SAND2013-7595C

An AMR Task Dependency Analysis and Communications Simulation Methodology

Cy P Chan[1], Joseph P Kenny[2], Gilbert Hendry[2], Vincent E Beckner[1], John B Bell[1], and John M Shalf[1]

[1]Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, Berkeley, CA

[2]Sandia National Laboratory*, P.O. Box 969, Livermore, CA

ABSTRACT

The ability to predict the performance of irregular, asynchronous applications on future hardware is essential to the exascale co-design process. AMR applications are inherently irregular in their computation and communication patterns, resulting in complex relationships between problem parameters, machine configuration, data distribution, and application performance. We have developed a methodology to simulate the performance impact of different data distributions on a variety of network topologies for asynchronous execution of AMR applications. We demonstrate this framework by simulating CASTRO, a compressible astrophysics application, indicating that a performance improvement of up to 20 percent may be obtained through the use of locality-aware data distributions for some network topologies on an exascale-class supercomputer.

1. INTRODUCTION

The trend in high performance computing toward massive parallelism presents many challenges for irregular scientific applications, not least of which is the network's ability to support an increasing amount of data movement. Co-design has emerged as a promising technique to optimize software and hardware together to allow users to achieve high performance on future architectures [14]. However, in order to co-design effectively, we must have tools to quickly predict the performance of our applications on the potential hardware configurations under consideration.

Data movement has emerged as one of the most important factors influencing performance on today's machines, and it will become even more important on exascale machines due to the relative performance and energy scaling of processor and memory technology [8, 16, 13, 2]. In the context of adaptive mesh refinement (AMR) applications, data movement is largely determined by the machine's network topology and the distribution algorithm used to assign boxes (or grids) of data to machine locations. Furthermore, the desire to hide the cost of data movement and improve parallel efficiency has spurred the development of asynchronous runtimes, which replace the traditional alternating phases of computation and communication with a data-driven execution style, making performance modeling even more difficult.

Given the complex time-dependent interactions of applications, runtime, and hardware, structural simulation is often employed as the only means to evaluate hypothetical system-wide performance. SST/macro [6, 3] is one tool that enables this analysis by using efficient, validated, coarse-grained models and running application code in an online execution-driven environment. Typically, simulation efficiency is enhanced by a process known as *skeletonization* [17], which reduces application code to the parts of interest, namely the code necessary to reproduce communication characteristics. However, for irregular asynchronous applications like AMR, this is virtually impossible.

In order to achieve highly efficient structural simulation while maintaining accurate, detailed application behavior, we have developed an AMR dependency analysis tool that takes an AMR box list from the BoxLib [5] library as input and generates two outputs. The first is a graph that illustrates the evolution of data (problem state) along with all of the necessary computation and communication events that must occur during program execution, as well as their dependencies. The second is an XML file that can be used to drive an asynchronous execution using SST/macro to simulate network traffic over a configurable network. Our framework allows us to test configurations for which it would be infeasible to collect traces from real machines. Furthermore, compared to MPI traces, our framework allows more flexibility in examining the fine-grained communications that occur during execution because traces gathered from real computations utilize message aggregation, making it difficult to analyze the effects of data distribution.

Kerbyson et. al. [7] developed a performance model for an AMR code using a bulk-synchronous execution model over a parameterized network, but it would be unable to give performance predictions for future machines with features that

*Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

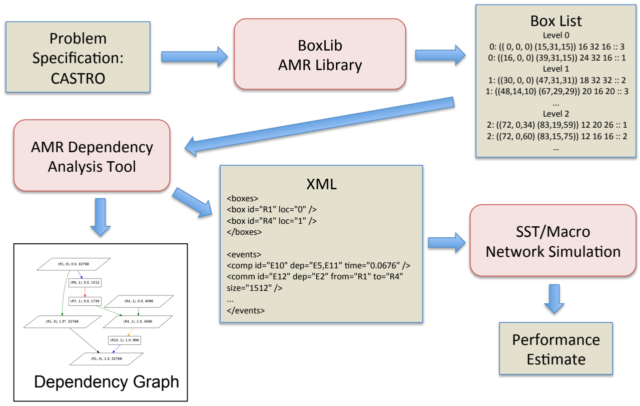

Figure 1: Workflow

exhibit complex runtime interactions, such as asynchronous execution or congestion-adaptive routing. Previous evaluation of hierarchical data distribution techniques on existing machines include various studies of static and dynamic load balancing [15, 11, 9, 10, 12]. Our work improves upon these works by providing an automated framework and methodology that spans from problem specification to dependency graph generation to network simulation, helping us to explore and evaluate our algorithms and architectures for co-design.

