Adaptive Mesh Refinement in Chapel

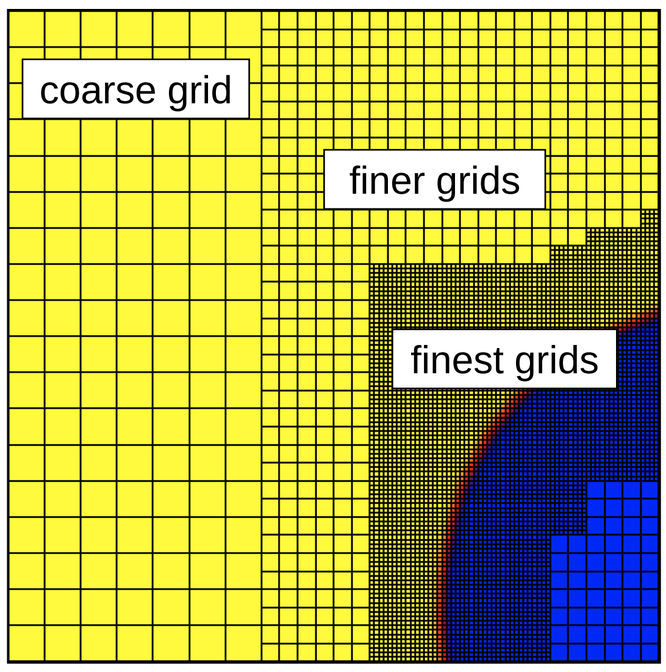
An Acid Test for High Productivity Computing

Jonathan Claridge (University of Washington) and John Lewis (Cray Inc.)

PGAS

Adaptive Mesh Refinement

- An important technique to reduce the computational cost of solving partial differential equations (PDEs)
- Provides varying spatial resolution – fine grids near interesting features, and coarse grids elsewhere
- Used in models of combustion, crystal formation, tsunami evolutation, and many other processes
- Very complex – simplest implementations measure tens of thousands of lines of code



Chapel

- Cray Inc.'s high productivity programming language, in development under the DARPA HPCS program

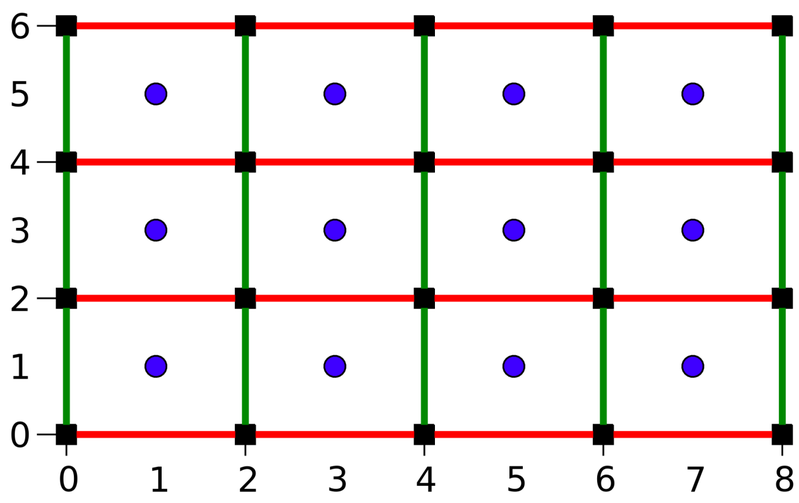
Project goal

- Assess Chapel's productivity by seeing how it can simplify development of an AMR code

Key Chapel Features for Code Simplification

Domains

- Strided indices allow robust description of continuous spaces




```
var cell_centers      = [1..7 by 2, 1..5 by 2];
var vertical_edges    = [0..8 by 2, 1..5 by 2];
var horizontal_edges  = [1..7 by 2, 0..6 by 2];
var vertices          = [0..8 by 2, 0..8 by 2];
```

- Easy iteration over multidimensional spaces without nested loops or manual entry of loop bounds

```
for edge in vertical_edges do ...
```

Applications of domains

- Define an array on any domain

```
var cell_values:       [cell_centers]      real;
var horizontal_fluxes: [vertical_edges]    real;
var vertical_fluxes:   [horizontal_edges]  real;
```

- Compiler will automatically iterate based on contiguous memory access for multidimensional arrays
- SPMD parallelism: Trivial with `forall` loops

```
var dx = 1.0/4;
var dy = 1.0/3;
forall cell in cell_centers do
  cell_values(cell) = sin(cell(1)*dx/2)
                    + sin(cell(2)*dy/2);
```

Assuming coordinate limits are 0 and 1...

