On Partitioning Dynamic Adaptive Grid Hierarchies*

Manish Parashar and James C. Browne
Department of Computer Sciences
University of Texas at Austin
{parashar, browne}@cs.utexas.edu
(To be presented at HICSS-29, January, 1996)

Abstract

This paper presents a computationally efficient run-time partitioning and load-balancing scheme for the Distributed Adaptive Grid Hierarchies that underlie adaptive mesh-refinement methods. The partitioning scheme yields an efficient parallel computational structure that maintains locality to reduce communications. Further, it enables dynamic re-partitioning and load-balancing of the adaptive grid hierarchy to be performed cost-effectively. The run-time partitioning support presented has been implemented within the framework of a data-management infrastructure supporting dynamic distributed data-structures for parallel adaptive numerical techniques. This infrastructure is the foundational layer 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. 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 decomposed and mapped across processors. The former enables efficient computational access to the grids while the latter minimizes the total communication and synchronization overheads. Furthermore, the dynamic nature of the adaptive grid hierarchy makes it is necessary to re-partition the hierarchy on-the-fly so that it continues to meet these goals.

The partitioning scheme presented in this paper is based on a recursive linear representation of a multi-dimensional grid hierarchy that can be efficiently generated and maintained. This representation is used to design distributed data-structures to support parallel adaptive methods, which are then dynamically partitioned and re-partitioned. The problem of partitioning and dynamically re-partitioning the multi-dimensional grid hierarchy is thus reduced to appropriately decomposing its one-dimensional representation. The partitioning scheme first defines an extendable, ordered index space using extendible hashing techniques [1]. Space-filling curves [2, 3] are then used to define a mapping from the multi-dimensional grid hierarchy to the linear index-space. The self-similar nature of these mappings is exploited to maintain locality across levels of the grid hierarchy while their locality preserving characteristics guarantees locality within partitions of individual grids in the hierarchy. The ordering of the one-dimensional DAGH representation is maintained under expansion and contraction of the grid hierarchy so that re-distribution can be performed with reduced (typically nearest-neighbor) communication.

*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 rest of this paper is organized as follows: Section 2 describes the adaptive grid structure defined by hierarchical adaptive mesh-refinement techniques, and outlines the operations performed on this hierarchy. Section 3 discusses the parallelization of these adaptive techniques and the decomposition of the adaptive grid hierarchy. Section 4 describes the design of the distributed dynamic data-structures that are used to implement the adaptive grid hierarchy, and presents a run-time partitioning scheme for these data-structures. Section 5 presents experimental results to evaluate the effectiveness of the run-time partitioning support and the associated overheads. Section 6 presents some concluding remarks.

2 Problem Description

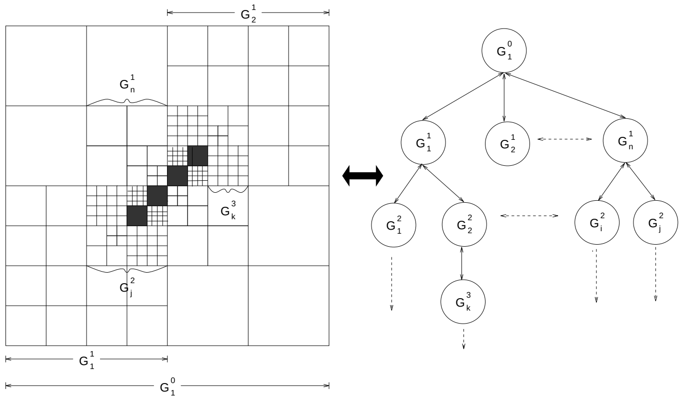


Figure 1: Adaptive Grid Hierarchy - 2D (Berger-Oliger AMR Scheme)

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. The adaptive grid hierarchy corresponding to the AMR formulation by Berger & Oliger [4] is shown in Figure 1. Operation on the hierarchy defined by this algorithm are outlined below:

Time Integration: Time integration is the update operation performed on each grid at each level of the adaptive grid hierarchy. Integration uses an application specific difference operator.

Inter-Grid Operations: Inter-grid operations are used to communicate solutions values along the adaptive grid hierarchy. Two primary inter-grid operations are *Prolongations* operations defined from a coarser grid to a finer grid and *Restriction* operations defined from a finer grid to a coarser grid.

