Petascale Block-Structured AMR Applications Without Distributed Meta-data

Brian Van Straalen, Phil Colella, Daniel T. Graves, Noel Keen

Applied Numerical Algorithms Group,
Lawrence Berkeley National Laboratory,
Berkeley, CA 94720, USA

Abstract. Adaptive mesh refinement (AMR) applications to solve partial differential equations (PDE) are very challenging to scale efficiently to the petascale regime.
We describe optimizations to the Chombo AMR framework that enable it to scale efficiently to petascale on the Cray XT5. We describe an example of a hyperbolic solver (inviscid gas dynamics) and an matrix-free geometric multigrid elliptic solver. Both show good weak scaling to 131K processors without any thread-level or SIMD vector parallelism.
This paper describes the algorithms used to compress the Chombo metadata and the optimizations of the Chombo infrastructure that are necessary for this scaling result. That we are able to achieve petascale performance without distribution of the metadata is a significant advance which allows for much simpler and faster AMR codes.

1 Introduction

PDE solvers using adaptive mesh refinement, AMR, on block structured grids, e.g. [3,4], are among the most challenging applications to adapt to massively parallel computing environments. Because the grids can be anywhere in the domain, metadata is required to describe where the data lives and what processor is responsible for it. Standard Chombo metadata is not distributed- all processors keep a redundant index of the distributed data layout. Previous results [10] have shown that Chombo AMR with scales well to 10K processors. The size (in memory) of the metadata in Chombo made further scaling impossible without significant metadata redesign. Other AMR infrastructures have distributed their metadata among the processors. PARAMESH [7] and SAMRAI [12] do this for large problems. BoxLib [5], in the CASTRO code [1], does not distribute metadata but extensively threads their code and uses bigger grids and were able to scale up to 200K processors using one grid per processor. In this paper, we use Chombo's flat MPI method and we compress Chombo's metadata. We show good weak scaling to petascale with many more boxes per processor (77 in the hyperbolic case, 53 in the elliptic case). Because we do not distribute our metadata, we avoid substantial increases in code complexity and communication time.

2 AMR Applications

Block-structured AMR, developed by Berger and Oliger [3, 4] for computational gas dynamics, is a multi-scale algorithm that achieves high spatial and temporal resolution in localized regions of dynamic multidimensional numerical simulations. A broad range of physical phenomena modeled by

PDE exhibit multi-scale behavior where variations in the solution occur over scales that are much smaller than the overall problem domain. Examples include flame fronts arising in the burning of hydrocarbon fuels, nuclear burning in supernovae, effects of localized features in orography or bathymetry on ocean currents, tracking of tropical cyclones, localized kinetic effects for plasma physics problems, and, in general, small scale effects due to nonlinear instabilities. In each of these problems, the fundamental mathematical description is given in terms of various combinations of PDE of classical type (elliptic, parabolic, hyperbolic). The Berger and Oliger AMR algorithm organizes refined regions into rectangular structured grids of several hundred to several thousand grid points per grid. High-resolution structured-grid methods (typically expressed as stencils) are used to advance the solution in time. Furthermore, the overhead of managing the irregular data is amortized over a relatively large number of floating point operations on the rectangular grids. For time-dependent problems, refinement is performed in time as well as space. Each level of spatial refinement has its own stable time step, with the time steps on a level constrained to be integer multiples of the time steps on all finer levels.

2.1 Chombo AMR Framework

AMR applications require a long-term sustained investment in software infrastructure to create scalable solvers that are capable of utilizing the full capabilities of the largest available HPC platforms. We have created a framework for implementing scalable parallel AMR calculations called `Chombo` [6] that provides an environment for rapidly assembling portable, high-performance AMR applications for a broad variety of scientific disciplines.

`Chombo` is a fully instrumented C++ library. There are a set of timer macros that can be used to time functions or sections of code. These timers attempt to use native instructions on the target architecture in order to minimize the overhead of collecting detailed performance data.

