ForestClaw: Hybrid forest-of-octrees AMR for hyperbolic conservation laws

Carsten BURSTEDDE [a,1], Donna CALHOUN [b], Kyle MANDLI [c] and Andy R. TERREL [c]

[a] *Institut für Numerische Simulation, Universität Bonn, Germany*
[b] *Boise State University, Idaho, USA*
[c] *Institute for Computational Engineering and Sciences, The University of Texas at Austin, USA*

Abstract. We present a new hybrid paradigm for parallel adaptive mesh refinement (AMR) that combines the scalability and lightweight architecture of tree-based AMR with the computational efficiency of patch-based solvers for hyperbolic conservation laws. The key idea is to interpret each leaf of the AMR hierarchy as one uniform compute patch in $\mathbb{R}^d$ with m^d degrees of freedom, where m is customarily between 8 and 32. Thus, computation on each patch can be optimized for speed, while we inherit the flexibility of adaptive meshes. In our work we choose to integrate with the `p4est` AMR library since it allows us to compose the mesh from multiple mapped octrees and enables the cubed sphere and other nontrivial multiblock geometries. We describe aspects of the parallel implementation and close with scalings for both MPI-only and OpenMP/MPI hybrid runs, where the largest MPI-only run uses 16,384 CPU cores.

Keywords. adaptive mesh refinement, hyperbolic conservation laws, clawpack, HPC, manycore

1. Introduction

With the advent of many-core chips such as GPUs and the MIC architecture comes the opportunity to sustain unprecedented rates of floating point operations at comparably high integration density and low cost. These architectures, however, require careful structuring of the data layout and memory access patterns to exhaust their multithreading and vectorization capabilities.

Consequently, it is not clear a priori how to accelerate PDE solvers that use adaptive mesh refinement. Of course, it was realized early that it helps to aggregate degrees of freedom (DOF) at the element level, as has been done with high-order spectral element [1], low order continuous Galerkin methods that accumulate many elements simultaneously [2], or discontinuous Galerkin [3] methods. GPU implementations of the latter have been proposed recently [4, 5]. The finite volume method has typically been implemented using a single degree of freedom per cell on structured [6, 7] or unstructured

[1] Corresponding author. E-mail: `burstedde@ins.uni-bonn.de`

meshes [8]; higher order methods have also been constructed by widening the stencil, for instance in WENO methods [9].

To facilitate hardware acceleration for parallel dynamic AMR, we build upon the forest-of-octrees paradigm because of its low overhead and proven scalability [10]. This approach identifies each octree leaf with a mesh element. The present work does not construct a traditional high-order element but defines each element to be a dense computational patch with m^d DOFs. In fact, this approach resembles block-structured AMR [11–14] except that the patches are not overlapping, which enables us to capitalize on our previous experience with scalable FE solvers for PDEs [15]. The CLAWPACK software [16] provides a popular implementation of such a patch. It has been designed to solve hyperbolic conservation laws and successfully used in the context of block-structured AMR [17–19].

In this paper we describe our design for the coupling of forest-of-octrees AMR with CLAWPACK at the leaf level. We comment on challenges that arise in enabling multiblock geometries and efficient parallelism and conclude with a range of numerical examples that demonstrate the conceptual advantages.

2. Design principles

The starting point of our work is defined by the `p4est` algorithms for forest-of-octrees AMR on the one hand, and the CLAWPACK algorithms for the numerical solution of hyperbolic conservation laws on the other. Both are specialized codes with the following characteristics:

	`p4est`	CLAWPACK
subject	hexahedral nonconforming mesh	hyperbolic PDE on $[0,1]^d$
toplevel unit	forest of octrees	patch of m^d FV cells
atomic unit	octree leaf	one DOF in each cell
parallelization	MPI	threads (Manyclaw variant)
memory access	distributed	shared on each MPI rank
data type	integers	floating point values
language	C	Fortran 77

Each leaf as the atomic unit of `p4est` houses a toplevel unit of CLAWPACK. The term cell is used to identify a single DOF within a CLAWPACK patch. The proposed 1:1 correspondence between a leaf and a patch thus combines two previously disjoint models in a modular way:

1. We permit the reuse of existing, verified, and performant codes.
2. We preserve the separation between the mesh on one hand and the discretization and solvers on the other.
3. The AMR metadata (`p4est`: under 1k bytes per octree, 8 bytes per MPI rank, 24 bytes per leaf. ForestClaw: $84 + 28d$ bytes per patch) is insignificant compared to the numerical data (m^d floating point values per patch).
4. The resulting parallel programming model is a hybrid (often referred to as MPI+X). Only rank-local leaves/patches are stored and computed on.

