity traversals and the (threshold-based) cluster generation, followed by our cluster-based data migration. We conducted a variety of studies—each time with different constellations and measurements—to gain insight into the simulations. All of these studies include the overheads of the function calls to measure the detailed runtime.

The first study is given in Figure 10, which shows the scalability per-se and contains different variants of cluster-generation strategies (threshold-based and scan-based; see above). The baseline for the linear scalability of this plot is given by the performance of all 28 compute cores on one MPI shared-memory domain.

For *Split* $T = 4096$, we can observe an efficiency of 95% for the standard threshold-based cluster generation method on 512 cores.

Next, we discuss the skip-based optimizations of the threshold- and scan-based approach; see *Split* $T = 4096$, *Skip* and *Scan, Skip*. Since our development strongly relies on a conforming grid, each time the grid structure changes this may require additional conformity traversals. There is one significant side-constraint of a grid communication scheme which is based on a stack-based communication approach: With the previously developed approaches, there are two constraints limiting possible optimizations: The first constraint is given by the entire partition to be traversed (see approaches used in [5, 36, 21]) since only one partition is allowed per memory context, and the communication scheme allows MPI only. The second constraint consists of a coloring of the shared interfaces, generating certain dependencies or a requirement of mutual exclusion on hyperfaces shared among the partitions requiring, e.g., an execution of the traversal (see [31]). We circumvent both constraints with our RLE clustering approach by (a) multiple partitions allowing for skipping traversals, (b) a replicated data layout also on shared-memory systems, and (c) using an RLE for efficient communication. With such a cluster-based optimization, we can efficiently skip conformity traversals of clusters, which results in a robust performance improvement of 25% (*Split* $T = 4096$ versus *Split* $T = 4096$, *Skip* in Figure 10).

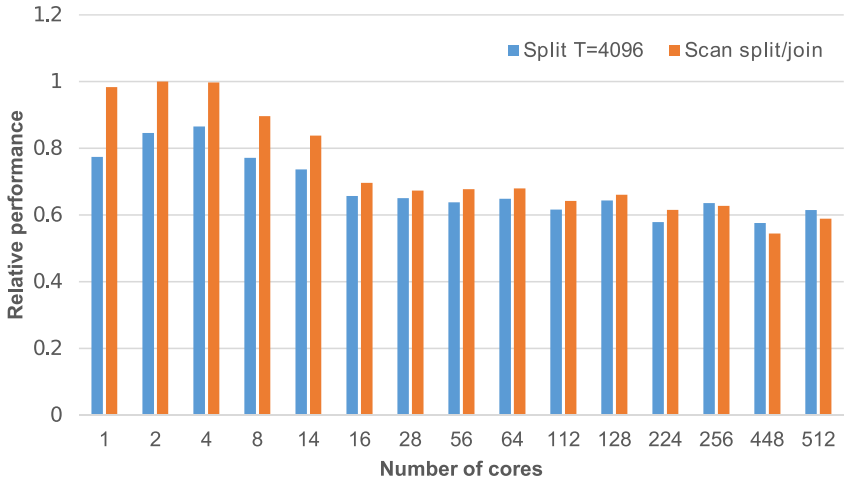

FIG. 11. *Performance comparison between threshold- and scan-based cluster generation methods. We observe a higher performance for scan-based clustering with low-core numbers, whereas the performance is higher for threshold-based clustering for higher core numbers. (Colored figures available in electronic version.)*

Concerning the scan-based cluster generation which splits clusters in a way to improve the load balance, we observe that the performance is below the threshold-split-based one. A detailed comparison of threshold versus scan-based cluster generations shows a dependency on the number of cores and is given in Figure 11. With the scan-based cluster generation (red) being less performant compared to the threshold-based method (blue) on 512 cores, we can observe a different behavior on fewer cores: Here, the scan-based strategy is up to over 20% more performant. We account for that by several reasons: First, our cluster-generation method is based on tree splits. With the clusters generated by tree splits, a close-to-optimal load balancing requires multiple tree splits. This also generates clusters far below the split-size T_s of the threshold-based method. Here we can observe this migration of many small clusters in overheads of the migration time: Runtimes for the cluster-based data migration are about two times larger for the scan-based method compared to the threshold-based one.

Next, we gain insight into the different phases (adaptivity, simulation time step, load balancing, etc.) and how this relates to the overall performance. Detailed timings are given in Figure 12. The left side shows the timings for the threshold-based clustering and the right side additionally with cluster-based optimizations (cluster skipping). We observe a similar runtime for the simulation time-step computations and for the adaptivity traversals; the skipping of traversals on already conforming clusters results in performance benefits also on distributed-memory systems.

We next evaluate possible performance differences when using OMP or Threading-Building-Blocks (TBB) in the context of a hybrid parallelization (MPI+X). Since our inter-rank data exchange via RLE connectivity information is only based on MPI ranks, the inter-rank shared interfaces have to be send/recv and processed (in the case of flux computations) by a single thread.[7] The operations (execute traversal,

[7] One obvious possible optimization of our RLE clustering is to extend this information, e.g., by global cluster IDs, rank-local cluster IDs, processing thread IDs, or other unique identifiers.

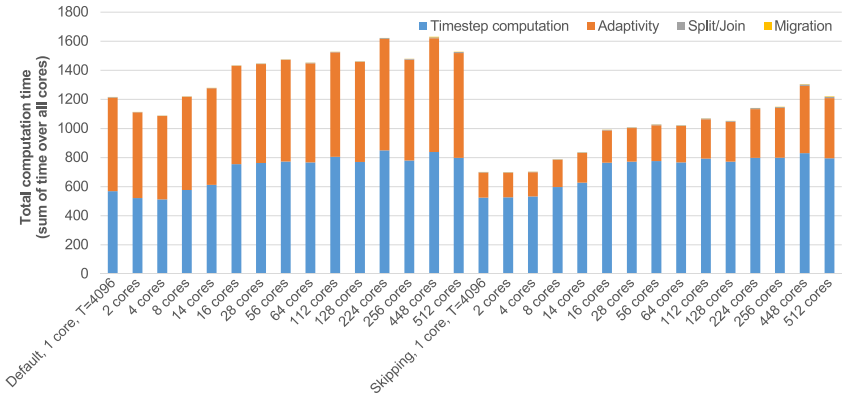

FIG. 12. *Detailed performance breakdown of the total time (sum over time on all cores). (Colored figures available in electronic version.)*

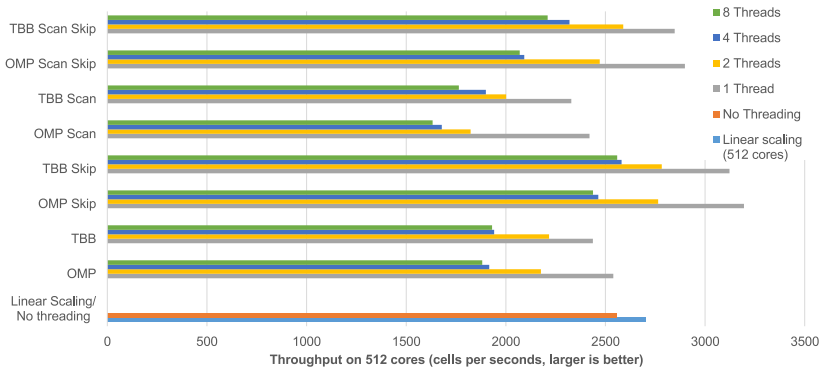

FIG. 13. *Performance results on 512 cores for different variants of threading libraries (OMP/Threading-Building-Blocks) and cluster generation. We observe that TBB always results in improved performance as soon as more than one thread is used. (Colored figures available in electronic version.)*

send/recv data, execute conformity traversals, etc.) on clusters are executed by a recursive OMP/TBB task generation (see [30] for "parfor" execution results). Figure 13 shows several performance results with a purely nonthreaded split-based method and linear scaling on 512 cores given at the bottom of the plot. We see that the peak-performance for threaded versions is the best for OMP only in the case of a single-threaded version. In fact, there are no significant overheads if executing the OMP-parallelized variant with only a single thread (see single-threaded bar next to OMP). In contrast, the TBB version always has overheads by the task generation (this always involves hand-coded packing of data into task classes to execute the operations via the worker threads). As soon as more than one thread is used in the simulation, the overheads of TBB are smaller compared to OMP. Again, we observe significant performance boosts for cluster-based optimizations by skipping traversals on an already conforming cluster.

