Procedia Computer Science

Volume 29, 2014, Pages 2241–2253

ICCS 2014. 14th International Conference on Computational Science

Cluster-based communication and load balancing for simulations on dynamically adaptive grids

Martin Schreiber and Hans-Joachim Bungartz

Technische Universität München

martin.schreiber@in.tum.de, bungartz@in.tum.de

Abstract

The present paper introduces a new communication and load-balancing scheme based on a clustering of the grid which we use for the efficient parallelization of simulations on dynamically adaptive grids.

With a partitioning based on space-filling curves (SFCs), this yields several advantageous properties regarding the memory requirements and load balancing. However, for such an SFC-based partitioning, additional connectivity information has to be stored and updated for dynamically changing grids.

In this work, we present our approach to keep this connectivity information run-length encoded (RLE) only for the interfaces shared between partitions. Using special properties of the underlying grid traversal and used communication scheme, we update this connectivity information implicitly for dynamically changing grids and can represent the connectivity information as a sparse communication graph: graph nodes (partitions) represent bulks of connected grid cells and each graph edge (RLE connectivity information) a unique relation between adjacent partitions. This directly leads to an efficient shared-memory parallelization with graph nodes assigned to computing cores and an efficient en bloc data exchange via graph edges. We further refer to such a partitioning approach with RLE meta information as *a cluster-based domain decomposition* and to each partition as a *cluster*. With the sparse communication graph in mind, we then extend the connectivity information represented by the graph edges with MPI ranks, yielding an en bloc communication for distributed-memory systems and a hybrid parallelization. For data migration, the stack-based intra-cluster communication allows a very low memory footprint for data migration and the RLE leads to efficient updates of connectivity information.

Our benchmark is based on a shallow water simulation on a dynamically adaptive grid. We conducted performance studies for MPI-only and hybrid parallelizations, yielding an efficiency of over 90% on 256 cores. Furthermore, we demonstrate the applicability of cluster-based optimizations on distributed-memory systems.

Keywords: dynamically adaptive grids, space-filling curve, high-performance computing, run-length encoding, meta information, finite volume simulation

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2014

doi:10.1016/j.procs.2014.05.209

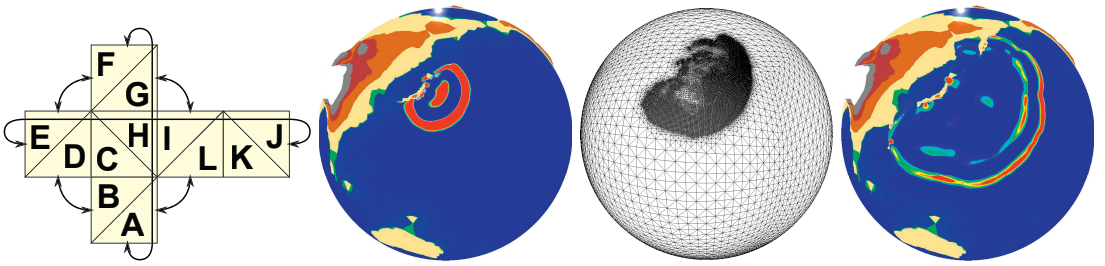

Figure 1: Left: Triangulated cubed sphere with the arrows annotating the connectivity information. Right: Visualization of the water surface and terrain with the colouring based on the water and terrain height and the underlying dynamically adaptive triangular grid of a finite-volume [13] simulation of the Tohoku Tsunami 2011 on a triangulated cubed sphere grid.

1 Introduction

Solvers for partial differential equations (PDE) which rely on a space-filling curve (SFC) decomposition of the underlying grid are very common in scientific computing [1,7,8,16,21,22,28]. These solvers recursively decompose the grid into typically non-overlapping primitives which are ordered along the SFC, hence resulting in a spacetree [12,20,21,21,30]. The leaves of such a spacetree then represent the grid cells and the traversal can then be either done recursively [28] or in a linearized way [7] traversing only the leaf nodes.

For a parallelization, a frequently chosen approach for distributed-memory systems is the partitioning of the simulation grid. Such a partitioning aims at optimizing the interfaces shared among partitions, e.g. by reducing the number of edge-cuts of the dual graph [26]. Using graph-based partitioners requires complex operations [5] and typically results in an NP-hard problem [32]. Since our simulations are wave-propagation dominated (see right image in Fig. 1), resulting in a frequently changing grid structure, these partitioners are not efficiently applicable. An alternative is given by SFCs inducing an ordering of the grid primitives. Then, an enumeration of them can be used for efficient load-balancing strategies for dynamically adaptive grids. Here, the one-dimensional grid representation is cut into partitions of a balanced amount of cells [5] and each partition also retains locality properties of the d-dimensional space [19]. Using such an SFC partitioning leads to faster load balancing for dynamically adaptive grids with partitions still close to optimality regarding the edge-cut of graph-based partitioners [18]. Therefore, we use an SFC-based partitioning approach with the one-dimensional intervals generated by tree splits which are naturally given by splitting the spacetree into sub trees.

