Engineering with Computers (2006) 22:181–195
DOI 10.1007/s00366-006-0038-6

ORIGINAL ARTICLE

Managing complex data and geometry in parallel structured AMR applications

Richard D. Hornung · Andrew M. Wissink · Scott R. Kohn

Received: 19 April 2005 / Accepted: 1 February 2006 / Published online: 30 August 2006

Abstract Adaptive mesh refinement (AMR) is an increasingly important simulation methodology for many science and engineering problems. AMR has the potential to generate highly resolved simulations efficiently by dynamically refining the computational mesh near key numerical solution features. AMR requires more complex numerical algorithms and programming than uniform fixed mesh approaches. Software libraries that provide general AMR functionality can ease these burdens significantly. A major challenge for library developers is to achieve adequate flexibility to meet diverse and evolving application requirements. In this paper, we describe the design of software abstractions for general AMR data management and parallel communication operations in SAMRAI, an object-oriented C++ structured AMR (SAMR) library developed at Lawrence Livermore National Laboratory (LLNL). The SAMRAI infrastructure provides the foundation for a variety of diverse application codes at LLNL and elsewhere. We illustrate SAMRAI functionality by describing how its unique features are used in these codes which employ complex data structures and geometry. We highlight capabilities for moving and deforming meshes, coupling multiple SAMR mesh hierarchies, and immersed and embedded boundary methods for modeling complex geometrical features. We also describe how irregular data structures, such as particles and internal mesh boundaries, may be implemented using SAMRAI tools without excessive application programmer effort.

Keywords Adaptive mesh refinement · Parallel computing · Complex geometry

This work was performed under the auspices of the US Department of Energy by University of California Lawrence Livermore National Laboratory under contract number W-7405-Eng-48 and is released under UCRL-JRNL-214559.

R. D. Hornung (✉) · A. M. Wissink · S. R. Kohn
Center for Applied Scientific Computing,
Lawrence Livermore National Laboratory,
P.O. Box 808, L-561, Livermore, CA 94551, USA
e-mail: hornung@llnl.gov

A. M. Wissink
e-mail: awissink@llnl.gov

S. R. Kohn
e-mail: skohn@llnl.gov

1 Introduction

In recent years, the availability of large-scale parallel computing resources has increased and numerical models and algorithms have advanced substantially. As a result, computer simulation has become a common tool for studying many science and engineering problems. Typical numerical models require the approximation of governing equations for a physical system on a discrete domain, or mesh. Mesh spacing is an important factor determining the accuracy and cost of a computation. Many problems give rise to numerical solutions with key features that require very fine meshes to resolve adequately. Often, such features reside in localized regions of a computational domain and are separated by large regions where the solution may be adequately represented on a coarser mesh. Adaptive mesh refinement (AMR) is a simulation methodology for dynamically increasing spatial (and often temporal) mesh resolution near key features. By focusing memory and computational resources, highly resolved

simulations may be achieved more efficiently than if the entire mesh is refined uniformly.

Despite the potentially large savings in memory and execution time that AMR offers, the range of computing applications in which it is used routinely is limited. This is due largely to the complexity of numerical algorithms that are required and difficulties in modifying AMR support software to meet new application requirements. Primary issues involve managing application data, which often varies for different problems, on irregular, locally-refined mesh configurations and achieving efficient parallel data (re)distribution and load balancing.

To support a variety of AMR development, several software libraries have emerged in recent years. These efforts have expanded the range of problems to which AMR is applied. Building a flexible and robust software infrastructure requires a substantial investment. So that the benefits of software reuse surpass the investment of development, the software must be designed to treat diverse computational problems or evolving requirements within a particular application area.

Basic linear algebra subroutines (BLAS) and linear and nonlinear solver packages are examples of general software libraries that are important tools in the numerical simulation community. By providing efficient, reusable algorithms, such libraries increase developer productivity and application performance. Typical libraries of this sort provide simulation codes with callable subroutines that perform specific computational tasks within a larger algorithmic structure. They impose few restrictions on an application code and often require simple data structures, such as arrays, for interoperability. In contrast, AMR libraries are much more invasive within the structure of an application. They control the mesh structure and perform complicated manipulations of data on the mesh. Thus, simulation codes are usually built up from data structures provided by these libraries. When appropriately designed, however, AMR libraries facilitate a variety of application development. Substantial decoupling between general adaptive meshing functionality and application-specific data and operations is both practical and beneficial.

In this paper, we describe the object-oriented software design and implementation of the data management and parallel communication infrastructure at the core of the SAMRAI structured AMR library developed at Lawrence Livermore National Laboratory (LLNL). The SAMRAI framework has achieved substantial reuse of software design and implementation across a wide range of applications. We show how complex geometrical features and irregular user-defined data types may be used in an AMR setting by describing how recently developed application codes use SAMRAI. Since new applications often require new algorithms and software functionality, we emphasize how SAMRAI can be extended and specialized to meet application needs.

We begin with a brief overview of the basic aspects of parallel structured AMR algorithms and support libraries. Then, we describe the design of the parallel data management and communication infrastructure in SAMRAI. Next, we illustrate various ways that SAMRAI is used by discussing implementation issues associated with various applications.

2 Structured AMR background

Structured AMR (SAMR) is a particular AMR methodology in which the computational mesh is comprised of structured mesh components. Most SAMR software and many SAMR algorithms are based on the work of Berger et al. [1, 2]. These researchers introduced techniques for using conservative shock capturing methods in the context of SAMR. This work is the basis for much of the evolution of SAMR development over the past two decades; see [3] for a brief survey.

In a typical SAMR code, the mesh consists of a hierarchy of levels, each of which corresponds to a single ''uniform'' degree of mesh spacing. Within the hierarchy, the levels are nested. The coarsest level covers the entire computational domain and each successively finer level covers a portion of the interior of the next coarser level. The mesh on each level is composed of a disjoint union of logically-rectangular regions, often called *patches* or *blocks*. Commonly, simulation data are stored in contiguous arrays associated with each patch so that the data map directly to the mesh without indirection. Mesh adaptivity involves selecting cells to refine on a given level, using some error estimation or feature detection criteria, and then clustering those cells into patch regions. These patch regions are used to form the next finer level in the hierarchy. Figure 1 shows the sort of mesh that is generated in a SAMR hydrodynamics calculation with highly-localized features.

Most SAMR algorithms employ numerical routines designed to treat data associated with an arbitrary patch in the mesh hierarchy. The computation is organized as a collection of numerical operations performed on the distributed patch regions and communication operations that pass information be-

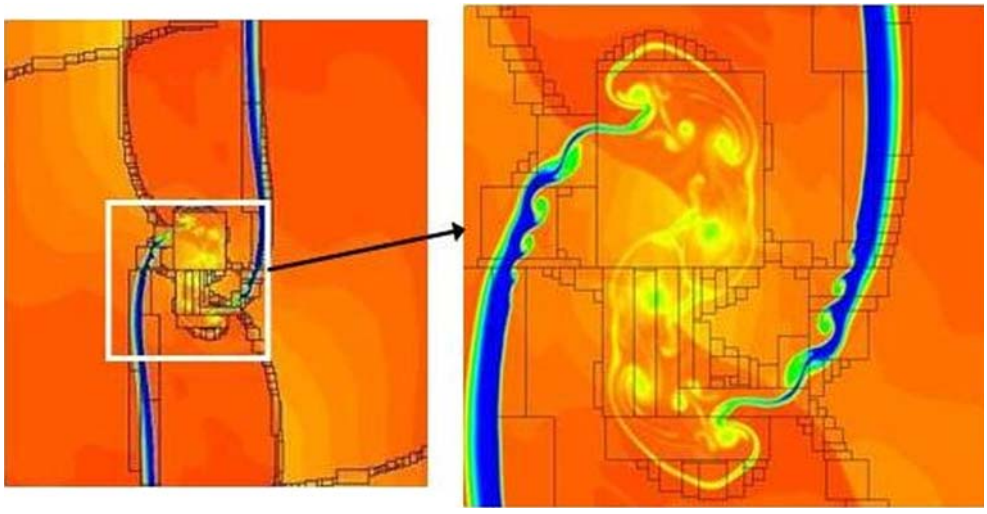

Fig. 1 An example SAMR patch configuration used to resolve an impulsively sheared contact surface in an Eulerian hydrodynamics simulation courtesy of Bob Anderson [5]. The figure on the *left* shows the local concentration of fine mesh level patches within the larger computational domain. The figure on the *right* shows finer detail in a localized region. The *black lines* are the finest level patch boundaries

tween those regions; for example, to fill ''ghost cells'' at patch boundaries. Numerical operations must account for internal boundaries between levels properly to produce an accurate, consistent solution across the AMR hierarchy. Communication operations must exchange data among irregular patch configurations within a single patch level as well as between different levels of mesh resolution using data refinement and coarsening. Multi-physics applications and problems involving complex geometry introduce additional complications. These problems typically combine mesh-based array data and irregular unstructured data such as particles, and require multiple solution procedures that share data and which use distinct data communication patterns.

