LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-JRNL-210250

Parallel Adaptive Mesh Refinement

L. Diachin, R. Hornung, P. Plassmann, A. WIssink

March 7, 2005

Parallel Processing For Scientific Computing, Chapter 8

Updated October 14, 2003

Chapter 1

Parallel Adaptive Mesh Refinement

1.1 Introduction

As large-scale, parallel computers have become more widely available and numerical models and algorithms have advanced, the range of physical phenomena that can be simulated has expanded dramatically. Many important science and engineering problems exhibit solutions with localized behavior where highly-detailed salient features or large gradients appear in certain regions which are separated by much larger regions where the solution is smooth. Examples include chemically-reacting flows with radiative heat transfer, high Reynolds number flows interacting with solid objects, and combustion problems where the flame front is essentially a two-dimensional sheet occupying a small part of a three-dimensional domain.

Modeling such problems numerically requires approximating the governing partial differential equations on a discrete domain, or grid. Grid spacing is an important factor in determining the accuracy and cost of a computation. A fine grid may be needed to resolve key local features while a much coarser grid may suffice elsewhere. Employing a fine grid everywhere may be inefficient at best and, at worst, may make an adequately resolved simulation impractical. Moreover, the location and resolution of fine grid required for an accurate solution is a dynamic property of a problem's transient features and may not be known *a priori*. Adaptive mesh refinement (AMR) is a technique that can be used with both structured and unstructured meshes to adjust local grid spacing dynamically to capture solution features with an appropriate degree of resolution. Thus, computational resources can be focused where and when they are needed most to efficiently achieve an accurate solution without incurring the cost of a globally-fine grid. Figure 1.1 shows two example computations using AMR; on the left is a structured mesh calculation of a impulsively-sheared contact surface and on the right is the fuselage and volume discretization of an RAH-66 Comanche helicopter [35]. Note the ability of both meshing methods to resolve simulation details by varying the local grid spacing.

Figure 1.2 illustrates a typical increase in efficiency that AMR provides. Here, the maximum element error is shown as a function of the number of grid points used

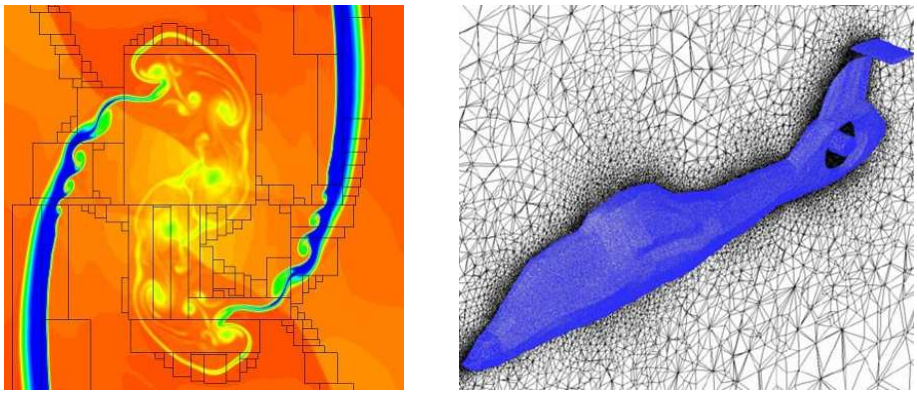

Figure 1.1. *Examples of AMR using structured and unstructured grids. The left figure shows fine detail in an impulsively-sheared contact surface computed using patch-based structured AMR. The right figure shows the accurate suface and volume representation of the fuselage and engine cowl of an RAH-66 Comanche helicopter with an unstructured AMR grid.*

in a finite element simulation of a three-dimensional pressure vessel with a crack. In the AMR case, grid refinement follows the propagation of a two-dimensional localized feature through the material volume as it transitions from elastic to plastic [8]. The plot compares calculations using a static, uniform grid and an adaptively refined mesh. Note that AMR requires significantly fewer elements to achieve a certain accuracy and that the slopes of lines through each set of points indicates that the uniform mesh solution is converging less rapidly as elements are added. AMR methods have also enabled simulation of problems that may be intractable with other computational approaches. For example, work by Bell et al. [11] has shown that highly-detailed features in laboratory-scale methane flames can be revealed using AMR that have not been seen in other simulations. Also, Norman et al. [17] use AMR in cosmology simulations that span twelve orders of magnitude of spatial resolution.

The basic parallel AMR algorithm is given in Figure 1.3 (adapted from [44]). Most implementations start with a uniform grid that must be partitioned across the processors of a parallel computer. The PDE is solved and the grid locally refined (and/or de-refined) as needed based on *a posteriori* estimates of the error or smoothness of the solution. The refined mesh must then be repartitioned to achieve a balanced load and the solution computed on the new grid configuration. Often, the process of re-gridding continues iteratively until a specified error tolerance is achieved or some specified finest grid spacing is reached. In time-dependent problems, re-gridding is performed periodically during the discrete time stepping sequence. The dynamic nature of the grid results in several difficulties for efficient parallel computation including dynamic load balancing, data (re-)distribution, and dynamic, complex data communication patterns. Unlike static grid computations, these overheads cannot be amortized over the duration of the calculation. The

Figure 1.2. *On the left is a comparison of maximum element error as a function of the number of grid vertices in an unstructured, tetrahedral mesh calculation. The AMR computation requires significantly fewer points to achieve a desired accuracy. On the right is an image of the two-dimensional version of this problem showing refinement around the transition region and areas of high elastic stress.*

algorithms developed to meet these challenges are discussed in the remainder of this paper for both Structured Adaptive Mesh Refinement (SAMR) and Unstructured Adaptive Mesh Refinement (UAMR) methods.

Partition the initial mesh M_0 on the processors of a parallel computer
Solve the PDE on M_0 and estimate the error on each element
while the maximum element error is larger than a given tolerance **do**
    Based on error estimates, determine a element or subgrid set, S, to
        refine or de-refine/merge. Act on these elements or subgrids
        and other elements/subgrids necessary to form the new mesh M
    Repartition the mesh M to achieve a balanced load
    Solve the PDE on M and estimate the error on each element
endwhile

Figure 1.3. *An outline of a parallel adaptive solution method for PDEs*

The remainder of the paper is organized as follows. We give an overview of the SAMR and UAMR methods in §1.2 and §1.3. We describe the basic features of each approach, along with issues associated with application development and parallel implementation. We also discuss some of the more widely available software packages for each. In §1.4 we compare and contrast the two methods to highlight the differences between them. In §1.5, we describe some current investigations by the research community and point out issues for future work including new algorithm development, software interoperability, and scalability to thousands of processors.