Regriding: The regriding operation consists of three steps: (1) flagging regions needing refinement based on an application specific error criterion, (2) clustering flagged points, and (3) generating the refined grid(s). The regriding operation can result in the creation of a new level of refinement or additional grids at existing levels, and/or the deletion of existing grids.

3 Parallelization of Adaptive Techniques

Parallelization of adaptive methods based on hierarchical AMR consists of appropriately partitioning the adaptive grid hierarchy across available computing nodes, and concurrently operating on the local portions of this domain. Parallel AMR applications require two primary types of communications:

Inter-grid Communications: Inter-grid communications are defined between component grids at different levels of the grid hierarchy and consist of prolongations (coarse to fine transfers) and restrictions (fine to coarse transfers). These communications typically require a gather/scatter type operations based on an interpolation or averaging stencil. Inter-grid communications can lead to serialization bottlenecks for naive decompositions of the grid hierarchy.

Intra-grid Communication: Intra-grid communications are required to update the grid-elements along the boundaries of local portions of a distributed grid. These communications consist of near-neighbor exchanges based on the stencil defined by the difference operator. Intra-grid communications are regular and can be scheduled so as to be overlapped with computations on the interior region of the local portions of a distributed grid.

3.1 Decomposing the Adaptive Grid Hierarchy

Key requirements of a decomposition scheme used to partition the adaptive grid hierarchy across processors are: (1) expose available data-parallelism; (2) minimize communication overheads by maintaining inter-level and intra-level locality; (3) balance overall load distribution; and (4) enable dynamic load re-distribution with minimum overheads. A balanced load distribution and efficient re-distribution is particularly critical for parallel AMR-based applications as different levels of the grid hierarchy have different computational loads. In case of the Berger-Oliger AMR scheme for time-dependent applications, space-time refinement result in refined grids which not only have a larger number of grid elements but are also updated more frequently (i.e. take smaller time steps). The coarser grid are generally more extensive and hence its computational load cannot be ignored. The AMR grid hierarchy is a dynamic structure and changes as the solution progresses, thereby making efficient dynamic re-distribution critical.

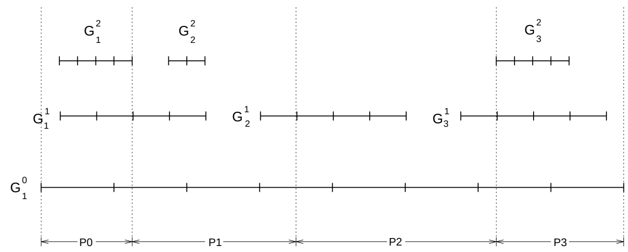


Figure 2: Composite distribution of the grid hierarchy

The *composite* decomposition of the adaptive grid hierarchy shown in Figure 2 addresses the issues outlined above. The primary aim of this decomposition is to alleviate the cost of potentially expensive inter-grid communications. This is achieved by decomposing the hierarchy is such a way that these communications become local to each processors. Parallelism across component grids at each level is fully exploited in this scheme. The composite decomposition scheme requires re-distribution when component grids are created or destroyed during regriding. This re-distribution can be performed incrementally and the data-movement is usually limited to neighboring processors. Alternate decompositions of the adaptive grid hierarchy are discussed in Appendix A.

Although the composite decomposition can efficiently support parallel AMR methods, generating and maintaining this distribution using conventional data-structure representations can result in large amounts of communications and data movement which in turn can offset its advantages. The following section presents a representation of the adaptive grid hierarchy, and distributed dynamic data-structures based on this representation, that enable a composite decomposition to be efficiently generated and maintained.

4 Run-Time Partitioning of Dynamic Adaptive Grid Hierarchies

Run-time partitioning support for decomposing dynamic adaptive grid hierarchies is based on a linear representation of the grid hierarchy and has been incorporated into the design of two distributed data-structures supporting dynamically adaptive numerical techniques:

- A Scalable Distributed Dynamic Grid (SDDG) which is a distributed and dynamic array, and is used to implement each grid in the adaptive grid hierarchy.
- A Distributed Adaptive Grid Hierarchy (DAGH) which is a dynamic collection of SDDGs and implements the entire adaptive grid hierarchy.

The SDDG/DAGH linear representation is generated using space-filling curves introduced below. A detailed discussion of the design of these data-structures is presented in [5].

