Implementation
Infrastructures for Parallel Adaptive Computation Methods

Jim Browne
Dept. of Computer Science/ICES/TACC
browne@cs.utexas.edu
www.cs.utexas.edu/users/dagh
Collaborators, Contributors and Motivators

Dwip Banerjee - initial experiments on distributed, dynamic data structures.
Carter Edwards - implementation of SDDA and finite element programming abstractions.
Manish Parashar - design and implementation of DAGH package.
Tinsley Oden and Abani Patra - hp-adaptive finite elements
Richard Matzner and Matt Choptuik - Einstein’s Equations and Berger-Oliger AMR
History of Infrastructure Development

1992 - Collaboration with Tinsley Oden started
1994 - First Implementation of Dynamic Distributed Array (DDA)
   Banerjee
1994 - Binary Black Hole Collaboration started
1995 - Composite Materials Collaboration started
1995-97 Hierarchical Dynamic Distributed Array (DDA/DAGH)
   Parashar
1995-97 Scalable Dynamic Distributed Array (DDA/hp-FE)
   Edwards
1997 - Dynamic Distributed Array Revisited
   Banerjee
1997 - NCSA Adoption of HDDA/DAGH
Problem Description - Coalescence of Astrophysical Black Holes

Goal
Accurate solution of general two-body problem in general relativity

Generate the waveforms of the gravitational radiation which is emitted.

Solution of Einstein’s Equations for gravitational waves resulting from coalescence of two black holes.
Problem Description - Coalescence of Astrophysical Black Holes

Computational Problem

Nonlinear partial differential equations with moving causal boundary conditions and singularities at inner boundaries.

Dramatic multi-scale numerical process.

Focus on hyperbolic evolution system and elliptic constraint equations.
Problem Description - Coalescence of Astrophysical Black Holes

Choice of computational methods
  Berger-Oliger Adaptive Mesh Refinement for evolution
  Multi-grid for elliptic solvers
Massively parallel implementation
  distributed memory multi-computers
Evolutionary development of codes
  physics and numerics both uncertain.
But certain computational requirements are known.
  parallel AMR
  parallel Multigrid on same system
Adaptive Mesh Refinement (AMR)
Dynamic Parallel Applications
Data-Management Requirements

- Reuse of existing FORTRAN 77 kernels
- Performance, Performance, Performance
- Locality, Locality, Locality!
Definition of Arrays

Array = (D, I, F)

- D = data set - a finite set of data objects
- I = an index space - a countable set of indices and a well-defined linear ordering relation
- F = mapping of D to I - F : D |--> I

NOTES:
1. D is dynamic
2. I is hierarchical
3. F is NOT bijective

Array Implementation
Storage for data objects in D
A function $F_c$ converse to F which maps I |--> D.
Index Spaces and Subspaces

Let I & J be index spaces.
Let the set of indices in J be a subset of the set of indices in I.

\[ i \leq j \iff i \leq j \quad \forall i, j \in J \subseteq I \]

Then J is an index subspace of I

Fortran Index Spaces

\[ I = \{1, 2, \ldots, m\} \times \{1, 2, \ldots, N\} \]

linear ordering is column major

\[(i,j) < (k,l) \iff j < l \text{ or } (j=1 \text{ and } i < k)\]
Hierarchical Index Space

I = { (d, (i_1, i_2, - - - i_d)) : d ∈ [0,L] i_j ∈ [i,N_0,i,j] 
   d = hierarchical depth of an index 
   i_j = ordered at depth j of the hierarchy 
A linear order 
   (k, (i_1, i_2, - - - i_k) ≻ (l, (j_1, j_2,- - - j1)) <=>
   i_1 = j_1 
   or i_1 = j_1 and i_1 < j_2 
   or i, i_2) = (j_1,j_2) and i_3 < j_3 
   or (i_1,i_2, - - - l_{min(k,l)}) = (j_1,j_2, - - - j_{min(k,l)}) 
   and k < l
Locality in Index Spaces

\[ \text{loc}(i,j) = (1 + \text{card} \{k \mid I: i<k<j \text{ or } j<k<i\})^{-1} \]

locality = "inverse" of distance

Converse Functions

\( F_c : \quad F(D) \rightarrow D \)

F(D) is dynamic

\( F_c \) - hashing
extendible hashing
lists
trees,
etc.
Discretization of N-dimensional space creates a partition of physical space.

Associate a point in index space with the “center” of each partition of physical space in the constrained space.

eg. - center of volume element
eg. - point on grid
Map from N-Dim to 1-Dim
Space Filling Curves

- Recursive mapping from N-d space to 1-d space
- Index locality implies geometric locality
- Locality preserved under expansion and contraction
- Computationally efficient mapping functions
Space Filling Curves

Degree 1

Degree 2

Degree 3

Degree 1

Degree 2
Hierarchical, Extendible Index-Space

- Encodes application locality
- Extendible, hierarchical & recursive
- Global name-space
- Linearized index space is basis for partitioning, communication, name resolution, refinement & coarsening
Storage and Access

- **Extendible hashing** on the linearized global index-space
- Index locality implies storage locality
- Locality preserved through expansion & contraction
- Array access semantics
What Data Does Each Processor Need?

