SAND2017-9583C

Evaluating Production Load Balancing Functions for Adaptive Mesh Schemes using Mini-Applications

Courtenay T. Vaughan and Simon D. Hammond

Application Performance Team
Center for Computing Research
Sandia National Laboratories
Albuquerque
New Mexico, USA

Email: {ctvaugh, sdhammo}@sandia.gov

***Abstract*—Finite difference and finite volume based application codes can be used to explore a broad range of physical phenomena in science and engineering. Many of these codes incorporate Adaptive Mesh Refinement (AMR) to steer computation to the critical parts of a physical simulation, enabling increased numerical accuracy of the solution while reducing memory consumption. However, mesh adaptivity comes at the cost of increased runtime and code complexity. In order to explore the design space offered by new computing environments, we have developed a mini-application called miniAMR. MiniAMR's implementation of adaptive mesh refinment is representative of that in CTH, a heavily used and trusted production shock hydrodynamics code. One issue we see with CTH is an imbalance between the computational weight of the blocks during complex simulation. In this paper, we modify miniAMR to have computational imbalances and introduce a new weight-based load balancing scheme. To evaluate the efficiency of the new balancer across computing architectures, we benchmark miniAMR on five generations of high-performance multi- and many-core processor.**

***Index Terms*—High performance computing; adaptive mesh refinement; scientific applications; parallel architectures; performance evaluation.**

I. Introduction

High-performance computational-engineering is littered with a long history of finite difference and finite volume methods being used to study complex physical phenomena. In its simplest form, the algorithms are applied to a static grid. The resolution of the grid must be sufficiently fine to capture the phenomena being studied anywhere in the domain. For complex problems which require extremely detailed analysis, the fineness of the mesh is so demanding that the memory requirements of a universally equivalent level of refinement become too high.

As illustrated in Figure 1, incorporating Adaptive Mesh Refinement (AMR [5], [6]) instead allows for only parts of the mesh to be highly refined, while other sections can remain much coarser. The effect is a focus of attention on the most critical regions of a simulated mesh, enabling increased numerical accuracy of the solution while significantly reducing memory requirements. Adaptivity, however, comes at the cost of increased application complexity, and runtime behavior becomes strongly tied to the particular problem or geometries being examined. As future machines become are expected to become larger into the Exascale-era, and the potential of the systems to execute larger problems, the topic of how best to employ refinement, and balance the appropriate dynamically changing computation is of particular interest.

In order to explore the design space of adaptive mesh-based applications, we have developed a proxy application called miniAMR. MiniAMR is a standalone code designed for the exploration of some important performance issues prevalent in finite difference or volume codes that utilize adaptive meshes. It is free of constraints of any specific physical behavior, with refinement driven by potentially multiple objects moving through the domain, and is purposefully not intended to be representative of the computation found in a complex scientific application. This provides a simplistic setting in which to quickly experiment and analyze the fundamental components of adaptive-mesh algorithms.

In this paper, we specifically focus on an investigation of load balancing cost functions that can be used in CTH – the parent shock hydro-dynamics code, on which miniAMR

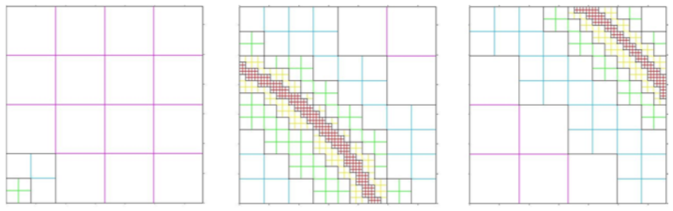
Fig. 1. Adapting Mesh as Shock Front Passes Through a Domain

is based. Our first investigate uses a very cheap method to balance refined blocks across MPI ranks based on ensuring the number of blocks per MPI rank is approximately equal. We compare this to a more sophisticated function which attempts to balance blocks across the MPI domain using a cost function associated with each block's expected computational work. As a basis for our benchmarking, we utilize five generations of processor used in our high-performance computing deployments.