In the standard Chombo framework, there is metadata associated with each computational region on a adaptive (called a "Box"). Each Box contains the integer locations of the lower left and upper right corners of the region. A collection of these regions along with their processor mapping (a "DisjointBoxLayout") is represented internally by a

```
Vector< pair<Box, int> >
```

This metadata is not distributed and therefore grows with the size of the problem. In previous scaling studies [10], the memory cost of this metadata was shown to become prohibitive in the 8K-32K processor range. A large part of the current work is to compress this metadata without distributing it. This allows the Chombo framework to scale to 131K processors for both elliptic and hyperbolic benchmarks without without the large memory cost and without the communication cost associated with distributed metadata.

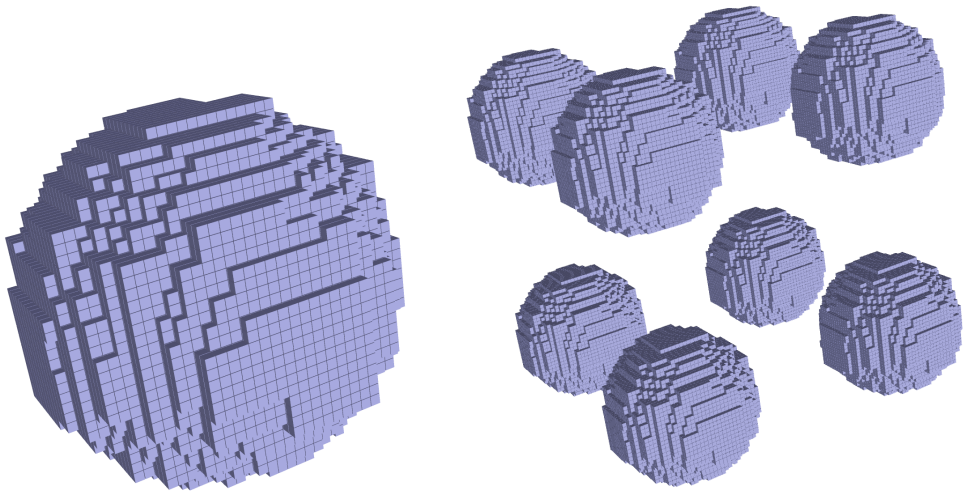
3 Benchmarking Methodology

In many applications that use PDE solvers, the primary motivation for using large numbers of processors is to achieve weak scaling. Even with AMR, many leading scientific problems remain out of reach due to inadequate grid resolution. In those cases, increasing the number of processors is used to increase the spatial resolution of the grids using the minimum number of processors necessary to fit the problem into the available memory. Therefore, we focus on a methodology for

constructing weak-scaled AMR benchmarks because this methodology models the dominant use-case for scientific problems that employ this computational method. We use two different examples for our benchmark, an explicit Godunov method for gas dynamics and a multigrid solver to solve Poisson's equation. These two are reasonable proxies for the two components of most complete AMR applications: explicit solvers for hyperbolic equations and implicit solvers for elliptic and parabolic equations.

3.1 Replication Scaling Benchmarks

Following [10] we use two benchmarks based on replication scaling. We take a grid hierarchy and data for a fixed number of processors and scale it to higher concurrencies by making identical copies of the hierarchy and the data, see Figure 1. The full AMR code (processor assignment, problem setup, etc.) is run without any modifications to guarantee it is not directly aware of the replicated grid structure. Replication scaling tests most aspects of weak scalability, is simple to define, and provides results that are easy to interpret. Thus, it is a very useful tool for understanding and correcting impediments to efficient scaling in an AMR context.



(a) (b)

Fig. 1. (a) Grids at the finest AMR level used in the hyperbolic gas dynamics benchmark – these grids cover the shock front of a spherical explosion in 3D. (b) Replicated grids at the finest AMR level used in the weak scaling performance study of the hyperbolic gas dynamics benchmark. There are 14902 boxes per processor at the finest level before replication. Each box is size 16^3.