1.2 Structured Adaptive Mesh Refinement Methods (SAMR)

Structured AMR (SAMR) methods are rooted in the work of Berger, Oliger, and Colella [12, 13]. The term *structured* refers to the use of logically-rectangular grid concepts in the implementation of the adaptive grid. SAMR utilizes a hierarchy of levels of spatial, and often temporal, grid spacing with each level composed of a union of logically-rectangular grid regions. SAMR codes generally adopt one of two implementation strategies that differ with respect to management of the grid hierarchy. We refer to them as the *patch-based* and *tree-based* approaches. While numerical methods for uniform structured grids are generally used in SAMR applications, SAMR algorithms are much more complex because they must treat internal mesh boundaries between coarse and fine levels properly to produce a consistent and accurate solution.

Originally, SAMR was developed for shock hydrodynamics problems [12, 13]. Since then, SAMR methods have been developed for many other problems including: compressible and incompressible fluid dynamics [2, 38, 57] porous media flow [50, 58], solid mechanics [27, 59], radiation diffusion and transport [34], laser-plasma interaction [23], combustion [10, 22], cosmology [17], and astrophysics [26].

Overview of the Approach

In both patch-based and tree-based SAMR approaches, the adaptive computational grid is organized as a hierarchy of grid levels, each representing a different grid spacing. The coarsest level covers the entire domain and the levels are nested so that each finer level covers a portion of the interior of the next coarser level. The formation of each finer level begins by identifying coarser cells that require refinement based on estimating the solution error or local smoothness. Then, these cells are clustered into logically-rectangular grid regions, often called *patches*, to form the finer level. Generally, boundaries of fine patches coincide with boundaries of grid cells on the next coarser level to simplify inter-level data communication and the construction of numerical approximations on the grid hierarchy.

In the patch-based scheme [12, 59], an integer index space for the computational domain emerges from the cells on the coarsest global grid level. Each finer level in the hierarchy is a refinement of some subregion of this global index space. Then, patch relationships can be described solely in terms of the level index spaces and grid refinement relations between them. Grid patches may be described compactly using only upper and lower cell indices. Thus, management of the patch hierarchy involves simple bookkeeping and requires very little overhead. The low storage requirements generally make it possible for each processor to own a complete description of the hierarchy and processor assignments. Thus, parallel data communication patterns can be computed efficiently in parallel without extra interprocessor communication.

In the tree-based scheme [24, 48], the grid is also organized into a hierarchy of refinement levels. However, in this case, the grid is usually decomposed into relatively small *blocks* of grid cells. Each grid block is refined into a set of blocks of

fine cells and the grid configuration is managed using a tree-based data structure that maintains explicit child-parent relationships between coarse and fine blocks. While conceptually simple and easy to implement, the tree structure involves greater storage overhead and potentially larger costs to adapt the grid than the patch-based approach. In parallel, it is generally not possible to store the entire tree structure on each processor so determining how best to split parent and child blocks across processors is important and requires additional inter-processor communication.

Typically, patch-based SAMR employs contiguous, logically-rectangular grid regions of various sizes and aspect ratios depending on the problem. Each of these patches usually contains thousands of grid cells. In contrast, tree-based SAMR usually employs relatively small, uniformly-sized blocks, each of which may contain up to a few hundred cells. While small blocks make it more difficult to exploit data locality on a large scale, they allow for more flexible grid configurations (e.g., fewer refined cells around local features) and easier computational load balancing. However, the use of smaller blocks and patches increases data communication overhead and storage overhead associated with ghost cells. In the end, both patch and tree approaches present challenges for achieving optimal parallel load balancing and scaling of SAMR applications.

Parallel SAMR

Parallel SAMR codes share aspects with non-adaptive, parallel, structured grid codes. Both employ numerical operations on contiguous, logically-rectangular regions of data and communication operations that pass information between the regions to fill ghost cells. However, data communication patterns in SAMR codes are dynamic and more complex because the grid configuration is irregular and changes during the computation. Load balancing numerical and communication operations are also critical for good parallel performance. While non-adaptive codes typically incur the cost of grid generation, load balancing, and constructing data communication dependencies once, these operations are performed frequently by adaptive codes. Thus, efficiency of these operations is paramount so that adaptive gridding overheads are acceptable on large numbers of processors.

Parallel SAMR applications are sufficiently complex and costly to develop that a number of libraries have been built to provide the underlying infrastructure for SAMR applications. While an exhaustive, detailed comparison of such software is beyond the scope of this paper, a few of the more well-known libraries include Chombo [21] and BoxLib [9] from Lawrence Berkeley National Laboratory, GrACE [47] from Rutgers University, PARAMESH [42] from NASA Goddard, and SAMRAI [33] from Lawrence Livermore National Laboratory.

These libraries support a wide variety of parallel SAMR applications, and all provide programming abstractions for managing parallel data decomposition and distribution on adaptive grid hierarchies. However, they differ in design and features, especially with respect to parallel implementation. GrACE and PARAMESH use the tree-based approach while BoxLib, Chombo, and SAMRAI are patch-based. GrACE developers have extensively researched partitioning strategies for SAMR using space-filling curves [46]. BoxLib is a pioneering SAMR software library and

many basic concepts found in other SAMR libraries, including Chombo and SAMRAI, are due to BoxLib. Also, BoxLib has been the foundation of an impressive history of SAMR algorithm and CFD application development [9].

SAMRAI and Chombo differ primarily in their design of data management and parallel communication capabilities. Chombo provides data containers templated on the grid data type. Parallel data transfers are performed individually by each data container object for the template parameter [21]. SAMRAI uses an object-oriented composition design pattern for its data management and parallel data communication infrastructure. The parallel data abstractions in SAMRAI are extensions of *communication schedule* ideas found in the KeLP library [5]. SAMRAI supports an arbitrary number of grid data objects within a single communication operation so that schedule construction costs can be amortized over all data transferred at a communication phase of a calculation. Thus, all data moves in one send-receive message pair per processor pair independent of the number and types of data objects involved. Also, parallel data communication functionality supports new data types, such as irregular user-defined particle data structures, without modifying or recompiling the library [33].

