

An Environment for Parallel Multi-Block, Multi-Resolution Reservoir Simulations

Manish Parashar
Department of Electrical & Computer Engineering
Rutgers, The State University of New Jersey
94 Brett Road, Piscataway, NJ 08854
Tel: (732) 445-5388; Fax: (732) 445-0593
parashar@caip.rutgers.edu

Ivan Yotov
Center For Subsurface Modeling & TICAM
University of Texas at Austin
2.400 Taylor Hall, Austin, TX 78712
Tel: (512) 471-2158; Fax: (512) 471-8694
yotov@ticam.utexas.edu

Abstract

This paper describes the design and implementation of a Problem Solving Environment (PSE) for developing parallel reservoir simulators that can handle multiple blocks, multiple physical models, and multiple resolution using dynamic locally adaptive mesh-refinements. The objective of the PSE is to reduce the complexity of building flexible and efficient parallel reservoir simulators through the use of a high-level programming interface for problem specification and model composition, object-oriented programming abstractions that implement application objects, and distributed dynamic data-management that efficiently supports adaptation and parallelism. The PSE has been used to develop a new generation parallel implicit multi-block, multi-phase flow simulator.

1 Introduction

The primary goal of the new generation of parallel multi-block, multi-scale reservoir simulators [1] is to support realistic, high-resolution reservoir studies with a million or more grid elements on massively parallel computers. Key requirements for these simulators include the ability to handle multiple physical models, generalized well management, multiple blocks, and dynamic, locally adaptive mesh-refinements. In this paper we describe the design and development of a Problem Solving Environment (PSE) to support the development of such reservoir simulators. Our research is motivated by the multi-disciplinary structure of the new generation simulators and the inherent complexity of their implementation. In a typical reservoir simulator on the order of tens of thousands of lines of code may be required to support a single physical model. It is difficult and inefficient for individual researchers to develop such a framework before even beginning to test their ideas for a physical model. Further, a large percentage of this framework which deals with managing adaptively, multiple blocks, parallelism, data distribution, and dynamic load-balancing is not directly related to the physical problem and can be reused. Clearly, a common infrastructure that provides computational support and parallel adaptive data-management will result in enormous savings in development and coding effort and will make individual research at universities and commercial

laboratories more efficient. The PSE presented in this paper provides such an infrastructure. Its primary objectives are:

1. To alleviate the complexity of implementing new reservoir simulators due to parallel data-management, dynamic mesh-refinement, and multiple blocks, by defining appropriate data-structures, programming abstractions and high-level programming interfaces.
2. To provide a high-level application programming interface for problem specification and model composition.
3. To provide a general framework for integrating input/output, visualization, and interactive experimentation with applications computation.
4. To use the most appropriate language and technology for individual tasks in the PSE. For example, to implement low level data-management in C, to build programming abstractions in an object-oriented language like C++, and to develop computational kernels in FORTRAN.

The PSE design is based on a clean separation of concerns across its functionality and the definition of hierarchical levels of abstraction based on this separation. The overall schematic of the design is shown in Figure 1.

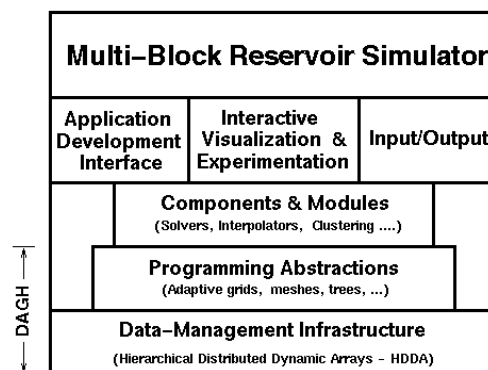


Figure 1 - Problem Solving Environment for Parallel Adaptive, Multi-Block Reservoir Simulation

The lowest layer of the PSE is a Hierarchical Distributed Dynamic Array (HDDA). HDDA provides pure array semantics to hierarchical, dynamic and physically distributed data, and has been successfully used as a data-management infrastructure for a variety of parallel adaptive algorithms

such adaptive mesh-refinement and multigrid methods, h-p adaptive finite element methods and adaptive multipole method. HDDA objects encapsulate distribution, dynamic load-balancing, communications, and consistency management. These objects have been extended with input/output and visualization capabilities so as to fundamentally integrate IO, visualization and interaction at the base layer of the PSE. The next layer, programming abstractions, adds application semantics to HDDA objects and implements application objects such as grids, meshes and trees. This layer provides object-oriented programming abstractions that can be used to directly implement parallel adaptive algorithms. The intermediate layers of the PSE implement application specific methods and components. The topmost layer is a high-level application programming interface (API) customized to parallel reservoir simulation. This API provides an overall skeleton and enables the developer to specify grid and grid block structures, adaptation strategies, grid elements and functions, and plug in FORTRAN or C computational kernels.