The contributions of this work are the following:

- **Breakdown of Functions in Adaptive Mesh-based Applications** - we utilize the miniAMR application to analyze the runtime contribution of the various functions associated with refinement and load-balancing in adaptive-mesh applications;
- **Comparison of Block-Count-based and Expected Computational Work-based Load Balancing Schemes** - we compare two approaches to load-balancing with varying overheads and effect on runtime performance. We conclude that the work-based balancing (as opposed to block-count-based) improves the performance of some parts of the application but degrades performance in others such that its use in the parent CTH hydro-code is unlikely to provide substantial benefit. In so doing, we demonstrate a real-world practical use of a mini-application in evaluating design choices for complex scientific codes;
- **Benchmarking of Multiple Generations of High-Performance Multi-/Many-Core Processors** - we compare the runtime performance of our miniAMR, adaptive-mesh mini-application on five generations of processor showing the gains in performance that recent hardware generations have provided.

The remainder of this work is laid out as follows: we start the paper with a brief evaluation of related work in Section I-A. A short description of CTH, the parent shock hydro-code of miniAMR is presented in Section II. Section III provides an overview of miniAMR. We present a case study of using miniAMR to evaluate load-balancing strategies in Section IV. Finally, we conclude the paper in Section V.

A. Related work

AMR was first introduced by Berger et.al. [5] [6]. Since then, several libraries have been developed including BoxLib [3], Chombo [7], SAMRAI [15], and others, to allow code developers to more easily develop adaptive mesh applications.

MiniAMR is available from the Mantevo project suite of mini-applications [13] – a suite of open-source implementations that span important classes of algorithms in high-performance computing. The project has recently transitioned to the Github source code hosting platform to enable a much broader level of community interaction.

CloverLeaf [10], [12] is a Mantevo miniapp that includes meaningful physics, solves the compressible Euler equations on a two dimensional Cartesian grid, using an explicit, second-order accurate method. CleverLeaf [2] adds a patch based AMR scheme to CloverLeaf using the SAMRAI toolkit.

Performance of SAGE AMR hydro-code from Los Alamos is modeled in [16] reflecting the importance of adaptive mesh schemes across a variety of domains and leading HPC centers.

MiniAMR is, in many ways, simpler than existing benchmarks, in keeping with the Mantevo goal of providing a tractable means for modifying the code, including swapping in new communication, refinement, and load-balancing strategies. The work presented in this paper is an example of its use in a practical setting.

II. The CTH Shock-Hydrodynamics Code

The application that lead to the development of miniAMR is CTH [17], [14], [1]. The CTH application is a multi-material, large deformation, strong shock wave, solid mechanics code developed and maintained at Sandia National Laboratories. CTH has models for multi-phase, elastic viscoplastic, porous and explosive materials, using second-order accurate numerical methods to reduce dispersion and dissipation and produce accurate, efficient results.

MiniAMR was developed to study CTH when it is run in adaptive-mesh mode. The AMR scheme employed by CTH utilizes an octree based scheme, where each processor has a number of small blocks, each of which comprises a few hundreds cells. As the calculation progresses, the number and placement of these blocks in the calculation can change. Each blocks has to communicate with its neighboring blocks in the mesh, so each MPI rank in the computation ends up performing communication within the rank (to neighboring blocks on the same rank) as well as to some number of neighboring ranks, which can evolve as the simulation progresses.

In [19], we compared miniAMR to CTH and showed that the communication for exchanging block boundary information is similar between CTH and miniAMR. This includes the length and number of messages in addition to the number of communication partners that each rank has. These communication patterns are extremely similar and the minor differences explained. The communication patterns for the refinement step showed differences which were attributable to the differing way the refinement is performed, but that is a small portion of the total run time. Futhermore, the proportion of time spent in computation and communication are similar between the codes.

For the purposes of this paper, we continue to use the *"sphere hits block"* problem, illustrated in Figure 2. In this problem we perform a hydro-dynamics calculation using two materials and a boundary exchange of 32 variables. The figure is colored by pressure, with red being areas of high pressure and green being areas of low pressure. While this problem is heavily simplified from the production use of CTH, its execution utilizes many of the routines which are important for our investigation.