Finally, we provide an in-depth analysis of the cluster-based data migration approach before discussing Figure 14. We briefly repeat a major aspect of our approach: Our novel cluster-based data migration avoids storing and migrating the connectiv-

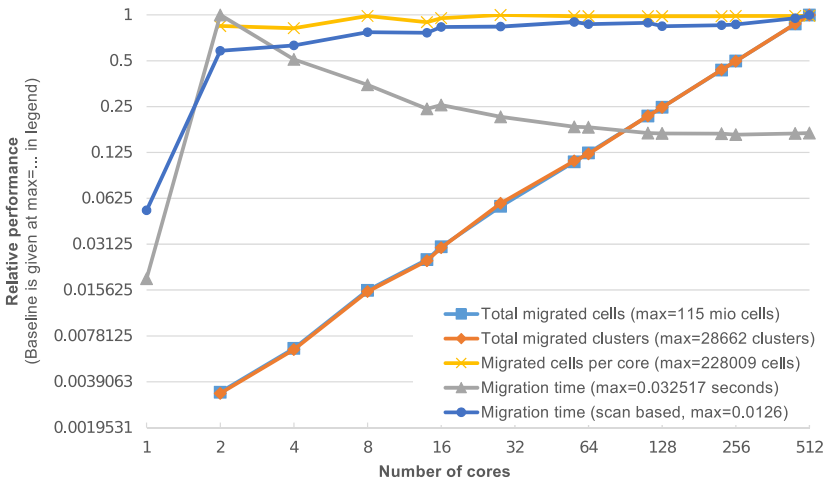

FIG. 14. *Max-normalized timings of migration overheads. (Colored figures available in electronic version.)*

ity information per-cell as is the case in all previously developed stack-based data exchange approaches ([21] based on [36]). Here, we shall also mention the p4est [9] development which does not rely on a stack-based communication but computes the connectivity information implicitly and avoids storing it in per-grid primitive before data migration. Our RLE clustering approach allows storing all the connectivity data only on a cluster-granularity level with an efficient RLE. Hence, additional data overheads from (a) explicitly storing the edge/node-based connectivity information and (b) migrating this additional data are avoided. In particular, for vertex/node-based communication, this can lead to significant bandwidth-reduction benefits in the case of low payload per cell (see [31] for a payload model) and to improvements on future high performance computing (HPC) architectures with network bandwidth becoming a significant performance bottleneck. In Figure 14, we plotted several categories for migration performance data over 100 simulation time steps (setup routines are not included) with the default threshold-split-based ($T_S = 4096$) cluster generation: The total number of migrated cells, the total number of migrated clusters, the migration time itself, and the migrated cells per core. For a better comparison, all values were normalized with the maximum of each category (see max = ... information in the legend). We observe a direct relation between the number of migrated clusters and the number of migrated cells. This is due to the threshold-based cluster generation, which generates a number of clusters depending on the number of cells per clusters, hence also relating both values for data migration. Regarding the migration time, we observe a maximum of migration time at two cores which is successively decreasing with an increasing number of cores for a split-based approach. The reason for this is the larger number of clusters per compute rank when involving only two ranks. Here, traversing over the larger number of clusters accounts for these overheads which successively decrease with a larger number of cores and hence fewer clusters. Using a scan-based cluster generation leads to steadily increasing data migration overheads (see scan-based migration time).

For these benchmarks, the data migration overheads and split/join operations are negligible (yellow in Figure 12). This is due to three reasons: First, the repartitioning is done with the RLE cluster algorithms. Second, as an implication of the RLE, the

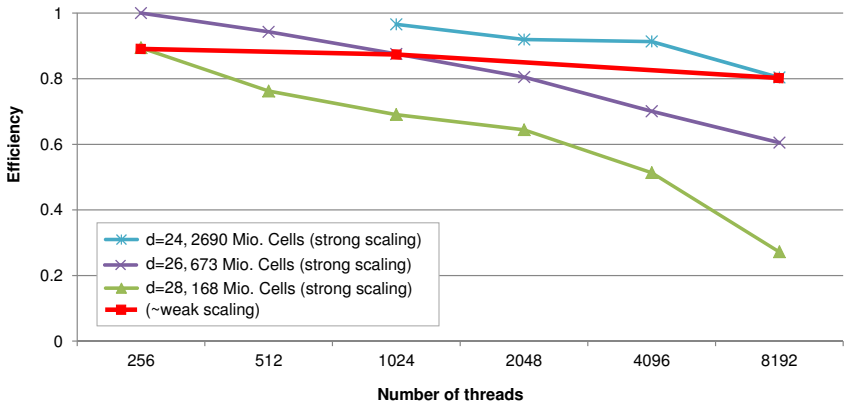

FIG. 15. *Scalability on up to one SuperMUC island with* 8192 *cores. We tested different initial refinement depths* (d) *with each benchmark marked with the average number of simulation cells of the dynamically adaptive grid over the runtime. (Colored figures available in electronic version.)*

required bandwidth for data migration of a single cluster with a certain number of cells is quasi-optimal since no connectivity information is stored in the cells. Third, the workload per core compensates overheads. This combination leads to a highly efficient data migration approach.

4.1.2. Large-scale scalability. The large-scale test is based on three strong scaling benchmarks, was executed on the Sandy-Bridge based SuperMUC,[8] and is intended to show limitations of a clustering approach based on tree splits. The simulation domain is a quadrilateral assembled by two triangles. A cluster split threshold size of $T_s := 32768$ was chosen, and we tested the scalability with different initial refinement depths $d := \{24, 26, 28\}$ and up to eight additional refinement levels for the dynamically changing grid. We executed the simulation for 100 time steps, and the benchmark measurement is started after no further cluster migration during the setup phase is done. It is important to mention that each time step of the simulation involves evaluating the adaptivity criteria, refining or coarsening the mesh according to the adaptivity criteria creating a consistent mesh, performing cluster generation, and applying data migration algorithms. A detailed analysis of the different phases can be found in the previous section. In this section we focus on limitations of the clustering approach.

Figure 15 shows the results for this scalability test with the initial refinement depth $d := 26$ used as the baseline for the 100% efficiency on 256 cores. The scalability of the $d := 24$ with about $168 * 10^6$ cells on average per time step shows a nonoptimal performance for a larger number of cores. This is mainly due to the cluster threshold size assigning an insufficient number of clusters to each computing core to allow load balancing on a single node with our threshold-based cluster generation. For $d := 26$ we observe an efficiency of 60% for a strong scaling with fully dynamic adaptive grid algorithms with mesh refinement and coarsening in each time step. On 8192 cores the average number of cells shows a load imbalance of over 20% due to the threshold-based tree splits and is the main limiting factor. Smaller threshold sizes or alternative

[8] http://www.lrz.de/services/compute/supermuc/

cluster shapes and cluster generation methods, e.g., based on range information [30] or SFC-cuts [5], can lead to improved performance. Using the benchmark $d := 28$ to get a pseudoweak scaling, we reach over 90% efficiency on 8192 cores compared to $d := 24$ with 168 mio. cells (see red line in Figure 15).

4.2. Stack-RLE for visualization. In this section, we describe the advantages of our stack-based RLE clustering approach concerning the visualization of simulation data. First, we discuss performance improvements for a Visualization ToolKit (VTK) binary file output for offline visualization by using writer tasks. Then, we present an online visualization via OpenGL which requires efficient online processing for a discontinuous solution stemming from finite-volume simulations.

4.2.1. VTK file backends. Using the VTK file backend, data is written to persistent memory. However, lower bandwidth is available to access such a persistent memory compared to the main memory. We evaluated five different implementations to analyze alternatives to overcoming the idling of cores that are waiting until the data have been written to persistent memory:

- *No output*: This benchmark does not write out any simulation data and is used to determine the maximum achievable performance.
- *Default (blocking)*: The default output method blocks the writing to persistent memory until the function which is called to write all simulation data to persistent storage finishes its execution.
- *pthread*: A separate thread is started which writes the simulation data in the background to hard disk. Here, the simulation continues on all available cores. This can lead to a single core shared among the writer and a simulation thread and hence conflicting computational resources.
- *pthread, lastcore*: This execution is similar to the aforementioned pthread execution but does not use the last core for the simulation. This aims to avoid resource conflicts; see above.
- *Writer task*: Using TBB for thread initialization, we can use TBB fire-and-forget tasks [20]. These tasks are enqueued to a working queue without any thread waiting for the finishing of the task.

The domain for the simulation is a quadrilateral split into two triangles, and the simulation grid is initialized with a regular refinement depth of 10 and with up to 16 additional refinement levels. The simulation computes a radial dam break with the Rusanov flux solvers for 201 time steps. Such a simulation results in 4.55 mio. cells processed on average per simulation time step and with binary VTK file sizes above 300 MB. The output data itself is preprocessed and written to consecutive output arrays in parallel using all available cores. We used an Intel platform (four socket system with each socket equipped with an Intel Xeon CPU E7-4850@2.00GHz with 10 cores per CPU) and wrote the simulation output data to persistent memory (Western Digital Hard Drive of the type Red 2 TB with a 64 MB cache and a theoretical transfer rate of up to 6 Gb/s). Results for different frequencies of writing output files to persistent memory are given in Figure 16 and are discussed in what follows.