Solving a PDE on such spacetree-induced grids requires efficient access of per-primitive stored simulation data as well as efficient exchange of data, e.g. via edges. Considering solvers for hyperbolic equations, a possible access pattern gathers data at edges which are shared by adjacent grid primitives. Such an access of edge-oriented data exchange should be designed in a way to reduce the amount of accessed memory under consideration of the memory hierarchy to tackle the increasing gap between computational power and memory bandwidth [6]. Different methods are researched in this context and are e.g. based on lists [7] or a stack- and stream-based approach [12, 21, 29] with the latter one used in this work.

In combination with a domain decomposition, connectivity information is required to exchange data via edges shared among adjacent partitions and we continue referring to this connectivity information as *meta information*. Such a meta information is either stored for

each grid primitive (e.g. using unstructured grids, resulting in a *dense connectivity graph*), or for each interface shared between partitions (see e.g. [17, 27] for edge communication, further denoted as a *semi-sparse connectivity graph*).

Furthermore, dynamically changing grids demand for a repartitioning and load balancing. Therefore, the meta information also has to be updated to assure a consistent connectivity. Methods studied so far for semi-sparse connectivity graphs are based on per-edge communicated adjacency information, and are efficiently applicable with MPI parallelization [17,27]. However, this update scheme of meta information still relies on the enumeration of all cells which is stored for each edge shared by partitions.

2 Contribution

In this work, we present a meta information management resulting in a *sparse connectivity graph* for a given SFC-based domain decomposition which supports the stack-based communication: Here, each partition is represented as a graph node and a consecutive number of shared interfaces among two partitions is represented with a unique run-length encoded (RLE) entry in the per-partition stored meta information.

After introducing the former work on stack- and stream-based simulations in Sec. 3.1, we present our cluster-based parallelization on distributed-memory systems:

(a) **Run-length encoded meta information and clustering (Section 3.2):**
We present an en bloc data exchange for efficient shared- and distributed-memory communication.

(b) **Updating meta information for dynamically adaptive grids (Section 3.3):**
With dynamically adaptive grids, our RLE meta information is *updated implicitly* without requiring the transfer of rank- or primitive-enumeration-related information.

(c) **Dynamic cluster generation (Section 3.4):**
The partition generation and the required synchronization of the meta information is further described in this Section.

(d) **Migration of clusters (Section 3.5):**
Our migration is based on sending the raw cluster data and pre- and post-processing of RLE meta information only. Storing the grid structure information only with two markers required for each leaf cell, we consider this to be a *highly efficient data migration of partitions* wrt. the amount of transferred data.

With a cluster-based domain decomposition and its RLE connectivity information, it is possible to efficiently keep *more than one partition* in a single program context with *replicated shared interfaces* (see [25]) of the partitions. This yields beneficial features compared to other developments based on a stack-based communication approach:

(a) Since operations on clusters can be executed individually, this allows application of cluster-based optimizations such as skipping of already conforming clusters (see [24]) also in a distributed-memory environment.

(b) Our RLE communication approach directly leads to a hybrid OpenMP and MPI parallelization with a threaded parallelization over the clusters.

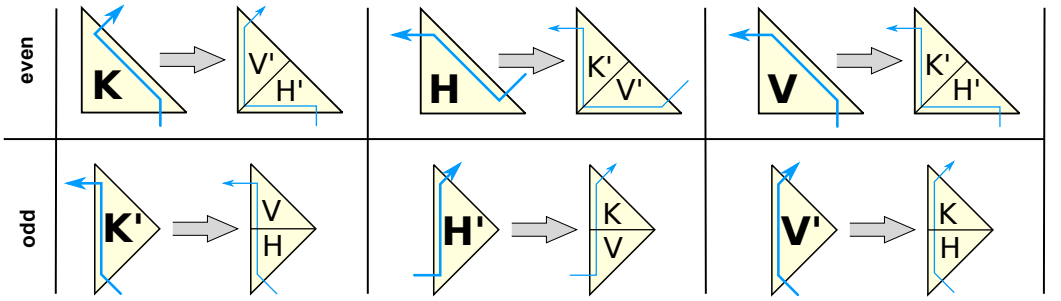

Figure 2: Grammar used in the spacetree for generation of the underlying simulation mesh. The thick gray arrow shows the derivative of the grammar rule to refine a cell in the refinement tree. Triangles without and with a prime are respectively of type *even* and *odd*.