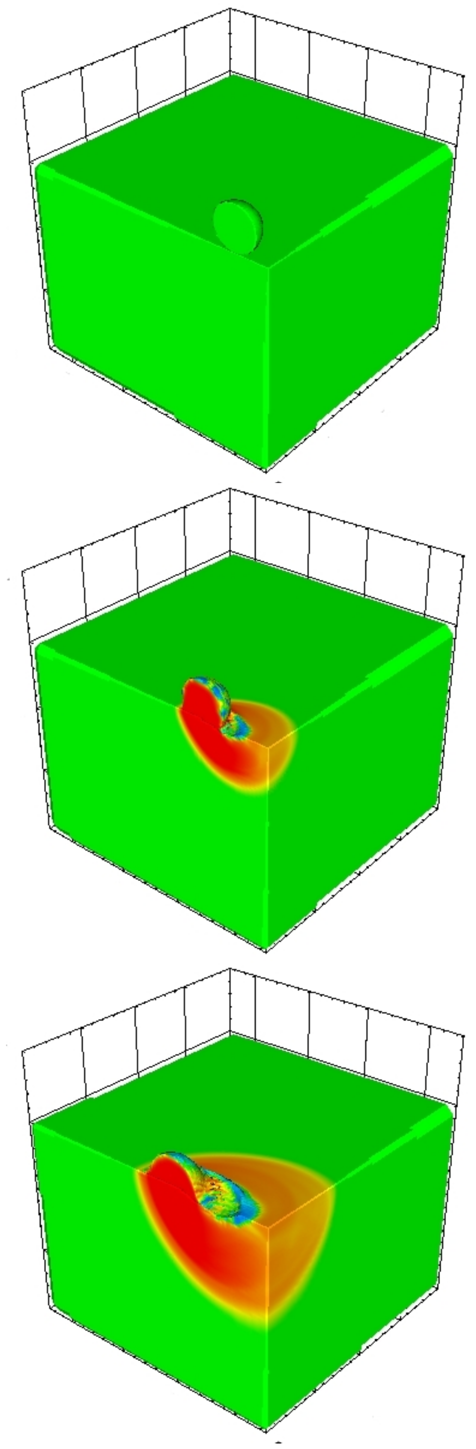
Fig. 2. CTH Sphere and Block Impact Simulation. Simulated time progresses from top to bottom.

A. Motivation

The "sphere hits block" problem was run with CTH in AMR mode, where the sphere and shock wave are the more refined areas of the calculation. Even with such a simple calculation, there is significant load imbalance in portions of the calculation. If we look at the three convection routines,

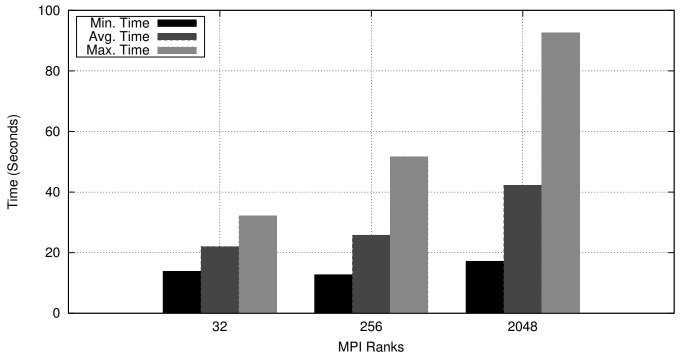

Fig. 3. Time Spent in Convection Routines in CTH

which collectively account for 20% to 25% of the calculation times of a typical simulation (shown in Figure 3), we see that the spread between the minimum and maximum time spent in the convection routines on different ranks increases as the number of ranks increase. The rest of the compute time in CTH is spread out over several routines with none taking more than 5% of the total time.

