Fast Methods with Sieve

Matthew G Knepley

Mathematics and Computer Science Division
Argonne National Laboratory

August 12, 2008
Workshop on Scientific Computing
Simula Research, Oslo, Norway
Challenges

- Can we establish good interfaces for all levels of the hierarchy?
- Do we need language extensions for more sophisticated problems?
- What information is required from each component?
- Is inter-language programming effective?
- Can we develop a general framework for boundary conditions?
Outline

1. Spatial Decomposition
2. Data Decomposition
3. Serial Implementation
4. Parallel Spatial Decomposition
5. Parallel Performance
FMM in Sieve

- The Quadtrees are a Sieve with optimized operations.
- Multipole are stored in Sections.
- Two Overlaps are defined:
  - Neighbors
  - Interaction List
- Completion moves data for:
  - Neighbors
  - Interaction List
Spatial Decomposition

FMM in Sieve

- The Quadtree is a **Sieve** with optimized operations
- Multipoles are stored in **Sections**
- Two **Overlaps** are defined
  - Neighbors
  - Interaction List
- Completion moves data for
  - Neighbors
  - Interaction List
FMM in Sieve

- The Quadtree is a Sieve with optimized operations
- Multipoles are stored in Sections
- Two Overlaps are defined
  - Neighbors
  - Interaction List
- Completion moves data for
  - Neighbors
  - Interaction List
The Quadtree is a Sieve with optimized operations.

Multipoles are stored in Sections.

Two Overlaps are defined:
- Neighbors
- Interaction List

Completion moves data for:
- Neighbors
- Interaction List
The Quadtree is a Sieve with optimized operations.

Multipoles are stored in Sections.

Two Overlaps are defined:
- Neighbors
- Interaction List

Completion moves data for:
- Neighbors
- Interaction List
The Quadtree is a **Sieve** with optimized operations.

- Multipoles are stored in **Sections**
- **Two Overlaps are defined**
  - Neighbors
  - Interaction List

Completion moves data for
- Neighbors
- Interaction List
The Quadtree is a *Sieve* with optimized operations.

- Multipoles are stored in *Sections*.
- Two *Overlaps* are defined:
  - Neighbors
  - Interaction List

Completion moves data for:
- Neighbors
- Interaction List
We use binary scheme to label cells (or vertices)

Relevant relations can be determined implicitly
- `cone()`
- neighbors
- parent
- interaction list

When vertices are not used, we can directly connect cells
- `cone()` becomes neighbor method
Tree Interface

- `locateBlob(blob)`
  - Locate point in the tree
- `fillNeighbors()`
  - Compute the neighbor section
- `findInteractionList()`
  - Compute the interaction list cell section, allocate value section
- `fillInteractionList(level)`
  - Compute the interaction list value section
- `fill(blobs)`
  - Compute the blob section
- `dump()`
  - Produces a verifiable representation of the tree
Outline

1. Spatial Decomposition
2. Data Decomposition
3. Serial Implementation
4. Parallel Spatial Decomposition
5. Parallel Performance
FMM requires data over the Quadtree distributed by:

- **box**
  - Box centers, Neighbors

- **box + neighbors**
  - Blobs

- **box + interaction list**
  - Interaction list cells and values
  - Multipole and local coefficients
FMM Sections

FMM requires data over the Quadtree distributed by:

- **box**
  - Box centers, Neighbors

- **box + neighbors**
  - Blobs

- **box + interaction list**
  - Interaction list cells and values
  - Multipole and local coefficients
FMM Sections

FMM requires data over the Quadtree distributed by:

- box
  - Box centers, Neighbors

- box + neighbors
  - Blobs

- box + interaction list
  - Interaction list cells and values
  - Multipole and local coefficients
FMM Sections

FMM requires data over the Quadtree distributed by:

- box
  - Box centers, Neighbors

- box + neighbors
  - Blobs

- box + interaction list
  - Interaction list cells and values
  - Multipole and local coefficients

Notice this is multiscale since data is divided at each level
Outline

1. Spatial Decomposition
2. Data Decomposition
3. Serial Implementation
4. Parallel Spatial Decomposition
5. Parallel Performance
Evaluator Interface

- initializeExpansions(tree, blobInfo)
  - Generate multipole expansions on the lowest level
  - Requires loop over cells
  - $O(p)$
- upwardSweep(tree)
  - Translate multipole expansions to intermediate levels
  - Requires loop over cells and children (support)
  - $O(p^2)$
- downwardSweep(tree)
  - Convert multipole to local expansions and translate local expansions on intermediate levels
  - Requires loop over cells and parent (cone)
  - $O(p^2)$
**Evaluator Interface**