We present the application of cluster-based optimizations on distributed-memory systems (see (a)) and furthermore discuss results for our hybrid parallelization method (see (b)) in Section 4 followed by an outlook of the necessity of such a hybrid parallelization in Section 6.

3 Cluster-based domain decomposition and load balancing

3.1 Stack- and stream-based simulations

We start with an introduction to the stack- and stream-based communication properties based on the bi-sectioning Sierpiński SFC, see [3]. The required basic triangle traversal types are given by $\mathbb{G} := \{K, V, H\}$ accounting for different possibly edges pierced by the SFC for entering and leaving each triangle, see Fig. 2. We further annotate each triangle with $\mathbb{O} := \{even, odd\}$ with the marker *odd* mirroring the traversal direction along the edge inserted with the newest vertex bisection. The tuple $\mathbb{T} := (\mathbb{G}, \mathbb{O})$ then defines all SFC cell traversal possibilities required for our communication scheme. A bisection of a triangular cell is then accomplished by deriving $\mathbb{T}$ for both children based on $\mathbb{T}$ of the bisected cell, see Fig. 2 for these rules.

For data exchange, we first distinguish edges to be either on the left or right side of the SFC (see [2, 23]). Here, the SFC representation in a cell is assumed to be closely drawn to the hypotenuse. The edges of the leaf cells are further annotated with communication types $\mathbb{C} := \{new, old, bc\}$ respectively for pushing data to the communication stack, fetching data from the communication stack and handling boundary conditions. These edge types are inherited to the child nodes during a bisection. Then data exchange can then be accomplished via the *left* and *right communication stacks*, with push/fetch operations on the respective communication stack, see [2, 3, 23].

3.2 Run-length encoded meta information and clustering

An SFC-based grid generation inherently leads to an ordering of all cells in each partition with cell C_{i+1} sharing exactly one edge with C_i. For each partition, we can set the edge types of the interfaces shared with other partitions to *new* for writing data to the communication stacks which are then exchanged with the adjacent partitions, or to *old* to process data by reading exchanged data from the communication stacks via fetch operations. Here, we use the same

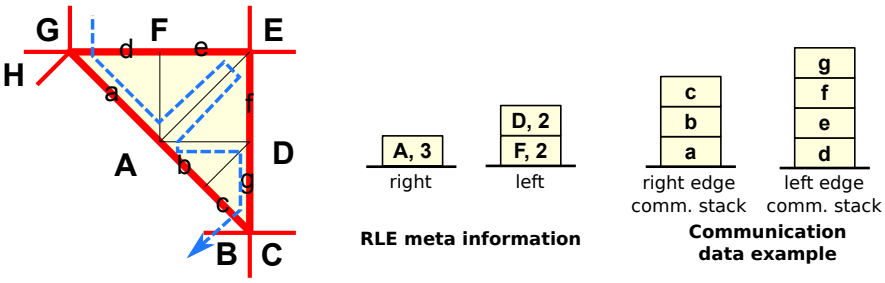

Figure 3: Exemplary RLE meta connectivity information for the partition highlighted with the thick red border. We give a description for the edges on the left side of the SFC. The edge data d and e is stored consecutively to the communication stack. Since this data has to be exchanged with cluster F, we use the RLE entry $(F, 2)$ to represent this communication interface. The next communication data f and g is shared with cluster D and we represent it with $(D, 2)$ in our meta information encoding.

stack for the inter- and intra-partition communication. Instead of storing the meta information for each edge shared among partitions [17, 27], we use a property induced by the required ordering of the communicated elements which are written to the communication stacks:

We formally annotate each communicated data with the index i of the cell C_i pushing the data to the stack. At any time during and after the grid traversal, these indices on the communication buffers are ordered. Two properties can then be inferred by using a stack-based communication approach. Without loss of generality, we only consider edge types of the interfaces shared with other partitions set to *new*, hence generating data to be forwarded to adjacent partitions.

(a) After the traversal, data associated to shared interfaces between adjacent partitions are stored *en bloc to the communication stack*.

(b) There is a *unique single consecutive block of data* for each adjacent partition.