A particular feature of ForestClaw is that the generic handling of multiblock geometries is inherited from `p4est`, identifying each octree as a block. Each block is understood as a reference unit cube with its own geometric mapping. The connectivity of the blocks can be created by external hexahedral mesh generators, eliminating the need to encode it by hand.

A main challenge is presented by the fact that the patch neighborhood is only known to `p4est`. This patch connectivity information needs to be propagated to the numerical code in ForestClaw that implements the interaction with neighbor patches via the use of a layer of (typically two) ghost cells surrounding each patch. To this end, we have designed an interface that provides access to the sequence of blocks, the list of patches per-block, and a constant-time lookup of neighbor patches (and their relative and, in general, nontrivial orientation between blocks). Suitably informed by `p4est`, ForestClaw stores only the patches local to each MPI rank.

Mesh modification directives, such as adaptive refinement and coarsening, are called from ForestClaw and executed as black-box operations inside `p4est`. The only information that flows into `p4est` is a set of per-patch refinement and coarsening flags, which are computed from the numerical state in ForestClaw. Our current algorithm requires neighbor patches to have a size difference of at most a factor of two (the 2:1 balance condition). This condition can have nonlocal effects on the refinement pattern and is enforced by special-purpose parallel algorithms in `p4est` [20].

3. Patch-based numerics at the leaf level

For hyperbolic problems, we integrate the solution on a single uniform patch, containing m^d cells, using the wave propagation algorithm described by R. J. LeVeque [21] and implemented in CLAWPACK [7, 22]. We assume a single degree of freedom per cell and reconstruct a piecewise constant solution to obtain left and right states at cell edges. At each edge, we solve Riemann problems to obtain left and right going waves propagating at speeds determined from the solution to the Riemann problem. For scalar advection, the speed of each wave is simply the local advection speed at the cell interface. For non-linear problems and systems, an approximate Riemann solver, such as a Roe solver [23], is typically used. Since much of the physics of an application can be contained in the Riemann solver, ForestClaw adopts CLAWPACK's interface to Riemann solvers effectively allowing problems solved with CLAWPACK to be solvable in ForestClaw. Wave limiters are used to supress spurious oscillations resulting from truncation error in a typical second order method.

Data exchanges between neighboring patches are done via layers of ghost cells extending the dimensions along the edges of each patch. The interior edge values of a given patch overlap the ghost cell region of a neighboring patch. For the second order wave propagation algorithm, two layers of ghost cells are sufficient. This implies that one layer of ghost patches is sufficient for $m \geq 4$. Neighboring patches at the same level of refinement simply copy their interior edge values into a neighbors' ghost cells. Neighboring fine grid patches average their interior edge data to a coarser neighbor's ghost cell values. And neighboring coarse grid patches interpolate data from their interior edge cells to their fine grid neighbor's ghost cell values. To avoid loss of conservation and the creation of spurious extrema, we use a standard conservative, limited interpolation

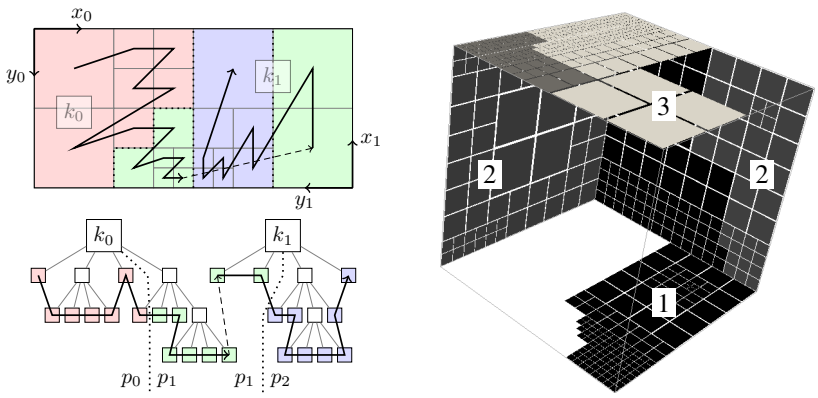


Figure 1. Left: Forest of two quadtrees, partitioned among three MPI processes. Each quadtree has its own coordinate orientation. The situation in 3D (octrees) is analogous. Right: The leaves in a forest of six quadtrees that serves as the computational domain for the cubed sphere. An adhoc refinement pattern has been 2:1 balanced in terms of neighbor sizes and partitioned between five MPI processes (the three middle ones out of $0 \ldots 4$ are shown with a color scale from black to white).