2. FRAMEWORK DESCRIPTION

Our methodology relies on an analysis framework that includes several components: the BoxLib library for problem specification and box distribution, an AMR dependency analysis tool, and the SST/macro network simulation tool.

Figure 1 shows the components in our workflow. A problem specification including initial state and boundary conditions is specified within the BoxLib [5] framework, as an application code normally would. BoxLib can then generate a list of boxes that covers the interesting areas of the domain following some parameters supplied by the user such as maximum and minimum box sizes and covering efficiency. The box distribution can be specified by one of the algorithms in BoxLib or optionally configured later in the tool chain. This list of boxes is then parsed by our AMR dependency analysis tool, which produces a dependency graph and an XML description of computation and communication. This XML description is then read by SST/macro, which reproduces the behavior of each simulated process running on a configurable network.

2.1 BoxLib

BoxLib is a hybrid C++/Fortran90 software framework that provides support for the development of parallel block-structured AMR applications. The fundamental parallel abstraction is the MultiFab, which holds data on the union of a set of boxes for each grid level. We utilized BoxLib to create box lists for the CASTRO application, which involved setting up the initial and boundary conditions of the simulation, tagging cells of interest, and covering the cells with boxes at the various hierarchy levels. The boxes are distributed among processes using one of the available box distribution schemes, which include round-robin (RR), knapsack (KS) and space-filling-curve (SFC). The round-robin and knapsack algorithms balance computational workload among the processes while the space-filling-curve algorithm assigns boxes that are near each other in space to adjacent processes in the machine in order to minimize data movement. Our study examines the effects of these distribution algorithms on application performance.

Listing 1: Example Box List Input File

```
Level 0    4 grids   40960 cells   100 % of domain
 0: (( 0,  0,  0) (15,31,15)) 16 32 16 :: 3
 0: ((16,  0,  0) (39,31,15)) 24 32 16 :: 1
 ...
Level 1    12 grids   146368 cells   44.668 % of domain
 1: ((30,  0,  0) (47,31,31)) 18 32 32 :: 2
 1: ((48,14,10) (67,29,29)) 20 16 20 :: 3
 ...
Level 2    78 grids   403440 cells   15.39 % of domain
 2: ((72,  0,34) (83,19,59)) 12 20 26 :: 1
 2: ((72,  0,60) (83,15,75)) 12 16 16 :: 2
 ...
```

Listing 1 shows an excerpt from an example box list output by BoxLib. This file specifies a three level AMR hierarchy where each box has a line that specifies its level, start and end points, dimensions, and process assignment.

2.2 AMR Dependency Analysis Tool

The AMR Dependency Analysis Tool takes the hierarchical list of boxes produced by BoxLib as input and generates an internal representation of the box hierarchy and two analysis outputs: a dependency graph and an XML file containing events and communications.

The internal representation generated by our tool from the box list is a hierarchical graph, where the nodes represent boxes and edges are drawn between pairs of boxes that interact during the course of the AMR computation. In order to determine the edge locations and properties, we built an algebraic box set library of parameterized dimension to compute interactions between logical regions of the domain space. Regions of space are represented by unions of disjoint rectahedral boxes encapsulated within BoxSet objects. The main operations on BoxSet objects supported by the library include intersections, unions, inversions, and set differences. Utility functions include extending the boundaries of a BoxSet by a specified number of cells, as well as constructing the ghost halo region around a BoxSet. These utility operations facilitate the identification of the interactions that occur during an AMR computation.

