Distributed Dynamic Data-Structures 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
(To be presented at HiPC, December, 1995)

Abstract

This paper presents the design and implementation of dynamic distributed data-structures to support parallel adaptive (multigrid) finite difference codes based on hierarchical adaptive mesh-refinement (AMR) techniques for the solution of partial differential equations. The abstraction provided by the data-structures is a dynamic hierarchical grid where operations on the grid are independent of its distribution across processors in a parallel execution environment, and of the number of levels in the grid hierarchy. The distributed dynamic data-structures have been implemented as part of a computational toolkit for the Binary Black-Hole NSF Grand Challenge project.

1 Introduction

Dynamically adaptive methods for the solution of partial differential equations that employ locally optimal approximations can yield highly advantageous ratios for cost/accuracy when compared to methods based upon static uniform approximations. Parallel versions of these methods offer the potential for accurate solution of physically realistic models of important physical systems. Sequential implementations of adaptive algorithms in conventional programming system abstractions have proven to be both complex and difficult to validate. Parallel implementations are then an order of magnitude more complex. Clearly there is an advantage in providing an infrastructure of appropriate dynamic data-structures and associated programming abstractions upon which parallel adaptive methods can be directly implemented.

*This research has been jointly sponsored by the Binary Black-Hole NSF Grand Challenge (NSF ACS/PHY 9318152) and by ARPA under contract DABT 63-92-C-0042.

The fundamental data-structure underlying dynamically adaptive methods based on hierarchical adaptive-mesh refinements is a dynamic hierarchy of successively and selectively refined grids, or, in the case of a parallel implementation, a Distributed Adaptive Grid Hierarchy (DAGH). The efficiency of parallel/distributed implementations of these methods is limited by the ability to partition the DAGH at run-time so as to expose all inherent parallelism, minimize communication/synchronization overheads, and balance load. A critical requirement while partitioning DAGHs is the maintenance of logical locality, both across different levels of the hierarchy under expansion and contraction of the adaptive grid structure, and within partitions of grids at all levels when they are partitioned and mapped across processors. The former enables efficient computational access to the grids while the latter minimizes the total communication and synchronization overheads.

The design and implementation of two basic data-structures are presented in this paper: (1) A Scalable Distributed Dynamic Grid (SDDG) which is a single grid in an adaptive grid hierarchy, and (2) A Distributed Adaptive Grid Hierarchy (DAGH) which is a dynamic collection of SDDGs and implements an entire adaptive grid hierarchy. The design of these data-structures uses a linear representation of the hierarchical, multi-dimensional grid structure which is generated using space-filling mappings [1, 2]. Operations on the grid hierarchy such as grid creation, grid refinement or coarsening, grid partitioning and dynamic re-partitioning, are efficiently defined on this one-dimensional representation. The self-similar nature of space-filling curves is exploited to maintain locality across levels of the grid hierarchy. Computational data associated with the grids in the hierarchy is maintained as a scalable distributed dynamic array (SDDA) that is based on extendible hashing [3] and guarantees preservation of locality under expansion and contraction. Space-filling indices used in the DAGH/SDDG representation serve as keys for indexing into the SDDA. The data-structures have been implemented as a C++ class library on top of the MPI [1][4] communication system. Architectures currently supported include the IBM SP2, Cray T3D, and clusters of networked workstations (IBM RS6000, SGI, SUN).

The rest of this paper is organized as follows: Section 2 describes the grid structure defined by hierarchical adaptive mesh-refinement techniques. Section 3 presents the design and implementation of SDDG/DAGH data-structures. Section 4 presents an experimental evaluation of these data-structures. Section 5 presents some concluding remarks.

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 adaptive mesh refinement (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 Distributed Data-Structures for Hierarchical AMR

3.1 Data-structure Representation

The SDDG/DAGH data-structure design is based on a linear representation of the hierarchical, multi-dimensional grid structure. This representation is generated using space-filling curves [1, 2, 5] which are a class of locality preserving mappings from d-dimensional space to 1-dimensional space, i.e. $N^d \rightarrow N^1$, such that each point in N^d is mapped to a unique point or index in N^1. Space-filling mapping functions are computationally inexpensive and consist of bit level interleaving operations and logical manipulations of the coordinates of a point in multi-dimensional space. Furthermore, the self-similar or recursive nature of these mappings can be exploited to represent a hierarchical structure and to maintain locality across different levels of the hierarchy. Finally, space-filling mappings allow information about the original multi-dimensional space to be encoded into each space-filling index. Given an index, it is possible to obtain its position in the original multi-dimensional space, the shape of the region in the multi-dimensional space associated with the index, and the space-filling indices that are adjacent to it.

