14 Scalable Structured Adaptive Mesh Refinement with Complex Geometry

Brian Van Straalen, David Trebotich, Andrey Ovsyannikov, and Daniel T. Graves

CONTENTS

14.1 INTRODUCTION

Cartesian structured adaptive mesh refinement (AMR) methods have become an increasingly popular modeling approach to solving partial differential equations (PDEs) in complex or irregular geometries. Though there are several Cartesian grid approaches (e.g., immersed boundary, immersed interface, and ghost fluid), we focus on the cut-cell, embedded boundary (EB) approach of [1], and, specifically, as it pertains to unsteady flow problems where the arrangement of mesh refinement is not expressible *a priori*. In the EB cut cell approach, finite volume approximations are used to discretize the solution in the cut cells that result from intersecting the irregular boundary with a structured Cartesian grid. Conservative numerical approximations to the solution can be found from discrete integration over the nonrectangular control volumes with fluxes located at centroids of the edges or faces of a control volume [1].

Block structured AMR is a technique to add grid resolution efficiently and dynamically in areas of interest while leaving the rest of the domain at a coarser resolution. Effectively, grid resolution is added where and when it is needed in a simulation depending on a refinement criterion such as gradients in the data. AMR was originally applied to finite difference methods for inviscid shock hydrodynamics [2] and has been extended to inviscid, incompressible flow [3], and viscous flow [4,5] in rectangular domains.

The goal of the *EB-AMR* approach is to maintain the high-performance computing (HPC) advantages of Cartesian structured AMR methods while permitting a wider range of applications requiring complex boundary representation. Building on top of the Chombo package (`chombo.lbl.gov`), we have extended the Single Program Multiple Data (SPMD) parallelism model to include alteration of the finite-volume stencils in the region adjacent to an irregular boundary.

In this chapter, we describe the primary framework optimizations that enable scalable simulations (Sections 14.2 through 14.6). The application code Chombo-Crunch is presented as an example of production scale computing with the EB-AMR approach (Section 14.9).

14.2 DISTRIBUTED GEOMETRY

The regions of the domain requiring highly refined finite volumes will be changing over the course of the simulation. For regular structured AMR, this leads to the familiar tag-regrid-advance modeling loop. For the EB-AMR approach, this basic cycle needs to be augmented with geometric moments from the constructive solid geometry implicit function [6].

The alteration to control volumes that interact with the EB does not interact directly with the implicit function. Instead, the implicit function is first preprocessed into the EB index space object, referred to as the *EBIndexSpace*. The EBIndexSpace contains the preprocessed geometric moments of an implicit function restricted to the intersection with a Cartesian grid. The formalism for this construct is described in [6]. For any implicit function geometric representation and a specified minimal grid spacing, the EBIndexSpace is a preprocessing step. It can be done in line with the fluid simulation calculation, or it can be done off line as a preprocessing calculation and read in from a file. While the algorithm is automatic and robust, it is a nontrivial amount of computation and is too large to be replicated across processing elements. Hence, EBIndexSpace generation must be computed in parallel and stored in a distributed fashion.

14.2.1 EBIndexSpace Generation

EBIndexSpace generation takes the implicit function as input. The active simulation domain is defined as all the space where the implicit function is greater than zero. While the EBIndexSpace might be a larger distributed data structure, the implicit data itself can be extremely compact. We first cover the case where the implicit function has compact representation and can be replicated across all processing elements.

EBIndexSpace generation is parallelized with domain decomposition. We employ the simple *domainsplit* or the more involved *recursive bisection*:

- *Domainsplit* segments the Cartesian box global domain that encloses the simulation volume into equal-sized subdomains. Within each subdomain, a processing element executes the moment generation algorithm and stores on its local memory its portion of the EBIndexSpace. While domainsplit is simple to understand, explain, and implement, it only executes in a load-balanced fashion for problems that have equal distribution of geometric complexity throughout the domain.

  A number of applications fall into this category (e.g., subsurface flow and transport at the pore scale). The geometric information for the micron scale pore geometry demonstrates self-similarity and produces geometric information that evenly spreads throughout the domain. Other examples from both nature and engineering also have these properties (e.g., coral, battery electrodes, paper manufacturing).

- *Recursive bisection* is an alternative domain decomposition technique. Starting with base domain and minimal grid spacing, the implicit function is queried as to the amount of geometric material that is contained within that volume. If larger than a desired threshold, the domain is split in its longest dimension and each subdomain is then queried in a similar fashion. The recursive algorithm is repeated redundantly across each processing element until an acceptable threshold for geometric complexity within a subvolume is reached. The threshold is determined experimentally.