2 Problem Description: Subsurface Modeling

Engineering solutions to fluid flow problems in heterogeneous porous media often require large scale computations to understand the design and implementation aspects of the proposed system. This applies to problems in both ground-water remediation and enhanced oil recovery. Typical engineering processes involve complex interactions among physical transport, micro-biological activity, chemistry and geochemistry, and phase behavior. These flow in porous media problems are modeled by degenerate parabolic and nearly hyperbolic (i.e. advection-dominated) partial differential equations with equality and inequality constraints, and are subject to hysteresis. Features that make the above problems difficult for numerical simulation include: multiple phases and chemical components, multi-scale heterogeneities, widely disparate time scales, e.g., the scales of reaction rates, precipitation, dissolution, and other phase changes, stiff gradients, irregular geometries with internal boundaries such as faults and layers, and multi-physics.

2.1 Computational approach

Our computational approach is based on a novel multi-block domain decomposition methodology for subsurface modeling, with multi-resolution grids defined on each block. The multi-block multi-resolution formulation of the discrete systems of equations for subsurface modeling allows efficient parallel domain decomposition solvers, and preconditioners that maximize data and computation locality, to be designed and applied. Recent work has demonstrated that the multi-block, multi-resolution domain decomposition is a viable approach for modeling subsurface flow and transport. Physical and mathematical considerations leads us to

emphasize locally mass conservative schemes, in particular mixed finite element (finite volume) methods for subdomain discretizations. Theoretical and numerical results for single phase flow indicate multi-block mixed finite element methods are highly accurate (superconvergent) for both pressure and velocity [2][3][4]. A parallel non-overlapping domain decomposition implementation provides an efficient scalable solution technique, and is extended to a degenerate parabolic equation arising in two phase flow. Grid structures associated with multi-block multi-resolution formulations are described below.

2.2 Multi-Block and Hybrid Domain Decomposition

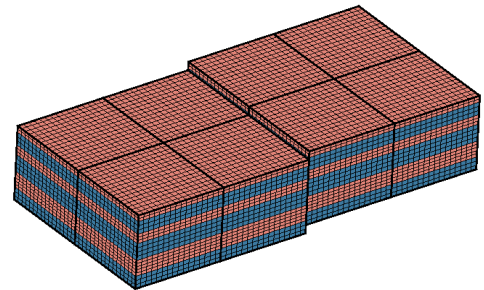


Figure 2 - Multi-Block Grid Structure with 2 Non-Matching Blocks.

In many cases geometrically highly irregular domains can be described as unions of relatively simple blocks. Each block is independently covered by a locally constructed grid. The grids do not have to match on the interfaces between blocks. Special approximating spaces (mortars) are introduced on interfaces of adjacent subdomains. This paradigm is consistent with a physical/engineering description of the mathematical equations: that is, the equations hold with their usual meaning on the sub-domains, which have physically meaningful interface boundary conditions between them. The multi-block approach is computationally attractive as the local grid structure allows more efficient and accurate discretization techniques to be employed. Moreover, structured and unstructured grids can be coupled if the geometry of a given block is very irregular as in the case of pinch-outs, to create hybrid grids. Since the numerical grids may be non-matching across interfaces, they can be constructed to follow large-scale geological features such as faults, heterogeneous layers, and other internal boundaries. The multi-block approach thus allows a rigorous coupling of different physical processes, mathematical models, or discretization methods in different parts of the simulation domain. The structure of a multi-block grid with 2 non-matching grids is shown in Figure 2. In this example we model displacement of oil by water in a faulted heterogeneous reservoir. A fault cuts through the middle of the domain and divides it into two blocks. The dark layer in this figure represents high permeability regions in the domain.