III. The MiniAMR Mini-Application

MiniAMR is a mini-application that we developed to look at the complexities of three dimensional adaptive-mesh refinement using an octree-based scheme. The computation is made up of a number of blocks, each of which has the same number of cells in each dimension. To refine the mesh in a given area, a block is replaced by eight blocks which have their cell size halved in each Cartesian dimension. Neighboring blocks differ by at most one level of refinement with interpolation being used to exchange data across a refinement boundary. The computation performed at each grid-point of a block is a seven point stencil, where the result for a cell is the arithmetic average of its value added to each of its six neighbors. Reflective boundary conditions are applied to the physical boundary of the global domain. Thus a constant sum across each variable can be computed for correctness. MiniAMR allows for this to be periodically checked during execution if optionally enabled by the user. After computation is done on the cells in all of the blocks local to an MPI rank, the ghost cells on each block are updated from its neighboring blocks.

Refinement is driven by geometric shapes moving across the domain. While this is a simplification over genuine runs of a code like CTH, it is sufficient to capture much of the characteristic behavior. As each shape moves through the mesh and, optionally, changes size, the boundary or volume can be used to define which blocks are refined. These can be used to model physical structures in the problem, such as modeling a shock-wave with an expanding sphere. During the refinement step, blocks are refined or recombined as needed. Finally a load-balancing operation is performed to ensure that the blocks are evenly distributed throughout the MPI ranks. Throughout this step, all of the bookkeeping needed to keep the block connectivity and communication data structures is updated.

We refer the reader to a more detailed description of miniAMR in [19].

A. Modification to Analyze Load Balancing

In order to experiment with load balancing, we have made modifications to the baseline miniAMR implementation. In order to create an imbalance within the calculation, the stencil calculation is changed so that the computation is repeated for each variable. The number of repetitions increases as the block becomes progressively more refined (*i.e.* the most refined blocks execute the most repetitions and therefore take the longest to execute). As it is written, the same calculation is done on each block for each variable. The correlates with our high-level mapping to methods used within CTH. The areas of interest are refined and those areas are the ones that likely have more things going on, such as cells with additional materials and conditions that require more calculations or additional checks. For the machines that we worked with, blocks with 5 levels of refinement took over twice as long to calculate as blocks without refinement.

For load-balancing, we use Recursive Coordinate Bisection (RCB) [4]. For the baseline miniAMR simulations, this process consists of a number of steps where a set of blocks is divided into equal **sized sets** and assigned to a group of processors. We term this a *balance-by-count* approach to load balancing our AMR domain.

We have added a second load-balancing approach which we term a *balance-by-weight* method. Here, we assign each block a weight based on the refinement level that reflects the computational time for that block. Then, for each step in the algorithm, a set of blocks is divided into sets of equal **weight**.

IV. Case Study: Analyzing Load Balancing Approaches using MiniAMR

The modifications to miniAMR force refined blocks take longer to compute and modify the load balancing to balance based on the computational weight of the blocks. In this short case study, we compare the performance of these two approaches in the context of assessing whether a move to *balance-by-weight* load balancing will provide a performance improvement to CTH. Our benchmarking is performed on several machines. The first machine is Chama, an Cray/Appro TLCC-2 [18] cluster with dual-socket 8-core, 2.6Ghz, Intel Sandy Bridge processors, connected by an Infiniband interconnect. The second machine is Volta, a Cray XC30 with Intel Ivy Bridge processors and a Cray Aries DragonFly interconnect. Each node has two 12-core Ivy Bridge processors running at 2.4 GHz. The third machine is Serrano, a Penguin NNSA CTS-1 commodity cluster with Broadwell processors and an Intel Omni-Path 1st generation interconnect. Each node had two 18-core Broadwell processors running at 2.1 GHz. The fourth machine used for benchmarking, is Trinity, a Cray XC40 machine with Cray Aries DragonFly interconnect [20], [11]. With this machine, we have the option of running either in the Haswell partition ("HSW"), where each node has dual-socket 16-core Intel Haswell nodes running at 2.3 GHz, or in the Knights Landing ("KNL") partition, where each node has a Intel Knight's Landing Xeon Phi 7250 processor with 68 cores available. Clock rate for the KNL is 1.4GHz. The runs use KNL nodes configured in the quadrant-flat memory modes where both high-bandwidth and slower capacity memory are available for allocations. We utilize the Intel 16.0 compiler on all runs as this was the baseline production compiler in the computing environment at the time the benchmarking runs were performed.