  This recursive bisection tiling of the simulation domain is then partitioned across the processing elements using a space-filling curve. The EBIndexSpace is computed in parallel as was done in the domainsplit algorithm. While having a higher upfront cost, the recursive bisection algorithm can result in a better distribution of the EBIndexSpace across the compute platform.

14.2.2 Local Geometry Caching

PDE simulations on a structure-adaptive mesh have their own mapping of processing elements to computational regions. An adaptive mesh computation with subcycling represents a different computational pattern than the computational geometry problem. This requires a separate parallelization strategy and load-balancing technique.

In the standard subcycled AMR compute pattern, the advance step advances the solution one time step at a level. If a regrid interval is reached, then a new collection of `Box`s called a `BoxLayout` are generated and then initialized. New stencils for the EB cut cells need to be generated. Although on a specific `Box`, there might be more than a dozen different stencils generated for any given finite volume based on the type of operator being discretized, they will all need to access the same underlying geometric moments.

`EBISLayout EBIndexSpace::fillEBISLayout(const ProblemDomain\& p, const BoxLayout\& b, int ghostCells)` is the Chombo operation that creates a local view of the EBIndexSpace for each specific patch. In this way, processing elements never need to know about the entire simulation process domain or about the EBIndexSpace. In this model, the pair `[p,b, ghostCells]` serves two roles: (1) indicates where geometric moments need to be communicated (what level of the refinement hierarchy and which patches are local to this processing element) and (2) provides a key into a caching map. `EBISLayout` is a refcounted object. In this way, we minimize how often the EBIndexSpace needs to be interrogated. The effects of local geometry caching for a high-fidelity incompressible viscous flow computation in a pore-scale geometry of packed spheres are shown in Table 14.1.

TABLE 14.1
Local Geometry Caching Effects

	Total Runtime (s)	Peak Memory Usage per Rank (MB)
Single-level, no caching	65	105
Single-level, caching	57	89
Adaptive mesh 2-levels, no caching	653	760
Adaptive mesh 2-levels, caching	348	413

14.3 FEEDBACK-BASED LOAD BALANCING

AMR modeling techniques pose a significant challenge to load balancing on distribute memory compute platforms. These algorithms have succeeded to the degree in which the compute load experienced by a processing element over a single patch of data is predictable. One can typically use the number of cells within a patch as a proxy for the expected compute load on a patch [7]. While not perfect, this approach has served well for AMR applications. Given a predictable compute load, various methods for distributing a load across processing elements like Knapsack and space-filling curve have been widely employed [8,9].

However, with the presence of the EB the number of cells on a patch is a poor proxy for compute load. Scaling tends to top out at 10X speedup. The first improvement to the cell count balancing is to apply a heuristic where irregular cells incur a larger penalty. This is meant to reflect the increased computation for irregular computation and the nonunit stride data access patterns. This heuristic model can become increasingly sophisticated, including the effects of cut cells crossing course-fine AMR interfaces, domain boundaries, and localized nonlinear physics. Though promising, this approach has not produced desired results. A performance model combining compile and run-time information with a tool like ExaSat [10,11] may illuminate better parameterizations than can be managed manually.

An alternative is to use a feedback-based load-balancing scheme. Each patch can construct a simple EB operator (e.g., Poisson). One step of a multigrid relaxation is computed and for each patch the actual compute time is measured. While the computational result is discarded, the compute time serves as the stand-in for the load-balance cost for that patch. This can be input into the existing load-balance algorithms, the operators are then reconstructed with the new distribution, and the simulation is allowed to advance (Table 14.2).

14.4 AGGREGATED STENCILS

The formalism described in [6] provides a powerful generalization of high-order finite volume methods for PDEs in arbitrary geometry. This expressive syntax is reflected in the EB-AMR class library in Chombo (EBChombo). While this powerful extraction layer is desirable in a framework that must target many application domains, it does come at a cost. The C++ abstractions for expressing arbitrary control volume discretizations imposes an unreasonable computational burden. While there are known techniques for faster execution of C++ abstractions, like template metaprogramming [12], these require knowledge of the stencil shape at compile-time. High-order EB stencils are produced from a least-squares algorithm at runtime based on the geometric moments and current mesh hierarchy. Therefore, EB stencil classes are passed through a run-time trace when they are bound to their target data structures. This process involves iterating through the abstract representation of a

TABLE 14.2
Different Load Balancers

	Total runtime (s)	Compute (%)/Commun. (%)
Standard, no ordering	41.60	45.83/54.17
Standard, Morton ordering	46.08	41.32/58.68
Feedback-based, no ordering	39.51	48.37/51.63
Feedback-based, Morton ordering	43.12	45.62/54.38

Note: Statistics are averaged over 10 iterations and five separate runs.