4.1 Space-Filling Curves

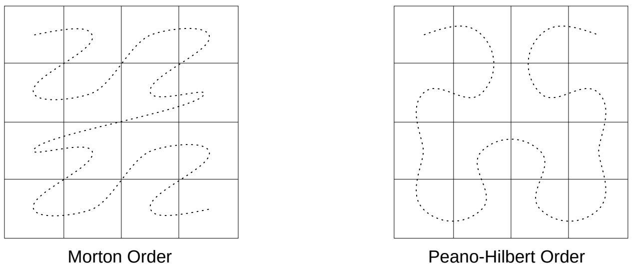
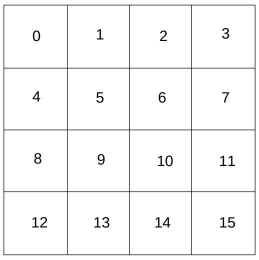


Figure 3: Space-Filling Curves - Examples

Space-filling curves [2, 3, 6] 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. Two such mappings, the Morton order and the Peano-Hilbert order, are shown in Figure 3. 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.

4.1.1 SDDG Representation:



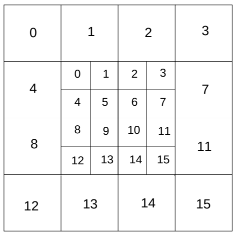
{0 1 4 5 2 3 6 7 8 9 12 13 10 11 14 15} (Morton)

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

Figure 4: 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 4 illustrates this representation for a 2-dimensional SDDG using 2 different space-filling curves (Morton & Peano-Hilbert).

4.1.2 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 5: 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 5 illustrates the composite representation for a two dimensional grid hierarchy.

4.2 SDDG/DAGH Associated Storage

SDDG/DAGH storage consists of two components; (1) storage of the SDDG/DAGH representation and (2) storage of the associated data. The overall storage scheme is shown in Figure 6. The SDDG/DAGH representation (which represents the structure of the adaptive grid hierarchy) is stored as an ordered list of SDDG or DAGH blocks. 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

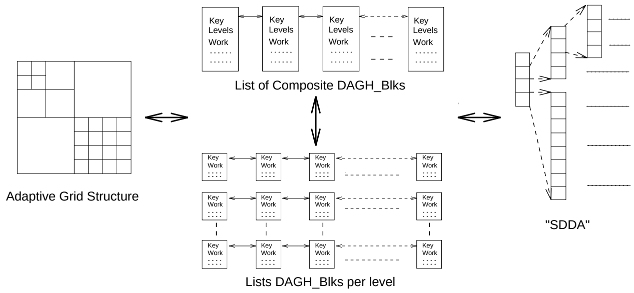


Figure 6: SDDG/DAGH Storage Scheme

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.

Data associated with the SDDG/DAGH data-structures is stored within a "Scalable Distributed Dynamic Array" (SDDA) which uses extendable hashing techniques [1] 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 SDDG/DAGH blocks and the array is indexed using associated keys. The SDDA data storage provides a means for efficient communication between SDDG/DAGH blocks.

4.3 Partitioning SDDGs/DAGHs

Partitioning a SDDG across processing elements using the one-dimensional representation consists of appropriately partitioning the SDDG block list so as to balance the 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.

In case of DAGHs, different decompositions can be obtained by partitioning the one-dimensional lists associated with either representation based on their assigned costs. In particular the desired composite decomposition can be easily generated by partitioning the composite DAGH representation to balance the total cost assigned to each processor. This decomposition using Morton and Peano-Hilbert space-filling ordering is shown in Figure 7. The resulting decomposition for a more complex DAGH is shown in Figures 8. Note that each numbered unit in this figure is a composite block of sufficient granularity. As inter-level locality is inherently maintained by the composite representation, the decomposition generated by partitioning this representation eliminates expensive gather/scatter communication and allows prolongation and restriction operations to be performed locally at each processor. Further, as the SDDG and DAGH one-dimensional representations consist of ordered grid blocks, the granularity of these blocks can be used to reduce communications.

A regriding operation using the defined DAGH representation is performed in four steps.