3.1.1 SDDG Representation:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

{0 1 4 5 2 3 6 7 8 9 12 13 10 11 14 15} (Morton)

{0 1 5 4 8 12 13 9 10 14 15 11 7 6 2 3} (Peano-Hilbert)

Figure 1: SDDG Representation

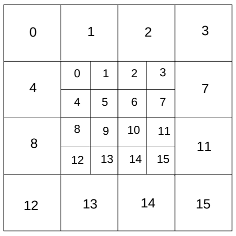
A multi-dimensional SDDG is represented as a one dimensional ordered list of SDDG blocks. The list is obtained by first blocking the SDDG to achieve the required granularity, and then ordering the SDDG blocks based on the selected space-filling curve. The granularity of SDDG blocks is system dependent and attempts to balance the computation-communication ratio for each block. Each block in the list is assigned a cost corresponding to its computational load. In case of an AMR scheme, computational load is determined by the number of grid elements contained in the block and the level of the block in the AMR grid hierarchy. The former defines the cost of an update operation on the block while the latter defines the frequency of updates relative to the base grid of the hierarchy. Figure 1 illustrates this representation for a 2-dimensional SDDG using 2 different space-filling curves (Morton & Peano-Hilbert).

Partitioning a SDDG across processing elements using this representation consists of appropriately partitioning the SDDG block list so as to balance the

[1] Message Passing Interface

total cost at each processor. Since space-filling curve mappings preserve spatial locality, the resulting distribution is comparable to traditional block distributions in terms of communication overheads.

3.1.2 DAGH Representation:



{0 1 4 {0 1 4 5} 2 3 {2 3 6 7} 7 8 {8 9 12 13} 12 13 {10 11 14 15} 11 14 15} (Morton)

{0 1 {0 1 5 4} 4 8 12 13 {8 12 13 9 10 14 11 15} 14 15 11 7 {7 6 2 3} 2 3} (Peano-Hilbert)

Figure 2: Composite representation

The DAGH representation starts with a simple SDDG list corresponding to the base grid of the grid hierarchy, and appropriately incorporates newly created SDDGs within this list as the base grid gets refined. The resulting structure is a composite list of the entire adaptive grid hierarchy. Incorporation of refined component grids into the base SDDG list is achieved by exploiting the recursive nature of space-filling mappings: For each refined region, the SDDG sub-list corresponding to the refined region is replaced by the child grid's SDDG list. The costs associated with blocks of the new list are updated to reflect combined computational loads of the parent and child. The DAGH representation therefore is a composite ordered list of DAGH blocks where each DAGH block represents a block of the entire grid hierarchy and may contain more than one grid level; i.e. inter-level locality is maintained within each DAGH block. Each DAGH block in this representation is fully described by the combination of the space-filling index corresponding to the coarsest level it contains, a refinement factor, and the number of levels contained. Figure 2 illustrates the composite representation for a two dimensional grid hierarchy.

The AMR grid hierarchy can be partitioned across processors by appropriately partitioning the linear DAGH representation. In particular, partitioning the composite list to balance the cost associated to each processor results in a composite decomposition of the hierarchy. The key feature of this decomposition is that it minimizes potentially expensive inter-grid communications by maintaining inter-level locality in each partition. A composite decomposition of a 2-dimensional DAGH is shown in Figure 3. Note that each numbered unit in this figure is a composite block of sufficient granularity. Other distributions of the grid hierarchy can also be generated using the above representation. For example, a distribution that decomposes each grid separately is generated by viewing the DAGH list as a set of SDDG lists.