TABLE 14.3
Total Runtime Spent in AMR V-Cycle of Elliptic Solver: AggStencil versus EBStencil

	EBStencil (s)	AggStencil (s)	Speedup (s)
128 × 128 × 128 mesh			
Dual-socket *Ivy Bridge* node	3.00	1.99	1.50
Dual-socket *Haswell* node	2.22	1.62	1.37
KNL node (DDR)	5.71	3.54	1.61
KNL node (HBM)	5.27	2.88	1.83

Note: Runtime is averaged over eight iterations and five separate runs.

complex stencil one time and recording the final floating-point data deference locations and caching these offsets in the *AggStencil* (Table 14.3).

These AggStencil objects can be reused throughout multiple PDE operators across multiple time steps and only need to be reconstructed at the same time that the AMR level is regridded. The AggStencil represents a compromise between compilation efficiency and the need to have runtime data dependency integrated. AggStencil construction cannot be done as an offline processing step since it represents the combination of the preprocessed EBIndexSpace and the run-time processing of the AMR grid hierarchy.

14.5 SPARSE PLOT FILE

In the same way that the AggStencil object represents the object of a preprocessed geometry database and a runtime adaptive mesh hierarchy, the plot file has to capture this combination of static and dynamic data structures. In our initial implementation of plot file generation, the geometric moments were turned into self-centered state variables and output with the rest of the simulation data. We referred to this as the dense output. The problem with this approach is that for a relatively simple PDE like compressible Navier–Stokes, you might have five state variables (rho, X-momentum, Y-momentum, Z-momentum, and energy.) You can easily have twice that number of geometric moment information (e.g., volume fraction, area fractions, area centroids, and volume centroids) for even a second-order algorithm. For most cells, this data is trivial because they are not participating in a cut cell.

The benefit is that the visualization and data analytics pipeline in the visit analytics program could utilize moment geometric information for more complex postprocessing. In particular, the visualization postprocessor could reconstruct the geometric information and projections of state data onto the boundary manifold. The drawback for this approach is a tremendously large file that takes a long time to write and store and move. The alternative is to only write out nontrivial geometric moments. We call this the sparse format.

The sparse format creates an additional dataset next to the state data dataset in the HDF5 file that represents an unstructured graph embedded in the structured AMR grid hierarchy. This is very much like a vertex-edge data structure you find in traditional unstructured grid methods except we exploit the fact that the nodes in this graph are embedded in a Cartesian index space. Finite volume faces are the analog to edges. The edges will only pass along one cardinal access. Volumes correspond to vertices. The graph still must be written out to the data file in a parallel write operation into a collective HDF5 dataset and thus requires at least one invocation of a parallel prefix sum to compute unique processing element offsets.

In practice, there is not one clear choice between these two approaches. For geometrically dense problem definitions, the sparse file format has little savings in terms of final storage and it incurs the collective communication burden of creating the prefix sum. The two options are available in Chombo, and users can experiment to suit their application (Table 14.4).

14.6 COVERED BOX PRUNING

The runtime mesh generation algorithm is free to construct patches anywhere within the computational domain. It is oblivious to the geometric implicit function or the EBIndexSpace itself. After a regrid phase, it is not impossible to have patches that have fallen completely outside of the region of positive implicit function. While EBChombo will not expend computational work executing the finite volume method within these boxes and hence no payment of a runtime penalty, Chombo will still generate attendant data structures and storage for these patches as if the state variable extended into these regions.

It might seem like a rare case for this to happen, but by the quirk of how domain boundary conditions are specified in most legacy Chombo application examples, it is assumed that Level 0 extends throughout the entire extent of the Cartesian computational domain. In fact, the original Berger–Rigoutsos mesh segmentation algorithm takes that as an axiom. For applications that use shallow grid hierarchies, a significant amount of the computational finite volumes can fall outside of the intersection of the active domain and the Cartesian computational domain. A simple improvement is called pruning where boxes are visited prior to load-balancing, and if they have fallen completely outside of the active simulation domain, then they are removed from that level's patches (Table 14.5).