1. Each processor locally refines its portion of the composite list of DAGH blocks.

2. A global concatenate operation is performed providing each processor with the complete new composite DAGH list.

3. Each processor partitions composite DAGH list and determines the portion of the DAGH assigned to all processors.

4. Processor perform required data-movement to update the new hierarchy.

Data-movement in the final step is performed using non-blocking communications since each processor has a global view of the hierarchy (entire composite list). Each processor first posts receives for all incoming data and then dispatches outgoing data. In order to allow refinements to accurately track solution features, AMR integration algorithms require regriding to be performed at regular intervals. As a result, there are small changes between the old and new grid hierarchies and data-movement typically consist of small amounts of near-neighbor exchanges.

Other distributions of the grid hierarchy (presented in Appendix A) can also be generated using the above representation. For example, a distribution that decomposes each grid separately (Independent Grid Distribution) is generated by viewing the DAGH list as a set of SDDG lists.

5 Experimental Evaluations

The run-time partitioning support described in this paper has been incorporated into a data-management infrastructure for distributed hierarchical AMR [7]. The infrastructure is implemented as a C++ class library on top of the MPI [8] communication system,

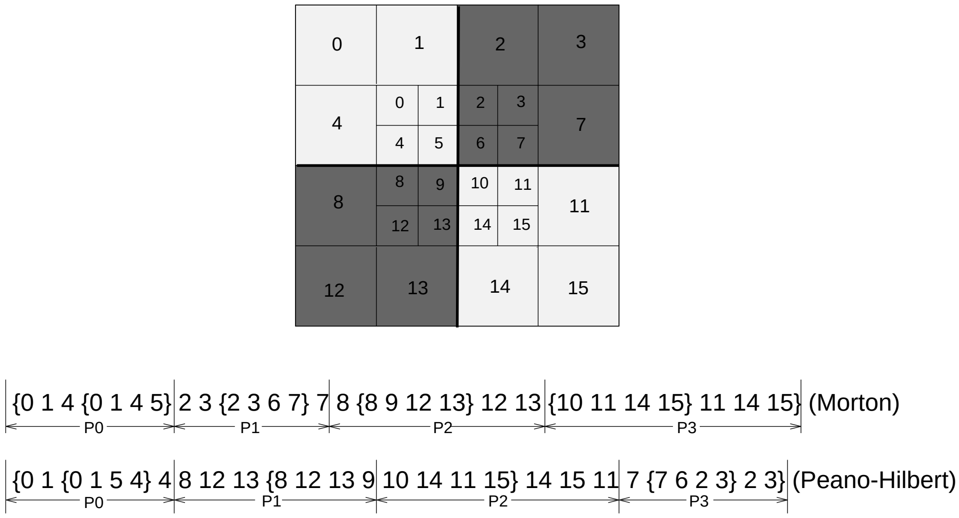


Figure 7: Composite partitioning of a 2-D DAGH

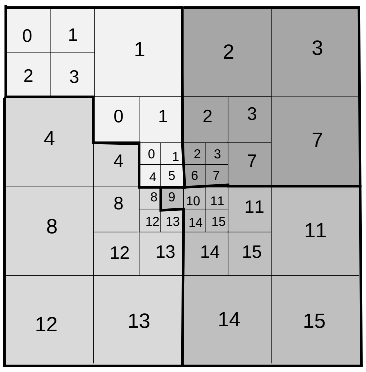


Figure 8: DAGH Composite distribution

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. The system currently runs on the IBM SP2, Cray T3D, clusters of networked workstations (SUN, RS6000, SGI). Results from an initial experimental evaluation of the infrastructure and the run-time partitioning scheme 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 and regular block partitioning for unigrid applications.

2. To evaluate the effectiveness of the partitioning scheme in terms of its ability to meet requirements outlined in Section 3; i.e. balance load, maintain locality and reduce communication requirements.

3. To evaluate the overheads of partitioning and dynamically re-partitioning the grid hierarchy using the scheme outlined.

5.1 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 [9]. 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.

