Adaptive Mesh Refinement in Chapel
Part II: A really hard problem, greatly simplified

Jonathan Claridge
University of Washington
March 2, 2011
Overview of two talks

- Previous talk:
  - Several AMR challenges that Chapel makes easy

- This talk:
  - A difficult part of AMR that Chapel sets us up to solve
Data refinement

- Data from a coarse grid provides boundary values to fine grids that it overlaps.
Data refinement

- Data from a coarse grid provides boundary values to fine grids that it overlaps
- Only need to fill fine ghost cells that are not overlapped by a fine sibling grid
Data refinement

- Data from a coarse grid provides boundary values to fine grids that it overlaps
- Only need to fill fine ghost cells that are not overlapped by a fine sibling grid
Data refinement

- Data from a coarse grid provides boundary values to fine grids that it overlaps
- Only need to fill fine ghost cells that are not overlapped by a fine sibling grid
Data refinement

- Data from a coarse grid provides boundary values to fine grids that it overlaps.
- Only need to fill fine ghost cells that are not overlapped by a fine sibling grid.
- Resulting region is a **union of rectangles**, most naturally defined by **set subtraction**.
Data refinement

- Data from a coarse grid provides boundary values to fine grids that it overlaps
- Only need to fill fine ghost cells that are not overlapped by a fine sibling grid
- Resulting region is a union of rectangles, most naturally defined by set subtraction
- Chapel: Define an object to store unions of domains, which supports domain subtraction in a set-minus fashion
Regridding

- New grids determined by a **partitioning algorithm** (Berger & Rigoutsos, 1991)
Regridding

- New grids determined by a *partitioning algorithm* (Berger & Rigoutsos, 1991)

  - set of flagged points
  - set of rectangles covering them
Regridding

- New grids determined by a *partitioning algorithm* (Berger & Rigoutsos, 1991)

Chapel:
- set of flagged points
- Boolean array on a large domain

→ set of rectangles covering them
→ set of subdomains containing all *true* entries
Regridding

- As with refinement, **unions** of rectangles (domains) are essential
Regridding

- As with refinement, **unions** of rectangles (domains) are essential.
- Subtractions in Berger-Rigoutsos always remove a subset that spans a domain in \( \text{rank-1} \) dimensions; general domain subtraction is convenient, but not necessary.
Regridding

- As with refinement, **unions** of rectangles (domains) are essential
- Subtractions in Berger-Rigoutsos always remove a subset that spans a domain in rank-1 dimensions; general domain subtraction is convenient, but not necessary
- However, domain subtraction **is** important after partitioning, when refining data onto a newly created level
Unions of domains: **MultiDomain class**

Precedents in AMR libraries:
Unions of domains: \textit{MultiDomain} class

Precedents in AMR libraries:

- Chombo \textit{BoxTools} library
  - Class \texttt{Box} represents rectangular sets of integer tuples (\texttt{IntVects})
  - Class \texttt{IntVectSet} represents irregular sets of integer tuples, supporting full set calculus
Unions of domains: **MultiDomain class**

Precedents in AMR libraries:

- **Chombo BoxTools library**
  - Class *Box* represents rectangular sets of integer tuples (*IntVects*)
  - Class *IntVectSet* represents irregular sets of integer tuples, supporting full set calculus

- **SAMRAI Hierarchy library**
  - Class *Box* (see above)
  - Classes *BoxArray*, *BoxList*, *BoxTree* represent unions of boxes, supporting various set operations
Unions of domains: MultiDomain class

- MultiDomain fields:
Unions of domains: **MultiDomain class**

- **MultiDomain fields:**

  ```
  param rank: int;
  param stridable: bool = false;
  ```

Parameters to specify child domains; compile time constants
Unions of domains: MultiDomain class

- MultiDomain fields:

  - `param rank: int;`
  - `param stridable: bool = false;`
  - `var stride: rank*int;`

Parameters to specify child domains; compile time constants

Child domains will have equal stride
Unions of domains: MultiDomain class

- MultiDomain fields:
  - `param` `rank`: `int`;
  - `param` `stridable`: `bool = false`;
  - `var` `stride`: `rank*int`;
  - `var` `subindices`: `domain(1)`;

Parameters to specify child domains; compile time constants
Child domains will have equal stride
Indices for array of child domains
Unions of domains: **MultiDomain class**