The *blocking version* shows a clear disadvantage compared to the other methods since cores idle until the function which writes the output data finishes writing the data. Such idling cores are partially compensated for by the two pthread versions showing similar improvement results. However, the dedicated writer core which we implemented to avoid oversubscription of cores (pthread, lastcore) leads to decreased performance of 0.85%, 3.68%, and 0.42%, respectively, for writing output files at each $B = (25, 50, 100)$ time step. Hence, avoiding resource conflicts does not result in a

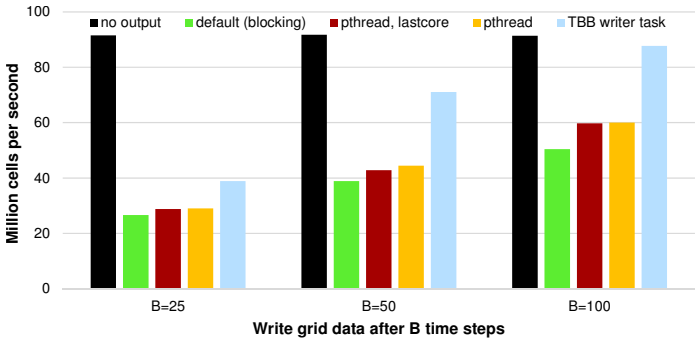

FIG. 16. *Benchmark statistics with a million cells per second processed and for different output backends. The parameter B specifies the number of time steps when data was written to persistent memory. (Colored figures available in electronic version.)*

robust performance improvement for the tested simulation parameters. The results indicate that the *oversubscription of cores should be used* for this scenario. With *TBB* fire-and-forget tasks, we get a robust performance improvement compared to all other writer methods. This can be explained by avoiding overheads of pthread creation and hiding the bandwidth bottleneck by conflict-free concurrent execution of the writer task and the simulation. Once the writer task finishes its execution, the clustering allows work stealing of tasks enqueued to other worker queues by the core that was writing the data. Furthermore, for $B = 100$, the performance loss for writing data to persistent memory is only 4% compared to writing no simulation data. It is obvious that it will decrease even more for $B > 100$ time steps.

4.2.2. OpenGL. With an online visualization using OpenGL, we can directly render the simulation results and interactively steer the simulation (see Figure 1) by changing the grid structure and updating the DoF of, for example, the water-surface height during the simulation. In this section, we focus on the algorithmic aspects of a reconstruction of a closed elevated surface for an online visualization with our stack-RLE clustering. Using a finite volume or discontinuous Galerkin method, a direct visualization of the water surface would lead to gaps and also to a distraction from the data; see Figure 17 for an explanatory visualization of a selected time step of a radial dam.

Our visualization is generated on dynamically changing simulation data stemming from finite volume (FV)/discontinuous Galerkin (DG) simulations. To obtain a continuous (gap-free) surface out of a discontinuous solution, we have to compute both vertex and normal data on the fly. The vertex data is based on averaging a particular datum stored in cells, and the normals are computed with additional traversals, based on the vertex data for shading purposes (see [18] for further information). Figures 17 and 1 (bottom left) show examples of the realized gap-free visualization.

Alternative algorithms which use the cell data and geometry as input and rely on sort operations are typically of $O(n \log n)$ complexity for n cells. With respect to computational complexity, all our algorithms concerning the pure grid traversal inside each cluster are of $O(n)$ complexity (see section 3.3.5). For the additional intercluster data communication, we obtain an $O(\sqrt{n})$ complexity for the execution of reduce operations on intercluster shared hyperfaces and the involved data exchange, since the

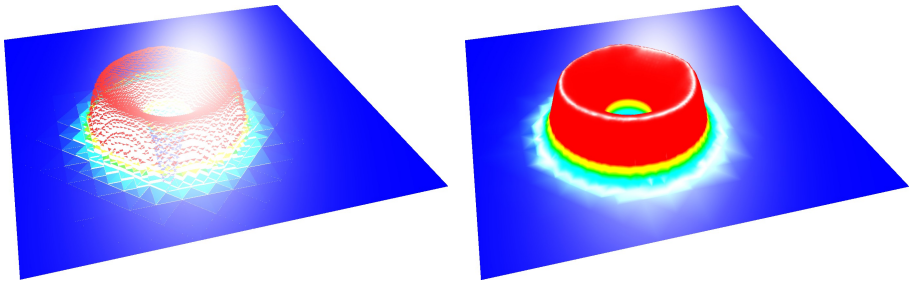

FIG. 17. *Visualization methods of the elevated surface of a shallow-water simulation. Left image: The direct visualization of the finite-volume simulation leads to white gaps in the surface. Right image: The closed surface leads to less distraction and improved possibility of analyzing the data. (Colored figures available in electronic version.)*

number of edges on the boundary of a cluster consisting of n cells can be estimated by $\sqrt{n}$. Furthermore, the reconstruction of consistent connectivity information is of constant complexity per cluster, and we further assume that the number of clusters is kept constant, e.g., by adopting the splitting threshold. Therefore, it is feasible to compute a closed surface with our vertex-based RLE connectivity information *in parallel* with an $O(n + \sqrt{n})$ complexity. For large problem sizes ($n \geq 18$ cells), the runtime for the reduce operation which is required for the replicated vertex data is compensated, which represents a clear improvement on the previous approaches for visualization methods.

We briefly discuss a method used in the AMR GeoClaw software [6] to visualize, e.g., tsunami data. Here, the cell-centered DoF are interpolated to a regular grid. Each element in the regular grid then stores the value of the leaf cells in a regularly refined spacetree. This would be applicable also for the tsunami simulations in this work since they would only require a moderate-sized regularly refined grid. However, as soon as the finest refinement level would lead to a resolution which would generate a regular grid which exceeds the available memory, algorithms such as the one presented in this section allow for an efficient construction of a closed surface. Also, the generation of an elevated surface as is shown in Figure 17 would not be possible with the data provided in the regular grid.

4.3. Analytic benchmark. We selected the Solitary Wave On Composite Beach (SWOCB) benchmark of the NOAA benchmarks [34] to quantify the benefits of the dynamic adaptive grids and the adaptivity criteria with SWEs [34][9] to show the applicability of the run-time adaptivity of the developed framework in the context of an analytic benchmark. Despite its high complexity with nonconstant bathymetry, an analytical solution is available [35]. We use this benchmark to show the feasibility of the adaptivity criteria and the potential of the dynamic adaptive mesh with refining and coarsening in every time step in a wave propagation benchmark scenario with a varying bathymetry.

4.3.1. Error indicator. With our main focus on computing the solution within given error bounds as fast as possible, only grid cells with a particular contribution (feature rich areas) to the final result should be refined. In this work, we developed a straightforward adaptivity criterion which is based on the steadiness of the height-quantity in each cell.

[9] http://nctr.pmel.noaa.gov/benchmark/Solitary_wave/

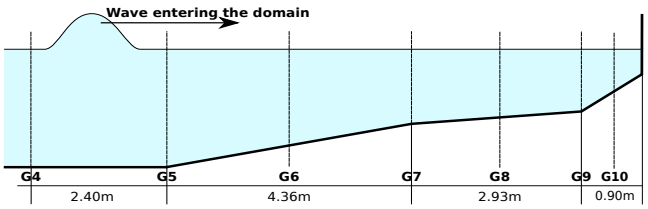

FIG. 18. *Scenario sketch for the "Solitary Wave on Composite Beach" benchmark. The placements of the gauge stations with selected ones plotted in Figure* 20 *are marked with G*. (Colored figures available in electronic version.)*

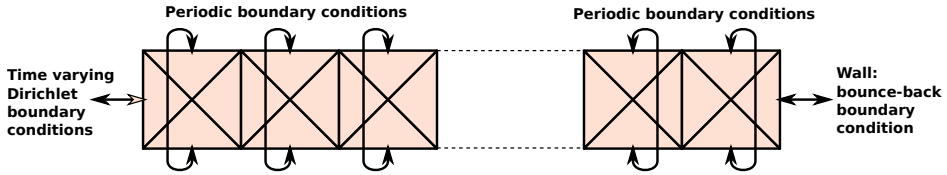

FIG. 19. *Base triangulation and boundary conditions for the "Solitary Wave on Composite Beach" benchmark. (Colored figures available in electronic version.)*

We denote the per-edge transferred net-update components Δh_i (the first height-related component of the augmented Riemann solver) with

$$I_{\text{net-update}} := \sum_{i=1,2,3} \Delta h_i |e_i| \ ,$$

where $|e_i|$ is the length of the triangle edge i. Then, each cell requests a refinement operation in each time step in case of $I > r$ and the cell commits to the coarsening process, in case of $I < c$. The effects of different parameters of r and c are going to be discussed below.

4.3.2. Scenario description. We use scenario A of the benchmark [35]; see Figure 18. The bathymetry is given—from left to right—by an area of a constant depth of 0.218 with three following, nonconstant bathymetry segments, with a slope of $\frac{1}{53}$, $\frac{1}{150}$, and $\frac{1}{13}$, respectively.