Since the mesh may change frequently in an adaptive calculation, computational overheads that cannot be amortized over an entire simulation are encountered. Non-adaptive codes usually incur the cost of grid generation, load balancing, and data communication dependency construction once. During an SAMR calculation, the mesh and patch configuration must be reconstructed and re-distributed, and the resulting data dependencies must be re-computed often. Efficiency of these operations is paramount so that adaptive gridding overheads are acceptable on large numbers of processors [4].

3 Structured AMR software libraries

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. In addition to SAMRAI, some well-known libraries are: Chombo [6] and BoxLib-based AMR software [7] from Lawrence Berkeley National Laboratory, GrACE [8] from Rutgers University, PARAMESH [9] from NASA Goddard, and Overture [10] from LLNL.

Each of these libraries provides software tools for treating parallel data decomposition and distribution on an adaptive patch hierarchy. At a basic level, the functionality of these libraries is similar. However, they differ substantially in their design and features, especially in terms of support for complex geometry and irregular, non-array data types, and parallel data communication infrastructure.

In BoxLib, Chombo, SAMRAI, and Overture, the patch configuration is maintained via integer lattice index space relationships. With the exception of Overture, these libraries are similar in the support they provide for basic SAMR algorithms and distributed array-based mesh data. Overture, built on P++ [11] which supports whole array operations on distributed parallel arrays, provides operations for overlapping configurations of various mesh types. BoxLib is a pioneering SAMR software effort serving as the basis for an impressive history of algorithm development for fluid dynamics problems. Support in other SAMR libraries for finite difference calculations on logically rectangular arrays is typically based on concepts found BoxLib. Chombo represents an extension of BoxLib by adding facilities for other SAMR calculations such as those involving embedded boundaries. GrACE and PARAMESH also employ a hierarchy of logically rectangular blocks, but use tree-based data structures to organize the hierarchy. GrACE developers have extensively researched dynamic partitioning and runtime management for SAMR calculations [12, 13].

Chombo is perhaps the most similar to SAMRAI of the aforementioned libraries, although it differs

significantly from SAMRAI in its design and implementation. The main difference that is germane to this paper involves data management and communication operations. Chombo provides a container class for data on a patch hierarchy level that is templated on the mesh data type [6]. Various classes supply specific interlevel data coarsening or refining operations for the data associated with a container object. In particular, parallel data transfers are defined and performed for each data container object individually. When a new data coarsening or refining operation is needed, a new class must be built to perform the interlevel communication.

In SAMRAI, a patch (rather than a level) is the fundamental container which holds all data objects associated with a box region of index space. Interlevel data communication operations are implemented using C++ inheritance-based specialization and extension mechanisms. This allows SAMRAI to support an arbitrary number of different data types and operations within a single communication schedule. Also, new user-defined data types and interlevel transfer operations can be combined with data types and operations supplied by SAMRAI without changing the library.

Previously, we have discussed performance overheads in AMR due to adaptive meshing and algorithms we have developed to improve SAMRAI performance [4], and object-oriented design patterns used in SAMRAI and how they enable algorithm reuse across different applications [14]. In the remainder of this paper, we describe key aspects of the SAMRAI data management and communication infrastructure and illustrate how these capabilities are used in various applications.

4 SAMRAI data management and communication infrastructure

In this section, we describe the design and implementation of software in SAMRAI for managing simulation data and communication. In [14], we discussed object-oriented software design patterns, such as Strategy and Factory [15], used in SAMRAI to achieve a loose coupling between application-specific routines and more general adaptive meshing and time integration operations provided by the library. The following discussion shows how similar design concepts are applied to achieve a flexible, parallel AMR data communication infrastructure.

Generally, an SAMR simulation consists of a sequence of interleaved processes involving numerical computations on patches and interpatch data communication. The interpatch data transfers required for each communication phase are defined by what data is needed to execute subsequent numerical operations. Examples of such communication scenarios are: filling ghost cells at patch boundaries before advancing the solution on patches, filling new patches after a new level is created during remeshing, coarsening data between levels to ensure consistency of the numerical solution across hierarchy levels, and summing values at patch boundaries when constructing a finite element stiffness matrix. SAMRAI is designed to support these and other data communication patterns within a common set of C++ classes that are independent of the data types and interpatch data transfer operations involved.

We note that a key SAMRAI design goal is to express each data communication phase of a computation similarly regardless of the types and number of data quantities involved or the details associated with particular data movement operations. Rather than communicate individual data quantities one-at-a-time, which is also possible, each communication procedure in SAMRAI represents an aggregation of all operations required move all the data involved in a communication phase of a calculation. To this end, each communication scenario is defined by specifying the complete set of data quantities involved and the associated operations needed (e.g., spatial coarsening, refining, time interpolation operations, etc.). These operations typically depend on the mesh coordinate system, the data type, and the numerical discretization used in an application. Thus, it is essential that the underlying software infrastructure be fairly general to allow such operations to be easily customized. To fix certain concepts before we describe the SAMRAI communication framework, we briefly describe essential mesh and patch data structures used to define a SAMR patch hierarchy.

4.1 Variable and data objects

The basic structural units of an SAMR patch hierarchy are: *patch*, *patch level* and *patch hierarchy* objects. Most, if not all, SAMR software libraries provide these software concepts, but their implementation and specific functionality differs among packages. Usually, a patch hierarchy object maintains a set of patch level objects. Each patch level object manages a collection of patch objects distributed across processors on a parallel machine. Simulation data associated with a level of mesh resolution in the hierarchy correspond to the patches that define the level.

A unique design aspect of SAMRAI is that a patch is a container for all data objects living on a logically

rectangular mesh region and that all such data is accessible via the patch. The flexibility of the SAMRAI data communication framework relies on the fact that every patch data type, whether a standard type supplied by the library or a type provided by a user, is managed by the framework in the same way. For our purposes, we consider cell-centered or node-centered array data types, for example, to be standard types whereas a user-defined type may be something like a unique particle data representation.

Each patch data object represents the instantiation of some simulation quantity on the patch region and each patch data type must obey the interface defined by the patch data base class. The SAMRAI patch data interface is a *Strategy* design pattern that defines the nominal set of operations required for the data object to be interoperable with the data management routines in the library. These operations include: data allocation over a box, data copying between two boxes on their intersection, and data packing and unpacking into and out of a message stream for parallel communication. Figure 2 illustrates the relationship between a patch and various patch data types.

There are two principle advantages to requiring the same creation mechanism and operational interface for all simulation data objects. First, all SAMRAI library routines that manipulate data on an AMR patch hierarchy are independent of the data type and are thus reusable. Second, a user may build a new application data type and essentially ''plug it in'' to the SAMRAI framework without rewriting or recompiling any library code [14].