Listing 2: Two-level, Two-box Example

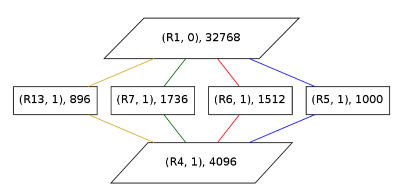

Figure 2: Compact representation of a two-box AMR hierarchy

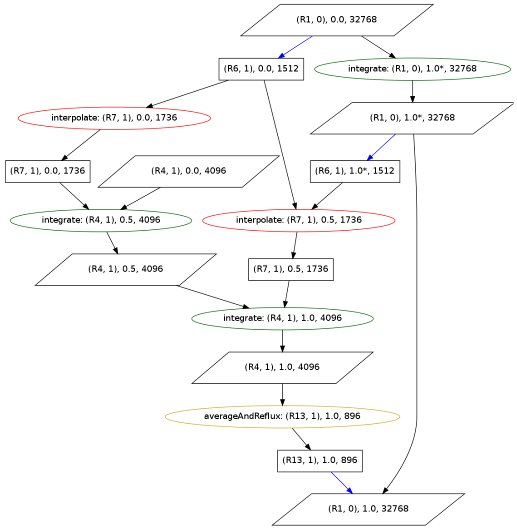

Figure 3: Example dependency graph output corresponding to the two-box AMR hierarchy for one time step with a refinement ratio of two. This dependency graph corresponds to an "unfurled" version of the internal representation shown in Figure 2.

```
Level 0   1 grids
 0: (( 0, 0, 0) (31,31,31)) 32 32 32 :: 0
Level 1   1 grids
 1: ((24,24,24) (39,39,39)) 16 16 16 :: 1
```

The internal graph representation of the AMR hierarchy is compact in the sense that it is agnostic to the refinement ratio and number of simulation time steps. Figure 2 depicts the internal representation of a two-box hierarchy corresponding to the input file in Listing 2. The parallelograms represent the boxes in the input file, while the rectangles represent intermediate regions where the boxes interact for operations such as ghost region interpolation, averaging, and refluxing. The different colors represent the different types of interactions that occur between the boxes. The compact representation can be "unfurled" to generate full dependency graphs for an AMR computation with arbitrary refinement ratio and number of time steps.

2.2.1 *Dependency Graph Output*

Figure 3 shows the output dependency graph of our tool run on the two-box compact graph representation shown in Figure 2, which represents an execution of the AMR simulation code for one coarse time step. The dependency graph output includes nodes for data, computations, and communications that occur during the execution of the simulation, and edges represent dependencies between the nodes. The nodes are annotated with metadata, such as size and physical location information.

Each box and intermediate region may appear multiple times in the output graph because their contents change as the simulation progresses. Thus, the nodes are also annotated with time stamps that correspond to the simulation time for which the data is valid. Note that communications are implicit in the figure, corresponding to the blue colored edges between data nodes. Depending on the context, it may be worthwhile to unfurl the dependency graph on an *as needed* basis to help reduce the program's memory footprint during execution.

2.2.2 *Dependency XML Output*

The dependency XML is a stripped version of the dependency graph that contains a list of boxes and a list of computation and communication events. The XML does not contain any direct information about the boxes themselves (such as their size or spatial extent), nor is there any notion of box hierarchy. It does however, specify a list of abstract *regions* within which computations may occur and between which communications may occur. The region and event information are sufficient to drive the next phase of the analysis: the SST/macro network event simulation tool.

Listing 3: Example XML Output

```xml
<boxes>
<box id="R1" loc="0" />
<box id="R4" loc="1" />
</boxes>
<events>
<comp id="E10" dep="E5,E11" type="integrate" at="R4"
     size="4096" time="0.0676" />
<comp id="E11" dep="E12,E8" type="interpolate" at="R4
     " size="1736" time="0.001" />
<comm id="E12" dep="E2" type="copy" from="R1" to="R4"
      size="1512" />
...
</events>
```

Listing 3 shows an example XML output containing two sections. The first section consists of a list of boxes as well as their initial process assignments, which may be modified at simulation time to explore alternative distribution strategies. The second section contains a list of computation and communication events. Each event is annotated with the type of the event, its location (or source and destination), the size of the data that needs to be processed or communicated, and (if the event is a computation) an estimate of the time to execute. The computation times may be generated by using a performance model such as the ExaSAT static analysis model [4]. This allows us to have a completely parameterized system model that captures both on-node and off-node performance. The compute times may also be estimated by other means, such as using performance profiles on current machines or a discrete event simulator.