14.7 HYBRID MPI+OPENMP REFACTORING

Our previously published work for scalable EB calculations has relied upon a pure message-passing interface (MPI) 1.1 Application program interface (API) (e.g., [1,13]). The result of this reliance is that redundant meta-data used to optimize the communication phase of our calculations stressed the limited memory availability per processing element. On Knights Landing (KNL), there are 16 GB

TABLE 14.4
Plot File Size and Averaged Write Time to Lustre PFS: Dense Plot Format versus Sparse Plot Format

# Cores	File Size (GB)		Write time (s)	
	Dense	Sparse	Dense	Sparse
512	7.37	1.85	6.00	15.99
1,024	14.75	3.81	6.45	18.55
2,048	29.5	7.72	12.75	21.47
4,096	59	10.53	16.41	30.38
8,192	118	21.54	25.63	63.02
16,384	236	43.5	32.52	76.53
32,768	472	67.1	46.74	82.27

TABLE 14.5
Effect of Pruning of Covered Boxes on Plot File Size, Checkpoint File Size, and Total System Memory Usage

# Boxes	# Covered Boxes	Plotfile (GB)		Checkpoint (GB)		Memory (GB)	
		Original	Pruning	Original	Pruning	Original	Pruning
512	32 (6.25%)	7.37	6.91	6.01	5.63	216.5	210.1
4,096	512 (12.5%)	59	51.62	48.09	42.08	1,459	1,369
32,768	6,862 (20.9%)	472	373.8	384.75	304.7	11,211	9,699

of high bandwidth memory available, which is approximately five times less than traditional DRAM capacities on current X86 processors such as Ivy Bridge or Haswell. In order to take advantage of this high bandwidth memory, which is five times faster than traditional memory, and keep the same memory footprint, we have moved to a hybrid MPI+OpenMP model. The logical change to the software would be for compute elements to share common metadata. Our options are to adopt an MPI-3 remote memory access API, or a hybrid MPI+OpenMP programming model. For complex dynamic memory-intensive algorithms written in C++, it can be difficult to utilize the shared memory semantics of MPI-3. The metadata in Chombo AMR makes significant use of the operating system's free store mechanisms. To use MPI-3, we would need to replicate a heap manager within our user-space MPI windows.

We have chosen to use MPI+OpenMP. It is a programming model that is recommended from vendors and National Energy Research Scientific Computing Center (NERSC). In course-grained hybrid OpenMP, threads take the place of MPI ranks but maintain the SPMD programming model. While we don't anticipate this to result in speed improvements in the code, we observe significant improvements in the total memory usage in our simulations. As an example, we solve the incompressible Navier–Stokes flow equations. On Edison, the memory usage for flat MPI (24 ranks) is a Resident Set Size of 2,880 MB/node. With Hybrid MPI+OpenMP this total drops to 1,672 MB/node.

14.8 EVALUATED PLATFORMS

Edison is a Cray XC30 MPP at NERSC. Each node contains two 2.6 GHz 12-core Xeon Ivy Bridge chips each with four DDR3-1600 memory controllers and a 30 MB cache. Each core implements the four-way AVX SIMD instruction set and includes both a 32 KB L1 and a 256B L2 cache.

Cori Phase 1 is a Cray XC40 MPP at NERSC. Each node contains two 2.3 GHz 16-core Xeon Haswell chips and four DDR4-2133 memory controllers and 80 MB of L3 cache (Table 14.6) [14].

14.9 CHOMBO-CRUNCH: PRODUCTION SUPERCOMPUTING

Chombo-Crunch is a high-performance simulation code used to model pore scale reactive transport processes associated with subsurface problems including carbon sequestration. It is based on adaptive, finite volume methods developed in the Chombo framework [1] and, thus, relies heavily on the EB infrastructure for treatment of very complex geometries. In particular, the cut cell approach allows for explicit resolution of reactive surface area between mineral and pore. Reactions are treated with the geochemistry module of CrunchFlow [15], which performs point-by-point computations, and thus scales ideally with Chombo solvers. The code makes use of a novel interface between Chombo

TABLE 14.6
Overview of Evaluated Platforms

Core Architecture	**Intel Ivy Bridge**	**Intel Haswell**
Clock (GHz)	2.40	2.3
DP (GFlop/s)	19.2 (17.1)	36.8 (26.2)
Data cache (KB)	32+256	64+512
Chip Architecture	**Intel Xeon E5-2695v2**	**AMD Opteron 6172**
Cores	12	6
Last-level cache	30 MB	5 MB
DP (GFlop/s)	230.4	50.4
STREAM bandwidth	45 GB/s	12 GB/s
Memory capacity	32 GB	8 GB
System	**Cray XC30 (Edison)**	**Cray XC40 (Cori Phase 1)**
CPUs/node	2	4
Compiler	icc 14.0.0	icc 13.1.3

Source: Y. J. Lo et al., *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, Springer, 2014.