To understand certain aspects of SAMRAI design, it is useful to distinguish between objects that are *persistent* and *dynamic* in the context of an adaptive simulation. For example, a computation may employ a variable, say ''density'', which has a fixed physical interpretation and numerical approximation. The data allocated to store density values depends on the mesh configuration and patch distribution, both of which may change frequently. Thus, a variable is viewed as persistent. Typically, it is created during problem initialization and describes the properties of a simulation quantity such as the type of the data (e.g., double, float, integer, etc.), mesh centering (e.g., node, cell, face, etc.) and its depth (i.e., the number of data values associated with each mesh point). These properties are independent of the AMR mesh configuration. The associated patch data objects that store the data values are dynamic; they are created and destroyed as the patch configuration changes. Patches share a *patch descriptor* that holds a common description of the data that may reside on a patch. The patch descriptor maintains a mapping between variables and the corresponding patch data objects. Typically, a variable creates a patch data object on any given patch using a *Factory* mechanism; see [14] for more details.

4.2 User-level data communication abstractions

In Sect. 4, we noted various data communication scenarios that may appear in SAMR applications. Such operations are composed of data transfers between data objects residing on different patches in a patch hierarchy. In parallel SAMR implementations, patches, or similar data containers, are distributed across processors. Thus, data movement involves interpro-

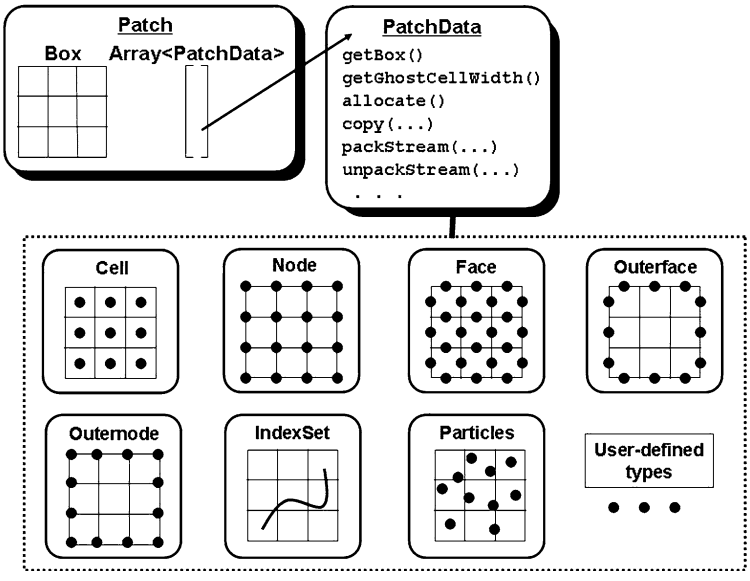

Fig. 2 A SAMRAI patch holds all data living on a box region of the mesh. Each patch data object represents the data for a simulation quantity on the patch. All patch data objects obey a common interface that defines the basic interaction between the library and the data

cessor data communication in addition to local copy operations.

Recall that a principle SAMRAI design goal is to allow users to define communication operations in a manner that represents all data motion required during a communication phase of a computation. The realization of this is that the SAMRAI communication infrastructure handles multiple data quantities and interlevel operations, including user-defined data and operations, simultaneously within a single communication operation. So that the software allows flexibility in treating user-defined data and interpatch transfer operations, a general, limited set of basic software interfaces used within the framework are exposed to an application developer. The essential elements of the SAMRAI patch data interface were described in Sect. 1. The mechanism employed by a user when providing new communication operations similarly uses C++ inheritance from abstract base classes.

A user of SAMRAI interacts with three main software abstractions during the process of defining and performing communication operations. These abstractions are: a *communication algorithm*, a *communication schedule*, and a *patch strategy*. Each abstraction appears in one of two forms indicating how it is used for inter-level data motion: there are algorithm, schedule, and patch strategy objects for coarsening data and for refining data. For completeness, we note that inter-patch communication on a single patch level or between two patch levels with the same mesh refinement (residing in different patch hierarchies, for example) is viewed as a special case of a refine operation.

A communication algorithm describes a data transfer phase of a calculation in terms of high-level mathematical concepts, such as variables and coarsen or refine operators. As such, a communication algorithm is *independent of the SAMR mesh configuration*. A communication schedule computes and stores the specific data transactions that must be performed to execute the communication described by the algorithm on a given patch configuration. Thus, a communication schedule is *dependent on the SAMR mesh configuration*. A patch strategy object provides a Strategy pattern interface for a user to supply arbitrarily complex data coarsening, refining, and physical boundary condition operations on individual patches during the execution of a communication schedule.

Typically, a communication algorithm is constructed during the problem setup phase of a simulation when variables are defined. An application developer creates the refine and coarsen algorithms needed during the calculation and registers the necessary data quantities and associated operations with each algorithm. Each communication algorithm object is later used to create, and re-create as necessary due to adaptive meshing, schedule objects to perform data motion. Coarsen and refine algorithm classes provide multiple routines for creating different schedule objects that perform interpatch communication on patch levels and patch hierarchies in various ways. In particular, a single algorithm can be used to build different schedules each of which represents a different realization of communication scenario involving the same set of data quantities and operations.

Note that the communication algorithm and communication schedule abstractions are analogous to the variable and patch data concepts described earlier. Like a variable, a communication algorithm *persists* throughout the duration of a simulation, typically. The separation of communication procedures into multiple abstractions allows an application developer to specify each distinct communication phase of a computation once in terms of high-level concepts (i.e., data quantities and operations). Communication schedules are created as needed to perform the specified data motion on a patch hierarchy. Thus, a communication schedule is a *dynamic* concept, similar to patch data, since it must be recreated when the AMR patch configuration changes. As long as the patch hierarchy remains unchanged, a schedule may be reused. Thus, the cost of computing data dependencies on an irregular AMR mesh is amortized over multiple communication cycles when possible. The notion of storing data transfer information in a schedule for parallel processing is not new. The design of SAMRAI communication schedules is based on extensions of ideas developed in other work, such as multi-block PARTI [16] and KeLP [17, 18].

The patch strategy abstraction provides an interface through which application-specific interlevel transfer operations and boundary conditions may be provided. This is useful when such operations are complex or involve functions of multiple simulation variables. For example, an application may use density and velocity variables and velocity should be interpolated so as to conserve momentum, the product of density and velocity. Arguably, such an operation is best implemented within user code since providing mechanisms to express general relationships among variables in a library is difficult to do in way that is both efficient and flexible. Using C++ inheritance, the patch strategy provides a simple mechanism for a user to provide custom interlevel transfer operations which are to be executed during communication schedule execution. Setting values at the physical domain boundary, which

is intimately tied to the numerical approximation, can be done similarly through the patch strategy interface. This scheme offers application developers myriad options for inter-level data interpolation operations and boundary conditions. We note that all functions declared in the patch strategy interfaces specify operations on single patches or transfers between two patches on the same processor. Thus, application developers are insulated entirely from the complexity of formulating data manipulations in parallel.

4.3 Data communication object decomposition

Before we discuss how SAMRAI data management and communication facilities are employed in applications, we describe key aspects of object-orientation employed in SAMRAI to illustrate how communication operations are extended and specialized. Figure 3 shows the main SAMRAI objects involved in data refinement. The object decomposition for data coarsening is similar.

To define a data refinement communication phase of a calculation, a developer creates a Refine Algorithm object and defines its behavior through a sequence of registration operations. Each registration specifies simulation data quantities to communicate and corresponding operations, such as inter-level spatial refinement. After the registration process is complete, the Refine Algorithm object can be used to create various Refine Schedule objects as needed.

To instantiate a realization of the data refinement procedures contained in the Refine Algorithm object on a particular configuration of an SAMR patch hierarchy, the developer creates a Refine Schedule object. This is done by calling one of several schedule creation methods in the Refine Algorithm class. These methods are distinguished by their patch level and patch hierarchy arguments. The schedule object can then be executed to perform the particular data communication process on the patch level(s) and/or parts of the patch hierarchy specified when it was created.