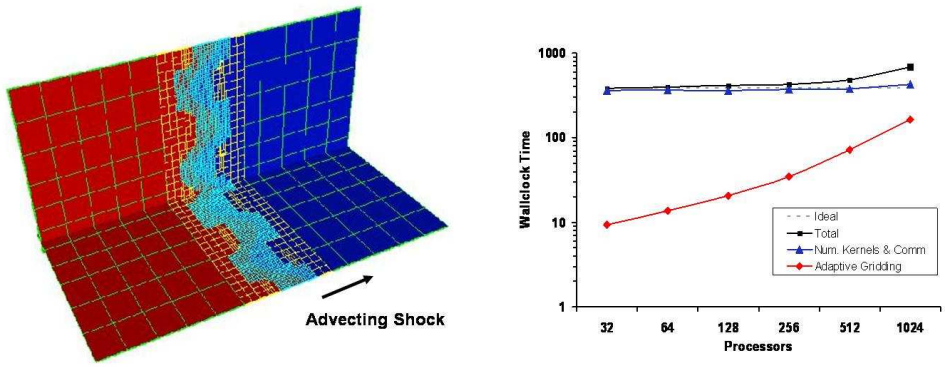

Figure 1.4. *Scaling properties of a three-level scaled SAMR simulation of a moving advecting sinusoidal front. Re-meshing occurs every two time-steps. Although the problem scales reasonably, adaptive gridding costs are clearly less scalable than numerical operations. Work to improve scaling of adaptive gridding operations is ongoing.*

A particularly critical issue in achieving good parallel performance of SAMR calculations is dynamic load balancing. Several researchers have proposed schemes that have shown to be effective. Steensland [56] developed partitioning strategies to optimize load balance and/or minimize data migration costs using space-filling curves in the GrACE library. Space-filling curves are also used in PARAMESH. Lan et al. [39] implemented a scheme based on sensitivity analysis of loads for the Enzo [17] code. Rendleman [18] proposed a knapsack algorithm to efficiently distribute different-sized patches, which is used in the BoxLib and Chombo libraries. SAMRAI provides greedy algorithms for spatially-uniform and spatially

non-uniform workloads and a space-filling curve algorithm to help reduce interprocessor communication.

In addition to load balancing, data redistribution and regeneration of data dependencies as the adaptive grid changes is very important for parallel scaling. Scaling studies of SAMR calculations with SAMRAI revealed that these overhead costs can be negligible on a few hundred processors, but become unacceptably large on 1000 or more processors [64]. This observation lead to the investigation of combinatorial techniques for reducing the operational complexity of adaptive gridding operations and improved scaling of large-scale parallel applications. Figure 1.4 shows the scaling properties for a linear advection problem run using SAMRAI. While this problem is particularly simple numerically—a typical physical problem would require much more computationally-intensive routines—it exposes the costs associated with adaptive gridding operations. Although numerical operations are minimal and regridding occurs every two time-steps on each grid level, nearly optimal scaling is achieved using more than 1000 processors. The University of Chicago ASCI Center FLASH code [26] which uses the PARAMESH library has also demonstrated scaling to several thousand processors [19].

1.3 Unstructured Adaptive Mesh Refinement Methods (UAMR)

As with SAMR methods, the goal of AMR strategies for unstructured meshes is place more grid points where the solution error is large. Unlike SAMR methods, the meshes used are called *unstructured* because the local connectivity of elements can vary. Another difference with SAMR methods is that typically the mesh must be *conforming*—the face of an element cannot be adjacent to a subdivided element face. Given the restriction to conforming meshes, the majority of unstructured AMR algorithms have concentrated on simplicial elements (tetrahedra in three dimensions), and we make this assumption in this section. Unstructured mesh algorithms [14] and adaptive strategies [36] have been extensively studied. In the following section we present an overview of UAMR methods from the perspective of developing scalable parallel implementations of these algorithms. In particular, we focus on aspects of these algorithms that exhibit commonality with SAMR methods.

Overview of the Approach

Given the constraint that unstructured meshes must remain conforming, mesh refinement algorithms broadly consist of two steps. First, elements are marked by an error estimation method for refinement (or de-refinement) and then these marked elements are subdivided (or merged). Second, the resulting non-conforming mesh is subsequently modified to be remain conforming. The most popular approaches for refinement for simplicial meshes include regular refinement [6] and element bisection [52, 53]. The difference between these two approaches is how they treat an element marked for refinement. With regular refinement, all element edges are simultaneously bisected, producing four triangles from one in two dimensions. With bisection,

only one edge is bisected at a time, producing two triangles in two dimensions, or two tetrahedra in three dimensions.

The problem with both of these schemes, is that following the refinement, neighboring elements become non-conforming as they contain a subdivided edge. To modify the mesh so that all elements are valid, the refinement must propagate through the mesh by additional refinement until all elements are valid. Although this propagation appears problematic, in practice it is typically limited and serves an additional property of maintaining mesh quality. In Figure 1.5 we give pseudocode for the bisection algorithm proposed by Rivara [52]. The propagation of the refinement of the mesh resulting from invalid elements is illustrated for a two-dimensional mesh in Figure 1.6. This process can be extended to three-dimensional, tetrahedral meshes.