Using the one-dimensional benchmark in a two-dimensional setting with a triangulated grid, we set the boundaries on the scenario's sides (top and bottom sides in Figure 19) to *periodic conditions*. Hence, this results in an infinite wide scenario. We use *reflective boundary conditions* on the right side of Figure 18. For the *input boundary condition* on the left side with the wave moving in, we use the boundary conditions of the analytical solution precomputed by the GeoClaw group.[10] For the simulations, we used the finite-volume augmented Riemann solvers [13] from the GeoClaw package.

Using user-specified data for each cluster (property (4)), we can set the different boundary conditions without testing for vertex coordinates. This relies on the property that each cluster is associated with at most one type of boundary condition which is fulfilled in the base triangulation, as shown in Figure 19.

To support RLE-cluster-based data exchange for base triangulations such as given in the left-hand image in Figure 4, we have to use four triangles to assemble a quadrilateral. We initialize the domain with a strip of 128 quadrilaterals, resulting in 2^9

[10] https://github.com/rjleveque/nthmp-benchmark-problems/

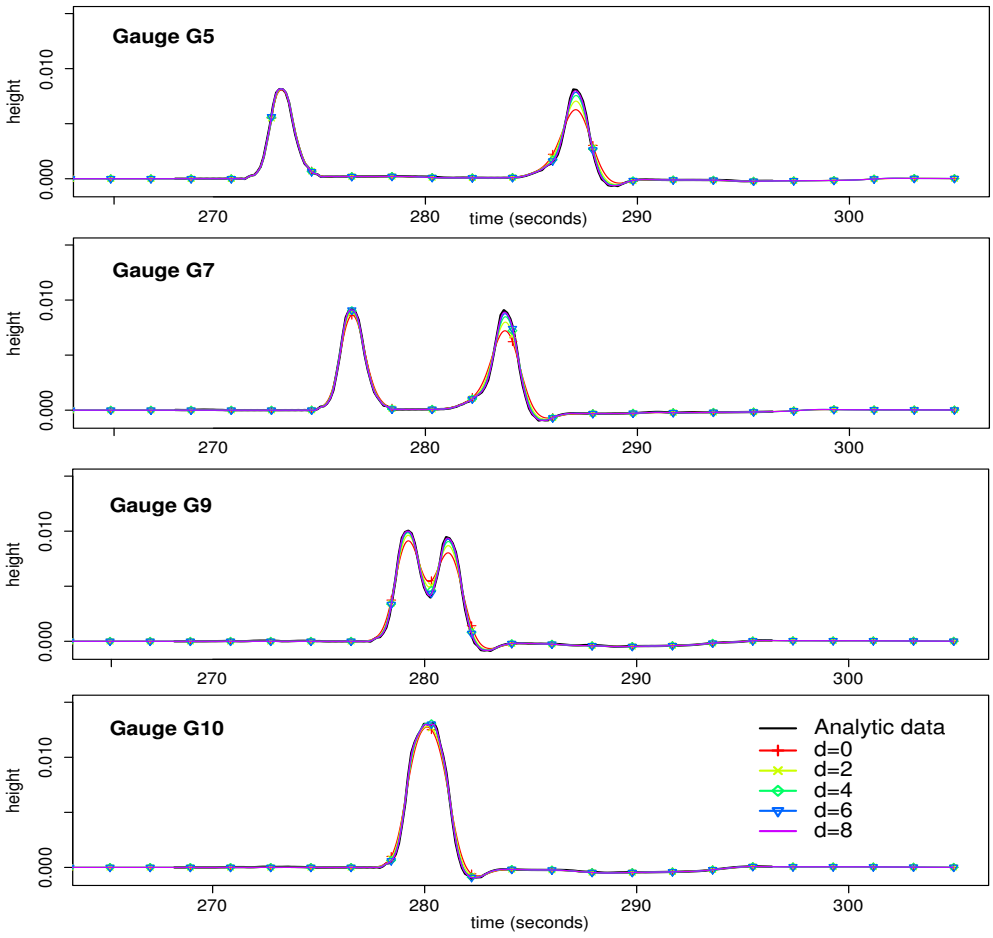

FIG. 20. *Analytic and computed solution for the surface elevation at water gauge stations G5, G7, G9, and G10. The numerically computed solution converges to the analytical solution with increased resolution* (d). *(Colored figures available in electronic version.)*

initial triangles for refinement depth 0. With an initial refinement depth d, the domain is initialized with $2^{(9+d)}$ triangle cells.

4.3.3. Benchmark runs with regular grids resolution. We first analyze the approximation behavior with different regular refinement depths without the dynamic adaptivity enabled. The plots for water surface height at selected water gauge stations are given in Figure 20, showing the convergence to the analytical solution. The averaged convergence rates are given by {G5: 1.6332, G6: 1.6376, G7: 1.7079, G8: 1.6294, G9: 1.6155, G10: 1.5012}.

A more detailed analysis of the convergence error based on the L1 norm with the analytic solution is given in Figure 21 for all water gauge stations, computed over the time interval $[270, 295]$.

4.3.4. Benchmarks runs with dynamic adaptivity. For testing the possibilities for dynamic adaptivity, we select water gauge G8, which is neither too close to nor too far from the reflective boundary. We executed several parameter stud-

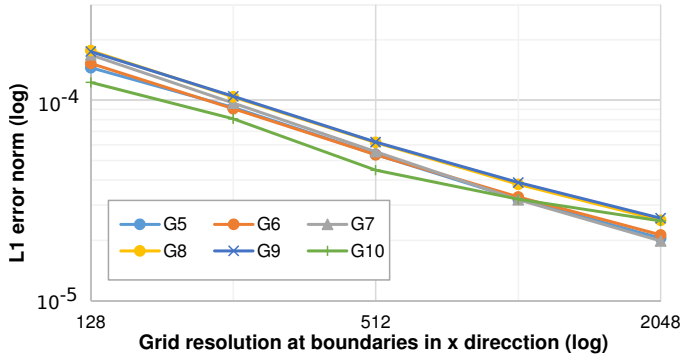

FIG. 21. *Absolute error (L1 norm) of the computed solution for the surface elevation at different water gauge stations. The resolution is given for the edges on one of the long quad strip boundaries. Both axes are in log scaling. (Colored figures available in electronic version.)*

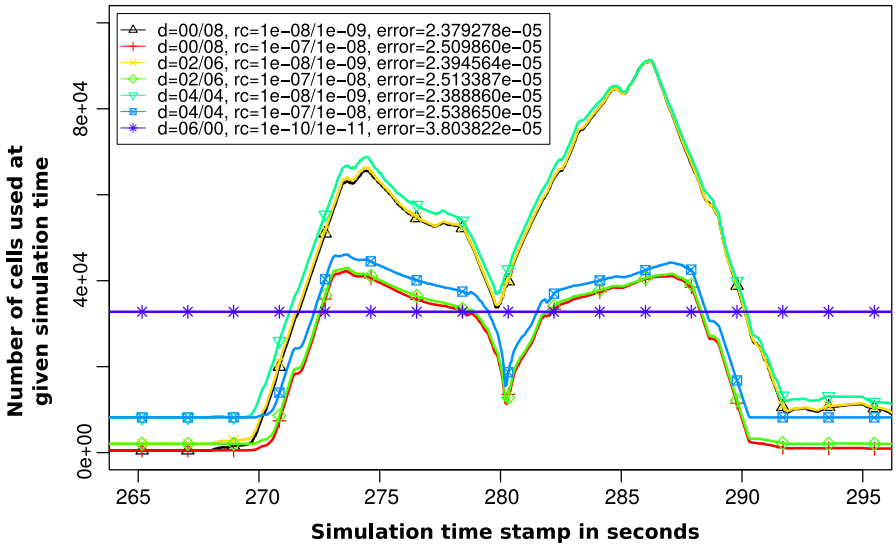

FIG. 22. *Cell distributions over time for the "Solitary Wave on Composite Beach" benchmark scenario. The elements in the legend are encoded with "d=[initial refinement depth]/[dynamic adaptive refinement levels], rc=[refine threshold]/[coarsen threshold], error=[L1 error to baseline]." For all adaptive scenarios, an increase of the number of cells is visible for the wave propagating into the simulation domain at* $t \in [265; 272]$. *In the phases before and after hitting the boundary on the right-hand side (*$t \approx 280$*), the different choices of the adaptivity criteria result in very different reduction behavior in terms of the number of cells. (Colored figures available in electronic version.)*

ies with initial refinement depths $d \in \{0, 2, 4, 6, 8\}$ and additional adaptive levels $a \in \{0, 2, 4, 6, 8\}$ and with the side constraint $d + a \leq 8$; i.e., we do not allow any spacetree depths exceeding 8.

A good justification for dynamic adaptivity is to show equivalent accuracy results with fewer cells involved in the computation. Following this idea, we executed the benchmark on a regular grid with $d = 6$, yielding $2^{9+6} = 32768$ grid cells, computed the L1 error of $3.80e{-}5$, and used this as a *baseline* for the comparison with simulations on dynamically adaptive grids.