scheme to interpolate values from the coarse grid to fine grid coarse cells [18, 24]. When sub-cycling, time accurate data between coarse grids is used to fill in ghost cells for fine grids. As we detail in the following section, this procedure can be extended transparently to distributed parallelism by defining an abstract exchange routine for ghost patch data.

Mesh refinement and coarsening requires interpolation from coarse grids to newly created fine grids, and the averaging of fine grid data to a newly created underlying coarse grid. This operation is rank-local, analogous to the general dynamic AMR procedures used in `p4est`-based FE codes [25, Fig. 4].

4. Parallelization

The MPI layer is addressed from within `p4est` and not exposed to the ForestClaw code. The leaf ordering is maintained in `p4est` according to a space filling curve. Each MPI rank has a local view on its own partition, augmented where necessary with information about one layer of ghost leaves (see Figure 1).

ForestClaw uses iterators over all rank-local leaves to execute numerical tasks one patch at a time, optionally restricted to a given refinement level. Random access to patches is possible and used when dereferencing the results from neighbor lookups. Looping over the patches in the order prescribed by the forest and accessing neighbors only relative to the current patch leads to a high percentage of cache reuse [26].

When ForestClaw accesses neighbor patches, they can be on the same or a different block. In the latter case, coordinate transformations are carried out. The structure of ForestClaw is oblivious to the fact that it only has a local view of the distributed mesh and data which relieves the developer from programming to the MPI interface.

With MPI parallelism, neighbor patches can be either local or assigned to a different process (ghost patches). Since this must make no difference numerically, we need to ensure that the values of parallel neighbor patches are current whenever they are accessed. To this end, we allocate storage for a layer of ghost patches in ForestClaw, which

we can pass to a general-purpose `p4est` routine that communicates local leaf data to all processes that view a particular leaf as a ghost. If we call this routine before we go through the local neighbor interactions, we can handle ghost values implicitly by neighbor lookups without querying if they are local or remote.

In the context of finite-element or finite-difference methods, there should be one such parallel data exchange per time step for a global value of the time step length $\mathrm{d}t$, or one exchange per discretization level per time step if $\mathrm{d}t$ is chosen per-patch depending on its size (this is sometimes called sub-cycling). In the tradition of block-structured AMR codes, interaction between neighbor patches is done in a hierarchy from coarse to fine levels, and then correction factors are propagated back from fine to coarse levels, requiring an exchange at each level. With sub-cycling, this entails a recursion with an operation count that is exponential in the difference between the largest and the smallest refinement level, with the benefit that $\mathrm{d}t$ matches the CFL condition at each level. While a large number of exchanges per time step can present a scaling bottleneck due to the inherent synchronization and latency losses, the expectation is still that the time to solution improves when switching from uniform to adaptive meshes, since we are computing with fewer patches, and improves again when enabling sub-cycling, since we take larger time steps on the coarser levels.

The space-filling curve paradigm allows for lightweight repartitioning algorithms. Even uniform meshes benefit from this approach since the number of processors does not need to be commensurable with the number of patches in each space dimension. For adaptive meshes, we repartiton after every refinement operation and transfer the numerical data accordingly; see [25, Fig. 4] for the overall procedure. When using sub-cycling, we have the option to assign a weight to each patch based on its level, determined by the expected number of sub-cycles per coarse-level time step, and partition to equidistribute the cumulative weight between the processors. Currently, we are not enforcing a load balance that is attained separately for each level, as it is done for example in Chombo [13] or recent geometric adaptive multigrid schemes [27].

The threaded parallelism over the degrees of freedom of a patch can be handled by ForestClaw alone without the need to involve `p4est`. For instance, a many-core implementation, such as Manyclaw [28], can be used for the integration of the hyperbolic system on a leaf-patch thereby allowing for hybrid parallelism. The design of leaf-patches then can enable efficient management of data local to many-core architectures (for example the Xeon Phi coprocessor) and the host.