3.2 Poisson Benchmark

We first benchmarked an AMR solver for Poisson's equation, based on a cell-centered multilevel discretization of Laplacian in three dimensions [8]. The solver itself used multigrid iteration suitably

modified for an AMR grid hierarchy [2, 11]. The benchmark applied ten iterations of the AMR-multigrid V-cycle, which is typical of the number of iterations required for the solver to converge, and corresponds to 1700 flops/grid point. This is a very demanding application from the standpoint of parallelism – requiring multiple communication and synchronization steps per multigrid iteration. The algorithmic features of this benchmark are typical of broad range of elliptic solvers arising in applications using AMR.

The grids used as the basis of for the Poisson replication benchmark are shown in Figure 3.3. There are three levels of AMR with a refinement ratio of four between each level. There is one unknown per grid point for a total of 15M grid points in the configuration with no replication.

3.3 Hyperbolic Gas Dynamics Benchmark

We benchmarked an explicit method for unsteady inviscid gas dynamics in three dimensions that is based on an unsplit PPM algorithm [9, 13]. This algorithm requires approximately 6000 flops/grid point. Since it is an explicit method, communication between processors is required only once per time step. We used the implementation of this method from the Chombo software distribution without significant modification. The grids used as the basis for the hyperbolic benchmark are shown in Figure 1.

The benchmark used three levels of AMR with a factor of 4 refinement between levels and with refinement in time proportional to refinement in space. We use fixed-sized 16^3 grids and and five unknowns per grid point, with 10^9 grid point updates performed for the single coarse-level time step. None of the grids at any level were changed during any of the time steps, i.e., there was no grid adaptation in time which is sometimes called "regridding". In the results given here, we are only timing the cost of computing a single coarse-level time step, which includes all intermediate and fine time steps on all AMR levels but excludes the problem setup and initialization times.

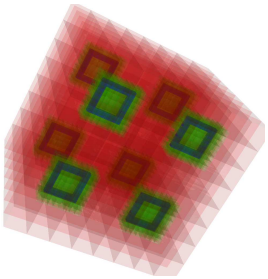


Fig. 2. Grids used in the Poisson benchmark before replication. The red is the level 0 grids, the green is level 1, the blue is level 2. This shows a 2x2 replication. There are 1280 boxes per processor at the finest level before replication. Each box is size 32^3.

4 Optimizing AMR for Scalability

To achieve our performance results for the two 3D Chombo applications discussed, several important changes were made to the standard code. A run-length compression method was used to greatly reduce the memory overhead associated with the metadata for the grids. There were also application-specific optimizations. For example, for our hyperbolic application, we optimized inter-level coarse-fine interpolation objects to take advantage of our new metadata structure. For our elliptic solver, we carefully control the number of communication steps necessary and greatly reduce the number of all-to-all communications in the multigrid algorithm for AMR.

4.1 Memory Performance: Compression

Moore's Law continued unabated for CPU design, but the gap between memory capacity and memory latency has grown every year. In such an environment it makes sense to work with the necessary metadata in an application in a compressed format and utilize the excess of processor cycles to uncompress this information on-the-fly as the processor needs it. This also makes better use of processor-to-memory resources, while decompression can happen in very fast local register storage.

Standard Chombo metadata holds the grid data for each level as explicit vectors of pairs of each box and its associated processor assignment. As the number of boxes becomes large, the memory associated with this representation of the grids grows linearly since this description is not distributed among processors. In [10], it was found that, for a typical Chombo application, the memory usage becomes untenable at between 8K and 16K processors (where the total number of boxes was between 1M and 10M).

We compress the metadata by first stipulating that every patch on a level must be of fixed size. We then create a bitmap of the domain coarsened by the box size and put a 1 where there is to be a box and a 0 where there is none. This bitmap is compressed using run-length compression. The load balancing is done by simply dividing the patches up evenly between processors. If there are N patches per processor, the first N in lexicographic order go to processor 0, the next N to processor 1 and so on. For applications where the load on a patch is more variable, a more flexible load balancing scheme may be necessary.