We execute several parameter studies for $t \in [250; 305]$ and compare the results

TABLE 1

Parameter studies for analytical solitary wave on composite beach benchmark: Column 3 holds the average number of cells used in a time step, column 4 shows the necessary number of time steps (due to the CFL condition), and the rightmost column lists the percentage of saved cells per time step on average compared to the baseline.

Benchmark parameter	Error	Avg. cells	Time steps	Saved cells per time step
d=0/8, rc=$10^{-8}/10^{-9}$	2.38e-5	23556.13	38839	28.1%
d=0/8, rc=$10^{-7}/10^{-8}$	**2.51e-5**	**11795.13**	**30009**	**64.0%**
d=2/6, rc=$10^{-8}/10^{-9}$	2.39e-5	24326.79	40303	25.8%
d=2/6, rc=$10^{-7}/10^{-8}$	2.51e-5	12924.97	31354	60.6%
d=4/4, rc=$10^{-8}/10^{-9}$	2.39e-5	28169.67	43500	14.0%
d=4/4, rc=$10^{-7}/10^{-8}$	2.54e-5	18075.68	36732	**44.8%**
d=6/0 (baseline)	**3.80e-5**	**32768.00**	**37553**	**0.0%**

using the steady-state-based error indicators (see section 4.3.1). The refinement adaptivity parameter was chosen as $r := 10^{-n}$ with $n \in \mathbb{N}$ and the coarsening parameter was chosen in a very conservative manner with $c := \frac{r}{10}$, and we only show improved results with the L1 error norm. Figure 22 shows significant changes in the number of cells over time, where

$$d = [\textit{initial refinement depth}]/[\textit{dynamic adaptive refinement levels}]$$

denotes the initial and adaptive refinement levels. For a decent choice in the adaptivity criteria, we observe the desired behavior with a reduction of the number of cells for the shrinking support of the wave in the domain. For selected runs (with the same or a slightly reduced L1 error), Table 1 shows the percentage of saved cells per time step in the last column. For the "d=0/8, rc=$10^{-7}/10^{-8}$" setting, this yields a small reduction of $\frac{2^{15}-11795.13}{2^{15}} \approx 64\%$ of the cells used on average per time step and a small reduction of the error. Concerning the total number of cells involved in the entire computation over time, the required number of cells is reduced by $\frac{37553 \cdot 2^{15} - 30009 \cdot 11795.13}{37553 \cdot 2^{15}} \approx 71.2\%$ for the considered parameters.

With a domain regularly resolved to refinement depth $d = 8$, the computed error is $2.52e$–5 after 75105 time steps. We compare this to the parameter study "d=0/8, rc=$10^{-7}/10^{-8}$" which yields the best result for the same order of magnitude. This leads to $\frac{75105 \cdot 2^{17} - 30009 \cdot 11795.13}{75105 \cdot 2^{17}} \approx 96.4\%$ fewer cells involved in the entire simulation run. These benefits have been obtained with a conservative choice of adaptivity parameters and relatively low grid resolutions. The benefits will be even higher for larger grid resolutions or more sophisticated adaptivity criteria.

4.4. A real-world test scenario: Tohoku tsunami. With the simulation of the Tohoku tsunami of March 11, 2011, a real-world test scenario is given to show the potential of the developed algorithms for dynamic adaptive mesh refinement. Here, we aim at showing the potential of parallel dynamically adaptive grids by significantly reducing the "time-to-solution," e.g., for buoy station parameter studies of tsunami simulations.

4.4.1. Bathymetry and multiresolution sampling. We start with indicating relevant details on pre- and online processing of bathymetry data and the initialization of the scenario. Due to dynamic adaptivity, sampling of bathymetry data has to be accomplished at run time, and transformations are required from Cartesian bathymetry grids to our triangular simulation grids. The bathymetry datasets

we used in this work are based on the global GebCo_08[11] dataset with its highest resolution requiring 2GB of memory. With an adaptive grid, different cell sizes and therefore different resolutions are required. To match the sampling rate of the simulation grid with the one used for bathymetry input, we use multiresolution bathymetry datasets. For our simulation running with triangular grids, we use a trilinear interpolation between multiresolution grid levels and in space. Using the GebCo dataset, the resulting memory requirement is considerably lower than the total memory available per shared-memory compute node on today's HPC systems. Therefore, we use a native loader for the input data, loading the entire bathymetry and displacement datasets into the memory, followed by multiresolution preprocessing.

For our Tohoku tsunami simulation, the GebCo dataset is preprocessed with the Generic Mapping Tools [39]; we map the bathymetry data given in longitude-latitude format to the area of interest (shown in Figure 23), conserving length with the origin placed at the displacement center.

4.4.2. Initialization. For the Tohoku tsunami, an earthquake resulted in displacements of the sea ground leading to a sudden change in the water surface height. We consider an earthquake-induced displacement model which assumes an instantaneous vertical displacement in the water surface. Hence, for the initial time step of the tsunami simulation, we require the information on the displacements describing this surface elevation.[12] We used the data provided by the UCSB[13] as input to the Okada model[32] to compute the displacements.
Our initialization of the simulation is based on an iterative loop:

(1) In each iteration, the conserved quantities (water surface height and momentum) are reset to the time $t = 0$. The bathymetry data is sampled from the multiresolution datasets, and both momentum components are set to zero.
(2) Then, a single time step is computed and the grid is refined based on error indicators (see section 4.3.1).
(3) If the grid structure changed, we continue at (1); otherwise we stop the initialization and start with the simulation.

4.4.3. Adaptivity parameters. Similar to the analytic benchmark in section 4.3, we conducted several simulation runs with different initial refinement parameters $d = \{10, 16, 22\}$ and up to $a = \{0, 6, 12\}$ additional refinement levels with $d + a \leq 22$. The refinement thresholds used by the error indicators are $r = \{50, 500, 5000, 50000\}$ and $c = \frac{r}{5}$ for the coarsening thresholds. Note that the absolute values for r and c differ considerably compared to the analytical benchmark due to the different scaling of the domain size for the Tohoku tsunami simulation.

4.4.4. Comparison with buoy station data. We first conducted studies by comparing the simulation data with the water-surface elevation measured at particular buoy stations. This elevation data is provided by NOAA.[14] The tidal waves are not modeled within our simulation but are included in the buoy station data. To remove this tidal-wave induced water-surface displacement, we use detide scripts (see [6]).

Figure 23 shows the simulation domain, the adaptive[15] grid, and the measured

[11] http://www.gebco.net/

[12] See the instructions provided within the Clawpack package http://depts.washington.edu/clawpack/users/quick_tsunami.html.

[13] http://www.geol.ucsb.edu/faculty/ji/big_earthquakes/2011/03/0311_v3/Honshu.html

[14] National Oceanic and Atmospheric Administration, National Data Buoy Center.

[15] Note that this visualization of the simulation grid and the water and bathymetry data is based on a simulation with adaptivity parameters chosen for a comprehensible visualization of the grid.