- Data it owns plus data its stencil overlaps with data owned by other domains/processors
Extendible Hashing

Dynamic hashing by splitting and coalescing buckets
Distributed Dynamic Storage

Application Locality

Index Locality

Storage Locality
Separation of Concerns => Hierarchical Abstractions

- AMR Application
  - Application Computations
  - Programming Abstractions
  - Dynamic Data Management
Separation of Concerns => Hierarchical Abstractions

HDDA

Index Space
- Partitioning
  - Name Resolution
- Expansion & Contraction

Storage
- Data Objects
- Interaction Objects
- Display Objects

Access
- Consistency
- Communication
Separation of Concerns => Hierarchical Abstractions

- Application Components
  - Modules
    - Solver
  - Kernels
    - Interpolator
    - Error Estimator
    - Clusterer

- Programming Abstractions
  - Grid Function
    - Cell Centered
  - Grid Structure
    - Main Hierarchy
    - Shadow Hierarchy
  - Grid Geometry
    - Region
    - Point
  - Multigrid Hierarchy

- Dynamic Data-Management
  - App. Objects
  - HDDA
    - Grid
    - Index Space
    - Mesh
    - Storage
    - Tree
    - Access

Application Specific
Method Specific
Adaptive Data-Mgmt

10/30/2011
Constructing Distributed Dynamic Data-Structures

Application Domain

Space-Filling Mappings

Hierarchical, Extendable, Global Index-Space

1-D Representation

Extendible Hashing

SDDG
(Scalable Distributed Dynamic Grid)

DAGH
(Distributed Adaptive Grid Hierarchy)

HDDA
(Hierarchical Distributed Dynamic Array)
Application Objects for AMR

- Scalable Distributed Dynamic Grid (SDDG)
- Distributed Adaptive Grid Hierarchy (DAGH)
Scalable Distributed Dynamic Grid

**Representation**: Ordered List of SDDG Blocks
- Block grid for granularity
- SDDG “Key”: SDDG Block -> Global Index-Space Cluster
- Order SDDG blocks using keys
- Assign computation “Work” to SDDG blocks

**Partitioning**:
- Partition SDDG block list to balance work per processor
Distributed Adaptive Grid Hierarchy

- **Representation**: Ordered List of Composite DAGH Blocks
  - Generate SDDG list for base grid
  - For each refine region:
    » Replace parent sub-list by child list
    » Modify sub-list work to reflect parent + child
    » Increment sub-list “Level” count

- **Partitioning**: Partition DAGH block list to balance work per processor
Partitioning Issues

- Locality
- Parallelism
- Load-balance
- Cost
Composite Distribution

- Inter-grid communications are local
- Data and task parallelism exploited
- Efficient load redistribution and clustering
- Overhead of generating & maintaining composite structure
Experimental Evaluation

- **Application:**
  - H3espresso - Einstein's equations for general relativity in 3D (Joan Masso, NCSA, UIUC)

- **DAGH overheads**
  - Performance competitive with and often better than conventional (FORTRAN + MPI) implementations

- **Inter-level communication overheads**
  - All inter-level communication local to processor
DAGH Performance

IBM SP2 - 32x32x32 (Unigrid)

Processors

Time

0 500 1000 1500 2000

DAGH
F90+MPI

☆ H3espresso (Einstein’s GR Eq. in 3D) Joan Masso, NCSA, UIUC
## Partitioning Overheads

<table>
<thead>
<tr>
<th>Processors</th>
<th>Update Time</th>
<th>Regridding Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>28.5 sec</td>
<td>1.84 sec</td>
</tr>
<tr>
<td>8</td>
<td>19.2 sec</td>
<td>1.58 sec</td>
</tr>
</tbody>
</table>

- \( \text{Regriding Time} = \text{refinement} + \text{re-partitioning} + \text{load-balancing} + \text{data-movement} \)
- Typically < 10% of computation time
- Worst case load-imbalance <20%
Load Balance

Composite Load Distribution
Load Balance

Composite Load Distribution

Norm. Load

1 2 3 4 5 6 7 8

-0.5

0.5

1.5

Processor

Ideal

Achieved

Ideal

Achieved
Summary

Dynamic Distributed Arrays - A basis for adaptive applications

Programming Abstractions - AMR, hp-FE, FMP

Competitive Performance

Integration of AMR and AFE has been accomplished.

Integration with visualization has been implemented.
Coming Next Week – Component-Based Application Structures

How to develop applications which are:

• Evolvable
• Portable
• Performant

Component-based Software Engineering

• Methodology
• Case Studies
Coming Next Week – Performance Optimization
Many Processor Architectures

Multicore Chips and Homogeneous Multichip Nodes
• Problem Definition
• Approach
• Case Studies

Heterogeneous Nodes with Accelerators
• Problem
• Approach
• Case Studies