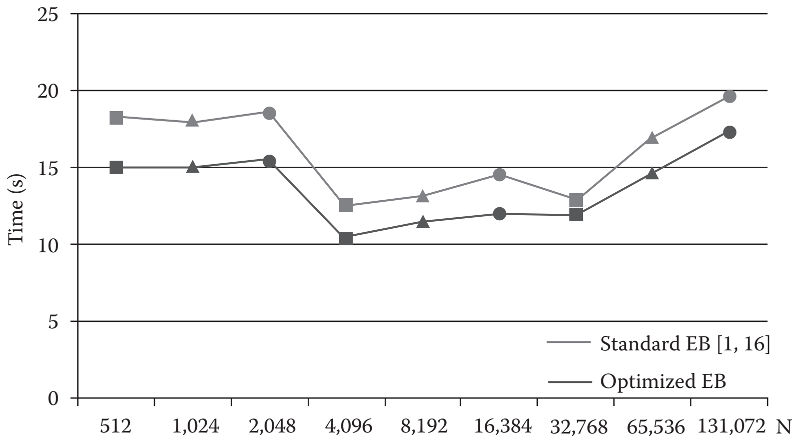
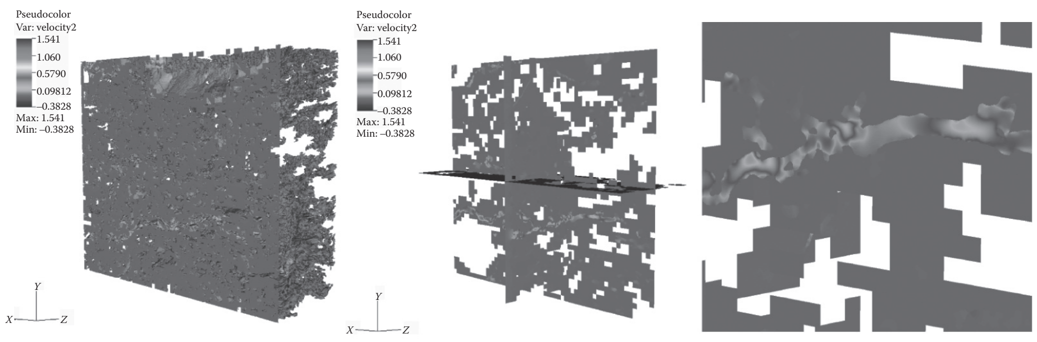


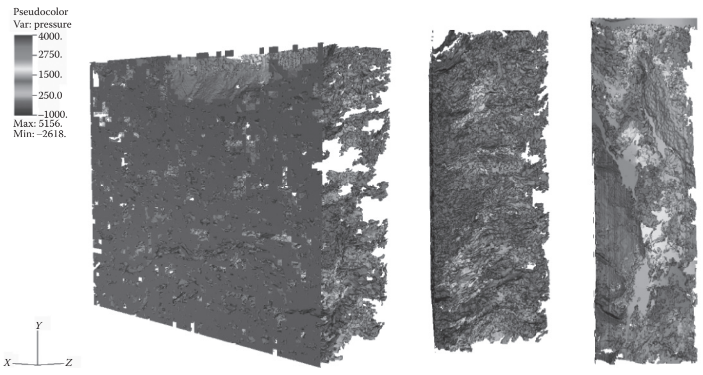
FIGURE 14.1 Weak scaling study of Chombo-Crunch on Cray XC30 NERSC Edison system.

and PETSc for access to algebraic multigrid solvers that do not possess the inherent shortcomings of geometric coarsening for very complex geometries [16]. Altogether, Chombo-Crunch is able to perform direct numerical simulation of real rock samples from image data at full-machine scale on DOE supercomputers. It has been validated by reactive transport experiments involving CO_2 injection [17,18].