2.3 Multi-Resolution Grids

Multi-resolution grids can be defined on the blocks of the multi-block structure by dynamically adapting each block. This is very convenient for the fast reconstruction of grids and calculation of stiffness matrices in time-dependent problems. Multi-resolution simulation grids contain multiple levels of resolution within the grid, and are produced by locally refining regions of the grid for more accurate computations within those subregions. Figure 3 shows the multi-resolution (adaptive) grid structure associated with the solution for a Buckley-Leverett quarter five spot test problem.

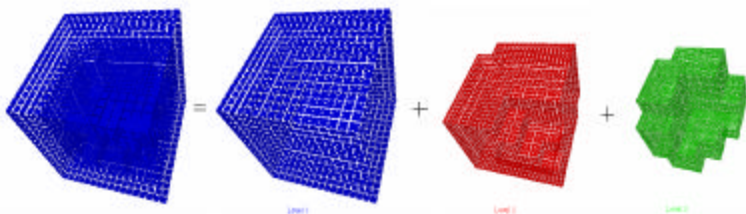


Figure 3 - Multi-Resolution Grid Structure – 3-D Buckley-Leverette Solution with 3 Levels

3 Data-Management for Parallel Multi-Resolution Techniques

Implementations of numerical algorithms for solving PDE's are typically formulated as operations on arrays. This is because vertices in a static grid can be directly represented as elements of an array. Such array based formulations have proven to be simple, intuitive and efficient, and are extensively optimized by current compilers. A similar formulation for dynamically adaptive algorithms then warrants a dynamic array; i.e. an array that can grow and shrink dynamically, and where each element of the array can be an array. Furthermore, parallel implementations of the algorithms require the array to be distributed. Like conventional arrays the dynamic distributed array must translate index locality (corresponding to spatial application locality) to storage locality. This array must additionally maintain this locality through its dynamics and distributions. The design of such are hierarchical, distributed and dynamic array is described below.

3.1 Hierarchical Distributed Dynamic Array (HDDA)

The HDDA data-structure implements a hierarchical distributed dynamic array, providing pure array access semantics to hierarchical, dynamic and physically distributed objects. The array is partitioned and distributed across multiple address spaces with communication, synchronization and consistency managed transparently. The lowest level of the array hierarchy is an object of arbitrary size and structure. The HDDA is designed to directly extract and maintain application data locality requirements. Its

design is based on the application of the fundamental software engineering design principle [5] of "Separation of Concerns". Applying separation of concerns to a convention array data-structure, it can be decomposed into an ordered index-space, storage corresponding to each index in the index-space, and accesses mechanism that enable retrieval of data objects associated with an index. Similarly, the HDDA is composed of (1) a hierarchical, extendible index-space and (2) associated distributed dynamic storage and access mechanisms.

3.1.1 Hierarchical, Extendible Index Space

The hierarchical extendible index space component of the HDDA is derived directly from the application domain using space-filling mappings [6], which are recursive mappings from N-dimensional space to 1-dimensional space. The solution space is first partitioned into segments. The space-filling curve then passes through the midpoints of these segments. A 2-dimensional mapping is shown in Figure 4. Space filling mappings encode application domain locality and maintain this locality through expansion and contraction. 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. The hierarchical index-space is used by the HDDA as the basis for application domain partitioning, as a global name-space for name resolution, and for communication scheduling. Space-filling mapping functions are computationally inexpensive consisting of bit-interleaving operations on the N-dimensional coordinates.

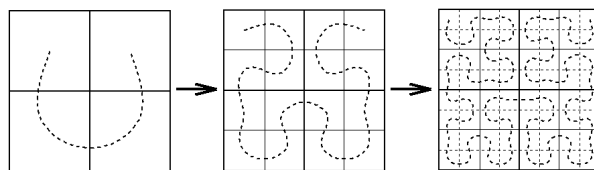


Figure 4 - Space Filling Mapping

3.1.2 Distributed Dynamic Storage and Access

Data storage is implemented using extendible hashing [7] techniques with contractions of the hierarchical index-space indices serving as hash keys. Extendible hashing provides efficient management mechanisms for dynamic databases. Spans of the hash keys space are mapped to units of storage called hash buckets and expansion and contraction of the key space are handled efficiently by splitting and merging these buckets. These operations are local involving at most two buckets. As spans of the index space are mapped to contiguous storage within a bucket by the hashing scheme, index locality (which encodes applications domain locality) is translated into storage locality. Data locality is preserved without copying.