5. Numerical results

We provide two examples that demonstrate the ForestClaw code. The numerical results to date have been designed to verify that interface layer between `p4est` and ForestClaw is sufficiently flexible and robust. Of particular importance was ensuring that all ghost cell transfers (including averaging and interpolation) are implemented correctly. The basic CLAWPACK algorithm and corresponding code are thoroughly tested and need no further verification in our context.

In both sets of numerical results, we solve a scalar advection equation,

$$q_t + (\mathbf{u} \cdot \nabla) q = 0, \tag{1}$$

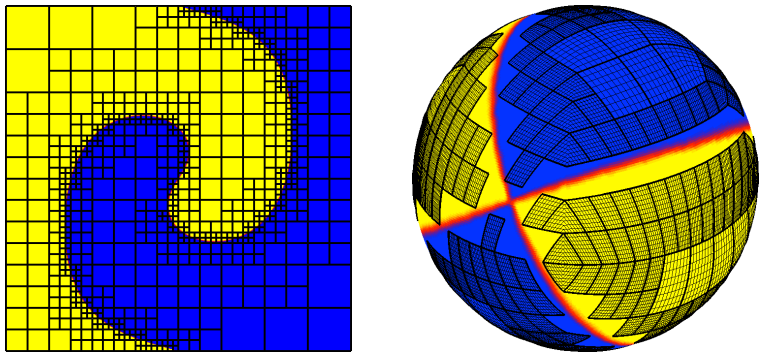


Figure 2. Numerical results for solving the advection equation. Left: Unit square (single quadtree) with a twisting velocity field. Right: Spherical ball with a rotational velocity field constructed from two mapped quadtrees. In both cases the concentration is color-coded with a sharp gradient shown in red. The adaptive mesh refinement follows the location of the gradient (the patches are not shown where they become too fine for display). Here we use $m = 8$ cells per CLAWPACK patch.

where the velocity field $\mathbf{u} = (u(\xi,\eta,t), v(\xi,\eta,t)$ is a prescribed function of computational coordinates. The relevant numerical parameters that are set in each case include the size of the patch on each leaf ($m = 8$, 16, or 32), and the minimum and maximum refinement levels, which in turn fix minimum and maximum effective mesh resolutions.

In both examples, a patch is tagged for refinement if the difference between its maximum and minimum values exceeds a prescribed threshold. A family of 2^d patches is coarsened to a parent patch if this patch would not meet the criteria for refinement.

Example 1: An initial concentration field q is set to 0 in the left half of a computational square and to 1 in the right half. A time dependent flow field is prescribed that distorts the interface between the 0 and 1 concentration values. Figure 2 shows the results at an intermediate time step, where the minimum and maximum levels of refinement are set to 3 and 6, respectively. This results in a minimum resolution of 64×64 and a maximum resolution of 512×512.

Example 2: We demonstrate the multiblock functionality of ForestClaw by considering flow on a sphere. The sphere mapping we use consists of two quadtrees, each defined in the computational space $[-1,1] \times [-1,1]$ [29,30]. Each quadtree is mapped to cover one hemisphere. The initial condition is $q = 0$ in one half of the sphere, and $q = 1$ in the other half, where the halves are not necessarily aligned with the equator of the mapping. The flow field u simulates rigid body rotation. We show the results at an intermediate time in Figure 2.

We begin our analysis of Example 1 with uniformly-refined mesh experiments run on TACC's Stampede supercomputer at different levels of resolution. We use 1 to 16 MPI processes on one 16-core compute node, obtaining a strong scaling efficiency for the time integration between 97%–103% depending on the resolution (from levels 1 and 8). Then, we examine processor counts from 16 to 256 using 16 cores per node. The scaling behavior remains linear for the time integration itself, and mostly linear for the local and parallel neighbor exchanges which require about 10% of the time integration run time; see Figure 3.