The results of this change in representation were striking. Figure 3 shows the memory usage for a sample weak scaling run of a gas dynamics solver. The problem has 77 boxes per processor at the finest level (at 196K processors, this amounts to 15.2 million boxes). Figure 5 shows the memory usage for a sample weak scaling run of an elliptic solver with 53 boxes per processor at the finest level (at 98K processors this amounts to 6.55 million boxes). We track the memory that Chombo allocates and measure the memory that the operating system is using for the application. The amount of memory reported by the operating system is substantially higher at high concurrencies and is largely due to MPI memory overhead. In both cases, the metadata compression was necessary to run at the highest concurrency, otherwise the memory of the compute nodes was exhausted.

Figure 4 shows a line labeled "MPI overhead". This is still a speculation on our part, but many tools were used to eliminate and quantify the use of memory in the benchmark applications.

4.2 Run Time Performance

To achieve better scaling of Chombo run time performance at higher concurrencies, optimizations were done for both the hyperbolic and elliptic solvers. For the hyperbolic solver, changes were made

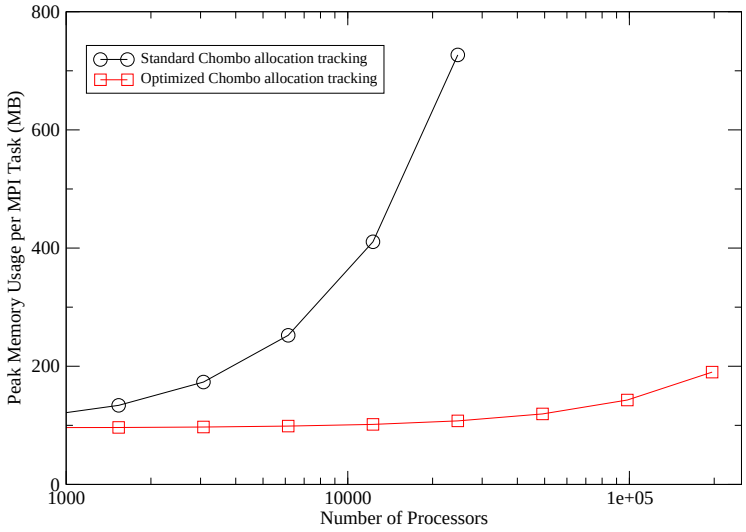
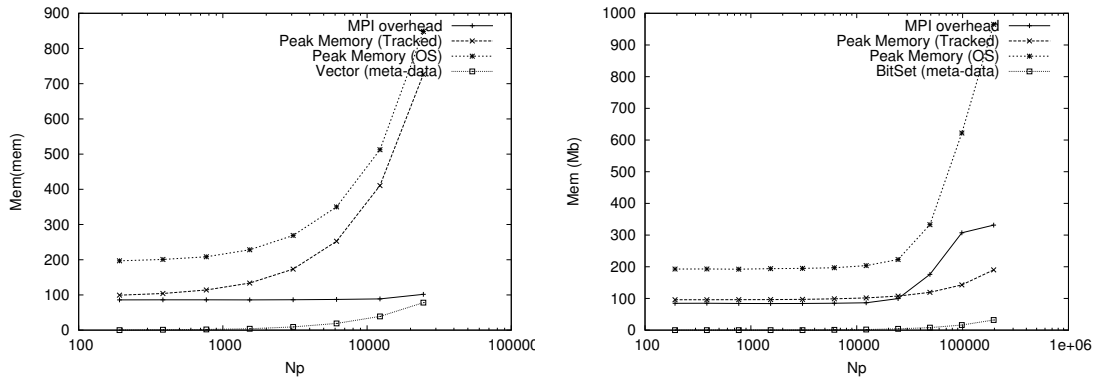
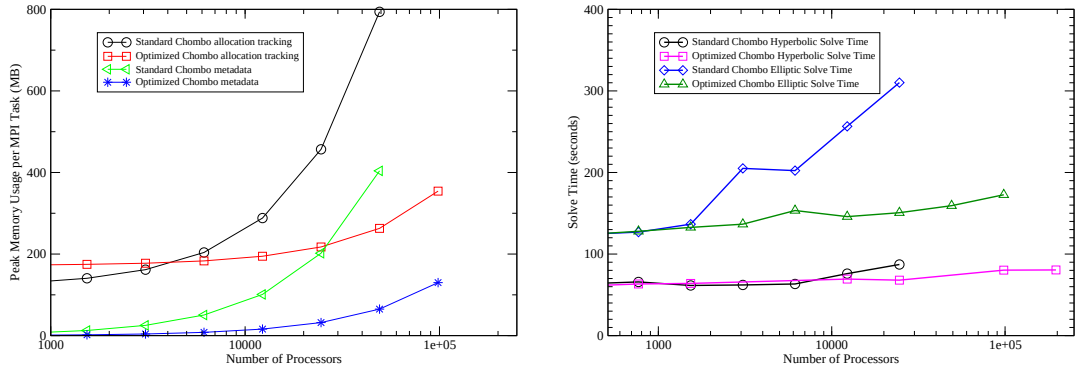


