Preprint
UCRL-JC-141940

AMRSim: An Object-Oriented Performance Simulator for Parallel Adaptive Mesh Renement

B. Miller, B. Philip, D. Quinlan, A. Wissink

This article was submitted to
Joint Association for Computing Machinery Java Grande-International Scientific Computing in Object-Oriented Parallel Environments Conference, Palo Alto, CA, June 2-4, 2001

January 8, 2001

AMRSim: An Object-Oriented Performance Simulator for Parallel Adaptive Mesh Re nement*

Brian Miller, Bobby Philip, Dan Quinlan, and Andy Wissink
Center for Applied Scienti c Computing
Lawrence Livermore National Laboratory
bjmiller,bobbyp,dquinlan,awissink@llnl.gov

ABSTRACT

Adaptive mesh refinement is complicated by both the algorithms and the dynamic nature of the computations. In parallel the complexity of getting good performance is dependent upon the architecture and the application. Most attempts to address the complexity of AMR have lead to the development of library solutions, most have developed object-oriented libraries or frameworks. All attempts to date have made numerous and sometimes conflicting assumptions which make the evaluation of performance of AMR across different applications and architectures difficult or impracticable. The evaluation of different approaches can alternatively be accomplished through simulation of the different AMR processes. In this paper we outline our research work to simulate the processing of adaptive mesh refinement grids using a distributed array class library (P++).

This paper presents a combined analytic and empirical approach, since details of the algorithms can be readily predicted (separated into specific phases), while the performance associated with the dynamic behavior must be studied empirically. The result, AMRSim, provides a simple way to develop bounds on the expected performance of AMR calculations subject to constraints given by the algorithms, frameworks, and architecture.

1. INTRODUCTION

Adaptive mesh refinement is a numerical technique for locally tailoring the resolution of computational grids. AMR permits the addition of finer grids to the global computational grid in an adaptive way so as to permit locally more accurate computations or the removal of global error introduced by local singularities. AMR as a numerical technique, is largely independent of the equations being solved, though numerous numerical and algorithmic issues are involved and are the subject of significant research. Unfortunately, AMR is not common place due largely to its inherent complexity.

Adaptive Mesh Refinement computations are complicated by their dynamic nature. In the serial environment they require substantial infrastructures to support the regridding processes, intergrid operations, and local bookkeeping of positions of grids relative to one another. In the parallel environment the dynamic behavior is more problematic because it requires dynamic distribution support and load balancing. Parallel AMR is further complicated by the substantial task parallelism, in addition to the obvious data parallelism, this task parallelism requires additional infrastructure to support efficiently[?]. The degree of parallelism is typically dependent upon the algorithms in use and the equations being solved. Different algorithms have significant compromises between computation and communication. Substantial research work is often required to define efficient methods and suitable infrastructure. AMR represents an impressive opportunity to demonstrate a potentially significant simplifying object-oriented mechanism to build adaptive mesh refinement applications.

The development of parallel algorithms for AMR is greatly complicated by the development of sufficient internal infrastructure to support the complexities of the numerical methods. Typically many details require an especially efficient implementation which only adds to the complexity and interferes with an understanding of the performance. This is especially true in addressing the complexity of parallel Adaptive Mesh Refinement. Our goal within this research work has been to build a sufficiently simple infrastructure to simulate the use of adaptive mesh refinement and provide sufficient flexibility to test many ideas about how to improve implementations and the AMR algorithms.

This approach to simulate AMR processes has not been attempted previously. Existing parallel AMR work has been forced to cope with such detail and complexity that numerous simplifying assumptions were critical to the successful implementation of each (reference AMR work at LBL, UC Berkeley, SAMRAI, AMR++, Overture, UT Austin, others?). In numerous cases object-oriented approaches were used to address the complexity of the implementation details, in most cases this was critical to the successful implementation despite numerous simplifying assumptions. For example, the most common among all implementations was the assumption of each grid wholely owned by a single pro-

*This work is funded (in part?) by the Department of Energy's Division of Mathematical, Information, and Computational Sciences under contract number ???.