Machine	Ranks	Balance-by-Count	Balance-by-Weight
Chama (SNB)	256	25238	25241
Volta (IVB)	288	22216	22215
Volta (IVB)	1080	29266	29317
Serrano (BDW)	288	21099	21114
Trinity HSW	256	26618	26578
Trinity HSW	2048	33558	33530
Trinity KNL	256	8427	8451
Trinity KNL	4096	8189	8182

TABLE I
MiniAMR Times in Seconds for Load Balancing Schemes

We run miniAMR with the "sphere hits block" problem on these platforms using each of the load balancing approaches. The resulting benchmarked runtimes are shown in Table I.

The only results to show improvement using the balance-by-weight scheme are those for the Trinity-Haswell partition (with a small difference of approximately 0.2%). The results for all other machines show a less than 0.2% decrease in performance for the new load balancing approach. Most of these results are in the order of the performance variation that we see from run-to-run variation and do not show significant improvement.

A. Comparison of Architectures

The results in Table I reflect our experiences with a broader range of codes on the machines listed (not just for adaptive mesh applications) – that subsequent generations of multi-core Xeon processor are providing similar performance for the same number of MPI ranks but that we require gradually fewer and fewer nodes to maintain a fixed runtime. For example, the 256 rank run on the Chama Sandy Bridge-based machine requires 16 nodes (since each node provides two sockets of 8 cores). Trinity Haswell however, provides a similar runtime for 256 ranks (only 5% slower) but requires only 8 nodes. This represents a roughly 2X per-die performance increase. The "tock" variants of each processor family (the Ivy Bridge and Broadwell), provide non-power of two cores so we have rounded these up to utilize full nodes to prevent skewing of our results when analyzing the average and maximum times of the runs – the effect is that these runs use slightly fewer nodes than each of their previous generations. The ability to maintain such a strong cadence of performance improvement has been of real utility to HPC centers like Sandia but the future of this pace of change is likely to begin to slow in future designs, motivating the analysis of threading and alternative programming models. We do not present benchmark results

Machine	Ranks	Avg Imbalance	Max Imbalance	Time Diff
Chama (SNB)	256	0.002	0.031	0.017
Volta (IVB)	288	0.002	0.050	0.023
Volta (IVB)	1080	0.002	0.071	0.030
Serrano (BDW)	288	0.002	0.050	0.035
Trinity HSW	256	0.002	0.032	0.039
Trinity HSW	2048	0.002	0.041	0.056
Trinity KNL	256	0.008	0.138	0.091
Trinity KNL	4096	0.014	0.227	0.193

TABLE II
VARIANCE SEEN IN STANDARD (BALANCE-BY-COUNT) MINIAMR

Machine	Ranks	Avg Weight Diff	Max Weight Diff	Time Diff
Chama (SNB)	256	0.074	0.589	0.113
Volta (IVB)	288	0.085	0.656	0.128
Volta (IVB)	1080	0.143	0.672	0.290
Serrano (BDW)	288	0.082	0.632	0.139
Trinity HSW	256	0.075	0.590	0.160
Trinity HSW	2048	0.148	0.596	0.424
Trinity KNL	256	0.104	0.635	0.201
Trinity KNL	4096	0.167	0.724	0.642

TABLE III
VARIANCE SEEN IN MINIAMR WITH UNBALANCED CALCULATIONS, BALANCING PERFORMED USING A BALANCE-BY-COUNT SCHEME

for some of these alternatives models here, but we plan to use miniAMR as basis for future publications in this theme.

B. Performance Analysis

To explain the observed runtimes, we start by running with miniAMR as it is normally run on the platforms to establish a baseline of how much variance should be expected during execution. The results for these runs are presented in Table II.