$i = 0$
$Q_i =$ a set of elements marked for refinement
$R_i = \emptyset$
while $(Q_i \cup R_i) \neq \emptyset$ **do**
    bisect each element in Q_i across its longest edge
    bisect each element in R_i across a nonconforming edge
    all incompatible elements embedded in Q_i are placed in R_{i+1}
    all other incompatible elements are placed in Q_{i+1}
    $i = i + 1$
endwhile

Figure 1.5. *The bisection algorithm*

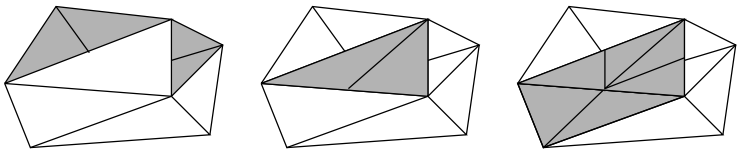

Figure 1.6. *From left to right the process of the bisection algorithm. In the initial mesh the shaded elements are refined; subsequently the shaded elements are refined because they are invalid.*

Like SAMR methods the error estimation procedures for UAMR methods is done via *a posteriori* error estimates [4]. The advantage of this type of error estimation scheme is that for elliptic problems these estimates bound the true error asymptotically, and their computation involves only local element information [7]. However, unlike SAMR methods, many of the technical aspects of unstructured AMR algorithms are based on mesh generation methods. For example, point insertion schemes, where new mesh vertices are placed at edge bisectors or element circumcenters are a fundamental tool for both unstructured mesh generation and

adaptive refinement.

A central concern for both mesh generation and mesh refinement methods is their ability to preserve mesh quality—typically this means maintaining well shaped elements. This problem is especially acute when it comes to the accurate representation of non-trivial (e.g., curved) boundaries [36]. Incorporating a combination of mesh improvement tools such as iterative vertex insertion, flip exchanges, and vertex smoothing into a single mesh refinement loop can be effective [14, 16, 25]. Implementation of these algorithms can be a complex task in three dimensions. For example, the process of flip exchanges for tetrahedra to improve mesh quality has a number of significant technical issues that have been extensively studied [40]. Point insertion schemes for adaptive refinement have also been studied [41].

Parallel UAMR

The basic approach for parallel UAMR methods is given in Figure 1.3. There are two broad areas to consider for these methods. First, the parallelization of the adaptive mesh refinement algorithm in a manner that ensures a high-quality, conforming mesh. And second, the issue of re-partitioning this mesh as it changes to ensure good load balancing without incurring too much communication overhead.

In early work on parallel UAMR methods, Williams [62, 63] gave an approach for parallel mesh refinement and implemented it in the parallel software package DIME. To ensure a consistent distributed mesh data structure, a parallel *voxel* database was maintained, using vertex coordinate information to help resolve point identity and element neighborhood information. However, a running time analysis of the algorithm was not given, and there are significant issues in maintaining a consistent distributed data structure for the mesh.

An alternative approach [37], uses the refinement of independent sets of elements to ensure that the distributed mesh data structure remains consistent during parallel refinement. A set of mesh elements are said to be independent if they do not share an edge. The independent sets can be efficiently chosen in parallel by a randomization strategy. Random numbers are assigned to every element marked for refinement; a element is in the independent set if its random number is larger than any neighboring marked elements. An analysis and computational experiments for this approach exhibit good parallel scaling to $O(500)$ processors [37].

The implementation of parallel software implementations of UAMR algorithms is a complex task, and several systems have been developed to support these algorithms. For example, the Parallel Algorithm Oriented Mesh Database (AOMD) developed at RPI is a mesh management database that provides a variety of services to mesh applications. Advanced features of AOMD include a flexible mesh representation, conforming/non-conforming mesh adaptation both in serial and parallel, and global and local mesh migration. AOMD supports mesh load balance with Zoltan [15], serial and parallel mesh I/O, and dynamic mesh usage monitoring. The software is written is in C++ and supports a wide variety of element types [51].

Another current system is NWGrid, a library of user-callable tools that provides mesh generation, mesh optimization, and dynamic mesh maintenance in three dimensions. A aspect of NWGrid is that geometric regions within arbitrarily com-

plicated geometries can be defined as combinations of bounding surfaces, where the surfaces are described analytically or as collections of points in space. A variety of techniques for distributing points within these geometric regions are provided. NWGrid provides grid partitioning functions, reconnection, mesh quality improvement, and remapping [60].

SUMAA3d is a library of scalable algorithms for parallel, adaptive simplicial mesh refinement for two- and three-dimensional unstructured meshes. The AMR scheme is based on edge bisection to refine and de-refine elements in conjunction with Rivara's technique for removing nonconforming points from the mesh. A geometric mesh partitioning heuristic used to generate a partition tree that strives to minimize both latency and transmission communication costs on distributed-memory computers. The code also includes parallel mesh quality improvement capabilities and has been used in a number of finite element simulations [36, 37].

A common aspect of all of these parallel UAMR implementations is that their efficient parallel performance depends critically on good mesh partitioning. Initially, the mesh must be partitioned to equalize the load on each processor and to minimize the number of elements that share edges but are assigned to different processors (often referred to as ghost elements). As the computation proceeds, the element refinement will not be balanced—some processors are more likely to have refinement. Thus, the mesh must be dynamically rebalanced to maintain the same load distribution as the initial partitioning, while minimizing the interprocessor communication required to move elements between processors.

A number of effective partitioning heuristics have been developed; overall these heuristics can be characterized as being geometric based (using the geometric positions of mesh points) or graph based (using the connectivity structure of the mesh). Mesh migration methods have also been proposed; however, it is generally more efficient in practice to completely repartition the mesh when the element assignments to processors becomes unbalanced. Several software systems have been developed that implement these algorithms including two widely used packages: Zoltan [15] and ParMetis [55].

1.4 A Comparison of SAMR and UAMR

Historically, SAMR and UAMR methods have been championed based on their relative strengths when used in particular application areas; a comparison of some of the relevant properties is summarized in Table 1.1. As discussed in §1.2, SAMR meshes are composed of a hierarchy of regular grids. There are several advantages to this approach compared to the UAMR approach including efficient storage of the mesh connectivity, fast and efficient computational kernels based on finite difference and finite volume methods, and the ability to easily reuse legacy code developed for Cartesian grids. One of the historical disadvantages of the SAMR approach is that complex geometries are difficult to accurately model, although this is being addressed with current research highlighted in §1.5. In constrast, complex geometries are typically easier to model using UAMR methods because of the increased flexibility in mesh generation. In addition, because all elements are at the same

Table 1.1. *A comparison of SAMR and UAMR methods*

Property	SAMR	UAMR
Mesh Configuration	Hierarchy of grids	Single level of elements
Mesh Connectivity	Regular: Implicit *ijk* connectivity	Irregular: explicit connectivity information must be stored
Discretization Method	Typically finite difference/ finite volume	Typically finite element/ finite volume
Kernel Efficiency	Typically highly efficient use of contiguous arrays	Typically less efficient because of indirect addressing
Refinement Issues	Internal boundary discretization	Maintaining mesh quality
Over-Refinement Potential	Clustering to form patches	Propagation to maintain conforming meshes
Decomposition Granularity	Blocks of many elements	Individual elements
Partitioning Heuristics	Space filling curves and bin packing	Graph- or geometry-based schemes
Repartitioning Strategy	Create a new level and move the data	Repartition entire mesh or migration strategies

level, legacy physics modules that incorporate finite-element and finite-volume discretization schemes can often be used with no modification when placed into an UAMR framework. However, because the mesh is unstructured, it often requires more resources for storage of the connectivity and geometry information and the computational kernels are typically not as efficient because indirect addressing is often used.