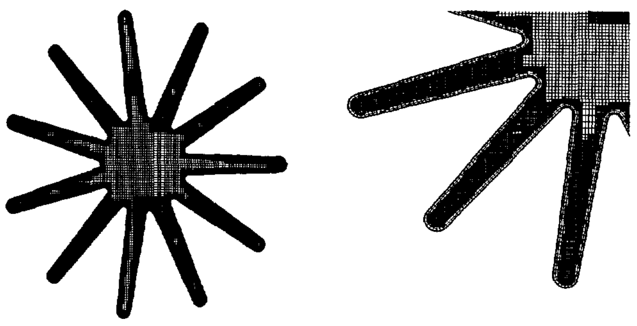

Figure 1: Solid rocket fuel grain cross-section with grid refinement.

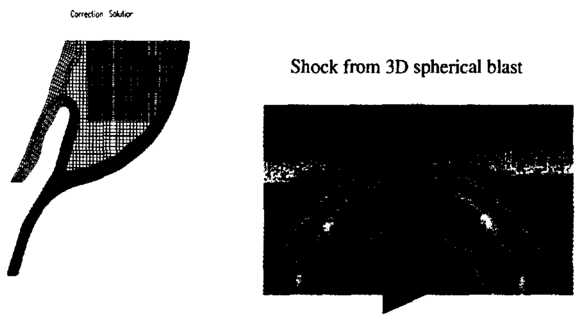

Figure 2: Cross-section of eye with reflenement from Overture Framework and spherical shock wave with refinement from SAMRAI Framework.

cessor (no distribution of AMR patches).

1.1 Taxonomy of the Adaptive Grid

The addition of local refinement adds a grid as a new level of refinement to an existing AMR grid. Initially the AMR grid is a global grid or collection of global grids. The addition of local refinement adds a grid which covers a smaller region of the global grid, the positioning of the local refinement is based upon the the evaluation of error (typically error estimates) on the previous adaptive grid. Through successive iteration in the solution process the adaptive grid is tailored in this way and evolves to be specific to the reduction of numerical error. Typically AMR can provide equivalent resolution to grids containing several orders of magnitude more grid points and which much less computational effort. The drawback is in the greater complexity of the dynamic computations and the increased sophistication of the algorithms.

Figure 1 shows the use of adaptive mesh refinement around the surface of the fuel grain in a star shaped cross section of a solid rocket motor. Figure 2 shows the use of AMR to resolve flow within the eye using AMR++. Figure 2 also shows the use of AMR within a SAMRAI application with refinement to resolve shockwaves around an expanding sphere.

2. PARALLEL AMR SIMULATOR

AMRSim is a simulator for evaluating both object-oriented framework approaches and different algorithms. While it implements all the components of common AMR algorithms for elliptic, parabolic, and hyperbolic equations is does not provide correct results for any particular partial differential equation. Its purpose is as a framework to evaluate, contrast, and compare different existing approaches on different architectures as well as develop and test new ideas.

Since AMRSim is object-oriented, it maps particularly well to numerous object-oriented frameworks specific to AMR. The simplicity gained by not correctly solving any equation is made up for in the additional complexity to address a range of implementation approaches in the handling of computations on AMR grids.

Some issues are specific to framework optimization and others are specific to algorithms. We separate the issues associated with these.

2.1 AMR Framework Optimizations

Current object-oriented frameworks introduce numerous assumptions (restrictions) which are important to evaluate. Each can be expect to be justified under specific applications or because it greatly simplifies the interface or the implementation of the framework itself. Each assumption is worthy of some evaluation of its effect on performance:

- distribution of grids
  Most AMR frameworks make the assumption that grids will not be distributed. To permit parallelism over more processors than grids, the existing grids are split up to form more grids. This process effectively can be considered to imply a distribution of the grid except that it leads to additional complexity in the domain calculus and has subtle effects on the behavior of some algorithms. Alternatively, grids can be distributed across processors and the computations handled as parallel computations local to a subset of processors over which the grid is distributed. This approach simplifies the user interface and maintains a constant complexity over the number of processors. But it requires substantially more infrastructure to support.

- partitioning strategies
  Where there is synchronized behavior in the processing of the AMR levels the partitioning of the grids in an adaptive grid can greatly effect the overall performance. The details of load balancing become particularly subtle and complex. AMRSim permits the evaluation of different strategies within a simpler and more controlled environment, and across a broad range of architectures.

2.2 AMR Algorithm Optimizations

The modeling of performance across a broad range of AMR algorithms requires the artificial introduction of computational load within specific phases of the AMR computation:

- computational work to solve individual grids
  All AMR algorithms require local processing of each grid, but depending upon the equations being solved and the algorithm being used, each requires a different amount of computation. In most algorithms the cost of the computation is linear in the number of points, but due to cache effects this is simulated using simple

relaxation methods within AMRSim. To address the requirements of a range of algorithms the number of relaxation sweeps over the grid is an input parameter to the simulation.