The results show that the amount of variance in time spent executing the the block calculation step as well as the calculated imbalance in the load seen on each run. The time and imbalance numbers are calculated as the maximum time minus the minimum time divided by the arithmetic mean time. For each of the machines shown the miniAMR runs perform 5 levels of refinement except for Trinity KNL, where 4 levels of refinement is used to keep the number of blocks within the smaller memory capacity per node. The lower maximum level of refinement on KNL creates the roughly 3X lower runtime shown in Table I. The maximum imbalance is larger than expected given the average since this calculation starts out with a small region of the calculation being refined and the refined area grows as the calculation (shock-wave) advances through the mesh. For sanity-checking we ran an additional small set of runs with a different simulation configuration in which the number of blocks at each level of refinement stays fairly constant and that resulted in a maximum imbalance of 0.001 and an average of 0.0004. Our observation is that the load balancer gives a fairly equal number of blocks to each processor during these simulations and most of the timings vary by less than 6% with the exception being the KNL runs with up to an almost 20% difference.

Table III shows the potential distribution of weights if we use a balance-by-count balancer but record the weights of the distributed blocks – *i.e.* we show the load imbalance that is potentially present in the code in the presence of blocks which take different times to compute but we are using the baseline balancer. The observation that there is a considerable difference across the MPI ranks in these results, implies that this is an area for further investigation. The imbalance in the number of blocks that the MPI ranks have is the same as Table II since that is the load balancer that was used.

In Table IV we show results for the balance-by-weight load balancer when applied to the same problem used for the

Machine	Ranks	Avg Weight Diff	Max Weight Diff	Time Diff
Chama (SNB)	256	0.003	0.071	0.009
Volta (IVB)	288	0.004	0.083	0.011
Volta (IVB)	1080	0.005	0.108	0.026
Serrano (BDW)	288	0.003	0.084	0.027
Trinity HSW	256	0.003	0.078	0.058
Trinity HSW	2048	0.005	0.102	0.089
Trinity KNL	256	0.017	0.348	0.037
Trinity KNL	4096	0.038	0.695	0.192

TABLE IV
VARIANCE SEEN IN MINIAMR WITH UNBALANCED CALCULATIONS, BALANCING PERFORMED USING A BALANCE-BY-WEIGHT SCHEME

results in Table III. These timings show that the distribution of weights is significantly reduced by using a balance-by-weight scheme and that the variation/timing differences between the MPI ranks is also reduced when compared to a balance-by-count approach.

However, Table V shows the imbalance in the number of blocks that each rank has when the blocks are balanced by weight. The number of blocks on each rank varies more than with load balancing by the number of blocks per rank. Given the nature of the RCB algorithm, the number and total volume of messages exchanged does not vary much between the two load-balancing schemes. The time difference in the interblock communication routines between the ranks that take the longest and those that take the shortest also decreases with load balancing by weight.

In Table VI we show the effect of changing the load balancer with weighted blocks. In all cases, the time to solution when load balancing by weight is larger than the time for load balancing by number of blocks, as an expensive traversal of the blocks present on each node must be performed (versus a tally of the count of blocks on each node). An example breakdown the timings from runs on Trinity-Haswell are shown in Table VII. Additionally, the number of blocks moved per rank is larger in the balance-by-weight approach. However, since the time for load balancing is about half of the total time for mesh refinement and the time for mesh refinement is less than one percent of the total time, the time difference makes negligible difference in the overall run time of the code.

Machine	Ranks	Avg. Imbalance	Max Imbalance
Chama (SNB)	256	0.095	1.151
Volta (IVB)	288	0.116	1.467
Volta (IVB)	1080	0.232	1.506
Serrano (BDW)	288	0.111	1.373
Trinity HSW	256	0.096	1.155
Trinity HSW	2048	0.243	1.211
Trinity KNL	256	0.140	0.990
Trinity KNL	4096	0.287	1.275

TABLE V
NUMBER OF BLOCKS IMBALANCE IN MINIAMR

		Bal. by Count		Bal. by Weight	
Machine	Ranks	Blocks Moved	Time	Blocks Moved	Time
Chama (SNB)	256	24650	31.49	24775	37.36
Volta (IVB)	288	25215	48.15	25377	49.13
Volta (IVB)	1080	37844	54.60	38226	58.19
Serrano (BDW)	288	25215	23.92	25349	25.07
Trinity HSW	256	24650	27.56	24775	28.26
Trinity HSW	2048	41909	75.41	42033	79.13
Trinity KNL	256	4718	11.00	4771	11.53
Trinity KNL	4096	8164	39.94	8275	46.63