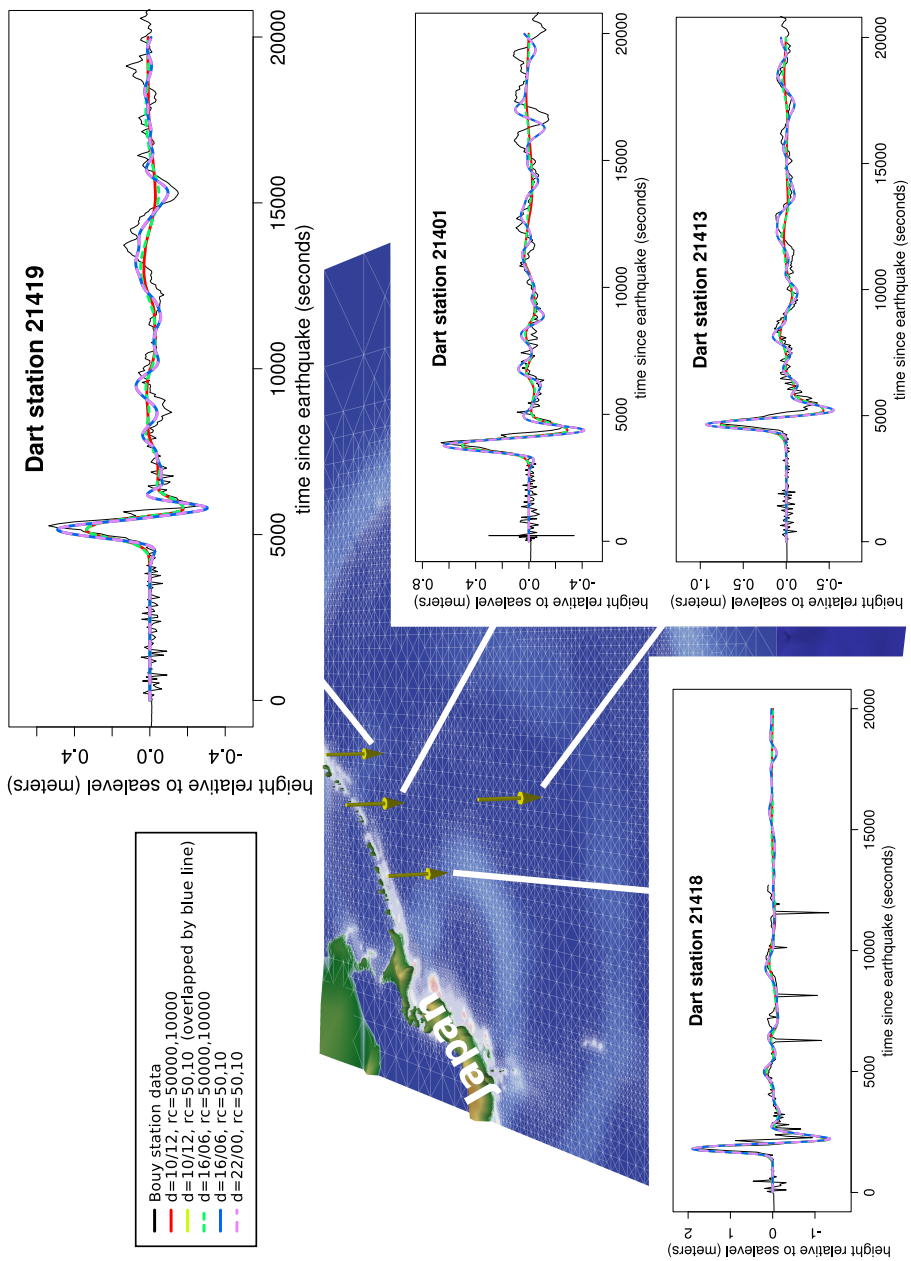

FIG. 23. *Tohoku tsunami simulation with different buoy stations marked with yellow arrows. The measured and simulated water surface displacements at the four relevant buoy stations are plotted for selected adaptivity parameters. (Colored figures available in electronic version.)*

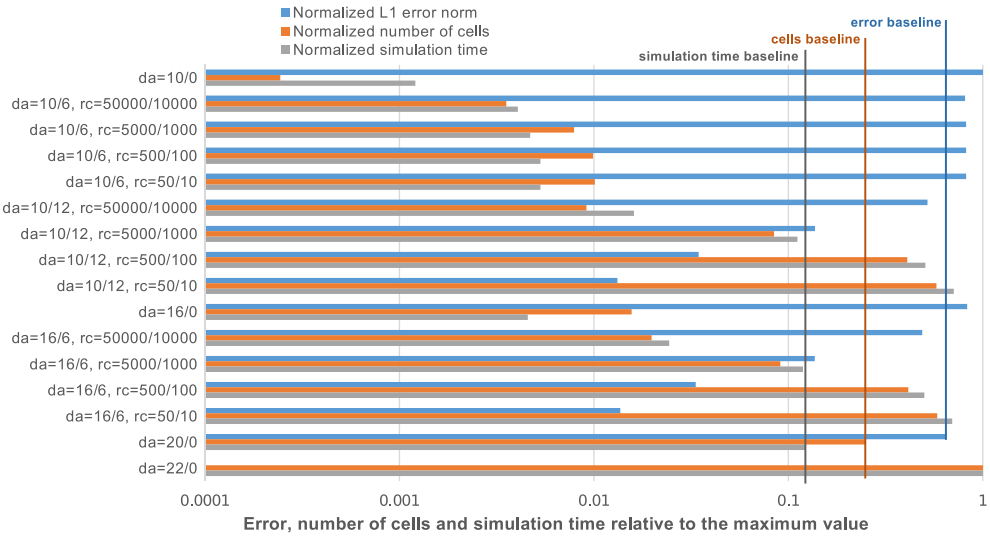

FIG. 24. *Barplot visualizing the normalized L1 error norm, the normalized number of cells, and the time to compute the simulation for different parameter combinations. The highest resolution ($d = 22/0$) is used as the reference result to compute the errors and to normalize the number of cells. The error of the coarsest resolution ($d = 10/0$) is used to normalize the yellow error bars. The number of cells shows a correlation to the computation time. The vertical lines represent the error baseline (blue) with its number-of-cells baseline (orange) involved in the simulation time (gray) for the benchmark $d = 20/0$. Competitive results require a decreased error, a decreased number of cells, and decreased computation time bars compared to our reference solution ($d = 20/0$); this holds for the four parameter settings $da = 16/6r = 5000/1000$, $da = 16/6r = 50000/10000$, $da = 10/12r = 5000/1000$, $da = 10/12r = 50000/10000$. (Colored figures available in electronic version.)*

displacements of the buoy stations. Regarding the wave arrival times at the buoy station, the results show a good agreement with the data recorded by the buoy stations.

4.4.5. Dynamically adaptive tsunami simulations. We executed tsunami simulations with the adaptivity parameters described in the previous section. To reduce the amount of data involved in our data analysis, we select buoy station 21419, with the peak of the first wave front arriving at the latest point in time compared to the other three considered buoy stations. We consider this to be the most meaningful and challenging buoy station, requiring longer simulation runtime compared to the other stations.

We compute the $L1$ error comparing different simulation parameters with a simulation running on a high-resolution grid on the time interval $[0s, 20000s]$ for several adaptivity parameters. Figure 24 shows corresponding bar plots of the errors and the normalized average number of cells relative to the maximum value. We use $da = 20/0$ (initial refinement depth of 20, no additional refinement levels) as the baseline for comparison with our dynamically adaptive grids. Comparing this baseline with the reference solution $da = 22/0$ shows (cf. Table 2) that about eight times more cells are involved in the computation. This is due to two additional refinement levels resulting in possibly four times more cells and to the reduced time-step size. All adaptive runs that resulted in a smaller error and fewer cells are highlighted in bold face in Table 2. Regarding the cell workload, we succeeded in saving more than 95% cells over a

TABLE 2

Different parameter studies and computed error relative to the reference solution da = 22/0 and saved cells relative to the baseline da = 20/0. If the error as well as the number of cells is less than the baseline, the corresponding row is in bold face.

Parameter study	L1 error	Processed mio. cells	Saved cells
da=22/0	0	88936.02	-719.16%
da=20/0 (baseline)	**0.0261210**	**10856.96**	**0%**
da=18/0 r=50/10	0.0306024	1298.14	88.04%
da=16/6 r=50/10	0.0005550	51775.64	-376.89%
da=16/6 r=500/100	0.0013608	36650.05	-237.57%
da=16/6 r=5000/1000	**0.0055551**	**7342.58**	**32.37%**
da=16/6 r=50000/10000	**0.0198121**	**1151.38**	**89.40%**
da=16/0	0.0337187	150.34	98.62%
da=10/12 r=50/10	0.0005356	51275.72	-372.28%
da=10/12 r=5000/1000	**0.0055124**	**6784.04**	**37.51%**
da=10/12 r=50000/10000	**0.0213234**	**452.25**	**95.83%**
da=10/6 r=500/100	0.0333127	95.11	99.12%
da=10/6 r=5000/1000	0.0333143	75.89	99.30%
da=10/6 r=50000/10000	0.0325013	22.86	99.79%
da=10/0	0.0406040	0.26	100.00%

TABLE 3

Timings for tsunami simulations separated into simulation, adaptivity, split/join stages, initialization, and output management phases. The overall wall clock time includes the entire wall clock time from start to end of the program including the additional output workload writing out data of buoy station displacements (see Figure 23). The column da = 10/12 describes the timings for the dynamic adaptive version, creating results which are competitive. The columns da = 20/0 and da = 22/0 show regular resolved domains with refinement depth 20 and 22, respectively. For the regular resolved domains, the adaptivity traversals were executed without changing the grid structure.

	Simulation parameters		
	da=10/12 r=50000/10000	**da=20/0**	**da=22/0**
Simulation traversals	20.88 sec	288.98 sec	2602.69 sec
Adaptivity traversals	7.35 sec	–	–
Split/join operations	3.10 sec	–	–
Approx. initialization time	2.68 sec	4.03 sec	5.79 sec
Output management	8.81 sec	32.77 sec	54.89 sec
Simulation wall clock time	**42.81 sec**	**325.78 sec**	**2663.37 sec**

simulation run.

Next, we continue with a detailed discussion of wall clock time results to determine the possible savings in time-to-solution. These benchmarks were conducted on an Intel Westmere-EX Xeon E7-4830 shared-memory system, with four sockets and eight cores per socket. We used compact affinities and used physical cores only with the icpc compiler ver. 15.0.4 and OMP parallelization. No further cluster-based optimizations such as cluster-based workstealing to compensate for different varying runtimes of Riemann solvers were used. We assigned each cluster to an OMP thread. A cluster split and join threshold of $T_s := 4096$ and $T_j := 2048$ is used.

