Dynamic Adaptively Refined Mesh Simulations on 1M+ Cores*

Brian T. N. Gunney
Lawrence Livermore National Laboratory
7000 East Avenue, L-561
Livermore, CA, USA
gunneyb@llnl.gov

ABSTRACT

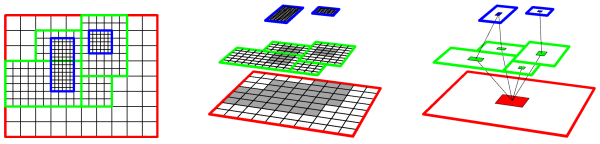
This poster presents a clustering algorithm and a partitioning algorithm for patch-based dynamically adaptive mesh refinement (AMR) and shows scaling results to more than 1M cores.

Keywords

scalability, parallel algorithm, adaptive mesh refinement, dynamic adaptivity, mesh partitioning, clustering

1. INTRODUCTION

Patch-based structured adaptive mesh refinement (SAMR) is a meshing approach that assembles meshes from simple unpartitioned structured grids. The SAMR mesh is a hierarchy of box-shaped grids of increasing resolutions (Figure 1a & b). An advantage of SAMR is that the mesh is succinctly described because each box can be specified with very little metadata: just the indices of two corners and a few ancillary attributes, such as the MPI process that owns the box.



a. Refined mesh b. Hierarchy of levels c. Interlevel proximity graph

Figure 1: Representations of a 3-level mesh.

We use the term mesh management (MM) for the generation, acquisition (via communication), storage of and operations on metadata. Ultimately, MM must provide the boxes and their proximity data, such as that represented by the graph in Figure 1c. Traditional MM approaches duplicate global metadata on every process and use sequential algorithms. This has proved unfeasible on more than $O(10^3)$ cores due to the linear growth of global metadata. Distributing metadata is not trivial, because traditional MM algorithms all required global metadata.

To address this problem, we previously developed a distributed MM approach [2]. By saving and using key information typically discarded by traditional algorithms, MM operations on global metadata could be re-cast in terms of distributed metadata. Proximity data was generated and evolved with the boxes instead of by global searches. Although this proved to be scalable, two key MM algorithms bogged down at $O(10^5)$ cores: the clustering algorithm [3] and the partitioning algorithm [1]. These were plug-in components, which the SAMR framework accessed by abstract interfaces. This poster highlights the new clustering and partitioning algorithms that allowed scaling to $O(10^6)$ cores.

2. TILE CLUSTERING

Clustering generates the initial set of boxes when creating or replacing a level. The user tags cells for the new level to cover, and the algorithm finds a set of boxes that cover those cells. We allow the boxes to contain some non-tagged cells to avoid requiring too many boxes, but the over-refinement extreme of too many non-tagged cells should also be avoided.

The new clustering algorithm was an extension of the highly scalable tile clustering approach by Luitjens [4]. The index space was divided into fixed-size tiles. We generated a box for each tile that overlapped at least one tag. We used tile sizes on the small side to reduce over-refinement. To reduce the number of boxes, we coalesced them. Thus, we achieved both the low over-refinement of small tiles and the low box-counts of large tiles. In addition, we extended Luitjen's algorithms to generate the edges necessary to integrate scalably in the MM framework. For flexibility, we allowed tiles to cross patch and processor boundaries.

3. CASCADE PARTITIONING

Cascade partitioning was a new approach. We used P2P communication to build on each process a $O(\lg N)$-sized picture of the global load distribution, including the load of each half of the machine. Transferring work between the halves balanced their loads. Applying the algorithm recursively balanced within each half.

*This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

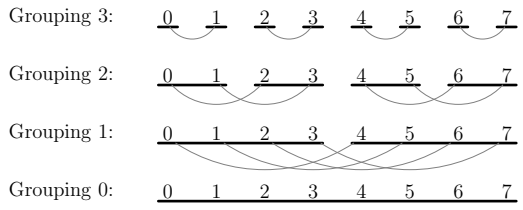


Figure 2: Cascade groups for ranks 0-7. Underscores indicate process groupings. Each group is a binary tree node. Each process belongs to 4 groups. Arcs connecting processes indicate inter-group communication pairs.