Partitioning of the applications domain and associated distribution of HDDA objects is achieved by partitioning the

index space among processing elements and accordingly assigning ownership to HDDA buckets. Buckets are used as the units of communications and caching. The overall HDDA distributed dynamic storage scheme is shown in Figure 5 [8].

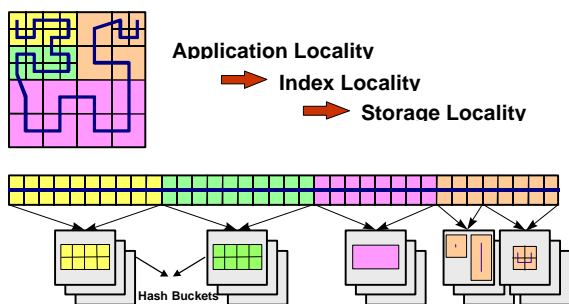


Figure 5 - HDDA Storage Mechanism

3.2 Adaptive Grids and Grid Hierarchies

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 then limited by the ability to partition the DAGH at run-time so as to expose all inherent parallelism, minimize communication and synchronization overheads, and balance load. A critical requirement while partitioning the adaptive grid hierarchy 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. Two fundamental applications objects for parallel adaptive methods are constructed on top of the HDDA.

1. A Scalable Distributed Dynamic Grid (SDDG) which is a single grid in an adaptive grid hierarchy.
2. A Distributed Adaptive Grid Hierarchy (DAGH) which is a dynamic collection of SDDGs implementing an entire adaptive grid hierarchy.

The design uses a linear representation of the hierarchical, multi-dimensional grid structure generated using the global index-space defined above. Operations on the grid hierarchy such as grid creation, grid refinement or coarsening, grid partitioning and dynamic re-partitioning, are then efficiently defined on this one-dimensional representation. The self-similar (i.e. recursive) nature of index-space is exploited to maintain locality across levels of the grid hierarchy. SDDG and DAGH representations are described below.

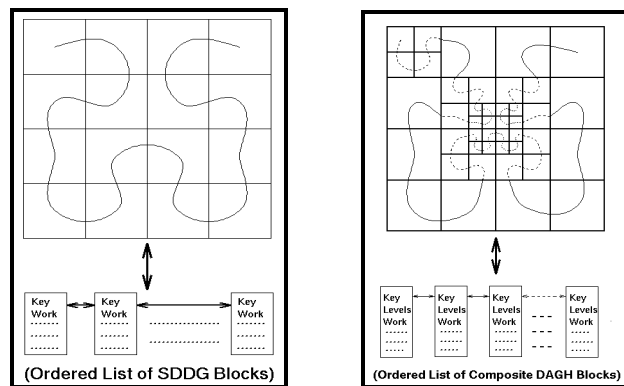


Figure 6 – SDDG & DAGH Representations

3.2.1 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 by mapping them to a span of the global index-space. 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 (adaptive mesh-refinement) 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 6 illustrates this representation for a 2-dimensional SDDG.

Partitioning a SDDG across processing elements using this representation consists of appropriately partitioning the SDDG block list so as to balance the total cost at each processor. Since space-filling curve (used to generate the global index-space) mappings preserve spatial locality, the resulting distribution is comparable to traditional block distributions in terms of communication overheads.

3.2.2 DAGH 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 global index-space: 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 6 illustrates the composite representation for a two dimensional grid hierarchy.

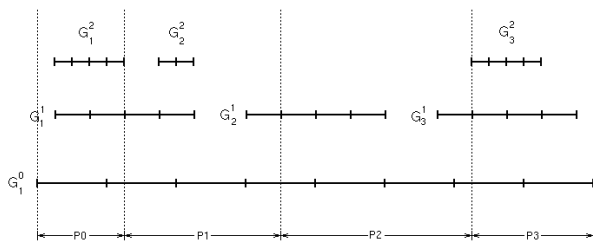


Figure 7 - DAGH Composite Distribution

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 with each processor results in a composite decomposition of the hierarchy [9]. 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 1-dimensional DAGH is shown in Figure 7. Note that each numbered unit in this figure is a composite block of some architecture dependent 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.