A user can optionally provide Refine Patch Strategy and Refine Transaction Factory objects to the schedule creation method. In the figure, the User Patch Strategy object denotes a user-supplied patch strategy. As described earlier, this object provides application-specific interlevel transfer and boundary condition operations that are called when the schedule executes. A transaction factory, such as the (Concrete) Transaction Factory object in the figure, is used to create transactions which describe individual inter-patch data exchanges.

During its construction, a Refine Schedule computes interpatch data dependencies needed to perform the communication operations defined by the Refine Algorithm on a given patch configuration. Each ''atomic'' data communication operation between two patch data objects is described by a Transaction object. Such operations include data copying on an overlap region or summation to accumulate values at patch boundaries necessary when constructing a finite element stiffness matrix, for example. Transaction objects are created for the Refine Schedule by a concrete transaction factory. The Factory mechanism used in the creation of interpatch data transactions allows the Refine Schedule abstraction to be adapted to new communication procedures without modification. Figure 4 illustrates the Strategy pattern relationship between the abstract transaction base class and concrete transaction objects.

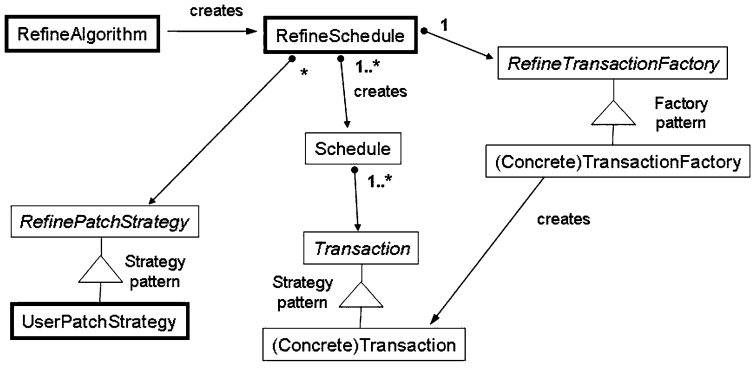

Fig. 3 Object decomposition associated with a SAMRAI data refinement communication procedure. The objects with *bold boxes* are those seen and used by a typical application developer. *Standard text* indicates a concrete object; *italicized type* indicates an interface (i.e., an abstract base class). The design pattern characterizing each instance of inheritance is noted. *Arrows* indicate object association. Each *arrow points* in the direction of usage; the *solid circle* indicates that the user owns a reference to another object. An identifier of ''1'' indicates ownership of exactly one object, ''1..*'' indicates ownership of one or more objects, and ''*'' indicates ownership of zero or more objects

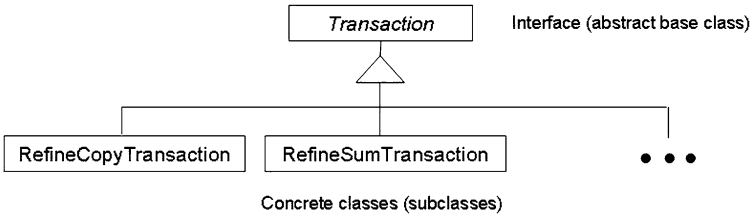

Fig. 4 Specific inter-patch data exchange operations are provided by concrete transaction objects which are derived from the abstract transaction base class

If no transaction factory is provided by the user, then a default factory that creates RefineCopyTransaction objects is used.

Transaction objects created are stored in Schedule objects. The SAMRAI Schedule class is a general abstraction providing all operations needed for message-based communication using MPI. During communication schedule execution, the set of Transaction objects owned by each Schedule define the data to be marshaled and un-marshaled for parallel communication. Actual patch data manipulation operations, such as local data copies and parallel message stream pack/unpack operations, are performed by the individual data objects, including user-defined data types, assigned to each transaction. The SAMRAI design feature whereby all patch data objects obey the same interface (recall Sect. 1) allows the entire SAMRAI data management and communication infrastructure to be generic with respect to the data.

5 New application development

In this section, we discuss several SAMR applications recently developed using SAMRAI. We use them to provide concrete illustrations of how aforementioned SAMRAI features are employed. For more information about these applications, we refer interested readers to the cited references.

5.1 ALE—AMR: moving, deforming meshes and multiblock AMR

Recently, a unique simulation code that combines Arbitrary Lagrange-Eulerian (ALE) hydrodynamics algorithms and structured AMR has been developed using SAMRAI [5]. An ALE method integrates the governing hydrodynamic equations using a Lagrangian formulation. Then, when the mesh is sufficiently deformed, it applies mesh relaxation and advection algorithms to map the solution to a new mesh configuration. ALE, by itself, is adaptive since mesh points follow flow features during the Lagrangian step. However, typical ALE codes are limited to a fixed number of mesh points during a computation. The ALE–AMR approach combines desirable aspects of ALE with complementary features of AMR. In particular, ALE–AMR exploits the benefits of ALE for evolving a multi-material system and uses AMR to dynamically add and remove mesh points for computational efficiency and enhanced accuracy. Figure 5 shows an example of a deforming locally-refined mesh produced by the ALE–AMR approach.

Arbitrary Lagrange-Eulerian methods differ from Eulerian hydrodynamics approaches, which have become commonplace for SAMR, because the mesh points themselves are solution variables. Maintaining consistency of the numerical solution on a SAMR patch hierarchy where the mesh may deform differently on different patch levels is the primary issue to address in the development of the ALE–AMR hybrid approach. In this section, we provide a brief overview of how numerical integration and interlevel data interpolation operations specific to ALE–AMR are built using the SAMRAI framework.

5.1.1 Integration and data interpolation

Managing the nodal mesh coordinates and simulation data on a mesh defined by those coordinates is a central concern for an ALE method. In SAMRAI, mesh coordinates are defined by an extensible *grid geometry* object hierarchy. Integer index space coordinates are managed in a general fashion at the lowest level of this hierarchy. The real-space mesh coordinates are designed to be customized via C++ class inheritance (i.e., Strategy design pattern). The ALE–AMR code uses this mechanism to provide its own spatial mesh coordinates which it represents using node-centered patch data.

The ALE–AMR code utilizes general adaptive algorithm and data communication functionality in SAMRAI and provides operations that are specific to ALE integration and nodal mesh coordinates. SAMRAI manages the time integration process and all adaptive meshing operations on the patch hierarchy. ALE–AMR supplies operations for integrating patch levels, including all patch-based numerical kernels.

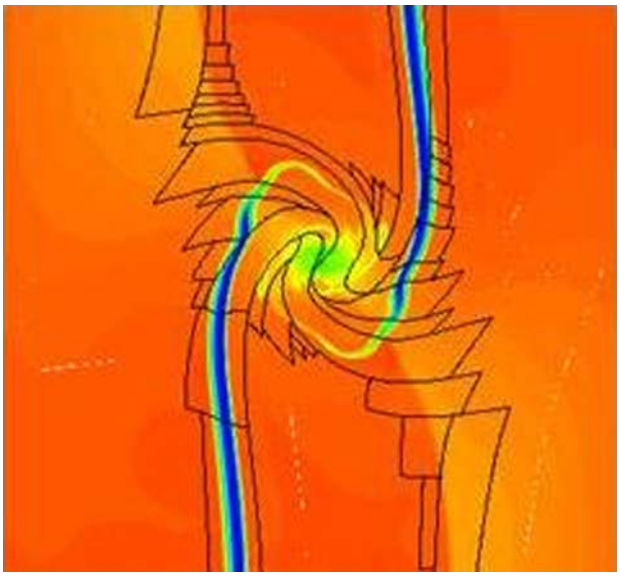

Fig. 5 The ALE–AMR code combines moving, deforming ALE meshes with structured AMR as illustrated in this calculation of an impulsively-sheared contact surface

During the execution of an ALE integration timestep, control is passed between the ALE–AMR code and the SAMRAI library multiple times, some key points of which are summarized in Table 1.