Chombo-Crunch scales to full machine capability on NERSC supercomputers Edison [1], Cori Haswell, and now Cori KNL. The optimization work described in this chapter has improved that performance by 10%–15% and reduced the memory footprint of the application code by approximately 75% in order to fit into the high bandwidth memory on Cori KNL. Figure 14.1 shows the results of



(a)



(b)

FIGURE 14.2 (See color insert.) Direct numerical simulation from image data of fractured shale with pruning. (From D. Trebotich and D. Graves. *Comm. App. Math. Comp. Sci.*, 10, 43–82, 2015.) (a) Velocity data mapped to surface and data slice. Pruned boxes are shown as white space. (b) Pressure mapped onto the mineral surface with side views. The simulation has been performed on NERSC Edison using 40,000 cores.

weak scaling on Edison. Comparison of the performance of standard EB from [1] and [16] versus optimized EB is provided.

As an example of Chombo-Crunch production capability, we perform direct numerical simulation of flow in the fractured shale example in [1] using the aforementioned optimizations. The computational domain has been discretized using nearly 2 billion grid points ($1,920 \times 1,600 \times 640$) and a resolution of 48 nanometers. Figure 14.2 shows velocity and pressure plots. Pruning of covered boxes, shown as white space, has resulted in a reduction of at least one third of both the total boxes and computational cores in [1]. (The memory bandwidth sweet spot for Chombo-Crunch due to domain decomposition and load balancing is 32^3 grid cells per box, one box per core on current Intel architectures Ivy Bridge and Haswell.)

14.10 BURST BUFFER AND IN-TRANSIT DATA PROCESSING WORKFLOW

Emerging exascale supercomputers have the ability to accelerate the time-to-discovery for scientific workflows. However, the I/O systems have not kept the same pace as computational systems: technology trends show a growing gap in performance of computational systems versus I/O systems. Moving forward exascale systems and the anticipated increase in the size of scientific datasets, the I/O constraint will become more critical. To address this problem, advanced memory hierarchies, such as burst buffers, have been recently proposed as intermediate layers between the compute nodes and the parallel file system. Recently, the Cray DataWarp Burst Buffer [19] has been deployed on NERSC's Cori Phase 1 system. It consists of 144 I/O nodes with overall more than 900 TB of non-volatile RAM (NVRAM). Keeping data in burst buffer, close to a processing element, allows a simulation to accelerate the checkpoint/restart process. We utilize Cray DataWarp burst buffer coupled with in-transit processing mechanisms, to demonstrate the advantages of advanced memory hierarchies in preserving traditional coupled scientific workflows. Figure 14.3 shows the schematic of in-transit workflow [20], which couples simulation (Chombo-Crunch) with on-the-fly visualization (VisIt). The burst buffer is used as a file-based coordination mechanism between workflow components. Figure 14.4 shows the amount of I/O in total simulation-visualization workflow runtime. The comparison of total I/O time in two cases when the Lustre file system and DataWarp burst buffer are used to store and exchange data between workflow components. Results are provided for different I/O intensities. As seen in Figure 14.4, the burst buffer provides a definite I/O improvement to the Chombo-Crunch+VisIt workflow at high-I/O intensity when a plotfile is written and processed

