Chapel HPC Challenge Entry: 2011

Chapel Team: Brad Chamberlain, Sung-Eun Choi,
Tom Hildebrandt, Vass Litvinov, Greg Titus
Benchmarks: John Lewis, Kristi Maschhoff, Jonathan Claridge

SC11: November 15th, 2011

What is Chapel?

- A new parallel programming language
  - Design and development led by Cray Inc.
- **Overall goal:** Improve programmer productivity
- Being developed as open source (BSD) at SourceForge
- **Target Architectures:**
  - multicore desktops and laptops
  - commodity clusters
  - Cray architectures (as well as those from other vendors)
- A work-in-progress

Hardware Platforms

NCCS Jaguar: Cray XT5™

- Dual AMD 6-core Istanbul Opteron processors
- Cray SeaStar™ interconnect
- 16 GB of memory per node
- 9984 nodes

Cray Internal system Kaibab: Cray XE6™

- Dual AMD 12-core Magny-Cours Opteron processors
- Cray Gemini™ interconnect
- 32 GB of memory per node
- 84 nodes

Hardware Platforms

NCCS Jaguar: Cray XT5™

- Dual AMD 6-core Istanbul Opteron processors
- Cray SeaStar™ interconnect
- 16 GB of memory per node
- 9984 nodes

Cray Internal system Kaibab: Cray XE6™

- Dual AMD 12-core Magny-Cours Opteron processors
- Cray Gemini™ interconnect
- 32 GB of memory per node
- 84 nodes

Life's now harder due to NUMA nodes/first-touch policies

Speaking of being a work-in-progress…

- We have several performance-critical irons in the fire, many aimed at the Cray XE/XK and Cascade lines:
  - lightweight user-level tasking layers
  - a native communication layer for Cray's Gemini™ interconnect
  - a bulk copy optimizations for array slices
  - new hooks for overlapping communication and computation
  - a hierarchical locale concept for NUMA nodes and other emerging node architectures