- **MultiDomain fields:**
  
  ```
  param rank: int;
  param stridable: bool = false;
  var stride: rank*int;
  var subindices: domain(1);
  var domains: [subindices] domain(rank, stridable=stridable);
  ```

  - Parameters to specify child domains; compile time constants
  - Child domains will have equal stride
  - Indices for array of child domains
  - Array of child domains
Unions of domains: MultiDomain class

- MultiDomain fields:

  ```
  param rank: int;
  param stridable: bool = false;
  var stride: rank*int;
  var subindices: domain(1);
  var domains: [subindices] domain(rank, stridable=stridable);
  ```

  - Parameters to specify child domains; compile time constants
  - Child domains will have equal stride
  - Indices for array of child domains
  - Array of child domains

- In principle, domains could be an associative domain of rectangular domains
Unions of domains: MultiDomain class

- MultiDomain fields:

  ```
  param rank: int;
  param stridable: bool = false;
  var stride: rank*int;
  var subindices: domain(1);
  var domains: [subindices] domain(rank, stridable=stridable);
  ```

  - Parameters to specify child domains; compile time constants
  - Child domains will have equal stride
  - Indices for array of child domains
  - Array of child domains

- In principle, domains could be an associative domain of rectangular domains

- Tree-based storage of domains, with bounding boxes at nodes, will allow better performance for set operations; direction for future improvement
Unions of domains: MultiDomain class

- MultiDomain operations:
  MultiDomain = domain;
  MultiDomain.add(domain);
  MultiDomain = domain - domain;
  MultiDomain.subtract(domain);
  MultiDomain.intersect(domain);
  etc...

- Most operations allow a MultiDomain as an argument as well
Domain subtraction

- Recursive procedure, reducing to $\text{rank}-1$ subtraction at each step
Domain subtraction

- Recursive procedure, reducing to rank-1 subtraction at each step

- Calculate Yellow - Red, working along the horizontal:
Domain subtraction

- Recursive procedure, reducing to rank-1 subtraction at each step
- Calculate Yellow − Red, working along the horizontal:
  - Yellow splits into 3 pieces: Y_lower, Y_inner, and Y_upper, any of which may be empty
Domain subtraction

- Recursive procedure, reducing to rank-1 subtraction at each step

- Calculate Yellow - Red, working along the horizontal:
  - Yellow splits into 3 pieces: Y_lower, Y_inner, and Y_upper, any of which may be empty
  - Y_lower and Y_upper consist of 0 or 1 domains, disjoint from Red
Domain subtraction

- Recursive procedure, reducing to rank-1 subtraction at each step

- Calculate Yellow - Red, working along the horizontal:
  - Yellow splits into 3 pieces: Y_lower, Y_inner, and Y_upper, any of which may be empty
  - Y_lower and Y_upper consist of 0 or 1 domains, disjoint from Red
  - Now calculate Y_inner - Red, but project onto remaining dimensions since Y_inner.dim(1) == Red.dim(1)

Y_inner - Red is much more complicated than this in higher dimensions
Class GridCFGhostRegion

- Represents ghost cells of a fine grid that will receive data from "coarse neighbor" grids
Class GridCFGhostRegion

- Represents ghost cells of a fine grid that will receive data from “coarse neighbor” grids
Class GridCFGhostRegion

- Represents ghost cells of a fine grid that will receive data from “coarse neighbor” grids
- Fields are:

  ```
  const grid: Grid;  // The fine grid in question
  const coarse_neighbors: domain(Grid);
  const multidomains: [coarse_neighbors] MultiDomain(dimension, stridable=true);
  ```
Class GridCFGhostRegion

- Represents ghost cells of a fine grid that will receive data from “coarse neighbor” grids.