Fig. 3. Memory performance of Chombo 3D inviscid gas dynamics solver before and after metadata compression. In both cases, there are three levels of refinement (factor of 4 between levels) and 77 boxes per processor at the finest level. The standard Chombo solver was not able to run at the highest concurrencies because of memory requirements.



(a) (b)

Fig. 4. A closer look at memory performance of Chombo 3D inviscid gas dynamics solver (a) before and (b) after metadata compression. In both cases, there are three levels of refinement (factor of 4 between levels) and 77 boxes per processor at the finest level. The original solver (a) was not able to run at the highest concurrencies because it ran out of memory.



(a) (b)

Fig. 5. a)Memory performance of Chombo 3D elliptic solver before and after metadata compression. In both cases, there are three levels of refinement (factor of 4 between levels) and 53 boxes per processor at the finest level. The standard Chombo solver was not able to run at the highest concurrencies because of memory requirements. b)Solve time of Chombo 3D hyperbolic and elliptic solvers before and after optimization. In both cases, there are three levels of refinement (factor of 4 between levels). The largest hyperbolic benchmark was run at 196K cores

to the definition of some inter-level objects to account for the new fixed box size representation of the layouts. These changes allowed excellent weak scaling results to 196K processors.

The standard Chombo elliptic solver required several optimizations to facilitate scaling to 98K. We used the fact that the equation is solved in residual-correction form to minimize how often inhomogeneous inter-level interpolation is done in the solve. Standard Chombo also does an extra coarse-fine interpolation before the refluxing step. Previous to our optimizations, there were 8 inhomogeneous coarse-fine interpolations per multigrid v-cycle. We were able to reduce this to only 2 inhomogeneous coarse-fine interpolations per multigrid v-cycle. The standard Chombo Poisson solver does not do box aggregation. Once the input grids are no longer coarsenable, a bottom solver is called. We introduce box aggregation and take multigrid down to a two-cell grid. At the bottom level we simply do two Gauss-Seidel relaxations. This saved a lot of communication time at higher concurrencies because other bottom solvers require substantial all-to-all communication to calculate norms. The AMR Multigrid is this case turns into a true multigrid solve. We reduced the amount of communication at relaxation steps by only doing ghost cell exchanges every other relaxation step. This did not substantially affect the multigrid convergence. The run time comparison between standard Chombo and optimized Chombo for 10 elliptic solves (each with 7 multigrid v-cycles) is given in figure

5. The standard Chombo solver was not able to run at the higher concurrencies because the memory requirements were too large for the machine.

Excellent weak scaling is observed on Jaguar (Cray XT5) to 196K processors for the hyperbolic problem and 98K for the elliptic problem as is shown in Fig. 5. There are 77 boxes per processor for the hyperbolic problem, and 53 boxes per processor for the elliptic problem.

As the problem size per processor remains constant for these weak scaling experiments, we can estimate that the number of flops per processor are also constant. For the hyperbolic solver, the number of total flops is estimated to be $7.4e10$. At a concurrency 196K, with a solve time of 80.5 seconds, the aggregate flop rate is 181 TFlops. For the multigrid elliptic solver, with the estimated number of flops at $4.4e10$, a solve time of 172.7 seconds, and 98K MPI processors, the aggregate flop rate is 25 TFlops.