3.3 The HDDA/DAGH Framework

The HDDA/DAGH framework provides programming and data-management support for parallel multi-block, multi-resolution applications. It supports a coarse grained data-parallel programming model where parallelism is achieved by operating on regions of the grid hierarchy in parallel. The HDDA/DAGH programming interface is developed so as to enable computations to be performed by traditional FORTRAN 77 and FORTRAN 90 kernels. The HDDA/DAGH application programming interface (API) supports object-oriented programming abstractions that can be used to express parallel adaptive computations based on adaptive mesh refinement and multigrid techniques. The API is designed to provide application developers with a set of primitives that are intuitive for expressing the application, while separating data-management issues and implementations from application specific operations. HDDA/DAGH performance is found to be comparable to and often better than conventional implementation (using FORTRAN) for non-adaptive applications[8]. More information on the HDDA/DAGH framework can be found at <http://www.caip.rutgers.edu/~parashar/DAGH/>.

3.4 Supporting Multi-Block Grids

The HDDA/DAGH framework provides a 2 level combined MIMD – SPMD programming model to support multiple blocks. At the top level processors are divided into groups and these groups operate on different blocks in an MIMD fashion. Within a processor group, computations on a single block proceed using the SPMD programming model. Each block is implemented as a complete grid hierarchy and can be independently refined. Special “mortar” grids are defined to couple adjoining blocks. These grids exist on the intersection of the adjacent faces of the two blocks and are shared by processors responsible for the block faces. Communication between blocks and the mortar grids is handled automatically by the PSE. Figure 8 outlines the support for multiple fault blocks. This picture shows 4 blocks and 4 mortar grids. The dotted oval boxes represent processors sharing a mortar grid.

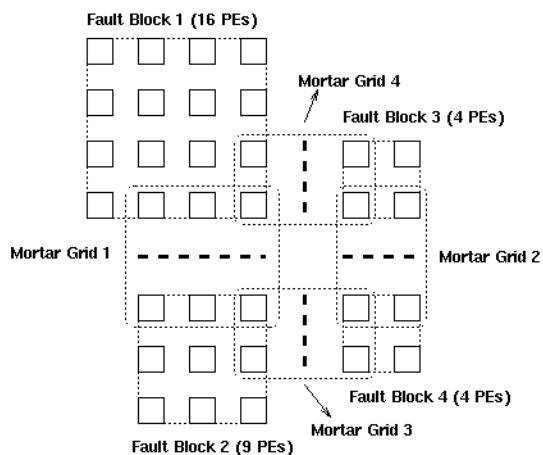


Figure 8 – Multi-Block Support

4 UT-MB: A Parallel Implicit Multi-Block Flow Simulator

The PSE described above has been used in the implementation of a parallel implicit multi-block two-phase flow simulator UT-MB. The mathematical model consists of a coupled system of highly nonlinear transient partial differential equations. The reservoir is described as a series of blocks and interfaces. The multi-block decomposition is induced by the geometrical and geological features of the reservoir. Each block is discretized locally by cell-centered finite differences on logically rectangular grids. Flux-matching conditions are imposed on the interfaces through mortar finite element spaces. The primary unknowns - oil pressure and oil concentration - are approximated on the interfaces by functions in the mortar spaces. Three different types of mortar spaces are implemented - continuous and discontinuous piecewise linear and discontinuous piecewise constants. Fully implicit time discretization leads to an unconditionally stable numerical scheme, but requires solving a nonlinear algebraic system of equations on each

time step. A non-overlapping domain decomposition algorithm reduces this system to a lower dimensional interface problem in the mortar spaces. The interface problem is solved by an inexact Newton method. The algorithm requires only subdomain (block) solves and inexpensive projections between mortar and subdomain spaces. This approach allowed for an already existing single block simulator Piers to be used as a subdomain solver [10]. In the example described here we model displacement of oil by water in a faulted heterogeneous reservoir. A fault cuts through the middle of the domain and divides it into two blocks. The numerical grids follow the geological layers and are non-matching across the fault (see Figure 2). Each block is covered by a $32 \times 32 \times 20$ grid. Continuous piecewise linear mortar finite elements are used along the fault on a 14×16 grid. The simulation was done on eight processors on IBM SP2, each block distributed among four processors. Oil concentration contours after 0.25 pore volume water injected (360 days) are shown in Figure 9 (water is injected at the right front corner and producer is placed at the left back corner).