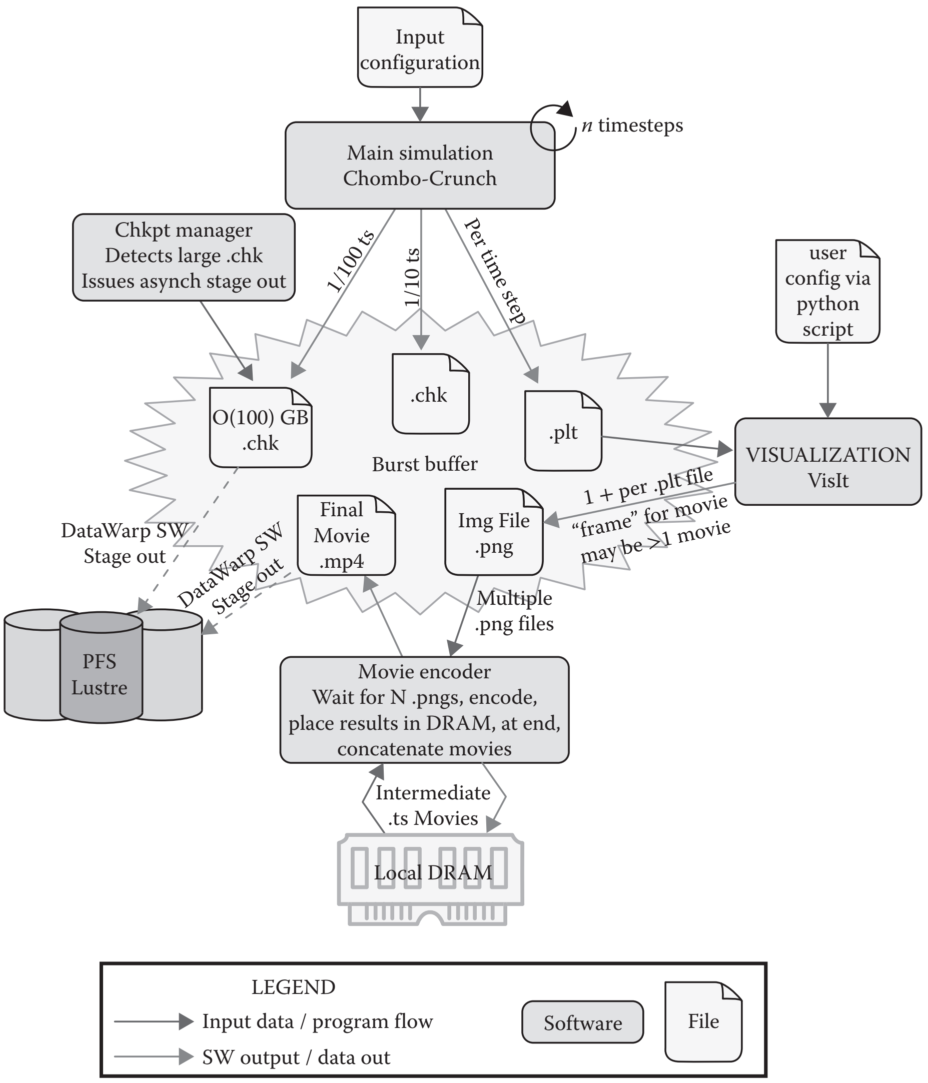


FIGURE 14.3 Chombo-Crunch and VisIt integrated burst buffer workflow diagram.

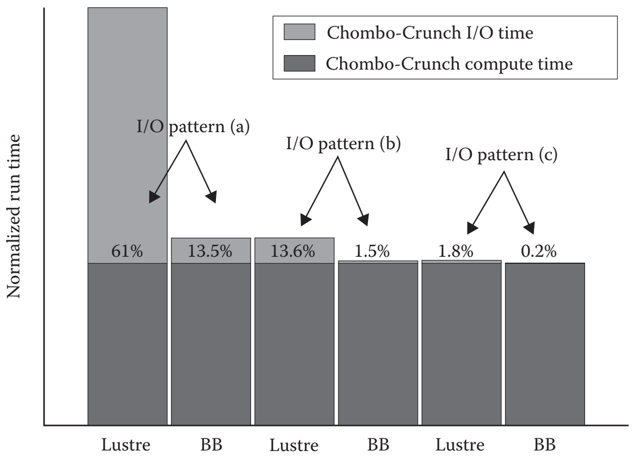


FIGURE 14.4 Summary on compute and I/O time for in-transit simulation-visualization workflow for different I/O intensities.

at every time step: 61% of I/O time for Lustre file system versus 13.5% of I/O time for for burst buffer; and medium I/O intensity when plot file is written and processed every 10 timesteps: 13.6% for Lustre file system and 1.5% of I/O time for burst buffer.

14.11 CONCLUSIONS AND FUTURE WORK

The combination of AMR and EB cut cell methods provides a powerful tool for multiscale, multiphysics simulation of PDEs in complex geometries. Mesh generation from arbitrary image data is not only tractable but even flexible in this approach. In order to scale to larger simulation domains and experimental time scales, however, we will need to alter our underlying software to accommodate new architectures on the path to exascale computing. A number of optimizations have been discussed in this chapter toward this end of distributed memory computing. Optimal mixture of these optimizations will be platform-dependent and will require further codesign with computer center experts and vendor support. We will be able to make use of a number of optimizations and features of these new machines. For example, off-chip NVRAM can be used for multicode coupling and more advanced workflows.

ACKNOWLEDGMENTS

This chapter is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, and in part by the Office of Basic Energy Sciences Energy Frontier Research Centers, and used resources of the National Energy Research Scientific Computing Center, all under contract number DE-AC02-05CH11231.

REFERENCES

1. D. Trebotich and D. Graves. An adaptive finite volume method for the incompressible Navier–Stokes equations in complex geometries. *Communications in Applied Mathematics and Computational Science*, 10(1):43–82, 2015.