5 Summary and Conclusions

We present petascale weak scaling results for two key AMR applications. With some modifications of the metadata representation of standard Chombo, we were able to show good weak scaling for both hyperbolic and elliptic problems without having to distribute metadata. Fully local metadata is maintained through the use of compressing the metadata format and decompressing the information as it is utilized in the calculation. The computational cost of working with loss-less compression techniques is not measurable in these applications. This results in efficient memory usage at a slight increase in local processing cycles. This approach involves far less complexity than distributed metadata designs that must balance communication and local caching algorithms, which are application-specific.

In general, we feel that coding theory will become an increasingly important aspect of HPC computing on future platforms where memory and communication costs will increasingly be the science-limiting characteristic of these machines. Trading off excess compute cycles for more efficient memory usage and lower communication costs will be a more common theme in future HPC codes.

While this has been a great success, we recognize that ultimately the flat MPI parallelism model is not extensible to the billion-way concurrency models needed to achieve exascale performance. We consider this work to be but one part of a larger exascale computing strategy where metadata is *not* distributed at the MPI parallelism layer. Within an MPI rank there are several further levels of parallelism to be exploited. The first level is threading the load currently handled sequentially by each MPI rank. The next level is fine-grain parallelism within the dimensional loops within a box. Both areas are being worked on currently. Finally there are instruction-level parallelism models to make use of vector processing within a larger threading model. These are all orthogonal optimization efforts that must all succeed to meet the goal of exaflop simulations.

6 Acknowledgments

We would like James Hack and the OLCF Resource Utilization Council for access to *Jaguar* via the Applied Partial Differential Equations Center project. The authors were supported by the Office of Advanced Scientific Computing Research in the Department of Energy under Contract DE-AC02-05CH11231.

References

1. A. Almgren, J. Bell, D. Kasen, M. Lijewski, A. Nonaka, P. Nugent, C. Rendleman, R. Thomas, and M. Zingale. Maestro, castro and sedona – petascale codes for astrophysical applications. In *SciDAC 2010, J. of Physics: Conference Series*, 2010.

2. A. S. Almgren, T. Buttke, and P. Colella. A fast adaptive vortex method in three dimensions. *J. Comput. Phys.*, 113(2):177–200, 1994.
3. M. Berger and P. Collela. Local adaptive mesh refinement for shock hydrodynamics. *J. Computational Physics*, 82:64–84, May 1989.
4. M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.
5. BoxLib Reference Manual. `https://seesar.lbl.gov/anag/eb/reference-manual/boxlib.html`.
6. P. Colella, D. Graves, T. Ligocki, D. Martin, D. Modiano, D. Serafini, and B. V. Straalen. Chombo software package for AMR applications: design document. `http://davis.lbl.gov/apdec/designdocuments/chombodesign.pdf`.
7. P. MacNeice, K. M. Olson, C. Mobarry, R. deFainchtein, and C. Packer. Paramesh : A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications*, 126:330–354, 2000.
8. D. Martin, P. Colella, and D. T. Graves. A cell-centered adaptive projection method for the incompressible Navier-Stokes equations in three dimensions. *Journal of Computational Physics*, 227:1863–1886, 2008.
9. G. Miller and P. Colella. A conservative three-dimensional Eulerian method for coupled solid-fluid shock capturing. *Journal of Computational Physics*, 183:26–82, 2002.
10. B. V. Straalen, J. Shalf, T. Ligock, N. Keen, and W.-S. Yang. Parallelization of structured, hierarchical adaptive mesh refinement algorithms. In *IPDPS:Interational Conference on Parallel and Distributed Computing Systems*, 2009.
11. M. C. Thompson and J. H. Ferziger. An adaptive multigrid technique for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 82(1):94–121, May 1989.
12. A. M. Wissink, R. D. Hornung, S. R. Kohn, S. S. Smith, and N. Elliot. Large scale parallel structured amr calculations using the samrai framework. In *SC01 Conference on High Perfomrance Computing*, 2001.
13. P. R. Woodward and P. Colella. The numerical simulation of two-dimensional fluid flow with strong shocks. *Journal of Computational Physics*, 54:115–173, 1984.