SAMRAI coordinates the parallel data communication operations for the ALE–AMR code. When a communication schedule is executed (e.g., the fillData() call in Table 1), SAMRAI performs all the necessary inter-patch data transfers. Numerical operations that refine and coarsen solution variables and set boundary conditions for patch regions defined by their nodal mesh points are provided by the ALE–AMR code. These routines are implemented in a patch strategy class as described in Sect. 2.

5.1.2 *Multiblock patch hierarchies*

A single logically rectangular index space is insufficient for some simulation problems, especially those to which the ALE–AMR code is suited. To support more flexible spatial geometry descriptions, SAMRAI provides domains consisting of multiple structured mesh regions. A *multiblock patch hierarchy* is comprised of a set of single-block AMR patch hierarchies, each with its own logically rectangular index space, and a set of translation and rotation transformations that describe the connectivity between the neighboring index spaces. Such domains may have *singularity points*, where mesh cells near block corners have either fewer or more neighbors than a conventional structured mesh domain with uniform cell connectivity. Figure 6 shows a five-block mesh where each mesh cell adjacent to the singularity point has more neighbor cells than in a standard logically-rectangular mesh.

The multiblock capabilities in SAMRAI are designed so that data management and communication operations on a multiblock patch hierarchy are similar to the single-block patch hierarchy case. Multiblock versions of patch hierarchy, patch level, and communication algorithm and schedule classes provide essentially the same interfaces as their single-block versions but are implemented by composing single-block functionality with additional operations for treating multiple blocks. Patch data transfers within each block is identical to the case of a single patch hierarchy. Between multiple blocks, data must be transformed between index spaces. However, the extension of single-block communication operations to multiblock operations required no changes to the existing SAMRAI communication classes. This is possible since SAMRAI data communication algorithms do not depend on AMR patch hierarchy structures and SAMRAI communication schedules can transfer data between arbitrary patch levels regardless of whether they reside in the same AMR patch hierarchy.

The main complexity for a user of SAMRAI multiblock functionality is to supply routines for filling internal boundary values near singularity points. This is done using the *multiblock patch strategy* interface, which is similar to the refine patch strategy interface

Table 1 Key interaction points between SAMRAI and the ALE–AMR code during a time integration step. First, the ALE–AMR code calls a SAMRAI routine to advance the solution on the AMR hierarchy. The SAMRAI time integration algorithm calls ALE–AMR for integrating each hierarchy level. Data communication operations are managed by SAMRAI which calls ALE–AMR operations that interpolate data and set boundary conditions on the mesh defined by nodal coordinates

ALE-AMR code		SAMRAI library
advanceHierarchy()	$\longrightarrow$	
	$\longleftarrow$	advanceLevel()
fillData()	$\longrightarrow$	
	$\longleftarrow$	{ postprocessRefine(), setPhysicalBoundaryConditions() }

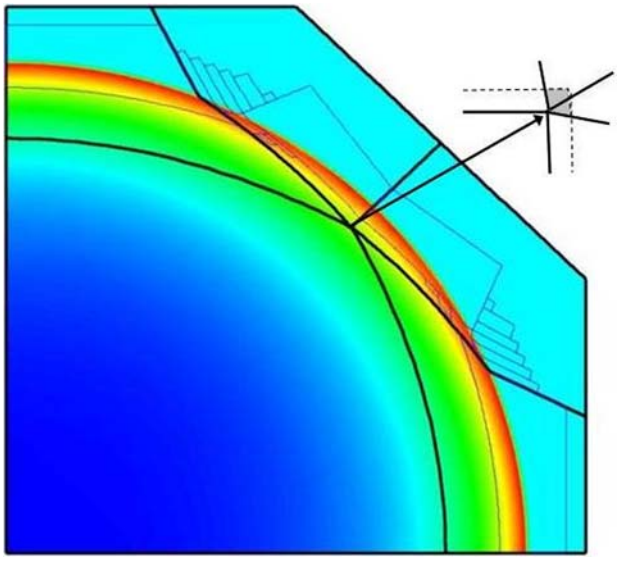

Fig. 6 A fluid front traveling through a multiblock domain consisting of five single-block patch hierarchies. Each mesh cell adjacent to the shared vertex has more neighbor cells than in a standard structured mesh. Setting ghost cell values for a patch near this point requires interpreting the irregular configuration, shown in the figure on the right, and interpolating values from each of the neighboring blocks

described earlier, except that it contains an extra routine for filling ghost data near singularity points. During communication, all data near a singularity point is gathered into a special structure which describes the intersections between conventional patch ghost cell regions and patches adjacent to a singularity point. Thus, multiblock functionality may be added to an existing AMR code by simply adding new operations to set ghost data for patches with irregular connectivity at corners.

5.2 Complex geometry and embedded boundaries

Many applications of practical engineering interest require representation of complex geometrical features. Constructing such features and meshes for them can be difficult and time consuming. Building a high quality mesh and numerical operations on it is particularly challenging. An emerging methodology for treating geometrically complex features in aerospace, geophysics, biology, and other problems areas embeds irregular geometry information into a Cartesian SAMR mesh [19–23]. This approach places internal boundary structures within the mesh by identifying cells that intersect the boundary. The main advantage of this approach is the computational speed with which the geometry representation can be constructed. Developments in two technologies have contributed to the popularity of this approach. They include fast geometry detection algorithms developed primarily for visualization, and Cartesian adaptive meshing algorithms that allow enhanced resolution near the geometry features. In this section, we describe a SAMRAI-based application for simulating flow around buildings in an urban landscape. The application demonstrates how embedded boundary data is managed in SAMRAI and how finite element operations are supported by the SAMRAI communication infrastructure.

Cells that intersect a surface generally form an irregular pattern on a structured mesh. To exploit the benefits of structured mesh methods, typical embedded boundary solution algorithms use conventional structured Cartesian mesh numerical methods on the entire mesh, as though the boundary were not present. Then, in a second step, special numerical operations are applied in cells near the boundary to correct the solution. Common correction algorithms use geometry information from each cut cell, such as the surface normal vector, centroid of the flow volume, volume fraction, and area fractions of the cut plane and surrounding faces. This additional data must be stored on an irregular set of cells on a patch; see Fig. 7.

One way to implement an embedded boundary structure in SAMRAI is to use the IndexData patch data type supplied by the library. This patch data type maintains a mapping between an irregular set of cells on a patch and data associated with those cells. It is a C++ class template in which the template parameter defines the data on each cell in the irregular index set. The IndexData type provides access to the data via standard cell indices on a patch or as an ordered list of the data items mapped to the indexed cells. The user-defined template parameter data is manipulated in parallel communication operations like any other data type. Figure 7 shows an example of embedded boundary patch data defined as IndexData<CutCell> where the CutCell parameter computes, stores, and provides access to quantities like the volume fraction, normal, and centroid of each cut cell.