Since all the communication data has to be ordered by the imaginary index due to stack-communication properties, also all the data associated to interfaces shared with an adjacent partition is stored consecutively to the communication buffer (property (a)). Property (b) is induced by reductio ad absurdum and the cell-indexed ordering of the elements on the communication stacks and the partitioning generated by SFC-intervals: two non-consecutive blocks of data associated to shared interfaces to two partitions would result in a violation of this cell-indexed ordering.

Based on this knowledge, we can then store the meta information for shared edges with a run-length encoding in a list of tuples for the left and right communication stack. Each tuple stores two values: (a) the information on the placement of the adjacent partition (pointer to the adjacent partition if its stored in the same memory, the MPI rank to know the rank for distributed-memory systems, etc.) and (b) the number of edges which are shared between both partitions which are not represented by the edge RLE meta information. An example for an edge communication and its corresponding RLE meta information is given in Fig. 3. This results in a sparse-graph representation of the meta communication information with an example given in Fig. 4. Since no meta information is required for intra-partition communication, the communication in each partition is represented by a node. Each RLE meta entry then represents an edge of the connectivity graph.

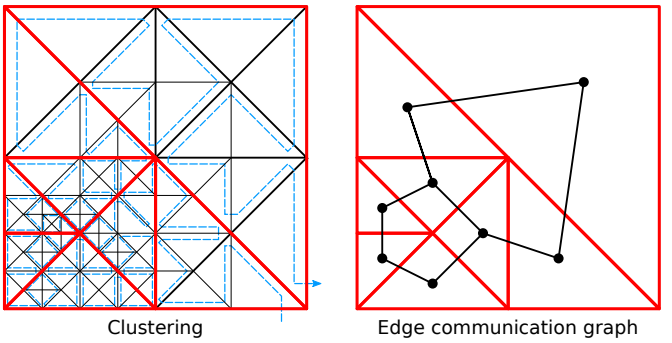

Figure 4: Left: partitions generated by spacetree split induced intervals of SFC. Right: sparse communication graph for partitions with meta information for edges shared by multiple partitions. Each red bordered area represents a partition and the lines the edge RLE communication information.

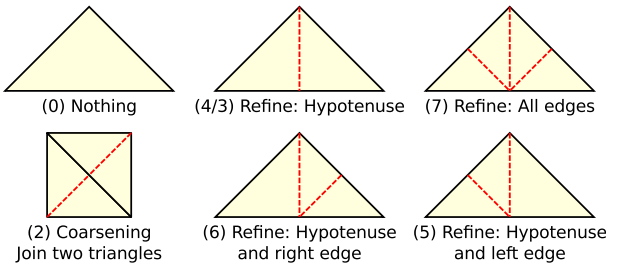

Figure 5: Different adaptivity states inserting or removing edges from triangle cells [23].

The information in each RLE then allows efficient data access on shared- and distributed-memory systems: On shared-memory systems, the communication data on the adjacent partition can be directly accessed via a pointer stored in the meta information. Then, the concrete location of the data on the adjacent communication stacks can be inferred by the corresponding adjacently stored RLE meta information (property (b)). On distributed-memory systems, the data has to be send and received with non-blocking commands to/from the rank specified in the communication meta information. The data transfer on shared- and distributed-memory can be done directly en bloc (property (a)).

We further keep all the data associated to a partition in a separately allocated memory area and allow multiple partitions in a memory context. In combination with our RLE meta information, this yields a software approach which is required for efficient cluster-based local-time stepping [9]. Therefore, we further refer to our method as a *cluster-based domain decomposition approach*.

With a forest of spacetrees [7], we can assemble different domains and also a grid for a cubed sphere (see left image in Fig. 1). Here, a quadrilateral is assembled by two triangles and six quadrilaterals are used for each cube face (see [4]). The meta information is then initialized for data exchange connecting the cube edges, as shown in the left image in Fig. 1.

3.3 Updating meta information for dynamically adaptive grids

Due to dynamically changing grids, edges can be removed or inserted to the grid, see Fig. 5 for all possible adaptivity states. Also our meta information has to be updated to account for these changes in grid structure. Our approach uses refinement and coarsening markers which are communicated via the stack-communication system: a refinement marker M_R is communicated via the edge if a refinement operation created a vertex on this edge (3)-(7). For coarsening operations joining two triangles, the coarsening markers M_C are communicated via the catheti of both coarsening triangles (2). By setting the edge types on the cluster boundary interfaces to *new*, the adaptivity markers forwarded via edges are finally stored on the left and right communication stacks. Hence, this stacks then contain the required information on *how to update the RLE meta information* to a consistent state: A single refinement marker M_R requires incrementing the corresponding RLE entry to account for an additional edge to be shared and two consecutive coarsening markers M_C require to decrement the corresponding RLE entry to account for removing an inter-cluster shared edge.