Fields are:

```plaintext
const grid: Grid;  // The fine grid in question
const coarse_neighbors: domain(Grid);
const multidomains: [coarse_neighbors] MultiDomain(dimension, stridable=true);
```

- Constructor also needs to know:
  - `parent_level` of grid
  - `coarse_level`
  - `ref_ratio`, the refinement ratio between `coarse_level` and `parent_level`
Class GridCFGhostRegion

for coarse_grid in coarse_level.grids {
    var fine_intersection =
        grid.extended_cells(refine(coarse_grid.cells, ref_ratio));

    if fine_intersection.numIndices > 0 {
        var boundary_multidomain = fine_intersection - grid.cells;

        for (neighbor, region) in parent_level.sibling_ghost_regions(grid) {
            if fine_intersection(region).numIndices > 0 then
                boundary_multidomain.subtract(region);
        }

        if boundary_multidomain.length > 0 {
            coarse_neighbors.add(coarse_grid);
            multidomains(coarse_grid) = boundary_multidomain;
        } else delete boundary_multidomain;
    }
}
Class GridCFGhostRegion

for coarse_grid in coarse_level.grids {
    var fine_intersection =
        grid.extended_cells( refine(coarse_grid.cells, ref_ratio) );

    if fine_intersection.numIndices > 0 {
        var boundary_multidomain =
            fine_intersection - grid.cells;

        for (neighbor, region) in parent_level.sibling_ghost_regions(grid) {
            if fine_intersection(region).numIndices > 0 then
                boundary_multidomain.subtract(region);
        }

        if boundary_multidomain.length > 0 {
            coarse_neighbors.add(coarse_grid);
            multidomains(coarse_grid) = boundary_multidomain;
        } else delete boundary_multidomain;
    }
}
Class GridCFGhostRegion

```python
for coarse_grid in coarse_level.grids {
    var fine_intersection =
        grid.extended_cells( refine(coarse_grid.cells, ref_ratio) );

    if fine_intersection.numIndices > 0 {
        var boundary_multidomain = fine_intersection;

        for (neighbor, region) in parent_level.sibling_ghost_regions(grid) {
            if fine_intersection(region).numIndices > 0 then
                boundary_multidomain.subtract(region);
        }

        if boundary_multidomain.length > 0 {
            coarse_neighbors.add(coarse_grid);
            multidomains(coarse_grid) = boundary_multidomain;
        } else delete boundary_multidomain;
    }
}
```

If `fine_intersection` is empty, there's no reason to continue
Class GridCFGhostRegion

for coarse_grid in coarse_level.grids {
    var fine_intersection =
        grid.extended_cells( refine(coarse_grid.cells, ref_ratio) );

    if fine_intersection.numIndices > 0 {
        var boundary_multidomain = fine_intersection - grid.cells;

        for (neighbor, region) in parent_level.sibling_ghost_regions(grid) {
            if fine_intersection(region).numIndices > 0 then
                boundary_multidomain.subtract(region);
        }

        if boundary_multidomain.length > 0 {
            coarse_neighbors.add(coarse_grid);
            multidomains(coarse_grid) = boundary_multidomain;
        }
        else delete boundary_multidomain;
    }
}
**Class GridCFGhostRegion**

```java
for coarse_grid in coarse_level.grids {
    var fine_intersection = 
        grid.extended_cells( refine(coarse_grid.cells, ref_ratio) );

    if fine_intersection.numIndices > 0 {
        var boundary_multidomain = fine_intersection - grid.cells;

        for (neighbor, region) in parent_level.sibling_ghost_regions(grid) {
            if fine_intersection(region).numIndices > 0 then
                boundary_multidomain.subtract(region);
        }

        if boundary_multidomain.length > 0 {
            coarse_neighbors.add(coarse_grid);
            multidomains(coarse_grid) = boundary_multidomain;
        } else delete boundary_multidomain;
    }
}
```

Iterate over the grid’s SiblingGhostRegion; a these() method has been defined to make the object iterable.
for coarse_grid in coarse_level.grids {
    var fine_intersection =
        grid.extended_cells(refine(coarse_grid.cells, ref_ratio));

    if fine_intersection.numIndices > 0 {
        var boundary_multidomain = fine_intersection - grid.cells;

        for (neighbor, region) in parent_level.sibling_ghost_regions(grid) {
            if fine_intersection(region).numIndices > 0 then
                boundary_multidomain.subtract(region);
        }

        if boundary_multidomain.length > 0 {
            coarse_neighbors.add(coarse_grid);
            multidomains(coarse_grid) = boundary_multidomain;
        } else delete boundary_multidomain;
    }
}
Class GridCFGhostRegion

```python
for coarse_grid in coarse_level.grids {
    var fine_intersection =
        grid.extended_cells( refine(coarse_grid.cells, ref_ratio) );

    if fine_intersection.numIndices > 0 {
        var boundary_multidomain = fine_intersection - grid.cells;

        for (neighbor, region) in parent_level.sibling_ghost_regions(grid)
            { if fine_intersection(region).numIndices > 0 then
                boundary_multidomain.subtract(region);
        }

        if boundary_multidomain.length > 0 {
            coarse_neighbors.add(coarse_grid);
            multidomains(coarse_grid) = boundary_multidomain;
        }
    else delete boundary_multidomain;
    }
}
```
Class GridCFGhostRegion