5.2 Representation Overheads

Performance overheads due the DAGH/SDDG representations are evaluated by comparing the performance of a hand-coded, unigrid, Fortran 90+MPI im-

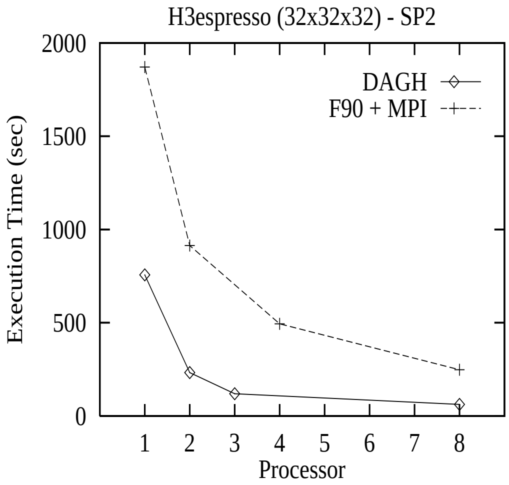


Figure 9: DAGH Overhead Evaluation

plementation 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 9 plots the execution time for the two codes.

5.3 Effectiveness of the Partitioning Scheme

Effectiveness of the partitioning scheme is evaluated by its ability to balance the computational load of the dynamic grid hierarchy across available processing elements while maintaining locality so as to reduce communications requirements. 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.

5.3.1 Load Balance

To evaluate the load distribution generated by the partitioning scheme we consider snap-shots of the distributed grid hierarchy at arbitrary times during integration. The structures of the DAGH in 5 such snap-shots are listed in Table 1. Efficiency at a grid level refers to the efficiency of regriding and is computed as one minus the fraction of the base-grid that is refined. Normalized computational load at each processor for the different snap-shots are plotted in Figures 10-14. 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

Num	Procs	DAGH Structure		
		Level	Efficiency	Load Metric
I	8	0	0.0	6268
		1	0.870095	13294
		2	0.969519	49908
		3	0.994657	139968
II	8	0	0.0	6396
		1	0.870095	13294
		2	0.969519	49908
III	4	0	0.0	6268
		1	0.874661	12570
		2	0.984111	25496
		3	0.994835	132616
		4	0.998852	471648
IV	4	0	0.0	6268
		1	0.874661	12570
		2	0.984111	25496
		3	0.994835	132616
		4	0.999617	157216
V	4	0	0.0	6396
		1	0.870095	13294
		2	0.969519	49908
		3	0.994657	139968
		4	0.999437	235824

Table 1: DAGH Snap-shots

latter value is computed as the total computational load of the entire DAGH divided by the number of processors.

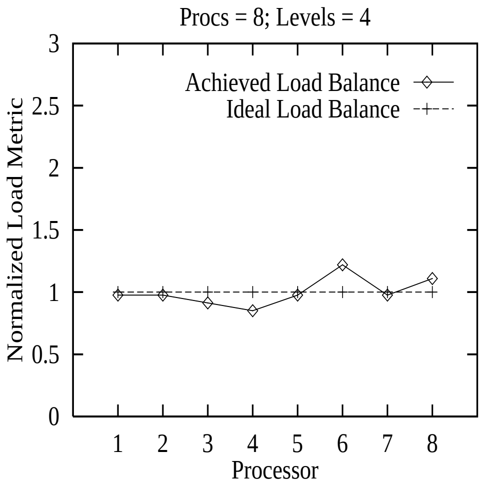


Figure 10: DAGH Distribution: Snap-shot I

The residual load imbalance in the partitions generated can be tuned by varying the granularity of the 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 a distribution.

