Object Oriented Programming Abstractions for Parallel Adaptive Mesh-Refinement

Manish Parashar and James C. Browne
Department of Computer Sciences
University of Texas at Austin
{parashar, browne}@cs.utexas.edu

Extented Abstract

Contents

Department of Computer Sciences & Center for Relativity • University of Texas at Austin
4.200 Taylor Hall, Austin, TX 78712-1081
Tel: (512) 471-5513; Fax: (512) 471-8694; parashar@cs.utexas.edu

1 Introduction

This paper describes high-level object oriented programming abstractions that can be used to directly implement parallel adaptive computations on dynamic hierarchical grid structures and demonstrates their application. Dynamically adaptive methods for solution of differential equations which employ locally optimal approximations have been shown to yield highly advantageous ratios for cost/accuracy when compared to methods based upon static uniform approximations. Parallel versions of these methods offer potential for accurate solution of physically realistic models of important physical systems. Sequential implementations of adaptive algorithms using conventional programming abstractions have proven to be both complex and difficult to validate. Parallel implementations are then an order of magnitude more complex, and introduce additional concerns such as partitioning, dynamic re-partitioning, load-balancing, and communication scheduling. It is commonly the case that over 75% of the code volume of a parallel adaptive code written in conventional programming systems is concerned with procedurally realizing dynamic distributed data structures on top of static data structures such as Fortran arrays. Furthermore this data-management code has little connection with the physics or engineering being solved. Clearly there is a need for high-level programming abstractions upon which parallel adaptive applications can be directly and simply developed.

In this paper we identify three fundamental object classes (or abstractions) that can be used to express parallel adaptive computations based on adaptive mesh refinement (AMR) and multigrid techniques (see Figure 1). The *Grid Geometry* abstractions represent regions in the computational domain and provide an intuitive means for addressing the regions and directing computations. A *Grid Hierarchy* abstraction

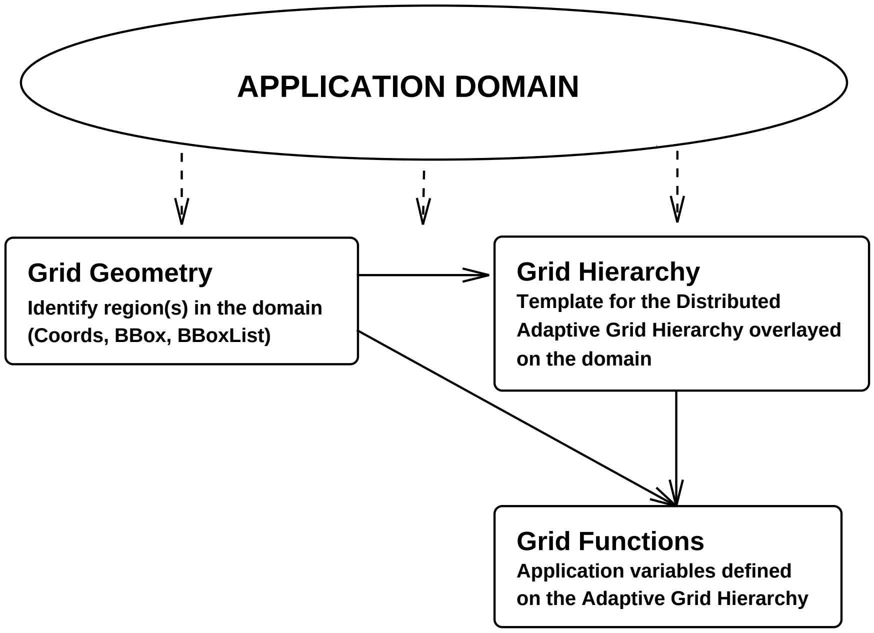

Figure 1: Programming Abstraction for Parallel Adaptive Mesh-Refinement

abstracts the distributed adaptive grid hierarchy (a directed acyclic graph (DAG) comprising of a dynamic number of levels of grid resolution, and a dynamic number of component grids at each level of resolution). Applications can directly index, operate on, and refine component grids within the hierarchy independent of its current structure and distribution. Parallelization issues such as partitioning and dynamic load balancing are encapsulated within Grid Hierarchy objects. The final abstraction is the *Grid Function* which represents an application variable defined on the distributed, hierarchical computational domain. A Grid Function object allocates distributed storage for the variable it represents according to the structure of the dynamic grid hierarchy, and enables the variable to be locally manipulated as simple Fortran 77/90 arrays. All user-level operations defined on these abstractions are independent of the structure of the dynamic grid or its distribution. Data-partitioning, load-balancing, communications and synchronization operations are transparent to the end user.

The programming abstractions have been implemented within DAGH [1][1], a data-management infrastructure to support parallel adaptive applications. The infrastructure incorporates distributed dynamic data-structures [2] that can efficiently support adaptive grid hierarchies. DAGH forms the foundational layer of a computational toolkit for the Binary Black-Hole NSF Grand Challenge project and is currently operational on the IBM SP2, Cray T3D and networked workstations (RS6000, Sun, SGI).

2 Problem Description

Dynamically adaptive numerical techniques for solving differential equations provide a means for concentrating computational effort to appropriate regions in the computational domain. In the case of hierarchical AMR methods, this is achieved by tracking regions in the domain that require additional resolution and dynamically overlaying finer grids over these regions. AMR-based techniques start with a base coarse grid with minimum acceptable resolution that covers the entire computational domain. As the solution progresses, regions in the domain requiring additional resolution are tagged and finer grids are overlayed on the tagged regions of the coarse grid. Refinement proceeds recursively so that regions on the finer grid requiring more resolution are similarly tagged and even finer grids are overlayed on these regions. The resulting grid structure is a dynamic adaptive grid hierarchy.