2.3 SST/macro

SST/macro [6, 3] is an open-source coarse-grained simulator for large parallel high-performance applications and machines that enables the exploration of current and future implementations of applications, libraries, and runtimes on performance models of typical supercomputer hardware. Typically, interfaces such as MPI are implemented in the simulator, effectively providing an on-line emulation environment for applications which can execute natively on the simulation host. SST/macro has been validated against existing HPC hardware [18] using formal Uncertainty Quantification techniques, demonstrating the effectiveness of coarse-grain modeling in efficiently capturing performance characteristics. In this work, we leverage SST/macro to provide network simulation capability to our analysis toolchain. An

Box Distribution Algorithm	Network Topology
Round-Robin (RR)	3D Torus
Knapsack (KS)	7D Torus
Space-Filling-Curve (SFC)	4-ary Fat-Tree
	16-ary Fat-Tree

Table 1: Experimental parameters explored

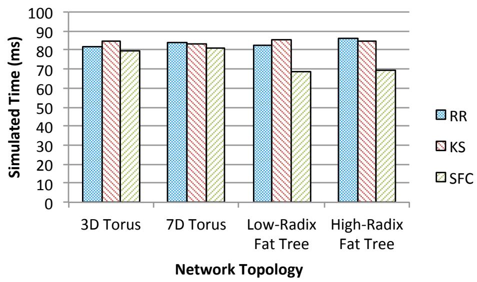

Figure 4: Simulated execution time for different network topologies and box distribution algorithms. RR = round-robin, KS = knapsack, SFC = space-filling-curve.

XML parsing application was written to be run in SST/-macro which interprets the computation and communication tasks output by the AMR dependency tool, and makes MPI calls accordingly.

3. RESULTS AND FUTURE WORK

To demonstrate our framework, we use the CASTRO AMR application code [1, 19, 20]. CASTRO is a radiation hydrodynamics code developed for computational astrophysics. The code uses compressible Eulerian hydrodynamics with self-gravity and multigroup flux-limited radiation diffusion. For the experiments considered here we only use the hydrodynamics component of the code. The discretization of the hydrodynamics uses an unsplit PPM integrator, which is an explicit finite volume algorithm that builds in nonlinear wave interactions. CASTRO uses a block-structured AMR paradigm in which points requiring additional resolution are grouped into larger aggregate regions that are represented as a union of logically rectangular patches of data. For computational efficiency and accuracy, the algorithm subcycles in time so that regions that are refined in space are also refined in time, with a synchronization step that occurs when adjacent levels reach the same physical time.

We have conducted some preliminary experiments using a variety of box distribution algorithms and network topologies for a fixed problem size of 967 boxes split over 3 AMR levels and distributed over 480 processes. Some of the major parameters explored are displayed in Table 1. Network parameters (i.e. bandwidths and latencies) were chosen to reflect estimates of exascale-class machine network capabilities. Computation event time estimates were generated by profiling the execution of the CASTRO code on NERSC's Hopper supercomputer and using a regression model to estimate the on-node performance on an exascale-class machine. Future work will incorporate using the ExaSAT analysis tool [4] to make more detailed estimates of the CASTRO application's compute performance.

Figure 4 shows the results of our simulations. The distribution algorithms perform similarly except for the space-filling-curve (SFC) algorithm, which performs roughly 18 percent faster than the next best algorithm on the fat tree topologies. This advantage is likely due to the SFC algorithm exploiting locality between boxes, effectively reducing network traffic, while the other algorithms focus exclusively on computational load balance.