- projection of data between levels (interpolation and restriction)
  All AMR algorithms exhibit communication between adaptive refinement levels as part of their solution process. The computational cost of this phase depends upon the number of levels of refinement, the number of patches at each level, their degree of connectedness, the distribution of the parent grids (relative to the child grids), etc. Each of these details form parameters to simulations in AMRSim.
- synchronization
  All AMR algorithms exhibit some synchronization between their different phases of computation (computation on the grids, computation of each level, interpolation of data to finer levels, restriction of data to coarser levels). Each algorithm differs both as to what extent the synchronization exists, and in between what phases of the solution process it is introduced. AMRSim permits the simple representation of multiple algorithms providing a consistent means to evaluate performance.

The development of new algorithms to address performance issues can also be modeled using AMRSim.

2.3 Simulation Inputs

Models built using AMRSim contain multiple inputs to simplify parameter studies of their performance. In many cases realistic inputs to the model can be taken from existing AMR application codes. These application specific inputs include:

- number of iterations on each level
- number of sweeps on each grid
- number of levels
- number of grids on each level
- relative connectedness of grids on a level (number of siblings)
- number of processors over which to share grids (relative overhead of the distribution of grids)
- cost of operations on each grid

In each case the input parameters are dependent upon the both the application (the equations) and the algorithms being used (FAC, AFAC, AFACX, Hyperbolic Methods). AMRSim applications are sufficiently flexible to support parameter evaluations.

3. IMPLEMENTATION OF AMRSIM

We base AMRSim on the A++/P++ array class library. This library greatly simplifies the development of parallel application codes. The array objects encapsulate the details of the distribution of the arrays over one or more processors, permitting a range of distributions to be specified and evaluated easily.

3.1 AMRSim built using A++/P++class library

A++/P++ is a portable C++ array class library for serial/parallel computers. A++ represents the serial array class and P++ represents the parallel array class, both have an identical interface. It provides a simple syntax for the creation and manipulation of arrays, similar to FORTRAN 90. In the serial environment the arrays represent contiguous storage and can be shared with other FORTRAN or C applications and provide for indexing, indirect addressing, complex views (sub-arrays), dynamic manipulation, etc. In the parallel environment the arrays form distributed objects whose distribution can readily be specified and manipulated dynamically. Within the array class, overloaded operators define a conventional array syntax with the usual array semantics of FORTRAN 90 like operations. In the parallel environment the operators handle communication, as required, for correct operation. A++P++ thus provides a simple and elegant mechanism that allows serial code to be reused in the parallel environment. It also provides a relatively high level of interface from which to optimize both serial and parallel performance using the array's serial and parallel semantics (see section ??).

We use the P++ array class library within **AMRSim** to manage the details of the distribution and inter-processor communication of distributed array data associated with each grid. Do as best represent the computations independent of any array class processing, all computations are done using the C data stored within each array object and specifically not using the array class's array operators. In doing so we generate the identical performance represented by preprocessing using a preprocessor built using ROSE [?].

3.2 Algorithmic Components

AMR algorithms can be readily classified and common components identified and modeled separately. Within AMRSim these individual components with some examples from the numerical solution of PDE's are:

- Projection (common to all AMR algorithms)
- Interpolation (common to all AMR algorithms)
- Local grid solution (common to all AMR algorithms)
- Synchronization (common to all AMR algorithms but different for each)
  - no synchronization (AFAC, AFACx)
  - synchronization by level (FAC)
  - synchronization with CFL constraint (hyperbolic methods)

3.3 Speci cation of speci c algorithms

While posing an analytical description of individual AMR algorithms using this classification their interdependence is difficult to analyze for a parallel computer because of the dynamics of the application and the difficulty in accurately modeling the interactions between processors. To simplify this step we model the individual AMR algorithm components with simulations. This combined approach to the performance modeling can be expected to be more realistic than an analytic model because it absorbs the numerous complex

```
for (level=0; level < maxLevel; level++)
   {
     SolveLevel(i);
   }

for (level=0; level < maxLevel; level++)
   {
     ghostBoundaryUpdate();
   }
```

Figure 3: Example Specification of AFACX Solver.

details of the interaction of multiple processors. It is in addition much more flexible than any of the existing AMR frameworks in permitting a more varied set of algorithms to be evaluated and compared. Further, many critical inputs to the simulation can be taken directly from simpler existing serial AMR applications (e.g. number of grids, number of levels, etc.).