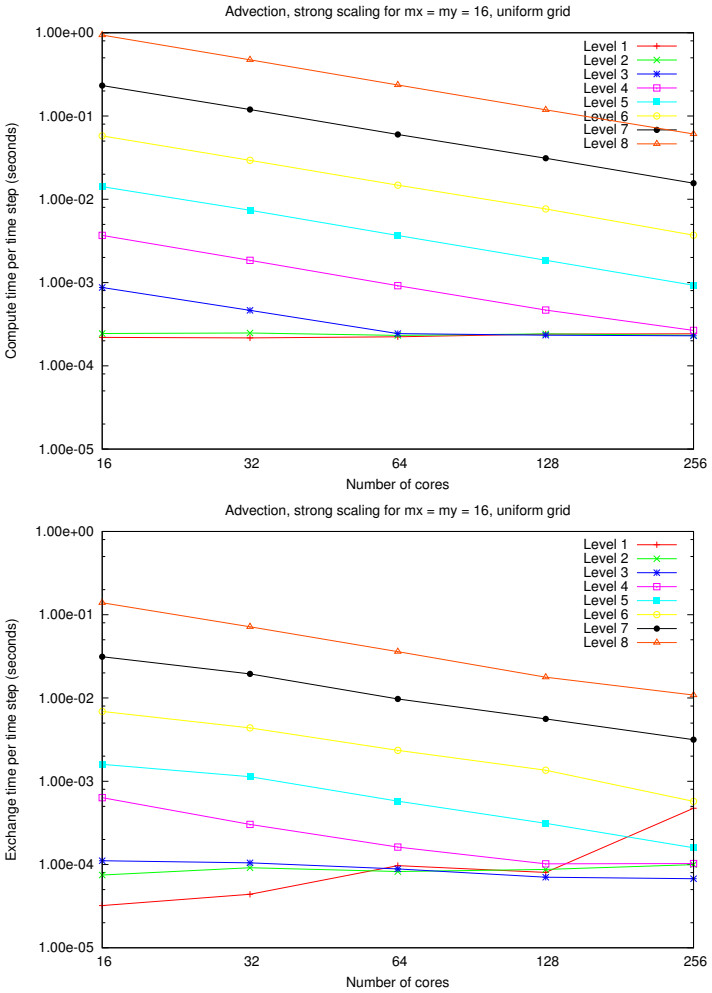


Figure 3. Strong scaling of MPI parallelism for advection on a uniformly refined mesh. Top: run time of time integration. Bottom: time spent in local and parallel neighbor exchanges. The number of patches is $2^{2\ell}$ at level ℓ. The flat lines for the smaller runs are caused by having fewer patches than MPI processes, leaving some of the processors idle. For each data point we use the time required by the slowest processor to verify that the load is balanced equally.

In Figure 4, we scale multi-node parallel calculations. We use the Manyclaw set of patch-based PDE solvers. Manyclaw provides a generic set of interators for Riemann problems and related algorithms.

We have examined weak scalability as well by using four times as many processes for each level increase, from level 6 at 64 cores to level 11 at 16,384. Up to 4,096 cores, the exchange times stay consistently below 16% of the time integration and the time in seconds per time integration step varies between .0578 and .0602, which yields a parallel efficiency of 96%. For the data point at 16,384 cores, the efficiency is still 82%.

Next, we examine how we can reduce the overall wall clock time of a simulation by switching from uniformly refined meshes to adaptive refinement with the same maximum

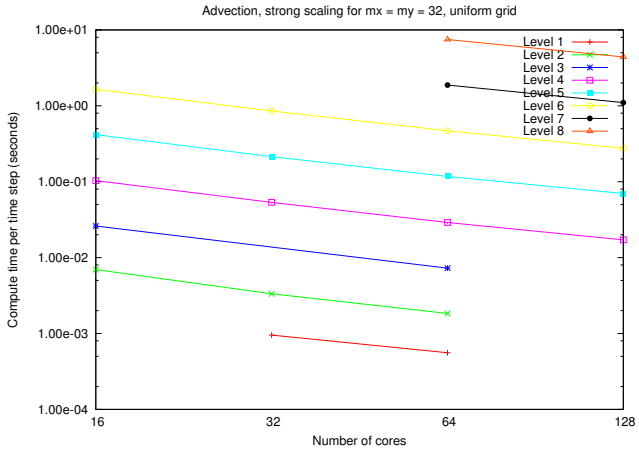


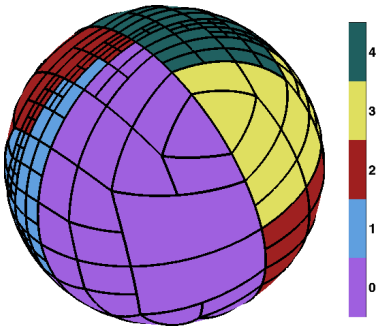
Figure 4. Strong scaling of OpenMP parallelism for advection on a uniformly refined mesh: MPI + OpenMp hybrid parallelization of the time integration.