The IndexData<CutCell> data type has been used to model building geometries in urban landscapes. Figure 8 shows an example domain in which a region of Manhattan containing more than one thousand buildings is represented as an embedded boundary within a SAMR mesh. The starting point for such a geometrical construction is an engineering-quality triangulated surface mesh in which buildings are defined by sets of triangles on a terrain map, or by polygons with an associated height. Fast computational geometry algorithms, similar to those used in computer graphics, quickly identify mesh cells that intersect surface triangles or polygons. The cut cells are further processed

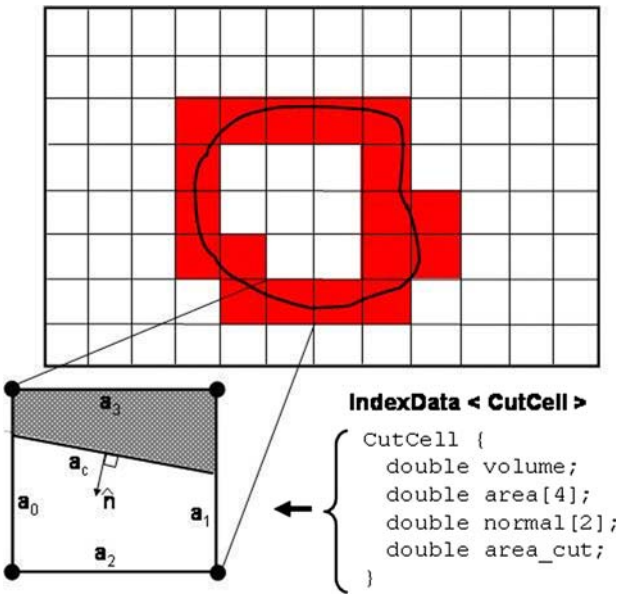

Fig. 7 The IndexData template class in SAMRAI manages data on an irregular set of cell indices on a patch. For example, the CutCell template parameter holds the data needed to represent the embedded boundary structure within a cell

to compute the geometric quantities on those cells [24, 25]. Since algorithms for constructing this information are fast and require no user intervention, the generation of the cut cells and AMR mesh required only about ten minutes on a single processor workstation.

To solve for flow around the buildings, an existing flow solver, developed at LLNL, was integrated into the SAMRAI-based application code. The original flow solver employs a finite element approximation of the incompressible flow equations. The finite element method requires communication procedures that accumulate sums of integrated quantities at nodes and edges from surrounding elements on neighboring patches within a single level as well as from patches on other levels at coarse-fine level boundaries. To implement the finite element sum communication pattern, the standard copy transaction is replaced with a *sum transaction* in existing SAMRAI communication schedule classes. This extension is implemented using the transaction factory mechanism described in Sect. 3.

5.3 Hybrid models

Traditional SAMR applications use local mesh refinement to increase resolution in single-physics calculations. In particular, the same mathematical model and numerical methods are applied at each mesh level. Advances in computer systems, numerical algorithms, and model development are driving increased development of multi-physics simulation capabilities. Often, such models employ *hybrid* formulations that couple multiple numerical models each of which represents a different aspect of the multi-physics system. The hierarchical structure of SAMR meshes can be a natural environment for hybrid model development whereby different numerical models are used on different levels in the mesh hierarchy.

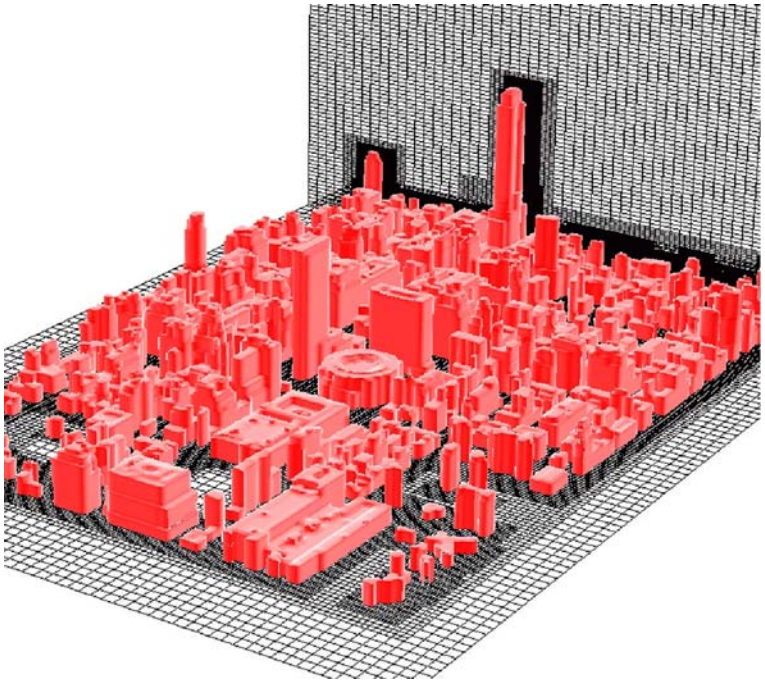

Fig. 8 An example complex geometry representation involving an urban area of several square blocks around Madison Square Garden in Manhattan. Buildings are represented as embedded boundaries in a SAMR mesh

In this section, we briefly discuss two such applications built using SAMRAI. One combines a compressible Euler continuum fluid model with a direct simulation Monte Carlo (DSMC) discrete particle model to resolve fluid interface dynamics. The other couples a continuum, incompressible Navier–Stokes fluid model to an immersed boundary method to simulate fluid–structure interactions. Figure 9 shows computations done with each code.

In each case, a continuum fluid model is used on all levels of the patch hierarchy. On the finest level, the continuum model is coupled to a different numerical model representing different physical processes. On coarser levels, AMR increases the local resolution of structured mesh-based fluid calculations. At the finest mesh level, each calculation transitions to a different numerical model that uses irregular data structures. In the following sections, we describe key aspects of these hybrid models that involve implementation of the irregular data structures on a SAMRAI patch hierarchy.

5.3.1 Continuum-particle hybrid model

The study of shock-accelerated interfaces between multiple fluids, such as the Richtmyer–Meshkov instability, motivated the development of the Euler–DSMC hybrid code [26]. Such problems often involve important physical mechanisms operating over a wide range of scales from hydrodynamic transport at the experimental scale (centimeters) to molecular diffusion at the shock thickness scale (nanometers). For some problems of interest, the standard Euler model is sufficiently accurate at coarser scales and away from fluid interfaces. Near those interfaces, a particle model like DSMC is required for physical accuracy. DSMC approximates the Boltzmann equation using a representative stochastic sampling of motion and collisions in a collection of fluid molecules [27]. The number of particles in a given region of the domain is determined by the local fluid density and stochastic sampling statistics. Often, the expense of DSMC simulation is prohibitive for large problem domains of interest. By using DSMC only on the finest mesh regions, local mesh refinement concentrates the atomistic calculations near fluid interfaces where they are needed to resolve dynamic, fine-scale flow features.

The Euler–DSMC hybrid code was constructed by merging two independently developed existing codes, a parallel AMR Euler code built with SAMRAI and a serial DSMC code that operates on a simple parallelepiped domain. The codes were integrated by building a wrapper class following the *Adapter* structural design pattern. The wrapper encapsulates the DSMC data structures and numerical routines and translates information between the continuum and particle representations. A simple patch data class owns a single instantiation of the wrapper and accesses the associated DSMC particle data via the wrapper methods.

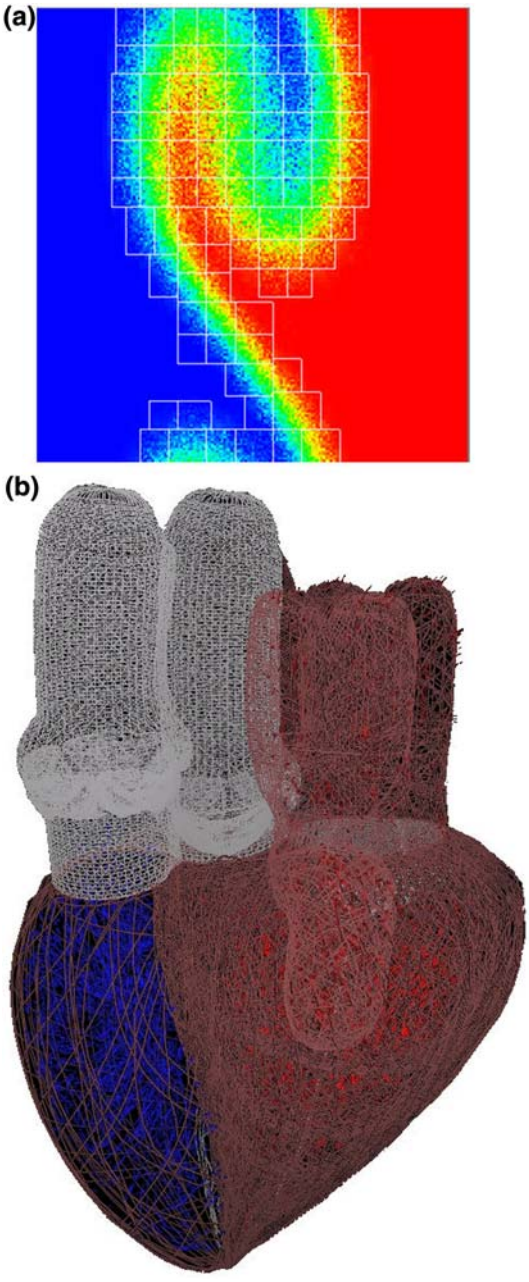

Fig. 9 Hybrid AMR applications use the mesh hierarchy to couple different numerical models in multi-physics simulations. **a** A sheared interface between two fluids produced by the Euler–DSMC code. White boxes are boundaries of patches holding DSMC particles on the finest mesh level. The continuum Euler fluid model is applied elsewhere in the domain. **b** An immersed boundary representation of elastic tissue fibers in a human heart. The structures were extracted from the SAMR mesh for clarity