3.4 Dynamic cluster generation

With a dynamically changing grid, generated load imbalances have to be compensated by reclustering. This requires updating meta information to a consistent state.

Regarding the *split operations*, we again profit from our stack-based communication which allows inferring the new RLE meta communication information for the local cluster on-the-fly during a traversal, see [23]: With the clusters generated by tree splits, we stop during the traversal of the grid at particular points. Based on the number of elements stored on the stacks, we can then infer the new meta information on-the-fly without requiring rank- or global cell-enumeration-related information. *Join operations* can be accomplished by concatenation of the RLE meta information of both clusters and by removing the meta information associated to the interfaces formerly shared by the joined clusters.

However, these updates on the meta information only consider the clusters involved in the split/join operation. Possible split and join operations on adjacent clusters require additional synchronization. Here, we distinguish between adjacent clusters stored in the same memory context and adjacent clusters on another rank for distributed-memory systems. If the adjacent cluster is stored *in the same memory context*, we generate a consistent meta information by accessing the adjacent clusters read-only to infer the required information: In case of a split, the synchronization is accomplished based on the meta information stored at both recently generated clusters. If the adjacent cluster was generated by joining other clusters, the meta information is synchronized with the new cluster generated by the join operation. We again like to emphasize, that all operations on the adjacent cluster are executed read-only. For clusters *stored on a different rank*, the split or joined clusters send the synchronization information, which was on shared-memory systems inferred by the read-only access, to the adjacent cluster. The meta information on the cluster on the other rank can then be updated to a conforming state based on the received information.

For deciding whether a cluster has to be split or joined, we use a massive splitting approach based on thresholds [23] (see [24] for an alternative, range-based cluster generation). As soon as the number of cells in a cluster exceed the threshold T_s, the cluster is split. If two sibling clusters both undershoot a threshold T_j, both clusters are joined. In this work, we use a join threshold $T_j := \frac{T_s}{2}$. Such a cluster generation leads to an efficient parallelization with cluster-based optimizations on shared-memory systems [24].

3.5 Migration of clusters

We present the algorithmic pattern for a cluster-based data migration:

1. *Destination annotation:*
   Each cluster is annotated with the rank to which it has to be migrated to.
2. *Prospectively updating the communication meta information:*
   For each cluster, the destination rank is send to the adjacent clusters if the adjacent cluster is stored on a different rank. The corresponding meta information in the adjacent clusters is then updated to the new rank.
3. *Cluster migration:*
   All cluster-associated data (stacks, meta information, user-specific data) can then be migrated directly to the new MPI rank. We like to emphasize, that the amount of this data is quasi optimal due to our SFC-induced partitioning and memory-efficient storage of the pure simulation data on a compact stream in each cluster (see [23] for further information).
4. *Synchronization of the cluster-local RLE information*:
   For clusters received at a rank, the pointers in the RLE meta information on interfaces shared by the received clusters and adjacent clusters on the same rank are synchronized. For clusters send to another ranks, the adjacent clusters which are still on the same rank update their RLE information with the rank to which the cluster was send to.

This pattern also allows utilization of generic load-balancing interfaces [11] and hence implementation of alternative well-studied load-balancing schemes (e.g. [10, 15, 31]). The load-balancing strategy which we use for our results is based on the parallel prefix sum over the per-rank processed number of simulation cells. We use this information to send the clusters to the next or previous MPI rank for improving the load balance, see [14] for further information.

Storing the grid structure requires only 2 flags per leaf cell with each flag stored in a single byte (this can be also encoded in a bit stream), and we derive the additionally required meta information on cell vertices, edge normals and communication flags derived on-the-fly during the traversal. With the user-specified cell data (e.g. the simulation data) stored compactly on a stream, the data migration of a cluster is quasi optimal regarding the amount of transferred memory.

4 Results

The presented results are based on shallow-water simulations executed on dynamically changing grids. We show a scalability test for our cluster-based data migration on distributed-memory systems.

We conducted a strong-scalability benchmark on the MAC cluster system[1] on the Intel compute nodes. The simulation domain is initialized with a refinement depth of 22 and 8 additional refinement levels. We split the cluster in case that they exceed a threshold size of 4096. The simulation is initialized with a radial dam break and is executed for 100 iterations. In each iteration, we successively execute a time step of the simulation-adaptivity traversals (see Sec. 3.3) and the threshold-based cluster generation (see Sec. 3.4), followed by a cluster-based data migration (see Sec. 3.5). We tested different parallelization models with the results given in Fig. 6 and further described next.