strategy				wall clock time	
mesh	remesh	partition	time step	$P = 16$	$P = 256$
uniform	none	by count	global	3961.	256.
AMR	every step	by count	global	252.	54.6
AMR	every 4	by count	global	178.	39.7
AMR	every step	by count	subcycle	99.9	17.3
AMR	every 4	by count	subcycle	87.2	14.0
AMR	every step	by weight	subcycle	95.7	18.2
AMR	every 4	by weight	subcycle	84.4	14.2

Table 1. Comparison of different meshing strategies for the advection example with a fixed maximum level 8, run on 16 and 256 cores, respectively. The wall clock time includes the whole run of the program beginning to end (file I/O disabled). The adaptive runs converge to a minimum level of 3.

level, effectively coarsening where high resolution is not needed. For an example with a sharp front as depicted in Figure 2 we expect considerable savings by AMR, which we confirm in Table 1. The wall clock times can be reduced by up to a factor of 50 by combining AMR with subcycling. The adaptive runs are still load balanced overall, which we infer from the linear scaling of the time integration alone (not shown) and from the fact that the weighted partition does not influence the run times. There is the tradeoff however, at least in the current implementation, that the adaptive runs exhibit reduced scalability associated with the ghost patch exchanges described earlier. Presently, ghost patch exchanges are made across all levels, even if only two levels are involved in the exchange. One enhancement of the code that is currently being developed is to rewrite the time stepping loop with the objective of minimizing the parallel neighbor exchanges.

To demonstrate a multiblock connectivity, we include results for Example 2 in Figure 5, together with a strong scaling table that shows that the spherical example scales reasonably well too.



P	time int./step
16	1.115
32	0.557
64	0.297
256	0.092

Figure 5. Left: Partition of the spherical Example 2 between five MPI processors, indicated by the color scale, at simulated time $T = 1$. The partition shows the mesh equator, while the refinement is invariant under the number of processors and the partition and follows the front, which is not aligned with the equator. Right: Strong scaling for a uniform run at level 8.

6. Conclusion

We have presented the integration of an MPI-based forest-of-octrees adaptive meshing code, `p4est`, with a numerical solver for hyperbolic conservation laws, CLAWPACK/Manyclaw, that implements threaded parallelism for a single compute patch on the other. All topological operations on the mesh are handled by `p4est`, while all numerical operations are handled by the CLAWPACK implementation. This combination naturally enables a hybrid MPI/threaded parallelization. We demonstrate scalability of this approach in various ways, where the MPI-only variant scales weakly up to 16,384 codes, and strong scalability is linear for up to 256 cores in the MPI-only and 128 in the threaded implementation

Acknowledgements

We would like to thank the Texas Advanced Computing Center (TACC) for access to the Stampede supercomputer under allocation TG-DPP130002 and TG-ASC130001 granted by the NSF XSEDE Program. The authors acknowledge valuable discussion with Randy LeVeque, Marsha Berger, and Hans-Petter Langtangen. We also acknowledge David Ketcheson and the KAUST sponsored HPC^3 numerics workshop at which the initial phases of this project were first discussed. The second author would like to also acknowledge the Isaac Newton Institute (Cambridge, UK), where much of the preliminary development work for ForestClaw was done. The leaf/patch paradigm was independently presented by B. as part of a talk at the SCI Institute, Salt Lake City, Utah, USA in July 2011. The fourth authors recognize Simula National Lab for funding.

References

[1] H. M. Tufo and P. F. Fischer. Terascale spectral element algorithms and implementations. In *Proceedings of the ACM/IEEE SC99 Conference on High Performance Networking and Computing*, 1999.

[2] Matthew G. Knepley and Andy R. Terrel. Finite Element Integration on GPUs. *Transactions of Mathematical Software*, 39(2), 2013.

[3] Jan S. Hesthaven and Timothy Warburton. Nodal high-order methods on unstructured grids. I. Time-domain solution of Maxwell's equations. *Journal of Computational Physics*, 181(1):186–221, 2002.

[4] A. Klöckner, T. Warburton, J. Bridge, and J. S. Hesthaven. Nodal discontinuous Galerkin methods on graphics processors. *Journal of Computational Physics*, 228(21):7863–7882, 2009.

[5] Carsten Burstedde, Omar Ghattas, Michael Gurnis, Tobin Isaac, Georg Stadler, Tim Warburton, and Lucas C. Wilcox. Extreme-scale AMR. In *SC10: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM/IEEE, 2010.