The main objects used to manage the DSMC particle data in the Euler–DSMC application code are illustrated in Fig. 10. The result of this implementation is that the DSMC particles are manipulated by SAMRAI communication operations just like any other data type. The DSMC code remains a standalone serial code. The Euler–DSMC hybrid is parallel and adaptive and allows multiple DSMC regions to be embedded within the computational domain. The only modifications required of the DSMC code were minor extensions to the way in which boundary conditions are applied.

5.3.2 *Immersed boundary heart model*

The last example we will present involves an adaptive mesh refinement immersed boundary application to simulate human cardiac dynamics by coupling electrophysiological and mechanical processes [28]. Realistic simulation of fluid–structure interactions between heart tissue and blood flow requires treatment of inhomogeneous elastic tissue with anisotropic orientation and moving tissue geometry. The immersed boundary (IB) approach was introduced by Peskin to study the hemodynamics of heart valve leaflets [29]. Blood flow is modeled using the viscous incompressible Navier–Stokes equations in an Eulerian reference frame on a Cartesian structured computational mesh. Heart fibers are modeled as viscoelastic structures in a Lagrangian reference frame on a curvilinear mesh where material points are coupled based on the structural configuration. The fluid and tissue models are coupled through terms representing the exchange of forces.

Adequate resolution of the flow and dynamics in a cardiac system using the IB method requires large-scale parallel computing resources. Finely spaced meshes are needed on large three-dimensional spatial domains to resolve narrow action potential fronts in heart fibers and important fine-scale fluid flow features near valves. Moreover, slow-moving wave fronts require long-running simulations, a situation which is exacerbated by the need for small, numerically-stable timesteps. Adaptive mesh refinement reduces the overall size of the calculation by providing fine mesh resolution near the elastic structures and by allowing coarser meshes away from the fluid-structure interface where the flow is smooth. The right side of Fig. 9 shows the complexity of the elastic structure in an IB simulation where the fibers have been extracted from the Cartesian mesh for clarity.

In the current implementation, the coupled Eulerian–Lagrangian system is advanced in time using a semi-implicit integration strategy. Each timestep requires solving linear systems of equations on the Cartesian SAMR mesh. The PETSc [30] library provides Krylov iterative solvers for SAMRAI mesh data natively using the PETSc–SAMRAI vector interface; see [31] for a description of how this interface works. The preconditioner is a custom FAC solver built for the SAMR Navier–Stokes discretization [28].

All data associated with the curvilinear IB structure mesh is stored in a PETSc vector object. The distribution of the PETSc vector data is specified by the decomposition of IB data among SAMR patches and the SAMRAI patch-processor mapping. The irregular IB data is managed similarly to the embedded structure in the urban dispersion model described in Sect. 2. In particular, an IndexData<Sentinel> patch data type is used to couple the Lagrangian IB data and the Eulerian AMR mesh. The Sentinel template parameter maps between a SAMRAI patch and the PETSc data structure holding the IB data on the patch region. The Sentinel class maintains additional indexing information to make interpolation between IB data and force and velocity mesh data efficient.

When the patch hierarchy is regenerated during adaptive mesh refinement, the IB (i.e., PETSc vector) data is redistributed so that each node of the curvilinear mesh is assigned to the same processor as the SAMR mesh patch in which it is physically located. Before remeshing, the sentinel data associated with each IndexData<Sentinel> object corresponds to the current curvilinear mesh configuration. Then, the patch hierarchy is remeshed and the Cartesian mesh data is

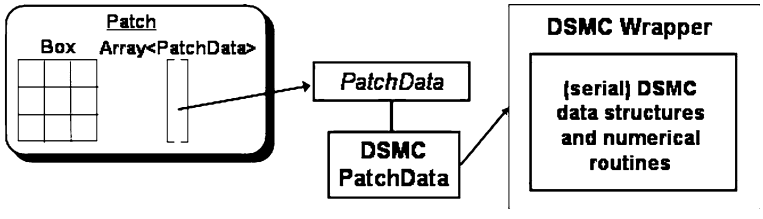

Fig. 10 In the Euler–DSMC code, the DSMCPatchData object provides access to the DSMC particle data and looks like any other patch data type to SAMRAI. Numerical routines that translate information between the Euler continuum and the DSMC particle representations reside in the DSMCWrapper object

redistributed. At this point, IndexData<Sentinel> patch data objects are created on the new SAMR patch configuration, but the IB data still resides on the old mesh configuration. Next, the assignment of the curvilinear IB mesh data to the new patch configuration is determined. This information is used to setup a PETSc parallel vector gather/scatter operation which redistributes the IB data to the new patch configuration. SAMRAI and PETSc parallel data management and communication facilities are used in concert in this application without duplication of IB data.

6 Conclusion

Adaptive mesh refinement has matured into an effective simulation technology for achieving efficient, highly-resolved calculations for a variety of computational science and engineering problems. As the range and sophistication of large-scale parallel applications increases, we believe that the AMR methods will become even more important. Robust and flexible software support libraries are essential for continued successful development of new applications.

This paper discusses the design and implementation of software in the SAMRAI library for managing complex data and communication in parallel for SAMR calculations. The capabilities have been illustrated in the context of several new and unique application efforts. The applications involve adaptive, deforming meshes, embedded boundaries for complex geometry, immersed boundaries for fluid–structure interactions, and hybrid coupling of continuum methods and discrete particle methods for multi-physics problems. We have shown how application-specific data can be manipulated in parallel in a straightforward manner using the general data management and communication infrastructure in SAMRAI. The SAMRAI infrastructure not only supports multiple, diverse application efforts simultaneously, it supports new data types and integration with existing application code and software libraries.

To allow application developers to exploit the software flexibility that we have developed, we have chosen to expose a set of powerful software interfaces through which users can extend and specialize framework capabilities to meet their needs. Object-oriented design patterns, such as *Strategy* and *Factory*, employed in SAMRAI have aided in achieving a clear separation of general SAMR functionality and application-specific operations. By reducing the imprint of particular SAMR algorithms and data types on the core of the library, SAMRAI has addressed new applications and provided a significant amount of design and code re-use across those applications. For example, although several of the applications we have described involve unique data representations, no additional code was required to distribute and communicate that data in parallel. Thus, both library code and pre-existing application code was reused.

In our experience, the software design choices we have made have helped to reduce application development time for users and have promoted the exploration of new application opportunities. Typical users concentrate most of their energy on application-specific concerns and are freed from the need to understand the complexity of the underlying parallel SAMR software infrastructure. We have also observed that software organization principles employed in the framework have influenced design decisions made by users during the development of application codes. For example, design patterns used in SAMRAI to decompose elements of the software are often adopted by application developers. This *design reuse*, whereby users emulate organizational features and software abstractions found in the framework, has enabled new algorithm development by increasing the flexibility of application codes.