In both SAMR and UAMR methods the grid can be over-refined, and more elements subdivided than necessary according to the error estimation process. In the case of SAMR meshes, this is due to the need to create rectangular patches; for UAMR meshes this is due to the need to propagate the refinement to create conforming meshes. Each technique also has discretization issues that must be considered in the refinement process. For example, for UAMR meshes, certain techniques for dividing elements can lead to elements with arbitrarily small angles which can adversely affect discretization accuracy. For SAMR meshes, special discretization techniques must be used at the internal coarse/fine interfaces to maintain stability and accuracy.

Parallelization of the two methods offer different challenges. Parallel SAMR methods work with grid blocks which can contain hundreds or thousands of elements. Typically space-filling or bin-packing algorithms are used to partition the data and when a new level is created, the blocks are distributed to the processors of the parallel computer. Because the level of granularity is large, there can be chal-

lenges associated with achieving good load balance on a parallel computer when the number of blocks does not greatly exceed the number of processors. This is much less of a problem for UAMR methods because partitioning is done on the level of individual elements. However, when a UAMR mesh is refined, it is not a matter of distributing the newly created data; typically the entire mesh is repartitioned or mesh migration strategies are used to maintain data locality and good load balance.

1.5 Recent Advances and Future Research Directions

To address new simulation problems, researchers are developing new algorithms that combine AMR with other solution methods. In §1.5.1, we discuss some of these approaches and the issues they present for parallel computing. To make the development of new methods easier in the future, researchers are developing tools to enable software to interoperate in new ways. In §1.5.2, we describe an effort to define a common abstract data model and interfaces for meshing and discretization tools. In §1.5.3, we discuss issues associated with scaling of AMR applications on new parallel architectures.

1.5.1 New AMR algorithms and parallel computing issues

AMR technology is being combined with other numerical techniques to expand the range of applicability of adaptive grid methods. Examples include combining AMR with ALE hydrodynamics methods [3] and coupling continuum and atomistic models via AMR to increase the range of applicability of those models. Other efforts combine structured AMR with embedded boundary methods or overlapping grids to treat problems with complex geometries. We highlight a few recent developments and discuss new research issues they raise.

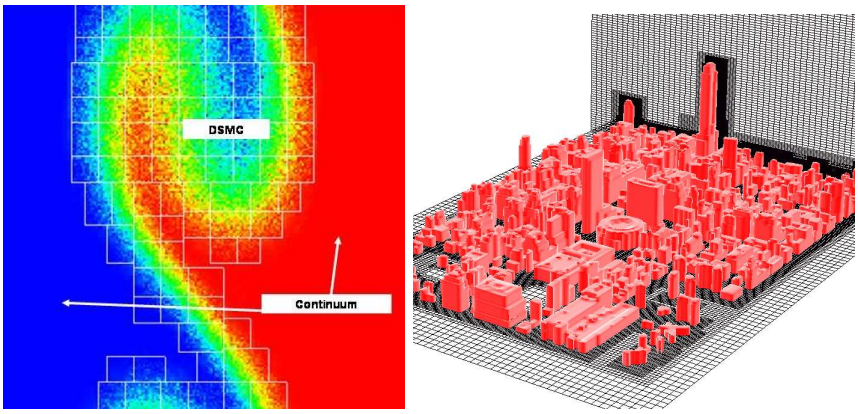

Figure 1.7. *Examples of new AMR applications. The left figure shows a continuum-atomistic hybrid coupling using AMR. The right figure shows an embedded boundary SAMR mesh around buildings in Manhattan used for flows in urban environments.*

Traditionally, AMR has been used to increase grid resolution for numerical methods based on continuum equations. However, many continuum models lack key physical processes that occur at fine scales and which are most accurately modeled using atomistic methods. However, atomistic schemes are often too expensive to apply globally in a computation. AMR techniques can be used to focus the application of atomistic methods in regions where they are needed [28, 54, 61]. The picture on the left in Figure 1.7 illustrates an approach where a continuum fluid model is coupled to a Direct Simulation Monte Carlo (DSMC) model using AMR to resolve molecular diffusion along a multi-fluid interface.

Another avenue of development that is being applied to problems in aerospace, geophysics, biology, and others combines AMR with embedded boundaries [1, 43, 45, 49]. These methods provide a viable alternative to unstructured and mapped grid methods for applications involving complex geometries. The picture on the right in Figure 1.7 shows an adaptive embedded boundary grid around a complex cityscape used for atmospheric calculations. Another technique for treating complex geometries with structured grids is to combine AMR with overlapping curvilinear grids [31]. This approach yields an accurate boundary representation similar to unstructured grids, yet allows structured grid numerical methods to accurately resolve features near the boundaries. Recently, this approach has been used to solve reactive flows in engineering geometries [32].

Apart from numerical challenges, these and other new applications of AMR technology add new considerations for parallel computing. For example, continuum-particle hybrids combine irregular particle data data structures and standard array-based data, and embedded boundary methods employ additional data in grid regions near the boundary. These approaches introduce additional complexity for dynamic load balancing, due to spatially *non-uniform* workloads, and parallel data communication. Traditionally, many load balancing schemes have assumed that the amount of work required to update the solution in each grid cell is uniform over the entire grid. However, the number of particles associated with a grid cell in an atomistic method typically varies based on local physics. Also, embedded boundary methods require extra numerical operations near boundaries to update the solution. New strategies for balancing non-uniform workloads are being developed to treat these issues. The unstructured grid research community has developed a number of useful schemes for partitioning non-uniform workloads and these techniques may prove useful for structured grid partitioning strategies.