[1] `http://www.mac.tum.de/wiki/index.php/MAC_Cluster`

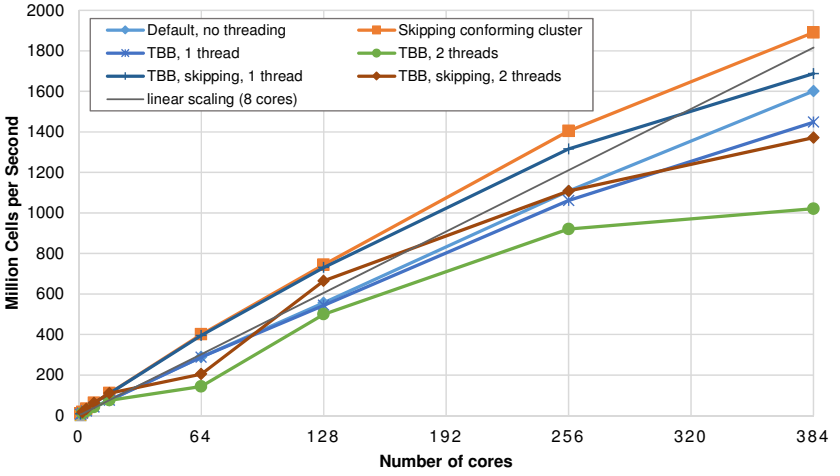

Figure 6: Strong scalability tests for different parallelization models and cluster-based optimizations, see text for further information. The x-axis depicts the overall used cores in the simulation and the y-axis the throughput in million triangle cells per second.

- *Default:*
  The multi-threading support for this benchmark is disabled. The efficiency is 91.5% for the execution on 256 cores and 88% on 384 cores with the baseline at 16 cores.

- *Skipping of conforming clusters:*
  With the requirement of a conforming grid, adaptivity traversals are executed on all clusters until a conforming grid state was generated, despite that some clusters already yield a conforming grid state. This method skips the conforming cluster traversals of this grids (see [24] for details). Such an algorithm leads to a robust performance improvement of 26.7% on 256 cores.

- *TBB, 1 thread:*
  This version activates the hybrid parallelization with Intel's Threading Building Blocks (TBB) and compiles the code with thread-enabled MPI parallelization. However, we only execute a single thread per MPI rank to measure the overheads involved by using tasking to run operations in parallel on each cluster. These overheads are 4.2% on 256 cores compared to our non-threaded default version.

  Activation of a cluster-based optimization via skipping of clusters with an already conforming state (TBB, skipping, 1 thread) achieves a damping of the overheads introduced by the conformity grid requirement and also leads to a performance improvement compared to the non-threaded version.

- *TBB, 2 threads:*
  Here, both threads execute the simulation-adaptivity traversals, but only a single one is used for exchanging the data with other MPI ranks to synchronize the meta information. On 384 cores, the serial processing of the MPI data exchange becomes more dominant, resulting in a decrease in performance. We account for this with the strong scalability benchmark: this leads to a longer time for the serial communication phase compared to the compute phase for an increasing number of cores. Similar to the single-threaded version,

the activation of the cluster-based optimizations also compensates these overheads with a similar performance compared to the non-threaded version for 256 cores.

With a hybrid parallelization, additional MPI data transfer is avoided for the shared interfaces of the clusters. However, this did not lead to performance improvements due to the introduction of additional overheads. We account for these overheads by additional tasks which are enqueued to worker queues as well as the non-parallel processing of the MPI data exchange and reduce operations on the shared interfaces.

With the skipping of conforming clusters, this again yielded a robust performance improvement for the threaded version in a hybrid parallelization environment. The *TBB, 2 threads* version with enabled cluster optimizations yielded a similar performance compared to the default, non-threaded version. Such a multi-threaded execution within one MPI rank allows further optimizations which is also discussed in the Section 6 as part of our ongoing work.

5 Summary

We introduced a cluster-based parallelization approach on distributed-memory systems with an underlying SFC-optimal sparse communication graph for a given partitioning. The benchmark shows an efficiency of 91.5% on 256 cores and 88% on 384 cores for a shallow-water simulation on a dynamically changing grids.