TABLE VI
LOAD BALANCING EFFECTS

V. SUMMARY AND FUTURE PLANS

Verified and trusted production application codes are typically large in size, complex in their coding and embody decades worth of gradual refinement and extension to improve accuracy and performance. CTH is one such code. It is written in complex Fortran, extends to hundreds of thousands of lines of code, has third party dependencies and has been refined over years to provide Sandia and the NNSA with an extremely high-performance, accurate code base. Moreover, CTH has been the subject of extensive verification and validation activities both at Sandia and in third party environments. Casual modification of such a code is therefore, quite an endeavor and one which implies extensive cost, not least because of the requirement to revalidate the numerical results. For this purpose, the concept of mini-applications, pioneered in the Mantevo suite, were developed to provide a cheaper, easier to modify code base that permitted very flexible experimentation. MiniAMR, which is part of the Mantevo suite, is designed specifically for this purpose – to provide a lightweight proxy for the complex CTH application in which future design experimentation can be performed.

In this paper, we utilized miniAMR to investigate potential future alternative load-balancing strategies for CTH when run in adaptive mesh mode. The complex calculations of CTH imply some degree of load imbalance and this study was designed to investigate whether a balance-by-weight approach may provide a superior execution time through decreased cross-MPI-rank imbalance.

As a baseline we compared our potentially new approach to the standard, balance-by-count method in which blocks

Routine	Bal. by Count	Bal. by Weight
Block Calculation	14106.70	14030.20
Communication	18530.20	18526.10
Grid Summation	780.48	828.30
Refinement	140.41	145.43

TABLE VII
BREAKDOWN OF ROUTINES ON TRINITY-HASWELL FOR MINIAMR USING BOTH BALANCING SCHEMES

were spread across ranks to ensure a roughly even distribution purely by block count per rank. The results suggest that, while we can provide a better load balancing of the calculation portion of the code, the results overall runtime of the code does not significantly improve because the constituent parts of the execution time remain largely unmodified. In particular, the interblock communication routines are complex enough to keep the total runtime of the code similar regardless of the balancing approach used.

The case study outlined in this paper is deceptively simple but behind it lies an important consideration – that the use of a mini-application in the context of modeling a genuinely complex production code, has saved developer time and effort. We have used our mini-application to evaluate potential alternative designs and found that although improves can be found in some areas, the general outcome will be marginal or useful in production. The result is a concrete, practical example of where a mini-application has been used to save the NNSA program valuable developer time and funds.

Future plans for miniAMR include an implementation of an MPI-OpenMP hybrid scheme (which is in progress and showing strong initial results), a task parallel implementation in which block calculations may be performed by a large collection of fine-grained tasks, and, the incorporation of other load balancing schemes, such as those found in the high-performance Zoltan [8], [9] partitioning library.

ACKNOWLEDGMENTS

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energys National Nuclear Security Administration under contract DE-NA0003525.

We are grateful to the Sandia Advanced Architecture Testbed team and production computing system admins for their support in environment configuration and time on the machines used for our experiments and benchmarking.

REFERENCES

[1] A Development Plan for a Massively Parallel Implementation of the Hydrocode CTH. Technical Report SAND90-0589, Sandia National Laboratories, July 1990.

[2] D.A. Beckingsale, O.F.J. Perks, W.P. Gaudin, J.A. Herdman, and S.A. Jarvis. Optimisation of Patch Distribution Strategies for AMR Applications. *Lecture Notes in Computer Science*, 7587:210–223, 2013.

[3] J. Bell et al. BoxLib Users Guide. Technical report, Lawrence Berkeley National Laboratory, 2012.

[4] M.J. Berger and S.H. Bokhari. A Partitioning Strategy for Nonuniform Problems on Multiprocessors. *IEEE Trans. Comput.*, 36:570–580, May 1987.

[5] M.J. Berger and P. Colella. Local Adaptive Mesh Refinement for Shock Hydrodynamics. *Journal of Computational Physics*, 82(1):64–84, 1989.