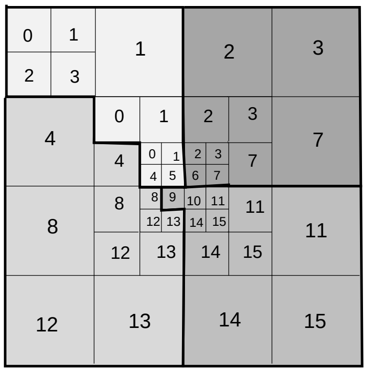


Figure 3: DAGH Composite distribution

3.2 Data-Structure Storage

Data-structure storage can be divided into two components; (1) storage of the adaptive grid structure, and (2) storage of the associated data. The overall storage scheme is shown in Figure 4. The two components are described below:

3.2.1 Adaptive Grid Structure Storage

The structure of the adaptive grid hierarchy is stored as an ordered list using the representation presented above. Appropriate interfaces enable the DAGH list to viewed as a single composite list or as a set of SDDG lists. The former enables a composite decomposition of the grid hierarchy to be generated, while the latter enables each level of the hierarchy to be addressed and operated on individually. Storage associated with each block consists of a space-filling index that identifies its location in the entire grid structure, an extent defining its granularity, the number of refinement levels contained (in case of DAGHs), and a cost measure corresponding to its computational load. Storage requirements for the SDDG/DAGH representation is therefore linearly proportional to the number of DAGH/SDDG blocks. This overhead is small compared to the storage required for the grid data itself.

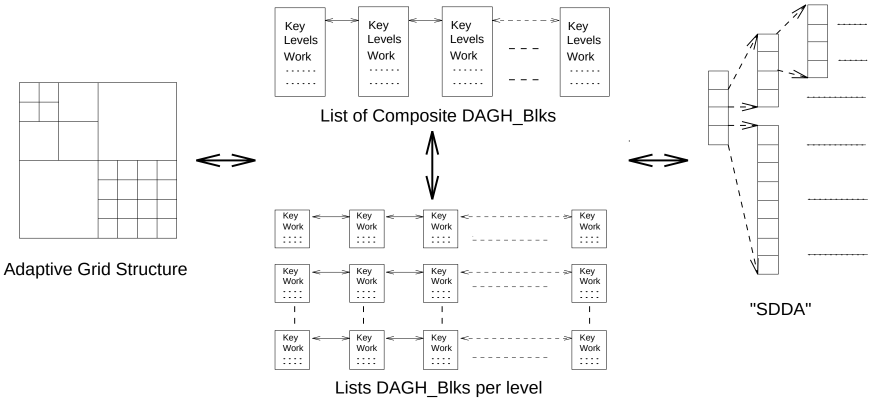


Figure 4: Storage Scheme

3.2.2 Data Storage

The data storage component of the storage scheme is implemented as a “Scalable Distributed Dynamic Array” (SDDA) and uses extendable hashing techniques [3] to provide a dynamically extendable, globally indexed storage. The SDDA is a hierarchical structure and is capable dynamically expanding and contracting. Entries into the SDDA correspond to DAGH blocks and the array is indexed using DAGH block keys. The SDDA data storage provides a means for efficient communication between DAGH blocks. To communicate data to another DAGH blocks, the data is copied to appropriate locations in the SDDA. This information is then asynchronously shipped to the appropriate processor. Similarly, data needed from remote DAGH blocks is received on-the-fly and inserted into the appropriate location in the SDDA. Storage associated with the SDDA is maintained in ready-to-ship buckets. This alleviates overheads associated with packing and unpacking. An incoming bucket is directly inserted into its location in the SDDA. Similarly, when data associated with a DAGH block entry is ready to ship, the associated bucket is shipped as is.

4 Experimental Evaluation of DAGH/SDDG

A data-management infrastructure for distributed hierarchical AMR has been developed based on the data-structures described in this paper. The infrastructure is implemented as a `C++` class library and provides high-level programming abstraction for expressing AMR computations. These abstraction manage the dynamics of the AMR grid structure and provide a fortran-like interface to the developer. This enables all computations on grid-data to be performed by Fortran subroutines. Results from an initial experimental evaluation of the infrastructure on the IBM SP2 are presented in this section. The objectives of the evaluation are as follows:

1. To evaluate the overheads of the presented data-structure representation and storage scheme over conventional static (Fortran) array-based structures for unigrid applications.

2. To evaluate the effectiveness of the composite partitioning of the AMR grid hierarchy in terms of its ability to balance load and maintain locality.

3. To evaluate the relative costs of partitioning and dynamically re-partitioning the grid hierarchy using the presented data-structure representation.