1.5.2 Software Interoperability

As applications and algorithms become increasingly complex, there is a growing need to combine tools developed by different groups into a single application code. Most of the tools for parallel adaptive mesh refinement highlighted in §1.2 fall into the *software framework* category as they provide an overarching infrastructure for AMR computations. Application scientists must buy into the entire framework to take advantage of the meshing services provided. A drawback of this approach is that it can be difficult to incorporate new or complementary tools developed outside the framework or to experiment with different approaches to determine which is best

suited for a given application. In general, interchanging meshing technologies, or combining different technologies together, is often a labor intensive and error prone process that results in a lengthy diversion from the central scientific investigation.

The recognition of this fundamental problem has led several groups to develop the technologies needed to create inter-operable and interchangeable *software components* for scientific computing. Components interact with each other only through well defined interfaces and are a logical means of facilitating the development of complex applications using software developed by independent groups. Three of the most notable examples of scientific component development are the Cactus Code group [30], the Common Component Architecture Forum [20], and the Terascale Simulation Tools and Technologies (TSTT) DOE SciDAC center [29]. The former two develop general environments for software components for scientific computing; the latter effort is focused on the development of meshing and discretization components, and we highlight that effort here.

The TSTT data model covers a broad spectrum of mesh types and functionalities, ranging from a non-overlapping, connected set of entities to a collection of meshes that may or may not overlap to cover the computational domain. A key aspect of the TSTT approach is that it does not enforce any particular data structure or implementation within the components, only that certain questions about the mesh data can be answered through calls to the component interface. The challenges inherent in this type of effort include balancing performance of the interface with the flexibility needed to support a wide variety of mesh types. Using these mesh component interfaces, TSTT researchers have developed new hybrid AMR technologies such as SAMR/front-tracking algorithms used in diesel jet spray break-up simulations and astrophysics calculations and insertable, parallel UAMR libraries used in accelerator design.

1.5.3 Scalability

AMR codes have shown that they scale effectively to $O(1000)$ processors, as previously discussed. In particular, AMR is generally considered to be effective on cluster-based parallel systems in common use today. However, considerable challenges exist for AMR application development to utilize new large-scale parallel architectures such as the IBM BlueGene/L at Lawrence Livermore National Laboratory and the Cray Red Storm at Sandia National Laboratory. These platforms have at least an order of magnitude more processors (30 to 120 thousand), with less memory per processor, than systems in routine use today. Algorithms used to dynamically load balance adaptive applications will have to be extended to efficiently operate on these larger processor systems. At the same time, current data decompositions used to maintain adaptive grid structures will no longer be acceptable within the small amount of memory available.

As AMR researchers begin to address scaling on ever larger parallel computer systems, it is likely that technologies and algorithms from different AMR methodologies will merge. For example, researchers have developed data structures based on graphs and trees, typically used in tree-based and unstructured AMR, to better manage spatial locality in patch-based AMR [64]. These schemes have significantly

improved the efficiency and scaling of communication operations on AMR grids. Other graph-based partitioning algorithms, which have traditionally been used in unstructured AMR, are being explored for tree-based SAMR [56]. These examples show that algorithms developed for one AMR approach can be used to improve the performance of another. The ultimate challenge for both SAMR and UAMR methods will be to achieve the high degree of scalability necessary to take full advantage of a new generation of massively parallel computers.

1.6 Conclusions

Adaptive Mesh Refinement (AMR) methods have proven to be a powerful technique for scientific simulations whose solutions exhibit dynamic, localized features. Two broadly defined approaches, Structured AMR (SAMR) and Unstructured AMR (UAMR) have developed their own set of algorithms and computational techniques. However, with the increasing availability of high-performance parallel computers, there is significant interest in the development of interoperable parallel algorithms and software to support these AMR methods.

In this paper we have reviewed the current state-of-the-art for both parallel SAMR and UAMR algorithms and existing software implementations. One of the most interesting current areas of research in these algorithms has been the crossover of partitioning techniques commonly used for UAMR methods to the SAMR community. With the use of large numbers (1,000s) of processors, the development of methods that are scalable for such machines is essential. A final important area of work has been to develop interoperable software as exemplified by the SciDAC TTST project. As scientists consider more complex multi-scale, multi-physics simulations, the ability to coordinate software efforts with this sort of interoperability will become increasingly more important.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract number W-7405-Eng-48. This work was partially supported by NSF grants DGE-9987589, CTS-0121573, EIA-0202007, and ACI-0305743.

Bibliography

[1] M. Aftosmis, J. Melton, and M. Berger, *Adaptation and surface modeling for cartesian mesh methods*, in Proceedings of the 12th AIAA Computational Fluid Dynamics Conference, San Diego, CA, 1995. AIAA Paper 95-1725.

[2] A. Almgren, J. Bell, P. Colella, L. Howell, and M. Welcome, *A conservative adaptive projection method for the variable density incompressible Navier-Stokes equations*, Journal of Computational Physics, 142 (1998), pp. 1–46.

[3] R. W. Anderson, N. S. Elliott, and R. B. Pember, *An arbitrary Lagrangian-Eulerian method with adaptive mesh refinement for the solution of the Euler equations*, Journal of Computational Physics, 199 (2004), pp. 598–617.

[4] I. Babuška and W. C. Rheinboldt, *Error estimates for adaptive finite element computations*, SIAM Journal of Numerical Analysis, 15 (1978), pp. 736–754.

[5] S. B. Baden, *The KeLP programming system*. See `http://www-cse.ucsd.edu/groups/hpcl/scg/kelp.html`.

[6] R. E. Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations. Users' Guide 6.0*, SIAM Publications, Philadelphia, PA, 1990.

[7] R. E. Bank, A. H. Sherman, and A. Weiser, *Refinement algorithms and data structures for regular local mesh refinement*, in Scientific Computing, R. Stepleman et al., ed., IMACS/North-Holland Publishing Company, Amsterdam, 1983, pp. 3–17.