[6] Phillip Colella and Paul R. Woodward. The Piecewise Parabolic Method (PPM) for Gas-Dynamical Simulations. *Journal of Computational Physics*, 54:174–201, 1984.

[7] AMRClaw website: `http://www.clawpack.org`.

[8] OpenFOAM website: `http://openfoam.org`.

[9] Chi-Wang Shu. High Order Weighted Essentially Nonoscillatory Schemes for Convection Dominated Problems. *SIAM Review*, 51(1):82–126, February 2009.

[10] Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.

[11] M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53:484–512, 1984.

[12] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.*, 82:64–84, 1989.

[13] Phillip Colella, Daniel T. Graves, Noel Keen, Terry J. Ligocki, Daniel F. Martin, Peter W. McCorquodale, David Modiano, Peter O. Schwartz, Theodore D. Sternberg, and Brian Van Straalen. *Chombo Software Package for AMR Applications. Design Document.* Applied Numerical Algoirthms Group, NERSC Division, Lawrence Berkeley National Laboratory, Berkeley, CA, May 2007.

[14] M. Berzins, J. Luitjens, Q. Meng, T. Harman, and C. A. Wight. Uintah: a scalable framework for hazard analysis. In *Proceedings of the 2010 TeraGrid Conference*, 2010.

[15] Carsten Burstedde, Georg Stadler, Laura Alisic, Lucas C. Wilcox, Eh Tan, Michael Gurnis, and Omar Ghattas. Large-scale adaptive mantle convection simulation. *Geophysical Journal International*, 2013. Accepted for publication.

[16] R. J. LeVeque. Wave propagation algorithms for multi-dimensional hyperbolic systems. *Journal of Computational Physics*, 131:327–353, 1997.

[17] M. J. Berger and R. J. LeVeque. A rotated difference scheme for Cartesian grids in complex geometries. AIAA Conference on Computational Fluid Dynamics, Honolulu, HI, CP-91-1602, 1991.

[18] AMRClaw website: `http://www.clawpack.org`.

[19] Randall J. LeVeque, David L George, and Marsha J Berger. Tsunami Propagation and inundation with adaptively refined finite volume methods. *Acta Numerica*, pages 211–289, 2011.

[20] Tobin Isaac, Carsten Burstedde, and Omar Ghattas. Low-cost parallel algorithms for 2:1 octree balance. In *Proceedings of the 26th IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2012.

[21] Randall J. LeVeque. Wave Propagation Algorithms for Multidimensional Hyperbolic Systems. *Journal of Computational Physics*, 131(2):327–353, March 1997.

[22] R. J. LeVeque. *Finite volume methods for hyperbolic problems*. Cambridge University Press, 2002.

[23] P. L. Roe. Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. *Journal of Computational Physics*, 43(2):357–372, October 1981.

[24] Chombo website: `http://seesar.lbl.gov/anag/chombo`.

[25] Carsten Burstedde, Omar Ghattas, Georg Stadler, Tiankai Tu, and Lucas C. Wilcox. Towards adaptive mesh PDE simulations on petascale computers. In *Proceedings of Teragrid '08*, 2008.

[26] Carsten Burstedde, Martin Burtscher, Omar Ghattas, Georg Stadler, Tiankai Tu, and Lucas C. Wilcox. ALPS: A framework for parallel adaptive PDE solution. *Journal of Physics: Conference Series*, 180:012009, 2009.

[27] Hari Sundar, George Biros, Carsten Burstedde, Johann Rudi, Omar Ghattas, and Georg Stadler. Parallel geometric-algebraic multigrid on unstructured forests of octrees. In *SC12: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM/IEEE, 2012.

[28] Andy R. Terrel and Kyle T. Mandli. ManyClaw: Slicing and dicing Riemann solvers for next generation highly parallel architectures. *TACC-Intel Symposium on Highly Parallel Architectures*, 2012.

[29] D. A. Calhoun, C. Helzel, and R. J. LeVeque. Logically rectangular grids and finite volume methods for PDEs in circular and spherical domains. *SIAM Review*, 50(4):723–752, 2008.

[30] M. J. Berger and R. J. LeVeque D. Calhoun, C. Helzel. Logically rectangular finite volume methods with adaptive refinement on the sphere. *Phil. Trans. R. Soc. A*, 367(1):4483–4496, 2009.