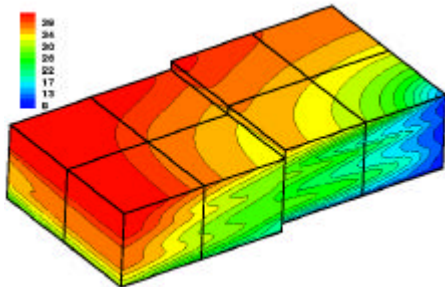


Figure 9 – Oil Concentration Contours

5 Conclusions

In this paper we described the design and implementation of a problem-solving environment to support the development of a new generation of reservoir simulators capable of realistic, high-resolution reservoir studies with a million or more grid elements on massively parallel multicomputer systems. The simulators can handle multiple physical models, generalized well management, multiple blocks, and dynamic, locally adaptive mesh-refinements. The PSE reduces the complexity of building flexible and efficient parallel reservoir simulators through the use of a high-level programming interface for problem specification and model composition, object-oriented programming abstractions that implement application objects, and distributed dynamic data-management that efficiently supports adaptation and parallelism. The PSE is currently operational on a number of high-performance computing platforms including the IBM SP2, CRAY T3E and SGI Origin 2000, as well as networked workstations

6 Acknowledgements

We gratefully acknowledge the financial support of the U.S. Department of Energy ER/MICS and ER/LTR programs that funded Argonne National Laboratory (ANL) and the University of Texas as part of the Advanced Computational Technology Initiative (ACTI) project New Generation Framework for Petroleum Reservoir Simulation. (ANL contract 951522401 to UT), and the ANL Enrico Fermi Scholarship awarded to Manish Parashar. The authors wish to thank the other participants of this project and in particular James C. Browne, Mary Wheeler, John Wheeler, Peng Wang, at the University of Texas at Austin.

7 References

- [1] P. Wang et al., "A New Generation EOS Compositional Reservoir Simulator," Proceeding of the Society of Petroleum Engineers, Reservoir Simulation Symposium, June 1997.
- [2] I. Yotov, "Mortar Mixed Finite-Element Methods on Irregular Multi-Block Domains," Third IMACS International Symposium on Iterative Methods in Scientific Computations, Academic Press, Ed. B. Chen, T. Mathew, J. Wang, 1997.
- [3] T. Arbogast, M.F. Wheeler, and I. Yotov, "Logically Rectangular Mixed Methods for Flow in Irregular, Heterogeneous Domains," in Computational Methods in Water Resources XI, Editors A.A. Aldama et al., Computational Mathematics Publications, Southampton, pp. 621-628, 1996.
- [4] T. Arbogast, I. Yotov, "A Non-Mortar Mixed Finite-Element Method for Elliptic Problems on Non-Matching Multi-Block Grids," In Comput. Meth. Appl. Mech. Engr., vol. 149, pp. 255-265, 1997.
- [5] M. Parashar and J.C. Browne, "System Engineering for High Performance Computing: The HDDA/DAGH Infrastructure for Implementation of Parallel Structures Adaptive Mesh Refinement," IMA Volumes in Mathematics and its Applications, Springer-Verlag, 1997.
- [6] H. Sagan, "Space Filling Curves," Springer-Verlag, 1994.
- [7] H.F. Korth, A. Silberschatz, "Database System Concepts," McGraw Hill. New York, 1991.
- [8] M. Parashar, J.C. Browne, "Distributed Dynamic Data-Structures for Parallel Adaptive Mesh Refinement," Proceedings for the International Conference on High Performance Computing, India, December, 1995, 22-27.
- [9] M. Parashar, J.C. Browne, "On Partitioning Dynamic Adaptive Grid Hierarchies," Proceedings of the 29th Annual Hawaii International Conference on System Sciences, Hawaii, January 1996, 604-613.
- [10] C.N. Dawson, H. Klie, C. San Soucie, M.F. Wheeler, "A parallel, implicit, cell-centered method for two-phase flow with a preconditioned Newton-Krylov solver," Comp. Geo-Sciences, 1998.