for coarse_grid in coarse_level.grids {
    var fine_intersection =
        grid.extended_cells( refine(coarse_grid.cells, ref_ratio) );

    if fine_intersection.numIndices > 0 {
        var boundary_multidomain = fine_intersection - grid.cells;

        for (neighbor, region) in parent_level.sibling_ghost_regions(grid)
        {
            if fine_intersection(region).numIndices > 0 then
                boundary_multidomain.subtract(region);
        }

        if boundary_multidomain.length > 0 {
            coarse_neighbors.add(coarse_grid);
            multidomains(coarse_grid) = boundary_multidomain;
        }
        else delete boundary_multidomain;
    }
}
Class GridCFGhostRegion

for coarse_grid in coarse_level.grids {
    var fine_intersection =
        grid.extended_cells( refine(coarse_grid.cells, ref_ratio) );

    if fine_intersection.numIndices > 0 {
        var boundary_multidomain = fine_intersection - grid.cells;

        for (neighbor, region) in parent_level.sibling_ghost_regions(grid)
        {
            if fine_intersection(region).numIndices > 0 then
                boundary_multidomain.subtract(region);
        }

        if boundary_multidomain.length > 0 {
            coarse_neighbors.add(coarse_grid);
            multidomains(coarse_grid) = boundary_multidomain;
        }
        else delete boundary_multidomain;
    }
}
Class GridCFGhostRegion

```chapel
for coarse_grid in coarse_level.grids {
    var fine_intersection = grid.extended_cells(refine(coarse_grid.cells, ref_ratio));
    if fine_intersection.numIndices > 0 {
        var boundary_multidomain = fine_intersection - grid.cells;
        for (neighbor, region) in parent_level.sibling_ghost_regions(grid) {
            if fine_intersection(region).numIndices > 0 then
                boundary_multidomain.subtract(region);
        }
        if boundary_multidomain.length > 0 {
            coarse_neighbors.add(coarse_grid);
            multidomains(coarse_grid) = boundary_multidomain;
        } else delete boundary_multidomain;
    }
}
else delete boundary_multidomain;
```

- MultiDomains greatly simplify the hard part
  - Internally, MultiDomains heavily rely on Chapel infrastructure for domains
  - Simple ≠ cheap; misuse of MultiDomains can be expensive
for coarse_grid in coarse_level.grids {
    var fine_intersection = grid.extended_cells(refine(coarse_grid.cells, ref_ratio));
    if fine_intersection.numIndices > 0 {
        var boundary_multidomain = fine_intersection - grid.cells;
        for (neighbor, region) in parent_level.sibling_ghost_regions(grid) {
            if fine_intersection(region).numIndices > 0 {
                boundary_multidomain.subtract(region);
            }
        }
        if boundary_multidomain.length > 0 {
            coarse_neighbors.add(coarse_grid);
            multidomains(coarse_grid) = boundary_multidomain;
        } else {
            delete boundary_multidomain;
        }
    }
} else {
    delete boundary_multidomain;
}
Conclusions

Final recap of code size:

<table>
<thead>
<tr>
<th>Language</th>
<th>Parallelism</th>
<th>SLOC(^1)</th>
<th>Tokens</th>
<th>Relative size (tokens)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ (D≤6)(^3)</td>
<td>Dist. mem.</td>
<td>40200</td>
<td>261427</td>
<td>100%</td>
</tr>
<tr>
<td>Fortran (2D+3D)(^2)</td>
<td>Serial</td>
<td>16562</td>
<td>151992</td>
<td>58%</td>
</tr>
<tr>
<td></td>
<td>2D</td>
<td>8297</td>
<td>71639</td>
<td>27%</td>
</tr>
<tr>
<td></td>
<td>3D</td>
<td>8265</td>
<td>80353</td>
<td>31%</td>
</tr>
<tr>
<td>Chapel (any D)</td>
<td>Shared mem.</td>
<td>1988</td>
<td>13783</td>
<td>5%</td>
</tr>
</tbody>
</table>