We grouped the tasks by recursive splitting, forming a binary tree wherein each node was a grouping (Figure 2). Data exchanges among intergroup communication pairs gave every process the load of each group it belonged to and of their sibling groups. After data exchanges using groupings 3, 2, then 1, all processes knew which half of the machine had more load and how much to transfer to the other half. Load transfer from the overloaded half to the underloaded used communication pairs in Grouping 1. Only metadata were exchanged, so all message lengths were short.

4. RESULTS

Figure 3 shows two scaling results obtained on a 1.5M-core Sequoia IBM BG/Q machine. The weak-scaling benchmark solved a linear advection equation, with 1 unknown/cell, propagating sinudoidal waves in the domain. The physics and numerics of the problem were simple. Regriding interval was once every 9 time steps. This benchmark averaged 1.64×10^6 cells/core. Regridding remained less than 25% of the total cost. The strong scaling benchmark solved the Euler equation, with 5 unknowns/cell, for an expanding spherical shock wave. Its mesh started with 4×10^9 cells and nearly doubled during the run. The regrid interval was 8 time steps. Both benchmarks used a simple forward Euler time-stepping scheme. Note that both were designed to be challenging for regridding, having little computation and frequent regridding. Changing computational work and/or regrid intervals would give different results.

Figure 4 shows scaling characteristics of mesh management for the scalar advection benchmark at 4 levels and 195×10^3 cells/core. All components, including the new clustering and partitioning components, scaled well and showed promise of continuing well past the largest runs performed. Note that if any one component failed to scale, regridding itself would also fail to scale. The strong scaling benchmark showed comparable performance but was harder to evaluate because in strong scaling, everything eventually degraded.

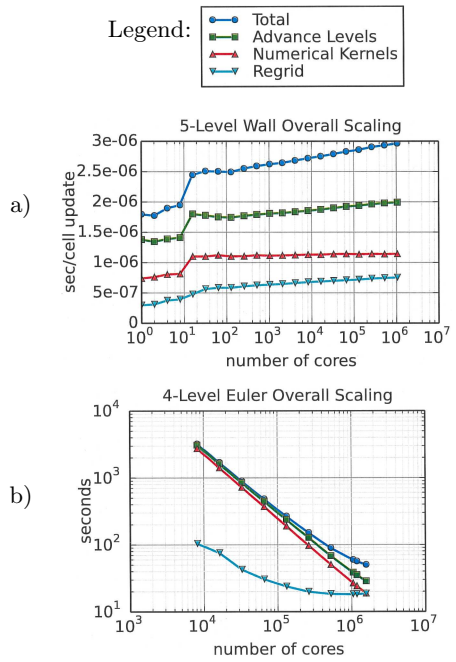


Figure 3: Scaling results. a) Weak scaling to 1M cores. Advection equation with 1 unknown/cell. 1.640×10^6cells/core. Regrid every 9 time steps. b) Strong scaling to 1.57M cores. Euler equations with 5 unknowns/cell. 4.03-8.41$\times10^9$ cells globally. Regrid every 8 time steps.

5. REFERENCES

[1] A. Bhatele, T. Gamblin, K. E. Isaacs, B. T. N. Gunney, M. Schulz, P.-T. Bremer, and B. Hamann. Novel views of performance data to analyze large-scale adaptive applications. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '12. IEEE Computer Society, Nov. 2012. LLNL-CONF-554552.

[2] B. T. N. Gunney. Scalable Mesh Management for Patch-based AMR. In *NECDC 2012 Proceedings*, October 2012.

[3] B. T. N. Gunney, A. M. Wissink, and D. A. Hysom. Parallel clustering algorithms for structured AMR. *Journal of Parallel and Distributed Computing*, 66(11):1419–1430, November 2006.

[4] J. Luitjens and M. Berzins. Scalable parallel regridding algorithms for block-structured adaptive mesh refinement. *Concurr. Comput. : Pract. Exper.*, 23(13):1522–1537, Sept. 2011.

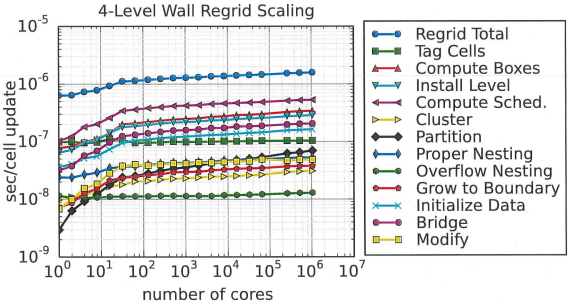


Figure 4: Regrid scaling results