[8] W. J. Barry, M. T. Jones, and P. E. Plassmann, *Parallel adaptive mesh refinement techniques for plasticity problems*, Advances in Engineering Software, 29 (1998), pp. 217–229.

[9] J. Bell and C. Rendleman, *CCSE Application Suite*. See `http://seesar.lbl.gov/CCSE/Software/index.html`.

[10] J. B. Bell, M. S. Day, C. A. Rendleman, S. E. Woosley, and M. A. Zingale, *Adaptive low mach number simulations of nuclear flame microphysics*, Journal of Computational Physics, 195 (2004), pp. 677–694.

[11] J. B. Bell, M. S. Day, I. G. Shepherd, M. Johnson, R. K. Cheng, V. E. Beckner, M. J. Lijewski, and J. F. Grcar, *Numerical simulation of a laboratory-scale turbulent v-flame*, Tech. Rep. LBNL-54198, Lawrence Berkeley National Laboratory, 2003.

[12] M. Berger and P. Colella, *Local adaptive mesh refinement for shock hydrodynamics*, Journal of Computational Physics, 82 (1989), pp. 64–84.

[13] M. J. Berger and J. Oliger, *Adaptive mesh refinement for hyperbolic partial differental equations*, Journal of Computational Physics, (1984), pp. 484–512.

[14] M. W. Bern and P. E. Plassmann, *Mesh generation*, in Handbook of Computational Geometry, J. Sack and J. Urrutia, eds., Elsevier Scientific, 2000, pp. 291–332.

[15] E. Boman, K. Devine, R. Heaphy, B. Hendrickson, W. Mitchell, M. St. John, and C. Vaughan, *Zoltan home page*. See `http://www.cs.sandia.gov/Zoltan`, 1999.

[16] F. J. Bossen and P. S. Heckbert, *A pliant method for anisotropic mesh generation*, in 5th Intl. Meshing Roundtable, Oct. 1996, pp. 63–74. See `http://www.cs.cmu.edu/~ph`.

[17] G. L. Bryan, T. Abel, and M. L. Norman, *Achieving extreme resolution in numerical cosmology using adaptive mesh refinement: resolving primordial star formation*, in Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM), Denver, CO, 2001, ACM.

[18] M. L. C.A. Rendleman, V.E. Beckner, *Parallelization of an adaptive mesh refinement method for incompressible fluid flows*, International Parallel and Distributed Processing Symposium, (2000).

[19] A. C. Calder, B. C. Curtis, L. J. Dursi, B. Fryxell, G. Henry, P. MacNeice, K. Olson, P. Ricker, R. Rosner, F. X. Timmes, H. M. Tufo, J. W. Turan, and M. Zingale, *High performance reactive fluid flow simulations using adaptive mesh refinement on thousands of processors*, in Proceedings of Supercomputing '00, IEEE Computer Society Press, 2000.

[20] *CCA Forum homepage*, 2004. See `http://www.cca-forum.org/`.

[21] P. Colella, D. Graves, T. Ligocki, D. Martin, D. Serafini, and B. V. Straalen, *Chombo software package for amr applications design document*, report, Applied Numerical Algorithms Group, NERSC Division, Lawrence Berkeley National Laboratory, Berkeley, CA, September 2003. See `http://seesar.lbl.gov/ANAG/chombo/index.html`.

[22] M. S. Day and J. B. Bell, *Numerical simulation of laminar reacting flows with complex chemistry*, Combust. Theory Modeling, 4 (2000), pp. 535–556.

[23] M. R. Dorr, F. X. Garaizar, and J. A. F. Hittinger, *Simulation of laser plasma filamentation using adaptive mesh refinement*, Journal of Computational Physics, 177 (2002), pp. 233–263.

[24] J. E. Flaherty, R. M. Loy, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz, *Adaptive local refinement with octree load-balancing for the parallel solution of three-dimensional conservation laws*, Journal of Parallel and Distributed Computing, 47 (1997), pp. 139–152.

[25] L. Freitag and C. Ollivier-Gooch, *Tetrahedral mesh improvement using face swapping and smoothing*, International Journal for Numerical Methods in Engineering, 40 (1997), pp. 3979–4002.

[26] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo, *Flash: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes*, Astrophysical Journal Supplement, 131 (2000), pp. 273–334.

[27] X. Garaizar and J. Trangenstein, *Adaptive mesh refinement and front tracking for shear bands in granular flow*, SIAM Journal on Scientific Computing, 20 (1999), pp. 750–779.

[28] A. L. Garcia, J. B. Bell, W. Y. Crutchfield, and B. J. Alder, *Adaptive mesh and algorithm refinement*, Journal of Computational Physics, 154 (1999), pp. 134–155.

[29] J. Glimm, D. Brown, and L. Freitag, *Terascale Simulation Tools and Technologies (TSTT) Center*, 2001. See `http://www.tstt-scidac.org`.

[30] T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf, *The cactus framework and toolkit: Design and applications*, in Vector and Parallel Processing - VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science, 2002. also available at `http://www.cactuscode.org/Showcase/Publications.html`.

[31] W. D. Henshaw, *Overture: An object-oriented framework for overlapping grid applications*, in Proceedings of the 32nd American Institute for Aeronautics Fluid Dynamics, St. Louis, MO, June 24-27 2002. See `http://www.llnl.gov/CASC/Overture`.

[32] W. D. Henshaw and D. W. Schwendeman, *An adaptive numerical scheme for high-speed reactive flow on overlapping grids*, Journal of Computational Physics, 191 (2003), pp. 420–447.

[33] R. D. Hornung and S. R. Kohn, *Managing application complexity in the samrai object-oriented framework*, Concurrency and Computation: Practice and Experience, 14 (2002), pp. 347–368. See also `http://www.llnl.gov/CASC/SAMRAI`.

[34] L. H. Howell and J. Greenough, *Radiation diffusion for multi-fluid eulerian hydrodynamics with adaptive mesh refinement*, Journal of Computational Physics, 184 (2003), pp. 53–78.

[35] S. Jindal, L. Long, and P. Plassmann, *Large eddy simulations for complex geometries using unstructured grids*, in Proceedings of the 34th AIAA Fluid Dynamics Conference, Portland, OR, 2004. AIAA Paper 2004-2228.