We measured the wall clock time without the initialization phase and focus on the algorithmic parts of the RLE-cluster generation (see section 3). Table 3 gives an overview of the time required for running the simulation, the adaptivity traversals, and the cluster split/join phases. The last row shows the wall clock time without the initialization phase but includes all measurement grid traversals necessary to generate the buoy station plots (see Figure 23).

These timings show that the adaptivity traversals and the time spent in the split/join operations for the cluster generation exceed the time invested for the simulation traversals. However, this relative overhead pays off due to the reduced overall computation time compared to the regular grid resolution.

(a) $da = 20/0$: We start by comparing the simulation-traversal time for running the wave propagation on the regularly resolved domain with the entire simulation time (time stepping, adaptivity, split/joins) required by our dynamically adaptive simulation $da = 10/12, r = 50000/10000$. This yields an improvement of $\frac{325.78}{42.81} = 7.6$.

We next analyze the theoretical maximum performance improvements based on the average number of cells per time step: for the simulation on a dynamically adaptive grid, on average only 69070 cells per time step are used. This is a significant reduction to the simulation executed on a regular grid which required 2097152 cells, yielding a factor of 30.4. The reasons for the observed lower speedup include the following three issues:

- The size of the dynamically adaptive grid with only 69070 cells on average per time step is very small, resulting in a relatively low workload per core; hence there are more threading overheads.
- Managing the dynamically changing grid, e.g., to bisect triangles, results in additional overheads.
- The number of required time steps is increased from 5177 for the $da = 20/0$ simulation to 6543 for the simulation on the dynamically adaptive grid because of smaller grid cells and the smaller time steps due to the cell-size related CFL condition.

(b) $da = 22/0$: For this comparison, we assume sufficiently accurate simulation data on the dynamically adaptive grids. This allows a comparison with the same maximal refinement depth for both simulation runs and hence avoids the aforementioned issue with smaller time steps. Comparing the runtime of 42.81 seconds for the dynamically adaptive simulation $da = 10/12, r = 50000/10000$ with the simulation runtime of 2663.37 seconds for a regularly resolved domain $da = 22/0$, we see a performance improvement of a factor of $\frac{2663.37}{42.81} = 62.2$.

4.4.6. Discussion and related work. The main motivation of this tsunami study was to show the potential of dynamically adaptive grids with refinement and coarsening in every time step. The solvers used for these studies are the augmented Riemann solvers from the GeoClaw software [6], which support wetting and drying. These solvers were tested in various benchmark scenarios by their developers (see [14] for a variety of different benchmarks).

Regarding the adaptivity criteria, the water surface height close to the shoreline tends toward zero, and our error indicator might not lead to refining the grid in this area. However, the development of advanced adaptivity criteria (e.g., by dividing with the average depth of each cell) for flooding and drying goes beyond this work, which is on evaluating the RLE clustering concept in various contexts.

The GeoClaw software also offers, in combination with AMRClaw [7], simulations of hyperbolic problems with adaptive mesh refinement, and we close this section with a brief discussion of this software. One obvious difference is the utilization of Cartesian grids instead of triangular grids. We see two main advantages in this: The first one is a slightly larger CFL condition (based on incircle radius) with a similar resolution compared to triangular grids. The second one is dimensional splitting which is directly offered by using Cartesian grids. The finest atomic geometry in GeoClaw is a patch

which is stored in each cell. This concept also allows a straightforward utilization of GPUs due to the regular data structure of the patch. Such a patch concept can also be applied to the dynamic adaptive mesh refinement with triangular grids. The RLE clustering in this work relies on conforming grids whose generation obviously involves overheads. Hanging nodes are mandatory in GeoClaw due to the Cartesian grids; also, allowing hanging nodes in the RLE clustering could reduce these overheads.

5. Conclusions and outlook. We gave a summary on the different aspects of the stack-RLE clustering approach and presented, evaluated, and discussed, for the first time, different application scenarios in detail showing the suitability of what had been only a concept before. The scalability benchmark shows a strong scalability of 95% on 512 cores on a small-scale cluster for a shallow-water simulation on dynamically changing grids. On the large-scale system SuperMUC, a cluster generation based on subtrees and a threshold-based cluster generation did not lead to high scalability for strong scalability runs. However, the weak scalability is still over 90% on 8192 cores with the baseline at 256 cores. To gain insight into simulations, efficient visualization backends are required to improve the performance over the entire simulation runtime. In this work, we presented how TBB's fire-and-forget tasks can lead to significantly improved performance for writing simulation data to persistent memory and how to use the node-based communication for an efficient OpenGL visualization.

We used the Solitary Wave On Composite Beach (SWOCB) analytic benchmark to first show the benefits of dynamically adaptive grids. For the considered gauges, this resulted in a savings of up to 96.4% cells compared to regularly resolved domains with the error in the L1 norm still of the same order of magnitude. As a real-world test scenario we selected the simulation of the Tohoku tsunami and used buoy station data to measure the feasibility of tracking the first wave front propagation. This simulation was executed on a shared-memory 32-core system in sub–real time. Here, we presented the applicability of our RLE clustering for such simulations which led to a savings of 95% cells and a reduction in time-to-solution of 7.6 in combination with an accuracy improvement regarding the buoy station elevations. Regarding the tsunami simulations, we focused on showing beneficial advantages of dynamical adaptive meshes with our RLE clustering approach by comparing the quality of buoy station elevations of a tsunami wave propagations. In this work we mainly focused on the accuracy of the simulation with dynamic adaptive meshes at the buoy stations by using existing solvers. Requirements such as flooding/drying would require further investigation into, e.g., solver development, adaptivity criteria, etc.

The algorithms from this work can be implemented in several variants. Since the results are only reproducible with the source code being made publicly available to accompany publication, this code can be found online (see [29]).

We expect that an extension to SFC cuts by using intervals of the one-dimensional SFC representation for partitioning results in significant improvements for strong scalability. We see a new range of cluster-based optimizations with our RLE clustering: workstealing via dynamic cluster reordering, cluster-based local time stepping in combination with dynamic adaptive mesh refinement, compensation of unpredictable workload per cell, skipping of cluster traversals for cluster-local residual corrections for iterative solvers, and field-of-volume skipping for visualization. We would like to point out the possibility of backporting and extending the presented RLE clustering algorithms to Cartesian grids (e.g., generated by the Morton and Peano curve), including a generalization to high-dimensional problems. Furthermore, our stack-RLE clustering approach is compatible with SIMD optimization possibilities based

on multilayers, higher-order elements, (mini)patches, ensemble runs, etc. Such SIMD optimizations could also make the utilization of GPUs efficient due to the increased and well-structured workload per grid primitive. Making the entire RLE clustering algorithm run on GPUs might be feasible by programming a thread group to act as a single CPU thread in RLE clustering: All stack-operations have to be identical across all threads in the group and also operate on the same stack to act as a single CPU thread. However, this then allows data operations (storing/loading of coalesced memory) and running computations (exploitation of SIMT) being executed parallel by all threads of the thread group. In order to exploit platforms such as the XeonPhi, simultaneous support for MPI and OMP is required and is now available for dynamically adaptive grids via our stack-RLE clustering approach. Such an RLE-cluster-based parallelization concept could be one of the possible algorithmic patterns which allow running simulations on dynamically adaptive grids on upcoming Exascale systems.

Acknowledgments. The authors gratefully acknowledge the CooLMUC2 Cluster and the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for supporting this project by providing computing time, respectively, on the CoolMUC Cluster and the GCS Supercomputer SuperMUC at Leibniz Supercomputing Centre (LRZ) (www.lrz.de). We are very grateful to Alexander Breuer from our department for his contributions regarding the implementation of the analytical benchmark and the tsunami simulations and for his continuous and very constructive feedback on this work. Furthermore, we thank the GeoClaw group for publishing their tsunami-related scripts and solvers as open source as well as the authors of the other freely available scripts. We also thank the anonymous reviewers for their feedback and in particular for requesting more detailed performance studies which greatly improved this work.

REFERENCES

[1] M. Bader, *Space-Filling Curves: An Introduction with Applications in Scientific Computing*, Texts Comput. Sci. Engrg. 9, Springer, Heidelberg, 2013.

[2] M. Bader, A. Breuer, O. Meister, K. Rahnema, and M. Schreiber, *Parallelization and Software Concepts for Tsunami Simulation on Dynamically Adaptive Triangular Grids*, http://www-old.newton.ac.uk/programmes/AMM/seminars/2012082414301.pdf, 2012; accessed 2015-06-01.

[3] M. Bader, K. Rahnema, and C. Attila Vigh, *Memory-efficient Sierpinski-order traversals on dynamically adaptive, recursively structured triangular grids*, in PARA '10: Proceedings of the 10th International Conference on Applied Parallel and Scientific Computing, Vol. 2, K. Jonasson, ed., Lecture Notes in Comput. Sci. 7134, Springer, Berlin, 2012, pp. 302–311.

[4] M. Bader, S. Schraufstetter, C. A. Vigh, and J. Behrens, *Memory efficient adaptive mesh generation and implementation of multigrid algorithms using Sierpinski curves*, Internat. J. Comput. Sci. Engrg., 4 (2008), pp. 12–21.