4. RESULTS

We demonstrate the use of AMRSim to model the load balanced computation phase common to all AMR algorithms. Some AMR algorithms enforce additional synchronization upon the ordering of the processing by level (e.g. FAC, and hyperbolic methods) while some to not (e.g. AFAC, AFACx). The results demonstrate the case of no additional synchronization and the effect of a load balancing based upon the equal distribution of grid points absent the effects of communication; typical with AMR computations.

figure*

5. CONCLUSIONS

The development of adaptive mesh refinement is complex due to its dynamic behavior and the expectations placed upon performance. This is especially problematic for parallel adaptive mesh refinement and has forced numerous simplifications and assumptions to be made in the development of object-oriented frameworks. AMRSim has been used to look at a few of these assumptions and provides a useful tool for the evaluation of numerous approaches to performance specific to individual architectures and algorithms. We have shown that there are advantages and disadvantages to the distribution of the AMR grids in the parallel environment and that specific algorithms can be expected to perform differently as a result.

Current work in progress for the final paper will look at the effects of the interpolation and restriction components within the AMR algorithms and put together a more complete picture of the performance of specific algorithms on different architectures.

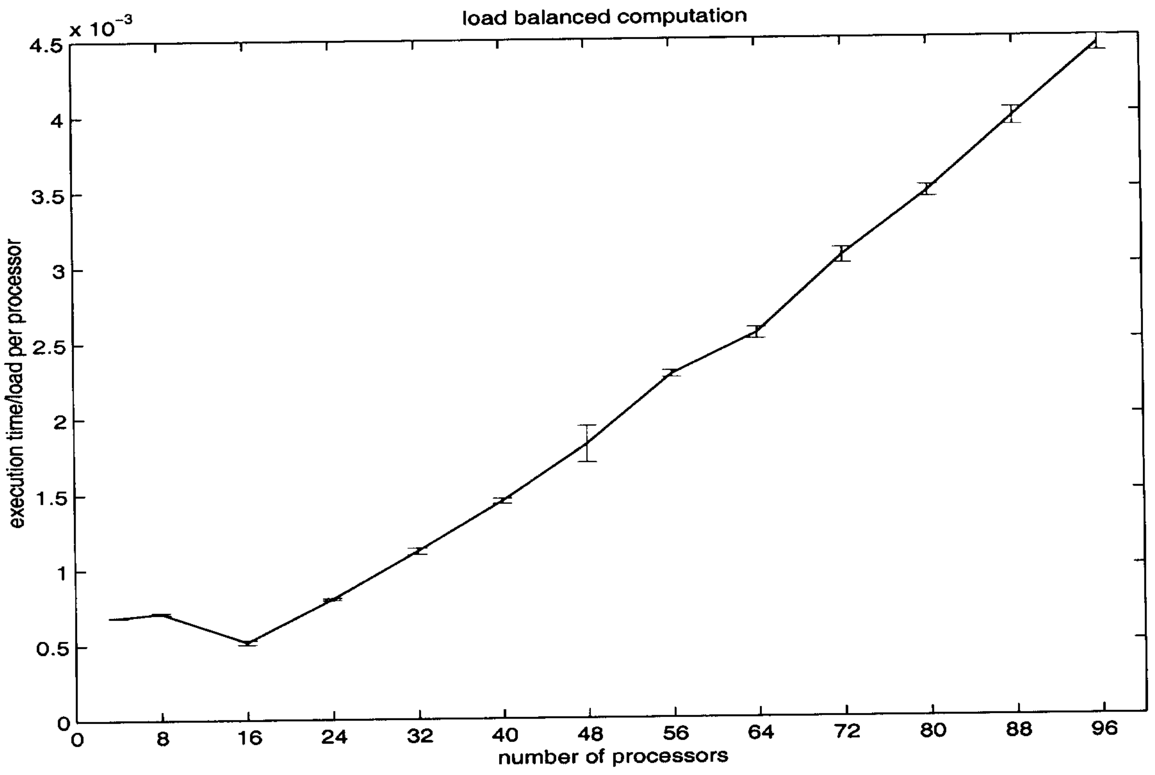

Figure 4: Performance of Load Balanced Computation for AFACx Solve on grids.