- **evaluateBlobs(tree, blobInfo)**
  - Evaluate direct and local field interactions on lowest level
  - Requires loop over cells and neighbors (in section)
  - $O(p^2)$

- **evaluate(tree, blobs, blobInfo)**
  - Calculate the complete interaction (multipole + direct)
## Kernel Interface

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2M(t)</td>
<td>Multipole expansion coefficients</td>
</tr>
<tr>
<td>L2P(t)</td>
<td>Local expansion coefficients</td>
</tr>
<tr>
<td>M2M(t)</td>
<td>Multipole-to-multipole translation</td>
</tr>
<tr>
<td>M2L(t)</td>
<td>Multipole-to-local translation</td>
</tr>
<tr>
<td>L2L(t)</td>
<td>Local-to-local translation</td>
</tr>
<tr>
<td>evaluate(blobs)</td>
<td>Direct interaction</td>
</tr>
</tbody>
</table>

- **Evaluator is templated over Kernel**
- **There are alternative kernel-independent methods**
  - kifmm3d
Outline

1. Spatial Decomposition
2. Data Decomposition
3. Serial Implementation
4. Parallel Spatial Decomposition
5. Parallel Performance
Parallel Tree Implementation

- Divide tree into a root and local trees
- Distribute local trees among processes
- Provide communication pattern for local sections (overlap)
  - Both neighbor and interaction list overlaps
  - Sieve generates MPI from high level description
Multiple local trees per process allows good load balance

Partition weighted graph
- Minimize load imbalance and communication
- Computation estimate:
  - Leaf: $N_i p \text{ (P2M)} + n_i p^2 \text{ (M2L)} + N_i p \text{ (L2P)} + 3^d N_i^2 \text{ (P2P)}$
  - Interior: $n_c p^2 \text{ (M2M)} + n_i p^2 \text{ (M2L)} + n_c p^2 \text{ (L2L)}$

Communication estimate:
- Diagonal: $n_c(L - k - 1)$
- Lateral: $2^d \frac{2^m(L-k-1)-1}{2^m-1}$ for incidence dimension $m$

Leverage existing work on graph partitioning
- ParMetis

- Good partitions exist for non-uniform distributions
  - 2D $\mathcal{O}(\sqrt{n} (\log n)^{3/2})$ edgecut
  - 3D $\mathcal{O}(n^{2/3} (\log n)^{4/3})$ edgecut

- As scalable as regular grids

- As efficient as uniform distributions

- ParMetis will find a nearly optimal partition

- Good partitions exist for non-uniform distributions
  - \( 2D \ C_i = 1.24^i C_0 \) for random matching
  - \( 3D \ C_i = 1.21^i C_0 \)?? for random matching

- 3D proof needs assurance that average degree does not increase

- Efficient in practice
Parallel Tree Implementation

Advantages

- Simplicity
  - Complete serial code reuse
  - Provably good performance and scalability
Parallel Spatial Decomposition

Parallel Tree Implementation

Advantages

- Simplicity
- Complete serial code reuse
- Provably good performance and scalability
Parallel Spatial Decomposition

Parallel Tree Implementation

Advantages

- Simplicity
- Complete serial code reuse
- Provably good performance and scalability
Parallel Spatial Decomposition

Parallel Tree Interface

- `fillNeighbors()`
  - Compute neighbor overlap, and send neighbors
- `findInteractionList()`
  - Compute the interaction list overlap
- `fillInteractionList(level)`
  - Complete and copy into local interaction sections
- `fill(blobs)`
  - Now must scatter blobs to local trees
  - Uses `scatterBlobs()` and `gatherBlobs()`
Parallel Spatial Decomposition

Parallel Data Movement

1. Complete neighbor section
2. Upward sweep
   1. Upward sweep on local trees
   2. Gather to root tree
   3. Upward sweep on root tree
3. Complete interaction list section
4. Downward sweep
   1. Downward sweep on root tree
   2. Scatter to local trees
   3. Downward sweep on local trees
initializeExpansions(local trees, blobInfo)
  Evaluate each local tree
upwardSweep(local trees, partition, root tree)
  Evaluate each local tree and then gather to root tree
downwardSweep(local trees, partition, root tree)
  Scatter from root tree and then evaluate each local tree
evaluateBlobs(local trees, blobInfo)
  Evaluate on all local trees
evaluate(tree, blobs, blobInfo)
  Identical
Outline

1. Spatial Decomposition
2. Data Decomposition
3. Serial Implementation
4. Parallel Spatial Decomposition
5. Parallel Performance
For large problems, a single root can be a bottleneck

We can recursively assign roots to subtrees
  - Bandwidth to root is controlled
  - What about utilization?

Root computation is similar to MG coarse solve