\(^1\) source lines of code, \(^2\) AMRClaw, \(^3\) Chombo BoxTools+AMRTools
Conclusions

Final recap of code size:

<table>
<thead>
<tr>
<th>Language</th>
<th>Parallelism</th>
<th>SLOC$^1$</th>
<th>Tokens</th>
<th>Relative size (tokens)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ (D≤6) $^3$</td>
<td>Dist. mem.</td>
<td>40200</td>
<td>261427</td>
<td>100%</td>
</tr>
<tr>
<td>Fortran (2D+3D) $^2$</td>
<td>Serial</td>
<td>16562</td>
<td>151992</td>
<td>58%</td>
</tr>
<tr>
<td></td>
<td>2D</td>
<td>8297</td>
<td>71639</td>
<td>27%</td>
</tr>
<tr>
<td></td>
<td>3D</td>
<td>8265</td>
<td>80353</td>
<td>31%</td>
</tr>
<tr>
<td>Chapel (any D)</td>
<td>Shared mem.</td>
<td>1988</td>
<td>13783</td>
<td>5%</td>
</tr>
</tbody>
</table>

$^1$ source lines of code, $^2$ AMRClaw, $^3$ Chombo BoxTools+AMRTools

- Hopefully you have a better sense of why the Chapel code is short – and how might translate to a full-featured AMR system
Conclusions

Final recap of code size:

<table>
<thead>
<tr>
<th>Language</th>
<th>Parallelism</th>
<th>SLOC$^1$</th>
<th>Tokens</th>
<th>Relative size (tokens)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ (D≤6) $^3$</td>
<td>Dist. mem.</td>
<td>40200</td>
<td>261427</td>
<td>100%</td>
</tr>
<tr>
<td>Fortran (2D+3D) $^2$</td>
<td>Serial</td>
<td>16562</td>
<td>151992</td>
<td>58%</td>
</tr>
<tr>
<td></td>
<td>2D</td>
<td>8297</td>
<td>71639</td>
<td>27%</td>
</tr>
<tr>
<td></td>
<td>3D</td>
<td>8265</td>
<td>80353</td>
<td>31%</td>
</tr>
<tr>
<td>Chapel (any D)</td>
<td>Shared mem.</td>
<td>1988</td>
<td>13783</td>
<td>5%</td>
</tr>
</tbody>
</table>

$^1$ source lines of code, $^2$ AMRClaw, $^3$ Chombo BoxTools+AMRTools

- Hopefully you have a better sense of why the Chapel code is short – and how might translate to a full-featured AMR system

Suggestion for future language evaluation

- Use rectangular set operations (“box calculus”) as a problem representative of, and more tractable than, AMR
Conclusions

What did Chapel do for us?
Conclusions

What did Chapel do for us?

- Integer tuples and rectangular sets thereof are native data types
  - Drastically simplifies construction of MultiDomains
Conclusions

What did Chapel do for us?

- Integer tuples and rectangular sets thereof are native data types
  - Drastically simplifies construction of MultiDomains

- Dimension-independence
  - After defining MultiDomains, spatial dimension only appears in variable declarations
Conclusions

What did Chapel do for us?

- Integer tuples and rectangular sets thereof are native data types
  - Drastically simplifies construction of MultiDomains

- Dimension-independence
  - After defining MultiDomains, spatial dimension only appears in variable declarations

- Clean, clear iteration syntax
  - Ability to define any object as an iterator with the `these()` method
Conclusions

What did Chapel do for us?

- Integer tuples and rectangular sets thereof are native data types
  - Drastically simplifies construction of MultiDomains

- Dimension-independence
  - After defining MultiDomains, spatial dimension only appears in variable declarations

- Clean, clear iteration syntax
  - Ability to define any object as an iterator with `these()` method

Recall Chapel’s main goal:

- **Improve programmer productivity**
Where can I learn more?

Chapel:

http://chapel.cray.com

Today’s presentations, and many more:

http://chapel.cray.com/presentations.html

Chapel source:

https://sourceforge.net/projects/chapel

Application studies:

http://chapel.svn.sourceforge.net/viewvc/chapel/trunk/…
AMR: …test/studies/amr
SSCA2: …test/users/jglewis/ssca2_version2
PTRANS: …test/studies/hpcc/PTRANS/jglewis