2. M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64–84, 23 May 1989.
3. D. Martin and P. Colella. A cell-centered adaptive projection method for the incompressible Euler equations. *Journal of Computational Physics*, 163(2):271–312, 2000.
4. A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. J. Welcome. A conservative adaptive projection method for the variable density incompressible Navier-Stokes equations. *Journal of Computational Physics*, 142(1):1–46, May 1998.
5. D. F. 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.
6. P. Schwartz, J. Percelay, T. J. Ligocki, H. Johansen, D. T. Graves, D. Devendran, P. Colella, and E. Ateljevich. High-accuracy embedded boundary grid generation using the divergence theorem. *Communications in Applied Mathematics and Computational Science*, 10(1):83–96, 2015.
7. B. Van Straalen, P. Colella, D. Graves, and N. Keen. Petascale block-structured AMR applications without distributed meta-data. *Euro-Par 2011 Parallel Processing*, pp. 377–386, Bordeaux, France: Springer, 2011.
8. A. Dubey and B. Van Straalen. Experiences from software engineering of large scale AMR multiphysics code frameworks. *arXiv preprint arXiv:1309.1781*, 2013.
9. A. Dubey, A. Almgren, J. Bell, M. Berzins, S. Brandt, G. Bryan, P. Colella, D. Graves, M. Lijewski, F. Löffler et al. A survey of high level frameworks in block-structured adaptive mesh refinement packages. *Journal of Parallel and Distributed Computing*, 74(12):3217–3227, 2014.
10. C. Chan, D. Unat, M. Lijewski, W. Zhang, J. Bell, and J. Shalf. Software design space exploration for exascale combustion co-design. In J. M Kunkel, T. Ludwig, and H. W. Meuer (eds.), *Supercomputing*, pp. 196–212. New York, NY: Springer, 2013.
11. D. Unat, C. Chan,W. Zhang, S. Williams, J. Bachan, J. Bell, and J. Shalf. Exasat: An exascale co-design tool for performance modeling. *International Journal of High Performance Computing Applications*, 29(2):209–232, 2015.
12. S. W Haney, P. F. Dubois. Beating the abstraction penalty in c++ using expression templates. *Computers in Physics*, 10(6):552–557, 1996.
13. D. Trebotich, B.Van Straalen, D. T. Graves, and P. Colella. Performance of embedded boundary methods for CFD with complex geometry. *Journal of Physics: Conference Series*, 125:012083, 2008.
14. Y. J. Lo, S. Williams, B. Van Straalen, T. J. Ligocki, M. J. Cordery, N. J. Wright, M. W. Hall, and L. Oliker. Roofline model toolkit: A practical tool for architectural and program analysis. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, pp. 129–148. Springer, 2014.
15. C. I. Steefel. New directions in hydrogeochemical transport modeling: Incorporating multiple kinetic and equilibrium reaction pathways. In L.R. Bentley, J. F. Sykes, C. A. Brebbia, W. G. Gray, and G.F. Pinder (eds.), *Computational Methods in Water Resources XIII*, Rotterdam: A. A. Balkema, 2000.
16. D. Trebotich, M. F. Adams, S. Molins, C. I. Steefel, and C. Shen. High-resolution simulation of pore-scale reactive transport processes associated with carbon sequestration. *Computing in Science and Engineering*, 16(6):22–31, 2014.
17. S. Molins, D. Trebotich, C. I. Steefel, and C. Shen. An investigation of the effect of pore scale flow on average geochemical reaction rates using direct numerical simulation. *Water Resources Research*, 48:W03527, 2012.
18. S. Molins, D. Trebotich, L. Yang, J. B. Ajo-Franklin, T. J. Ligocki, C. Shen, and C. I. Steefel. Pore-scale controls on calcite dissolution rates from flow-through laboratory and numerical experiments. *Environmental Science and Technology*, 48(13):7453–7460, 2014. PMID: 24865463.
19. W. Bhimji, D. Bard, D. Paul, M. Romanus, A. Ovsyannikov, B. Friesen, M. Bryson, J. Correa, G. Lockwood, V. Tsulaia et al. Accelerating science with the NERSC burst buffer early user program. *Annual Cray User Group Meeting*, 2016.
20. A. Ovsyannikov, M. Romanus, B. Van Straalen, G. Weber, and D. Trebotich. Scientific Workflows at DataWarp-Speed: Accelerated Data-Intensive Science Using NERSC's Burst Buffer. *PDSW-DISCS*, to appear, 2016.