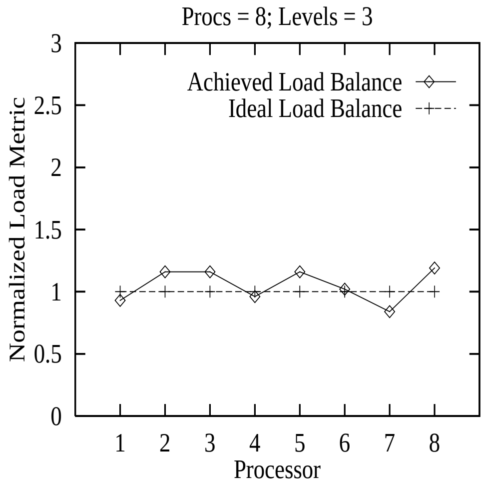


Figure 11: DAGH Distribution: Snap-shot II

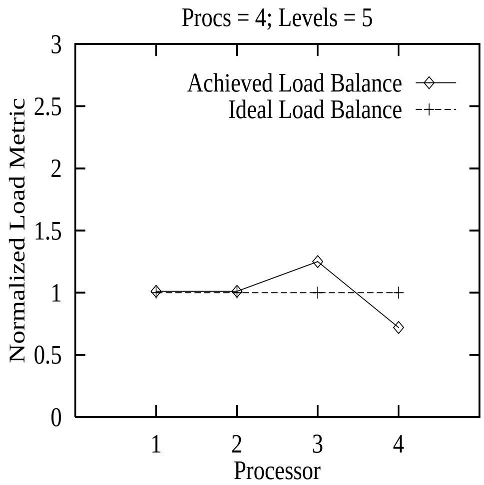


Figure 12: DAGH Distribution: Snap-shot III

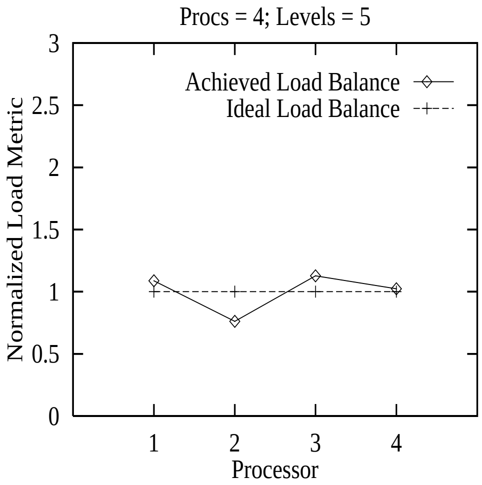


Figure 13: DAGH Distribution: Snap-shot IV

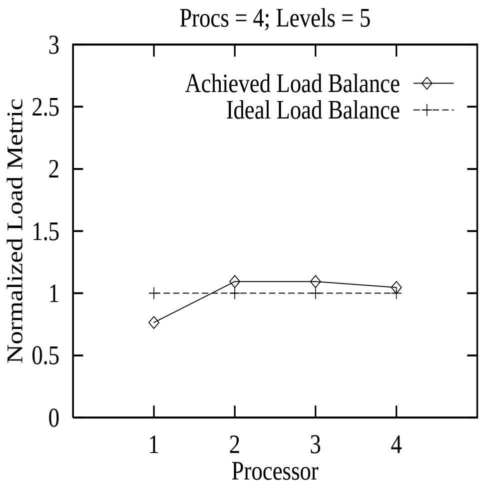


Figure 14: DAGH Distribution: Snap-shot V

5.3.2 Inter-Grid Communications Requirements

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

5.4 Partitioning Overheads

Partitioning is performed initially on the base grid, and on the entire grid hierarchy after every regrid operation. 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 2 compares 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.

6 Conclusions

This paper presented a run-time partitioning scheme for the Distributed Adaptive Grid Hierarchies that underlie adaptive multigrid techniques based on adaptive-mesh refinements. The partitioning scheme is based on a recursive one-dimensional representation of the adaptive grid structure generated using a hierarchical, extendable index-space. Extendible hashing techniques are used to define the extendable index-space; while space-filling curves are used to map a n-dimensional grid hierarchy to the index-space. This representation is used to design dis-

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

Table 2: Dynamic Partitioning Overhead

tributed data-structures to support parallel adaptive methods, which are then dynamically partitioned and re-partitioned. Partitions generated using this representations are shown to maintain 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. This reduces the amount of communications required. Further, re-distribution of the grid hierarchy required during regriding can be performed cost-effectively. A representative application from numerical general relativity is used to experimentally evaluate the partitioning scheme. Initial evaluation shows that the presented data-structure representation has no significant overheads. Further, the evaluation shows that the scheme generated an imbalance of at most 25% with less than 10% of the total application integration time being spent on partitioning and load-balancing. The resulting partitions of the adaptive grid hierarchy require no communications during inter-grid operations.

References

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

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

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

[4] Marsha J. Berger and Joseph Oliger, "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations", *Jounal of Computational Physics*, pp. 484–512, 1984.

[5] 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.

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

[7] 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, 2.400 Taylor Hall, Austin, TX 78712, 1995, Available via WWW at http://godel.ph.utexas.edu/Members/parashar/toolkit.html.

[8] 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.