Our framework allows users to evaluate the performance of various AMR codes on potential network configurations without requiring physical access for benchmarking, which is a valuable capability for software/hardware co-design, especially in advance of hardware arrival. We plan to utilize our simulation methodology to help evaluate many other box distribution algorithms and network topologies. We may also apply our framework to the evaluation of dynamic load balancing schemes coupled with more detailed on-node performance estimation tools.

4. REFERENCES

[1] A. S. Almgren et al. CASTRO: A New Compressible Astrophysical Solver. I. Hydrodynamics and Self-gravity. *Astrophysical Journal*, 715:1221–1238, June 2010.

[2] S. Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23–29, 1999.

[3] C. L. Janssen et al. Using simulation to design extremescale applications and architectures: programming model exploration. *SIGMETRICS Perform. Eval. Rev.*, 38:4–8, March 2011.

[4] C. Chan, D. Unat, M. Lijewski, W. Zhang, J. Bell, and J. Shalf. Software design space exploration for exascale combustion co-design. In *ISC*, pages 196–212, 2013.

[5] C. for Computational Sciences and E. L. B. N. Laboratory. Boxlib. `https://ccse.lbl.gov/BoxLib/`.

[6] C. L. Janssen et al. A simulator for large-scale parallel computer architectures. *IJDST*, 1(2):57–73, 2010.

[7] D. J. Kerbyson et al. Predictive performance and scalability modeling of a large-scale application. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pages 37–37, New York, NY, USA, 2001. ACM.

[8] P. Kogge et al. Exascale computing study: Technology challenges in achieving exascale systems, 2008.

[9] Z. Lan, V. Taylor, and G. Bryan. Dynamic load balancing for structured adaptive mesh refinement applications. In *International Conference on Parallel Processing.*, pages 571–579, 2001.

[10] X. Li and M. Parashar. Hierarchical partitioning techniques for structured adaptive mesh refinement applications. *The Journal of Supercomputing*, 28(3):265–278, 2004.

[11] X.-Y. Li and S.-H. Teng. Dynamic load balancing for parallel adaptive mesh refinement. In A. Ferreira, J. Rolim, H. Simon, and S.-H. Teng, editors, *Solving Irregularly Structured Problems in Parallel*, volume 1457 of *Lecture Notes in Computer Science*, pages 144–155. Springer Berlin Heidelberg, 1998.

[12] J. Luitjens and M. Berzins. Improving the performance of uintah: A large-scale adaptive meshing computational framework. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–10, 2010.

[13] D. A. B. Miller. Rationale and challenges for optical interconnects to electronic chips. In *Proceedings of the IEEE*, pages 728–749, 2000.

[14] M. Mohiyuddin et al. A design methodology for domain-optimized power-efficient supercomputing. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 12:1–12:12, New York, NY, USA, 2009. ACM.

[15] M. Parashar and J. Browne. On partitioning dynamic adaptive grid hierarchies. In *System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on* ,, volume 1, pages 604–613 vol.1, 1996.

[16] J. Shalf, S. S. Dosanjh, and J. Morrison. Exascale computing technology challenges. In *VECPAR*, volume 6449 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2010.

[17] M. Sottile, A. Dakshinamurthy, G. Hendry, and D. Dechev. Semi-automatic extraction of software skeletons for benchmarking large-scale parallel applications. In *ACM SIGSIM PADS*, May 2013.

[18] J. J. Wilke et al. Validation and uncertainty assessment of extreme-scale hpc simulation through bayesian inference. In *Proceedings of EuroPar*, August 2013.

[19] W. Zhang, L. Howell, A. Almgren, A. Burrows, and J. Bell. CASTRO: A New Compressible Astrophysical Solver. II. Gray Radiation Hydrodynamics. *Astrophysical Journal Supplement*, 196:20, Oct. 2011.

[20] W. Zhang, L. Howell, A. Almgren, A. Burrows, J. Dolence, and J. Bell. CASTRO: A New Compressible Astrophysical Solver. III. Multigroup Radiation Hydrodynamics. *Astrophysical Journal Supplement*, 204:97, 2013.