3 Object Oriented Programming Abstractions for Parallel Adaptive Mesh-Refinement

We have two key objectives in developing the high-level abstractions described in this section: Our first objective is to provide application developers with a set of primitives that intuitively complement the problem, i.e. application objects $\equiv$ abstract datatypes. The second objective is a separation of data-

[1]For information about DAGH see `http://godel.ph.utexas.edu/Members/parashar/DAGH/dagh.html`

management issues and implementations from application specific computations.

3.1 Grid Geometry Abstractions

The purpose of the grid geometry abstractions is to provide an intuitive means for identifying and addressing regions in the computational domain. These abstractions can be used to direct computations to a particular region in the domain, to mask regions that should not be included in a given operation, or to specify region that need more resolution or refinement. The grid geometry abstractions represent coordinates, bounding boxes and doubly linked lists of bounding boxes.

Coordinates: The coordinate abstraction (implemented as `C++` class *Coords*) represents a point in the computational domain. Operations defined on this class include indexing and arithmetic/logical manipulations. These operations are independent of the dimensionality of the domain.

Bounding Boxes: Bounding boxes (implemented as `C++` class *BBox*) represents regions in the computation domain and is comprised of a triplet: a pair of *Coords* defining the lower and upper bounds of the box and a *step array* that defines the granularity of the discretization in each dimension. In addition to regular indexing and arithmetic operations, scaling, translations, unions and intersections are also defined on bounding boxes. Bounding boxes are the primary means for specification of operations and storage of internal information (such as dependency and communication information) within DAGH.

Bounding Boxes Lists: The *BBoxList* `C++` class implements a doubly linked list of bounding boxes and represents a collection of regions in the computational domain. Such a list is typically used to specify regions that need refinement during the regriding phase of an adaptive application. In addition to linked-list addition, deletion and stepping operation, reduction operations such as intersection and union are also defined on a BBoxList.

3.2 Grid Hierarchy Abstraction

The grid hierarchy abstraction represents the distributed dynamic adaptive grid hierarchy that underlie parallel adaptive applications based on adaptive mesh-refinement. This abstraction enables a user to define, maintain and operate a grid hierarchy as a first-class object. Grid hierarchy attributes include the geometry specifications of the domain such as the structure of the base grid, its extents, boundary information, coordinate information, and refinement information such as information about the nature of refinement and the refinement factor to be used. When used in a parallel/distributed environment, the grid hierarchy is partitioned and distributed across the processors and serves as a template for all application variables or grid functions. The locality preserving *composite distribution* [3] based on recursive *Space-filling Curves* [4] is used to partition the dynamic grid hierarchy. Operations defined on the grid hierarchy include indexing of individual component grid in the hierarchy, refinement, coarsening, recomposition of the hierarchy after regriding, and querying of the structure of the hierarchy at any instant. During regriding, the re-partitioning of the new grid structure, dynamic load-balancing, and the required data-movement to initialize newly created grids, are performed automatically and transparently.

3.3 Grid Function Abstraction

Grid Functions represent application variables defined on the grid hierarchy. Each grid function is associated with a grid hierarchy and uses the hierarchy as a template to define its structure and distribution. Attributes of a grid function include type information, and dependency information in terms of space and time stencil radii. In addition the user can assign special (Fortran) routines to a grid function to handle operations such as inter-grid transfers (prolongation and restriction), initialization, boundary updates, and input/output. These function are then called internally when operating on the distributed grid function. In addition to standard arithmetic and logical manipulations, a number of reduction operations such as *Min/Max*, *Sum/Product*, and *Norms* are also defined on grid functions. GridFunction objects can be locally operated on as regular Fortran 90/77 arrays.

4 Outline of the Paper

The rest of the paper will be comprised of 3 additional sections: The first will present the design and implementation of the object-oriented abstractions described above, as `C++` classes within the DAGH infrastructure. The next section will illustrate the use of the abstractions and the programming interface provided by the infrastructure. The abstractions will be used to construct components of a Berger-Oliger AMR driver. The final section will present an experimental evaluation of the programming abstractions and the DAGH data-management infrastructure.

References

[1] Manish Parashar and James C. Browne, "An Infrastructure for Parallel Adaptive Mesh-Refinement Techniques", Technical report, Department of Computer Sciences, University of Texas at Austin, TICAM, 2.400 Taylor Hall, Austin, TX 78712, 1995, Available via WWW at http://godel.ph.utexas.edu/Members/parashar/toolkit.html.

[2] Manish Parashar and James C. Browne, "Distributed Dynamic Data-Structures for Parallel Adaptive Mesh-Refinement", *Proceedings of the International Conference for High Performance Computing*, Dec. 1995.

[3] Manish Parashar and James C. Browne, "On Partitioning Dynamic Adaptive Grid Hierarchies", *Proceedings of the* 29^{th} *Annual Hawaii International Conference on System Sciences*, Jan. 1996.

[4] Hanan Samet, *The Design and Analysis of Spatial Data Structures*, Addison - Wesley Publishing Company, 1989.

Department of Computer Sciences & Center for Relativity • University of Texas at Austin
4.200 Taylor Hall, Austin, TX 78712-1081
Tel: (512) 471-5513; Fax: (512) 471-8694; parashar@cs.utexas.edu