[36] M. T. Jones and P. E. Plassmann, *Adaptive refinement of unstructured finite-element meshes*, Finite Elements in Analysis and Design, 25 (1997), pp. 41–60.

[37] ——, *Parallel algorithms for adaptive mesh refinement*, SIAM Journal on Scientific Computing, 18 (1997), pp. 686–708.

[38] R. Klein, J. Bell, R. Pember, and T. Kelleher, *Three dimensional hydrodynamic calculations with adaptive mesh refinement of the evolution of Rayleigh Taylor and Richtmyer Meshkov instabilities in converging geometry: Multi-mode perturbations*, in Proceedings of the 4th International Workshop on Physics of Compressible Turbulent Mixing, Cambridge, England, March 1993, 1993.

[39] Z. Lan, V. Taylor, and G. Bryan, *Dynamic load balancing for adaptive mesh refinement applications: Improvements and sensitivity analysis*, Computer Physics Communications, 126 (2000), pp. 330–354.

[40] A. Liu and B. Joe, *Quality local refinement of tetrahedral meshes based on bisection*, SIAM Journal on Scientific Computing, 16 (1995), pp. 1269–1291.

[41] R. Löhner, *An adaptive finite element scheme for transient problems in CFD*, Computer Methods in Applied Mechanics and Engineering, 61 (1987), pp. 323–338.

[42] P. MacNeice, K. M. Olson, C. Mobarry, R. deFainchtein, and C. Packer, *Paramesh : A parallel adaptive mesh refinement community toolkit*, Computer Physics Communications, 126 (2000), pp. 330–354. See `http://ct.gsfc.nasa.gov/paramesh/Users_manual/amr.html`.

[43] P. McCorquodale, P. Colella, and H. Johansen, *A cartesian grid embedded boundary method for the heat equation on irregular domains*, Journal of Computational Physics, 173 (2001), pp. 620–635.

[44] W. F. Mitchell, *A comparison of adaptive refinement techniques for elliptic problems*, ACM Transactions on Mathematical Software, 15 (1989), pp. 326–347.

[45] D. Modiano and P. Colella, *A higher-order embedded boundary method for time-dependent simulation of hyperbolic conservation laws*, in Proceedings of the ASME 2000 Fluids Engineering Division Summer Meeting, Boston, MA, June 2000. ASME Paper FEDSM00-11220.

[46] M. Parashar, *GrACE–Grid Adaptive Computational Engine*. See `http://www.caip.rutgers.edu/~parashar/TASSL/`.

[47] M. Parashar and J. C. Browne, *System engineering for high performance computing software: The hdda/dagh infrastructure for implementation of parallel structured adaptive mesh refinement*, in IMA Volume on Structured Adaptive Mesh Refinement Grid Methods, 2000, pp. 1–18.

[48] ——, *On partitioning dynamic adaptive grid hierarchies*, in Proceedings of the 29th Annual Hawaii International Conference on System Sciences, Maui, HI, January, 1996, IEEE Computer Society Press, pp. 604–613.

[49] R. Pember, J. Bell, P. Colella, W. Crutchfield, and M. Welcome, *An adaptive cartesian grid method for unsteady compressible flow in irregular regions*, Journal of Computational Physics, 120 (1995), pp. 278–304.

[50] R. Propp, P. Colella, W. Crutchfield, and M. Day, *A numerical model for trickle-bed reactors*, Journal of Computational Physics, 165 (2000), pp. 311–333.

[51] J.-F. Remacle and M. Shephard, *An algorithm oriented mesh database*, International Journal for Numerical Methods in Engineering, 58 (2003), pp. 349–374.

[52] M.-C. Rivara, *Algorithms for refining triangular grids suitable for adaptive and multigrid techniques*, International Journal for Numerical Methods in Engineering, 20 (1984), pp. 745–756.

[53] M.-C. Rivara and C. Levin, *A 3-D refinement algorithm suitable for adaptive and multigrid techniques*, Communications in Applied Numerical Methods, 8 (1992), pp. 281–290.

[54] R. E. Rudd and J. Q. Broughton, *Concurrent coupling of length scales in solid state systems*, Phys Status Solidi B, 217 (2000), p. 251.

[55] K. Schloegel, G. Karypis, and V. Kumar, *ParMETIS*, 1999. See `http://www-users.cs.umn.edu/~karypis/metis/parmetis/index.html`.

[56] J. Steensland, S. Chandra, and M. Parashar, *An application centric characterization of domain-based sfc partitioners for parallel samr*, IEEE Transactions on Parallel and Distributed Systems, (2002), pp. 1275–1289.

[57] M. Sussman, A. Almgren, J. Bell, P. Colella, L. Howell, and M. Welcome, *An adaptive level set approach for incompressible two-phase flows*, Journal of Computational Physics, 148 (1999), pp. 81–124.

[58] J. Trangenstein and Z. Bi, *Multi-scale iterative techniques and adaptive mesh refinement for flow in porous media*, Advances in Water Resources, 25 (2002), pp. 1175–1213.

[59] J. A. Trangenstein, *Adaptive mesh refinement for wave propagation in nonlinear solids*, SIAM Journal on Scientific and Statistical Computing, 16 (1995), pp. 819–839.

[60] H. E. Trease, L. L. Trease, and J. D. Fowler, *The P3D Code Development Project*. See `http://www.emsl.pnl.gov/nwgrid/`.

[61] H. S. Wijesinghe, R. D. Hornung, A. L. Garcia, and N. G. Hadjiconstantinou, *3-dimensional hybrid continuum-atomistic simulations for multiscale hydrodynamics*, J. Fluid Eng., 126 (2004), pp. 768–777.

[62] R. Williams, *DIME: Distributed Irregular Mesh Environment*, California Institute of Technology, 1990.

[63] ——, *A dynamic solution-adaptive unstructured parallel solver*, Report CCSF-21-92, Caltech Concurrent Supercomputing Facilities, California Institute of Technology, Pasadena, Calif., 1992.

[64] A. M. Wissink, D. Hysom, and R. D. Hornung, *Enhancing scalability of parallel structured AMR calculations*, in Proceedings of the 17th ACM International Conference on Supercomputing (ICS03), San Francisco, June 2003, pp. 336–347.