Loop executes in parallel

- Nested parallelism: Once domain `grids` has been distributed to processors...

```
forall grid in grids do
  forall cell in grid.cells do
    // Data manipulation goes here
```

Distributed execution

Multithreaded execution on each processor

(not yet implemented in AMR code)

Dimension-independence

- Supports code development for an arbitrary number of space dimensions, greatly simplifying maintenance

Step 1: Base parameters to create an N-dimensional grid. 'config' variables are specified by a flag at compile-time (param) or run-time (const, var).

```
config param N: int = 5;
config const num_cells = 20;
const dimensions = [1..N];
const dx = 1.0/num_cells;
```

Step 2: Create an N-dimensional domain. Key element of dimension-free iteration.

```
var cell_centers: domain(dimension);
var ranges: dimension*range;
for d in dimensions do
  ranges(d) = (1.. by 2) #num_cells;
cell_centers = ranges;
```

Easy, bounds-free creation of 1-dimensional iteration space.

Step 3: Declare and assign an array.

```
var cell_values: [cell_centers] real;
forall cell in cell_centers {
  for d in dimensions do
    cell_values += sin( cell(d)*dx/2 );
}
```

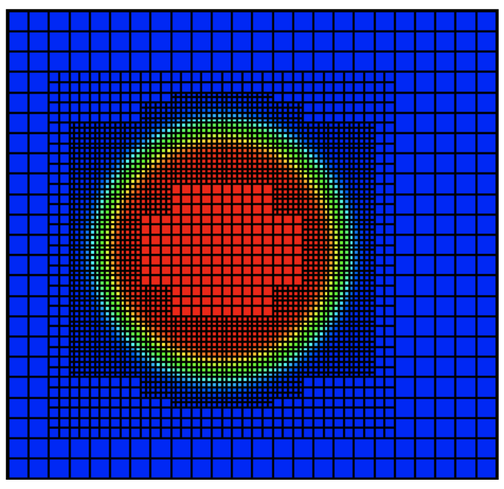
This loop is N-dimensional!

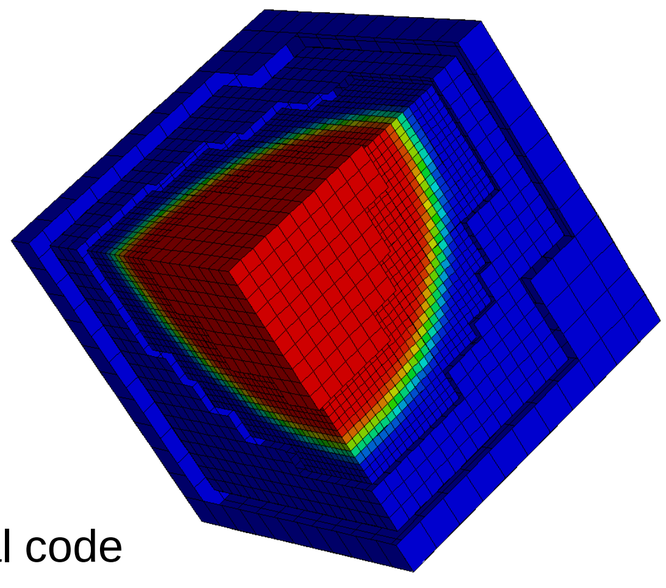
Results

- Wrote working, **dimension-independent** AMR framework in under 4 months with no prior Chapel experience
- Code is drastically shorter than implementations in traditional languages

Language	SLOC[1]	Tokens	Relative size (tokens)
Chapel (any D)	1988	13783	1
Fortran (2D) [2]	8297	71639	5.20
Fortran (3D) [2]	8265	80353	5.83
C/C++ (any D) [3]	40200	261427	20.22

[1] source ines of code, [2] AMRClaw, [3] Chombo BoxTools+AMRTools





2D and 3D outputs produced by identical code

APPLIED MATHEMATICS
UNIVERSITY of WASHINGTON

DARPA HPCS CRAY

This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0001. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.