Our connectivity information is run-length encoded and is implicitly updated by transferred adaptivity markers. With the stack-based communication system, the data which is to be transferred to adjacent clusters is stored consecutively in the memory and can be transferred directly en bloc without preprocessing by using the RLE connectivity information. This leads to an en bloc data transfer for shared- and distributed-memory parallelization models. Furthermore, the clustering leads to a hybrid parallelization with a threaded parallelization over the clusters (e.g. executing a task for each cluster).

Our presented cluster-based data migration is efficient for two major reasons: First, the cluster-based data migration only requires migrating the *raw cluster data* without further pre- and post-processing for intra-cluster connectivity. Since the simulation data is stored compactly on a stream and with all vertex and normal information inferred during the spacetree traversal, hence not requiring to be migrated, the amount of migrated data is quasi optimal. Second, the connectivity information requires only pre- and post-processing of the RLE compressed connectivity information.

Whereas multiple clusters are required for the data migration approach, we further presented one possible optimization by keeping multiple clusters in a single program context: our clustering weakens the constraint of traversing the entire partition which is stored per MPI rank for stack-based communication approaches. By clustering the partition, this allows a reordering or skipping of the execution of operations on the clusters [24]. In this work, we presented the optimization of the traversal-skipping of already conforming clusters also in a distributed-memory environment and used such an optimization to compensate overheads introduced by the hybrid parallelization.

6 Outlook

We currently investigate simulation scenarios where a hybrid parallelization method is required: considering e.g. the upper limit of 16GB on each MPI node on the SuperMUC with 16 physical cores, runtime-persistent datasets have to be by far smaller than 2GB. However, e.g. the GebCo

bathymetry datasets[2] are larger than 2GB with their highest resolution. With a hybrid parallelization, we can store the bathymetry data e.g. for earth-scale Tsunami simulations (see Fig. 1) only once per MPI rank. Also the elimination of the overheads of the non-parallel processing of the MPI communication is further investigated by extending the RLE connectivity information with unique cluster or thread ids for parallel parallel processing of the distributed-memory data exchange with a hybrid parallelization.

We further envision cluster-based optimizations in combination with a hybrid-parallelization: E.g. cluster-based local-time stepping, local-residual-aware iterative solvers and improved performance for unpredictable workload via work-stealing on cluster level. Such a work stealing can get beneficial for dynamically loading boundary values during the adaptivity phase and for flux computations with variable computational intensity (see e.g. [13] used in Fig. 1). Furthermore, the upcoming accelerator cards such as the XeonPhi require a hybrid parallelization. Hence, an evaluation of a cluster-based hybrid parallelization on such architectures is also part of our future work.

For sake of reproducibility, the software is freely available as open source at `http://www5.in.tum.de/sierpinski/` and is tagged with version *2014_03_12_iccs*.

7 Acknowledgements

We like to thank the Munich Centre of Advanced Computing for for funding this project by providing computing time on the MAC Cluster. This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89).

References

[1] M. Bader, C. Böck, J. Schwaiger, and C. A. Vigh. Dynamically Adaptive Simulations with Minimal Memory Requirement - Solving the Shallow Water Equations Using Sierpinski Curves. *SISC*, 32(1), 2010.

[2] Michael Bader, Kaveh Rahnema, and Csaba Attila Vigh. Memory-efficient sierpinski-order traversals on dynamically adaptive, recursively structured triangular grids. In Kristjan Jonasson, editor, *Applied Parallel and Scientific Computing - 10th International Conference, PARA 2010*, volume 7134 of *Lecture Notes in Computer Science*, pages 302–311. Springer, March 2012.

[3] Michael Bader, Stefanie Schraufstetter, Csaba Vigh, and Jörn Behrens. Memory efficient adaptive mesh generation and implementation of multigrid algorithms using sierpinski curves. *International Journal of Computational Science and Engineering*, 4(1):12–21, 2008.

[4] Jörn Behrens. *Adaptive atmospheric modeling: key techniques in grid generation, data structures, and numerical operations with applications*. Springer, 2006.

[5] Jörn Behrens and Jens Zimmermann. Parallelizing an unstructured grid generator with a space-filling curve approach. In *Euro-Par 2000 Parallel Processing*, pages 815–823. Springer, 2000.

[6] S. Borkar and A. A. Chien. The future of microproc. *Commun. ACM*, 54, 2011.

[7] C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees. *SISC*, (3), 2011.

[8] Carsten Burstedde, Omar Ghattas, Michael Gurnis, Georg Stadler, Eh Tan, Tiankai Tu, Lucas C Wilcox, and Shijie Zhong. Scalable adaptive mantle convection simulation on petascale supercomputers. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 62. IEEE Press, 2008.