4.1 Application Description

An adaptation of the *H3expresso* 3-D numerical relativity code developed at the National Center for Supercomputing Applications (NCSA), University of Illinois at Urbana, is used to evaluate the data-management infrastructure. *H3expresso* is a “concentrated” version of the full *H* version 3.3 code that solves the general relativistic Einstein’s Equations in a variety of physical scenarios [6]. The original *H3expresso* code is non-adaptive and is implemented in Fortran 90. A distributed and adaptive version of *H3expresso* has been implemented on top of the data-management infrastructure by substituting the original Fortran 90 arrays by DAGHs provided by the `C++` class library, and by adding a Berger-Oliger AMR driver. The new version retains the original Fortran 90 kernels that operate on grids at each level.

4.2 Representation Overheads

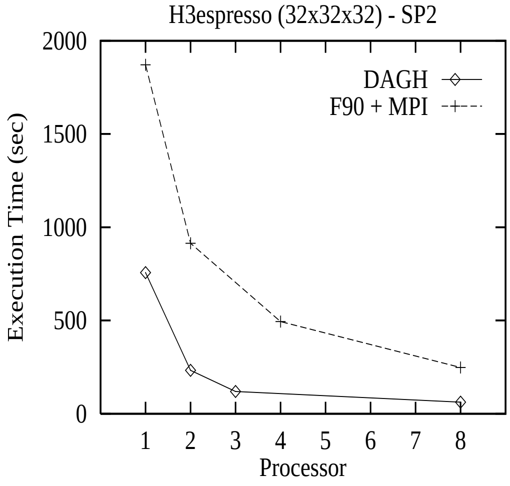


Figure 5: DAGH Overhead Evaluation

The overheads of the proposed DAGH/SDDG representation are evaluated by comparing the performance of a hand-coded, unigrid, Fortran 90+MPI implementation of the *H3expresso* application with a version built using the data-management infrastructure. The hand-coded implementation was optimized to overlap the computations in the interior of each grid partition with the communications on its boundary by storing the boundary in separate arrays. Figure 5 plots the execution time for the two codes.

4.3 Composite Partitioning Evaluation

The results presented below were obtained for a 3-D base grid of dimension $8 \times 8 \times 8$ and 6 levels of refinement with a refinement factor of 2.

4.3.1 Load Balance

To evaluate the load distribution generated by the composite partitioning scheme we consider snap-shots of the distributed grid hierarchy at arbitrary times during integration. Normalized computational load at each processor for the different snap-shots are plotted in Figures 6-9. Normalization is performed by dividing the computational load actually assigned to a processor by the computational load that would have been assigned to the processor to achieve a perfect load-balance. The latter value is computed as the total computational load of the entire DAGH divided by the number of processors.

Any residual load imbalance in the partitions generated can be tuned by varying the granularity of the

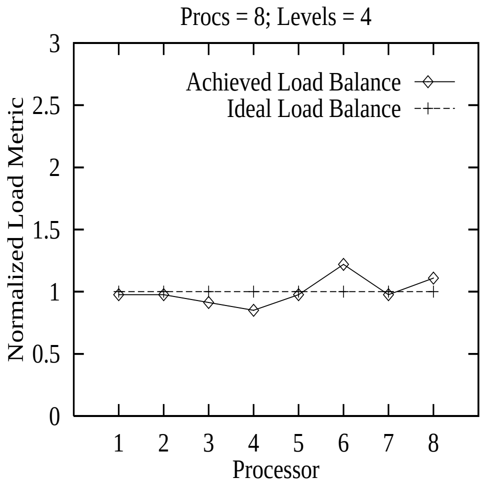


Figure 6: DAGH Distribution: Snap-shot I

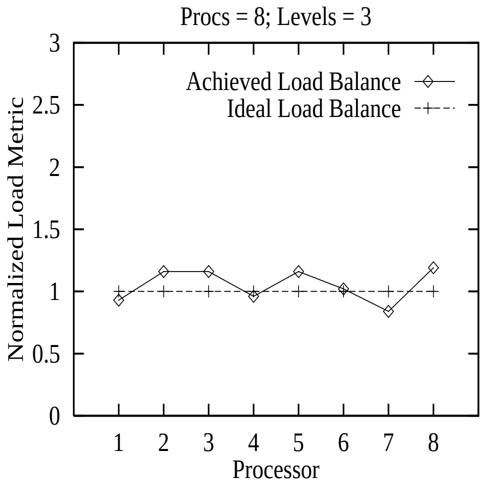


Figure 7: DAGH Distribution: Snap-shot II

SDDG/DAGH blocks. Smaller blocks can increase the regriding time but will result in smaller load imbalance. Since AMR methods require re-distribution at regular intervals, it is usually more critical to be able to perform the re-distribution quickly than to optimize each distribution.