In the end, the benefits of using a large, complex software infrastructure like SAMRAI depend on the goals of individual application developers. The capabilities of SAMRAI and other SAMR support libraries reflect the capacity of their developers to anticipate application requirements and provide good software solutions for future extensions and enhancements. Some SAMR algorithms have become commonplace and support for them may be found in multiple libraries. However, new and unique applications force developers to consider the trade off between creating code from scratch and the learning curve required to make productive use of a sophisticated code base built by others. Using a software library may require one to consider different ways of conceptualizing the constructing an application code. This is often beneficial. In this paper, we have demonstrated that the advantages of exploiting general software capabilities as well as providing application-specific specializations when appropriate can be substantial.

Acknowledgments We gratefully acknowledge the contributions of our collaborators whose application development work is discussed in this paper and who provided figures. Boyce Griffith is developing the immersed boundary AMR code to model dynamics of the human heart. He has recently completed his Ph.D. thesis at the Courant Institute at New York University under the direction of Professor Charles Peskin. Bob Anderson and others are continuing to develop the ALE-AMR code at

LLNL. Noah Elliott is primarily responsible for the multi-block hierarchy development in SAMRAI. The ability to represent complex, urban landscapes geometry on a SAMRAI mesh utilizes the Eleven library in the Overture package, with the help of Kyle Chand (LLNL), and also uses the Cart3d/Cubes package, with the help of Marsha Berger (Courant Institute, NYU). The efforts of these individuals and other users of SAMRAI have creatively demonstrated the utility and flexibility of the library and their contributions make continued development of SAMRAI possible.

References

1. Berger MJ, Colella P (1989) Local adaptive mesh refinement for shock hydrodynamics. J Comp Phys 82:64–84
2. Berger MJ, Oliger J (1984) Adaptive mesh refinement for hyperbolic partial differental equations. J Comp Phys 53:484–512
3. Diachin L, Hornung R, Plassman P, Wissink A (2005) Parallel adaptive mesh refinement. In: Heroux M, Raghavan P, Simon H (eds) Frontiers of parallel processing for scientific computing. SIAM book series on software, environments, and tools. Society for Industrial and Applied Mathematics, Philadelphia (in press)
4. Wissink AM, Hysom D, Hornung RD (2003) Enhancing Scalability of Parallel Structured AMR Calculations. In: Proceedings of the 17th ACM international conference on supercomputing (ICS03), San Francisco, pp 336–347
5. Anderson RW, Elliott NS, Pember RB (2004) An arbitrary Lagrangian–Eulerian method with adaptive mesh refinement for the solution of the Euler equations. J Comp Phys 199:598–617
6. Colella P, Graves D, Ligocki T, Martin D, Serafini D, Straalen BV (2003) Chombo software package for AMR applications design document. Report, Applied Numerical Algorithms Group, NERSC Division, Lawrence Berkeley National Laboratory, Berkeley
7. Rendleman CA, Beckner VE, Lijewski M, Crutchfield WY, Bell JB (2000) Parallelization of structured, hierarchical adaptive mesh refinement algorithms. Comput Vis Sci 3:147–157
8. Parashar M, Browne JC (2000) System engineering for high performance computing software: the HDDA/DAGH infrastructure for implementation of parallel structured adaptive mesh refinement. In: Baden SB et al (eds) IMA Volume 117: structured adaptive mesh refinement (SAMR) grid methods. Springer, Berlin Heidelberg New York, pp 1–18
9. MacNeice P, Olson KM, Mobarry C, deFainchtein R, Packer C (2000) PARAMESH: a parallel adaptive mesh refinement community toolkit. Comp Phys Comm 126:330–354
10. Henshaw WD (2002) Overture: an object-oriented framework for overlapping grid applications. In: Proceedings of the 32nd American Institute for Aeronautics Fluid Dynamics, St. Louis
11. Lemke M, Quinlan D (1992) A C++ virtual shared grids based programming environment for architecture-independent development of structured grid applications. Lecture Notes in Computer Science. Springer, Berlin Heidelberg New York
12. Li X, Parashar M (2003) Dynamic load partitioning strategies for managing data of space and time heterogeneity in parallel SAMR applications. In: Proceedings of the 9th international Euro-Par conference (Euro-Par 2003), Klagenfurt, Austria. Springer, Berlin Heidelberg New York
13. Chandra S, Sinha S, Parashar M, Zhang Y, Yang Y, Hariri S (2002) Adaptive runtime management of SAMR applications. In: Sahni S, Prasanna VK, Shukla U (eds) Proceedings of the 9th international conference on high performance computing (HiPC 2002). Lecture Notes in Computer Science, vol 2552. Springer, Berlin Heidelberg New York, Bangalore, pp 564–574
14. Hornung RD, Kohn SR (2002) Managing application complexity in the SAMRAI object-oriented framework. Concurr Comput Pract Exp14:347–368
15. Gamma E, Helm R, Johnson R, Vlissides J (1995) Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman, Inc., Menlo Park
16. Saltz J, Berryman H, Wu J (1991) Multiprocessors and runtime compilation. Concurr Pract Exp 3:573–592
17. Baden SB, Colella P, Shalit D, Van Straalen B (2001) Abstract KeLP. In: Proceedings of the 10th SIAM conference on parallel processing for scientific computing, Portsmouth
18. Fink SJ, Baden SB, Kohn SR (1996) Flexible communication schedules for block structured applications. In: Third international workshop on parallel algorithms for irregularly structured problems (IRREGULAR '96), Santa Barbara, California
19. Aftosmis M, Melton J, Berger M (1995) Adaptation and surface modeling for cartesian mesh methods. In: Proceedings of the 12th AIAA computational fluid dynamics conference, San Diego. AIAA Paper 95-1725
20. Aftosmis M, Berger M (2002) Multilevel error estimation and adaptive h-refinement for cartesian meshes with embedded boundaries. In: Proceedings of the 40th AIAA aerospace sciences meeting and exhibit, Reno. AIAA Paper 2002-0863
21. Johansen H, Colella P (1998) A cartesian grid embedded boundary method for poissons equation on irregular domains. J Comp Phys 147:60–85
22. McCorquodale P, Colella P, Johansen H (2001) A cartesian grid embedded boundary method for the heat equation on irregular domains. J Comp Phys 173:620–635
23. Zeeuw DD, Powell KG (1993) An adaptively refined cartesian mesh solver for the euler equations. J Comp Phys 104:56–68
24. Aftosmis M, Berger M, Melton J (1998) Adaptive cartesian mesh generation. In: Thompson JF, Soni BK, Weatherill NP (eds) Handbook of grid generation. CRC Press LLC, Boca Raton. ISBN 0-8493-2687-7
25. Petersson NA (2003) Rapsodi: geometry preparation and grid generation. In: Proceedings of the 2nd SIAM conference on computational science and engineering, San Diego
26. Wijesinghe HS, Hornung RD, Garcia AL, Hadjiconstantinou NG (2004) 3-dimensional hybrid continuum-atomistic simulations for multiscale hydrodynamics. J Fluid Eng 126:768–777
27. Alexander F, Garcia A (1997) Direct simulation Monte Carlo. Comput Phys 11:588-593
28. Griffith B, Hornung R, McQueen D, Peskin C (2006) An adaptive, fornally second-order sccurate version of the immersed boundary method J Comp Phys (to appear)
29. Peskin CS (2002) The immersed boundary method. Acta Numer 11:479–517
30. Balay S, Buschelman K, Eijkhout V, Gropp WD, Kaushik D, Knepley MG, Curfman McInnes L, Smith BF, Zhang H (2004) PETSc users manual. Report, Argonne National Laboratory, Argonne. ANL-95/11 - Revision 2.1.5
31. Pernice M, Hornung R (2005) Newton-Krylov-FAC methods for problems discretized on locally-refined grids. Comput Vis Sci 8(2):107–118