[5] J. Behrens and J. Zimmermann, *Parallelizing an Unstructured Grid Generator with a Space-Filling Curve Approach*, in Euro-Par 2000 Parallel Processing, Springer, New York, 2000, pp. 815–823.

[6] M. J. Berger, D. L. George, R. J. LeVeque, and K. T. Mandli, *The GeoClaw software for depth-averaged flows with adaptive refinement*, Adv. Water Res., 34 (2011), pp. 1195–1206.

[7] M. J. Berger and R. J. LeVeque, *Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems*, SIAM J. Numer. Anal., 35 (1998), pp. 2298–2316, https://doi.org/10.1137/S0036142997315974.

[8] C. Burstedde, O. Ghattas, M. Gurnis, G. Stadler, E. Tan, T. Tu, L. C. Wilcox, and S. Zhong, *Scalable adaptive mantle convection simulation on petascale supercomputers*, in Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, IEEE Press, Piscataway, NJ, 2008, 62.

[9] C. Burstedde, L. C. Wilcox, and O. Ghattas, *p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees*, SIAM J. Sci. Comput., 33 (2011), pp. 1103–1133, https://doi.org/10.1137/100791634.

[10] C. E. Castro, M. Käser, and E. F. Toro, *Space-time adaptive numerical methods for geophysical applications*, Philos. Trans. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci., 367 (2009), pp. 4613–4631.

[11] P. Fischer, J. Lottes, D. Pointer, and A. Siegel, *Petascale algorithms for reactor hydrodynamics*, J. Phys. Conf. Ser., 125 (2008), 012076.

[12] A. Frank, *Organisationsprinzipien zur Integration von Geometrischer Modellierung, Numerischer Simulation und Visualisierung*, dissertation, Institut für Informatik, Technische Universität München, München, Germany, 2000.

[13] D. L. George, *Augmented Riemann solvers for the shallow water equations over variable topography with steady states and inundation*, J. Comput. Phys., 227 (2008), pp. 3089–3113.

[14] F. I. González, R. J. LeVeque, P. Chamberlain, B. Hirai, J. Varkovitzky, and D. L. George, *Validation of the GeoClaw model*, GeoClaw Tsunami Modeling Group, University of Washington, Seattle, WA, 2011.

[15] F. Günther, M. Mehl, M. Pögl, and C. Zenger, *A cache-aware algorithm for PDEs on hierarchical data structures based on space-filling curves*, SIAM J. Sci. Comput., 28 (2006), pp. 1634–1650, https://doi.org/10.1137/040604078.

[16] D. F. Harlacher, H. Klimach, S. Roller, C. Siebert, and F. Wolf, *Dynamic load balancing for unstructured meshes on space-filling curves*, in Proceedings of the 26th International Parallel and Distributed Processing Symposium Workshops & Ph.D. Forum (IPDPSW), IEEE, Washington, DC, 2012, pp. 1661–1669.

[17] G. Jin and J. Mellor-Crummey, *SFCGEN: A framework for efficient generation of multi-dimensional space-filling curves by recursion*, ACM Trans. Math. Softw., 31 (2005), pp. 120–148.

[18] S. Jin, R. R. Lewis, and D. West, *A comparison of algorithms for vertex normal computation*, Visual Computer, 21 (2005), pp. 71–82.

[19] W. B. March, K. Czechowski, M. Dukhan, T. Benson, D. Lee, A. J. Connolly, R. Vuduc, E. Chow, and A. G. Gray, *Optimizing the computation of n-point correlations on large-scale astronomical data*, in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, IEEE Computer Society, Washington, DC, 2012, 13372324.

[20] A. Marochko and A. Kukanov, *Composable parallelism foundations in the intel threading building blocks task scheduler*, in Proceedings of ParCo 2011, Ghent, Belgium, 2011, pp. 545–554.

[21] O. Meister, K. Rahnema, and M. Bader, *A software concept for cache-efficient simulation on dynamically adaptive structured triangular grids*, in Applications, Tools and Techniques on the Road to Exascale Computing (ParCo 2012, Ghent, Belgium), Adv. Parallel Comput. 22, IOS Press, Amsterdam, 2012, pp. 251–260.

[22] R.-P. Mundani, *Hierarchische Geometriemodelle zur Einbettung verteilter Simulationsaufgaben*, Dissertation, Faculty of Informatics, Elektrotechnik und Informationstechnik, University of Stuttgart, Stuttgart, Germany, Shaker, Aachen, 2006.

[23] T. Neckel, *The PDE Framework Peano: An Environment for Efficient Flow Simulations*, dissertation, Institut für Informatik, Technische Universität München, München, Germany, 2009.

[24] A. Rahimian, I. Lashuk, S. Veerapaneni, A. Chandramowlishwaran, D. Malhotra, L. Moon, R. Sampath, A. Shringarpure, J. Vetter, R. Vuduc, D. Zorin, and G. Biros, *Petascale direct numerical simulation of blood flow on* 200*k cores and heterogeneous architectures*, in Proceedings of the 2010 ACM/IEEE International Conference for HPC, Networking, Storage and Analysis, SC '10, IEEE Computer Society, Washington, DC, 2010, pp. 1–11.

[25] H. Sagan, *Space-Filling Curves*, Universitext, Springer-Verlag, New York, 1994.

[26] M. Schreiber, *Cluster-Based Parallelization of Simulations on Dynamically Adaptive Grids and Dynamic Resource Management*, dissertation, Institut für Informatik, Technische Universität München, München, Germany, 2014.

[27] M. Schreiber and H.-J. Bungartz, *Cluster-based communication and load balancing for simulations on dynamically adaptive grids*, in Proceedings of the International Conference on Computational Science (ICCS'14), Procedia Comput. Sci. 29, Elsevier, New York, 2014, pp. 2241–2253.

[28] M. Schreiber, H.-J. Bungartz, and M. Bader, *Shared memory parallelization of fully-adaptive simulations using a dynamic tree-split and -join approach*, in the 19th IEEE International Conference on High Performance Computing (HiPC) (Puna, India), IEEE, Washington, DC, 2012, 13485445 .

[29] M. Schreiber et al., *Website and Repository with Source Code Related to this Work*, http://www5.in.tum.de/sierpinski/, http://www.martin-schreiber.info/sierpinski/ (mirror), https://bitbucket.org/schreiberx/sierpi/.

[30] M. Schreiber, T. Weinzierl, and H.-J. Bungartz, *Cluster optimization and parallelization of simulations with dynamically adaptive grids*, in Euro-Par 2013 Parallel Processing, Springer, New York, 2013, pp. 484–496.

[31] M. Schreiber, T. Weinzierl, and H.-J. Bungartz, *SFC-based communication metadata encoding for adaptive mesh*, in Parallel Computing: Accelerating Computational Science and Engineering, Adv. Parallel Comput. 25, IOS Press, Amsterdam, 2013, pp. 233–242.

[32] G. Shao, X. Li, C. Ji, and T. Maeda, *Focal mechanism and slip history of the* 2011 *mw* 9.1 *off the Pacific coast of Tohoku earthquake, constrained with teleseismic body and surface waves*, Earth Planets Space, 63 (2011), pp. 559–564.

[33] H. D. Simon, *Partitioning of unstructured problems for parallel processing*, Comput. Syst. Engrg., 2 (1991), pp. 135–148.

[34] C. E. Synolakis, E. N. Bernard, V. V. Titov, U. Kânoğlu, and F. I. González, *Standards, Criteria, and Procedures for NOAA Evaluation of Tsunami Numerical Models*, NOAA Tech. Memo. OAR PMEL-135, NOAA/Pacific Marine Environmental Laboratory, Seattle, WA, 2007.

[35] K. Utku and C. E. Synolakis, *Long wave runup on piecewise linear topographies*, J. Fluid Mech., 374 (1998), pp. 1–28.

[36] C. A. Vigh, *Parallel Simulation of the Shallow Water Equations on Structured Dynamically Adaptive Triangular Grids*, Ph.D. thesis, Institut für Informatik, Technische Universität München, München, Germany, 2012.

[37] T. Weinzierl, *A Framework for Parallel PDE Solvers on Multiscale Adaptive Cart. Grids*, dissertation, Institut für Informatik, Technische Universität München, München, Germany, 2009.

[38] T. Weinzierl and M. Mehl, *Peano—a traversal and storage scheme for octree-like adaptive Cartesian multiscale grids*, SIAM J. Sci. Comput., 33 (2011), pp. 2732–2760, https://doi.org/10.1137/100799071.

[39] P. Wessel and W. H. F. Smith, *Free software helps map and display data*, Eos Trans. Amer. Geophys. Union, 72 (1991), pp. 441–446.

[40] G. Zumbusch, *On the quality of space-filling curve induced partitions*, ZAMM Z. Angew. Math. Mech., 81 (2000), pp. 25–28.