[6] M.J. Berger and J. Oliger. Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations . *Journal of Computational Physics*, 53(3):484 – 512, 1984.

[7] P. Colella, D. T. Graves, J. N. Johnson, H. S. Johansen, N. D. Keen, T. J. Ligocki, D. F. Martin, P. W. Mccorquodale, D. Modiano, P. O. Schwartz, T. D. Sternberg, and B. Van Straalen. Chombo Software Package for AMR Applications Design Document. Technical report, 2003.

[8] K. Devine, B. Hendrickson, E. Boman, M. St.John, and C. Vaughan. Zoltan: A Dynamic Load-Balancing Library for Parallel Applications; User's Guide. Technical Report SAND99-1377, Sandia National Laboratories, 1999.

[9] Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson, and Courtenay Vaughan. Zoltan Data Management Service for Parallel Dynamic Applications. *Computing in Science & Engineering*, 4(2):90–97, 2002.

[10] W.P. Gaudin, A. Mallinson, O.F.J. Perks, J.A. Herdman, D.A. Beckingsale, J. Levesque, M. Boulton, S. McIntosh-Smith, and S.A. Jarvis. Optimising Hydrodynamics Applications for the Cray XC30 with the Application Tool Suite. In *Proc. 56th Cray User Group Meeting*, 2014.

[11] K Scott Hemmert, Michael W Glass, Simon D Hammond, Rob Hoekstra, Mahesh Rajan, Shawn Dawson, Manuel Vigil, Daryl Grunau, James Lujan, David Morton, et al. Trinity: Architecture and Early Experience. *Cray Users Group*, pages 2008–2010, 2016.

[12] JA Herdman, WP Gaudin, Simon McIntosh-Smith, Michael Boulton, David A Beckingsale, AC Mallinson, and Stephen A Jarvis. Accelerating Hydrocodes with OpenACC, OpenCL and CUDA. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 465–471. IEEE, 2012.

[13] M.A. Heroux, D.W. Doerfler, P.S. Crozier, J.M. Willenbring, H.C. Edwards, A. Williams, M. Rajan, E.R. Keiter, H.K. Thornquist, and R.W. Numrich. Improving Performance via Mini-applications. Technical Report SAND2009-5574, Sandia National Laboratories, September 2009.

[14] E.S. Hertel, Jr., R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. Mcglaun, S. V. Petney, S. A. Silling, P. A. Taylor, and L. Yarrington. CTH: A Software Family for Multi-Dimensional Shock Physics Analysis. In *Proceedings, 19th International Symposium on Shock Waves*, pages 377–382, 1993.

[15] R.D Hornung and S.R. Kohn. Managing Application Complexity in the SAMRAI Object-Oriented Framework. *Concurrency and Computation: Practice and Experience*, 14(5):347–368, 2002.

[16] Darren J Kerbyson, Henry J Alme, Adolfy Hoisie, Fabrizio Petrini, Harvey J Wasserman, and Mike Gittings. Predictive Performance and Scalability Modeling of a Large-scale Application. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, pages 37–37. ACM, 2001.

[17] J Michael McGlaun, SL Thompson, and MG Elrick. CTH: a Three-Dimensional Shock Wave Physics Code. *International Journal of Impact Engineering*, 10(1-4):351–360, 1990.

[18] Mahesh Rajan, DW Doerfler, Paul T Lin, Simon D Hammond, Richard F Barrett, and Courtenay T Vaughan. Unprecedented Scalability and Performance of the new NNSA Tri-Lab Linux Capacity Cluster 2. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 417–425. IEEE, 2012.

[19] C.T. Vaughan and R.F. Barrett. Enabling Tractable Exploration of the Performance of Adaptive Mesh Refinement. In *Workshop on Representative Applications at IEEE Cluster*, 2015.

[20] CT Vaughan, DC Dinge, PT Lin, KH Pierson, SD Hammond, J Cook, CR Trott, AM Agelastos, DM Pase, RE Benner, et al. Early Experiences with Trinity - The First Advanced Technology Platform for the ASC Program. *Proc. of the Cray User Group (CUG)*, 2016, 2016.