[2] `http://www.gebco.net/data_and_products/gridded_bathymetry_data/`

[9] CE Castro, M Käser, and EF Toro. Space–time adaptive numerical methods for geophysical applications. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367, 2009.

[10] George Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of parallel and distributed computing*, 7(2):279–301, 1989.

[11] Karen Devine, Bruce Hendrickson, Erik Boman, Matthew St John, and Courtenay Vaughan. Design of dynamic load-balancing tools for parallel applications. In *Proceedings of the 14th international conference on Supercomputing*, pages 110–118. ACM, 2000.

[12] Anton Frank. *Organisationsprinzipien zur Integration von geometrischer Modellierung, numerischer Simulation und Visualisierung*. Dissertation, München, 2000.

[13] David L George. Augmented riemann solvers for the shallow water equations over variable topography with steady states and inundation. *Journal of Computational Physics*, 227(6):3089–3113, 2008.

[14] Daniel F Harlacher, Harald Klimach, Sabine Roller, Christian Siebert, and Felix Wolf. Dynamic load balancing for unstructured meshes on space-filling curves. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1661–1669. IEEE, 2012.

[15] G Horton. A multi-level diffusion method for dynamic load balancing. *Parallel Computing*, 19(2):209–218, 1993.

[16] W. B. March et al. Optimizing the comp. of n-point correlations on large-scale astronomical data. In *Proc. of the Int. Conf. on High Perf. Comp., Netw., Stor. and Analysis*, SC '12. IEEE Computer Society Press, 2012.

[17] Oliver Meister, Kaveh Rahnema, and Michael 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*, volume 22 of *Advances in Parallel Computing*, pages 251–260, Gent, May 2012. ParCo 2012.

[18] William F Mitchell. A refinement-tree based partitioning method for dynamic load balancing with adaptively refined grids. *Journal of Parallel and Distributed Computing*, 67(4):417–429, 2007.

[19] Bongki Moon, Hosagrahar V Jagadish, Christos Faloutsos, and Joel H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *Knowledge and Data Engineering, IEEE Transactions on*, 13(1):124–141, 2001.

[20] Ralf-Peter Mundani. *Hierarchische Geometriemodelle zur Einbettung verteilter Simulationsaufgaben*. Dissertation, Aachen, 2006.

[21] Tobias Neckel. *The PDE Framework Peano: An Environment for Efficient Flow Simulations*. Dissertation, Institut für Informatik, Technische Universität München, June 2009.

[22] 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 200k cores and heterog. arch. In *Proc. of the 2010 ACM/IEEE Int. Conf. for HPC, Networking, Storage and Analysis*, SC '10, pages 1–11. IEEE Computer Society, 2010.

[23] Martin Schreiber, Hans-Joachim Bungartz, and Michael Bader. Shared-memory parallelization of fully-adaptive simulations using a dynamic tree-split and -join approach. Puna, India, December 2012. IEEE International Conference on High Performance Computing (HiPC), IEEE Xplore.

[24] Martin Schreiber, Tobias Weinzierl, and Hans-Joachim Bungartz. Cluster optimization and parallelization of simulations with dynamically adaptive grids. In *Euro-Par 2013 Parallel Processing*. Springer, 2013.

[25] Martin Schreiber, Tobias Weinzierl, and Hans-Joachim Bungartz. SFC-based Communication Metadata Encoding for Adaptive Mesh. In *ParCo 2013*, October 2013. in review.

[26] Horst D Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(2):135–148, 1991.

[27] Csaba A. Vigh. *Parallel Simulation of the Shallow Water Equations on Structured Dynamically Adaptive Triangular Grids.* PhD thesis, Institut für Informatik, Technische Universität München, 2012.

[28] T. Weinzierl. *A Framework for Parallel PDE Solvers on Multiscale Adaptive Cartesian Grids.* Verlag Dr. Hut, 2009.

[29] T. Weinzierl and M. Mehl. Peano – A Traversal and Storage Scheme for Octree-Like Adaptive Cartesian Multiscale Grids. *SIAM Journal on Scientific Comp.*, 33(5):2732–2760, October 2011.

[30] Tobias Weinzierl. *A Framework for Parallel PDE Solvers on Multiscale Adaptive Cart. Grids.* Dissertation, Institut für Informatik, Technische Universität München, München, 2009.

[31] Gengbin Zheng and Laxmikant V Kale. *Achieving high performance on extremely large parallel machines: performance prediction and load balancing.* Citeseer, 2005.

[32] Gerhard Zumbusch. *On the quality of space-filling curve induced partitions.* Sonderforschungsbereich 256, 2000.