4.3.2 Inter-Grid Communications

Both prolongation and restriction inter-grid operations were performed locally on each processor without any communication or synchronization.

4.4 Partitioning Overheads

Partitioning is performed initially on the base grid, and on the entire grid hierarchy after every regrid.

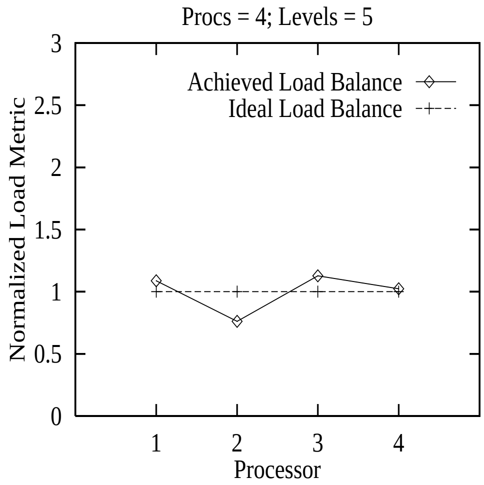


Figure 8: DAGH Distribution: Snap-shot III

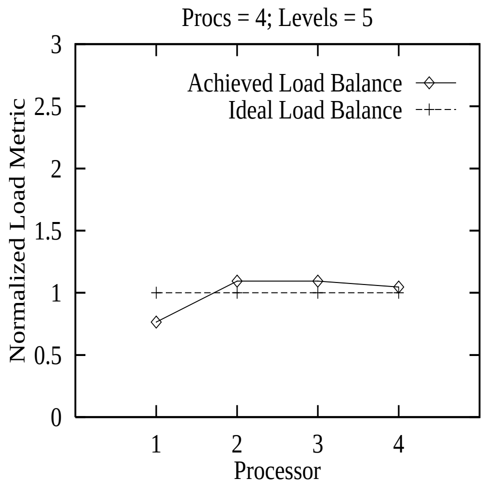


Figure 9: DAGH Distribution: Snap-shot IV

Regriding any level l comprises of refining at level l and all level finer than l; generating and distributing the new grid hierarchy; and performing data transfers required to initialize the new hierarchy. Table 1 compares the total time required for regriding, i.e. for refinement, dynamic re-partitioning and load balancing, and data-movement, to the time required for grid updates. The values listed are cumulative times for 8 base grid time-steps with 7 regrid operations.

5 Conclusions

This paper presented the design and implementation of dynamic distributed data-structures to support parallel adaptive finite difference applications based on hierarchical adaptive mesh-refinement

Procs	Update Time	Regriding Time
4	28.5 sec	1.84 sec
8	19.2 sec	1.58 sec

Table 1: Dynamic Partitioning Overhead

(AMR) techniques for the solution of partial differential equations. The design of the data-structures uses a linear representation of the hierarchical, multi-dimensional grid structure, which is generated using a space-filling mapping. Computational data associated with the grids in the hierarchy is maintained as an scalable distributed dynamic array (SDDA) that is based on extendible hashing and guarantees preservation of locality under expansion and contraction. A data-management infrastructure for distributed hierarchical AMR has been developed based on the data-structures described in this paper. A representative application from numerical relativity is used to experimentally evaluate the infrastructure. Initial evaluation shows that the presented data-structure representation has no significant overheads. The composite distribution generated by partitioning the DAGH representation produces an load imbalance of at most 25%, and requires less than 10% of the actual computation time for partitioning and dynamic load-balancing. The resulting partitions maintain locality and require no communications during inter-grid operations.

References

[1] Giuseppe Peano, "Sur une courbe, qui remplit toute une aire plane", *Mathematische Annalen*, **36**:157–160, 1890.

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

[3] R. Fagin, "Extendible Hashing - A Fast Access Mechanism for Dynamic Files", *ACM TODS*, **4**:315–344, 1979.

[4] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard", Technical Report CS-94-230, Computer Science Department, University of Tennessee, Knoxville, TN, Mar. 1994.

[5] Hans Sagan, *Space-Filling Curves*, Springer-Verlag, 1994.

[6] J. Massó and C. Bona, "Hyperbolic System for Numerical Relativity", *Physics Review Letters*, **68**(1097), 1992.