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

A Decompositions of the Dynamic Adaptive Grid Hierarchy

A.1 Independent Grid Distribution

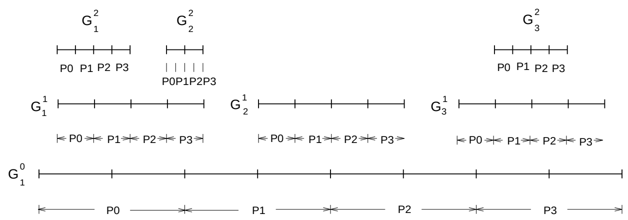


Figure 15: Independent grid distribution of the grid hierarchy

The independent grid distribution scheme shown in Figure 15 distributes the grids at different levels independently across the processors. This distribution leads to balanced loads and no re-distribution is required when grids are created or deleted. However, the decomposition scheme can be very inefficient with regard to inter-grid communication. In the adaptive grid hierarchy, a fine grid typically corresponds to a small region of the underlying coarse grid. If both, the fine and coarse grid are distributed over the entire set of processors, all the processors (corresponding to a fine grid distribution) will communicate with the small set of processors corresponding to the associated coarse grid region, thereby causing a serialization bottleneck. For example, in Figure 15, a restriction from grid G_2^2 to grid G_1^1 requires all the processors to communicate with processor *P3*

Another problem with this distribution is that parallelism across multiple grids at a level is not exploited. For example, in Figure 15, grids G_1^1, G_2^1 & G_3^1 are distributed across the same set of processors and have to integrated sequentially.

A.2 Combined Grid Distribution

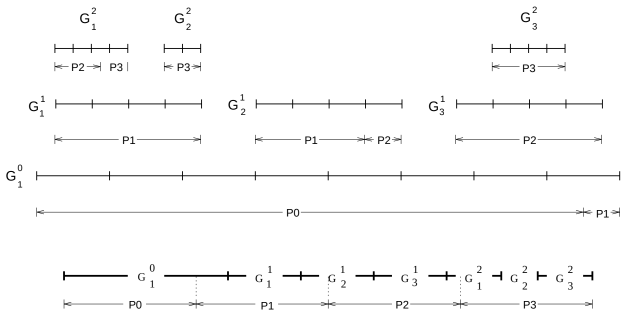


Figure 16: Combined grid distribution of the grid hierarchy

The combined grid distribution, shown in Figure 16, distributes the total work load in the grid hierarchy by first forming a simple linear structure by abutting grids at a level and then decomposing this structure into partitions of equal load. The combined decomposition scheme also suffer from the serialization bottleneck described above but to a lesser extent. For example, in Figure 16, G_1^2 and G_2^2 update G_1^1 requiring $P2$ and $P3$ to communicate with $P1$ for every restriction. Regriding operations involving the creation or deletion of a grid are extremely expensive in this case as they requires an almost complete re-distribution of the grid hierarchy.

The combined grid decomposition does not exploit the parallelism available within a level of the hierarchy. For example, when G_1^0 is being updated, processors $P2$ and $P3$ are idle and $P1$ has only a small amount of work. Similarly when updating grids at level 1 (G_1^1, G_2^1 and G_3^1) processors $P0$ and $P3$ are idle, and when updating grids at level 2 (G_1^2, G_2^2 and G_3^2) processors $P0$ and $P1$ are idle.

A.3 Independent Level Distribution

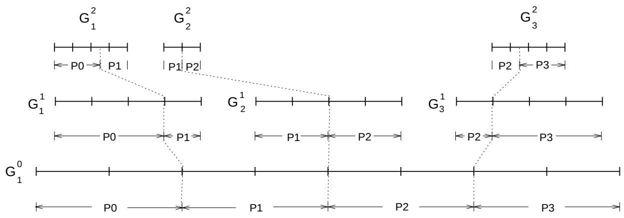


Figure 17: Independent level distribution of the grid hierarchy

In the independent level distribution scheme (see Figure 17), each level of the adaptive grid hierarchy is individually distributed by partitioning the combined load of all component grids at the level is distributed among the processors. This scheme overcomes some of the drawbacks of the independent grid distribution. Parallelism within a level of the hierarchy is exploited. Although the inter-grid communication bottleneck is reduced in this case, the required gather/scatter communications can be expensive. Creation or deletion of component grids at any level requires a re-distribution of the entire level.