- Look for these in our 1.5 release, April 2011

Chapel Codes for 2011

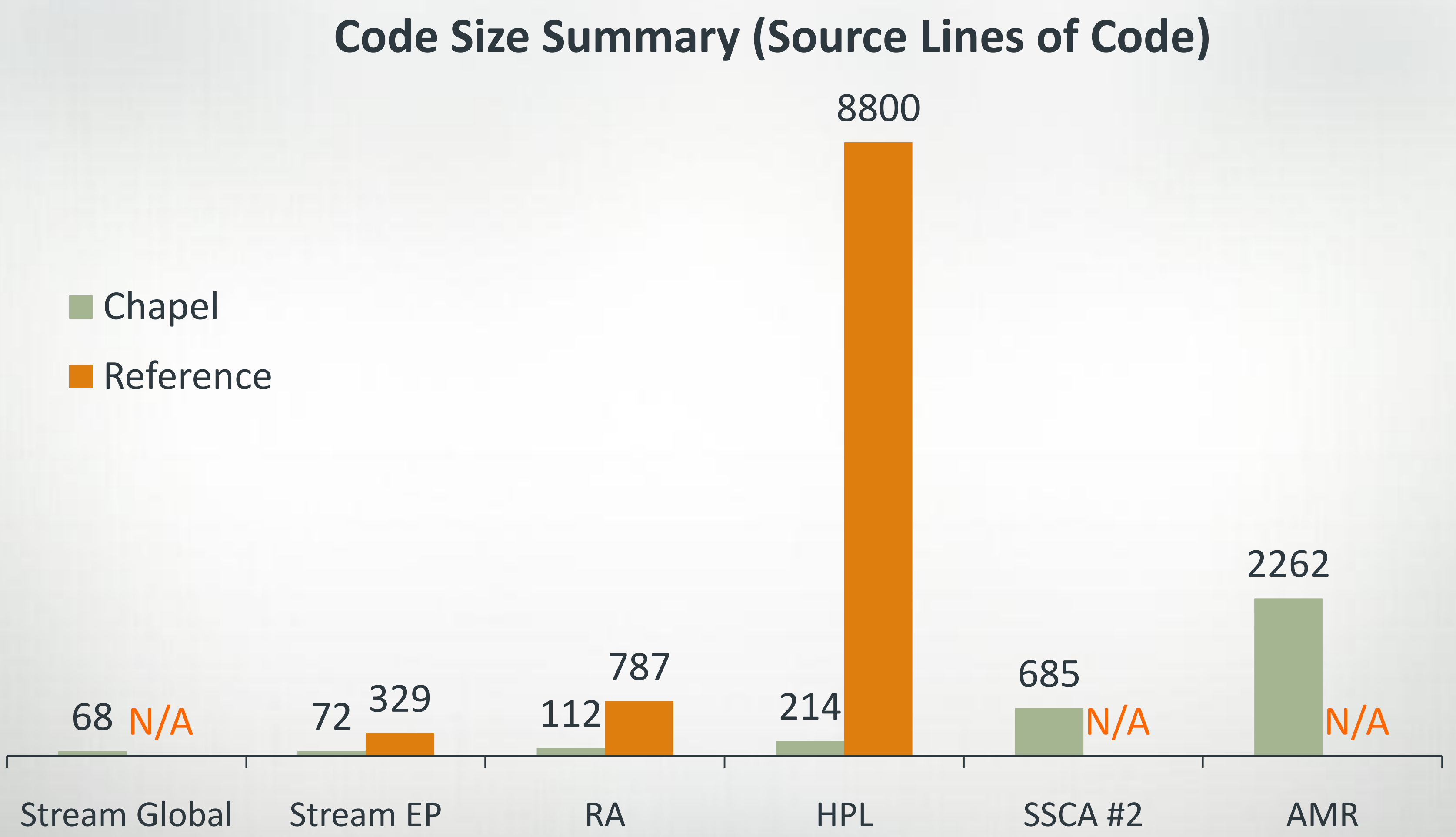
HPCC:

1. Stream Triad
   - EP
   - Global
2. Random Access (RA)
3. HPL
   - focusing on Schur Complement computation

Others:

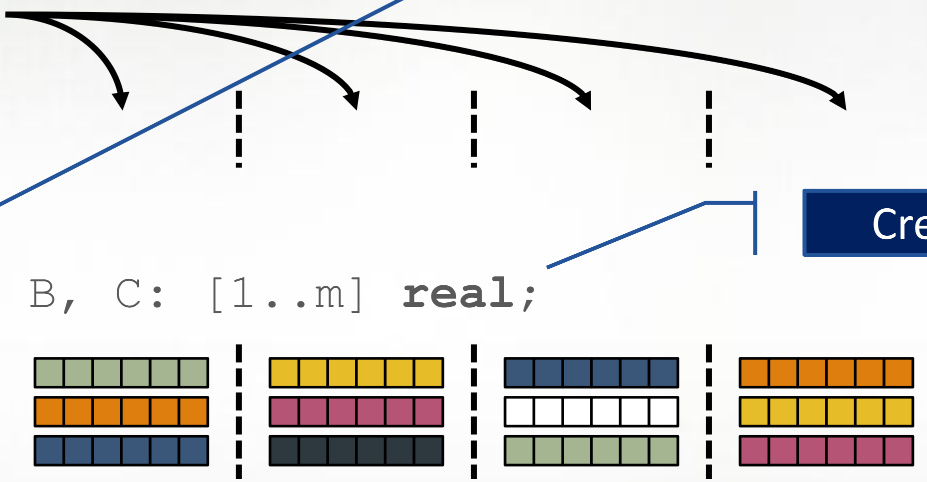
4. SSCA #2 (an unstructured graph benchmark)
   - focusing on Kernel 4: Betweenness centrality
5. Adaptive Mesh Refinement (AMR) Framework

Chapel Source Code Sizes



EP STREAM Triad in Chapel (Excerpts)

```
coforall loc in Locales do
  on loc {
```



```
    local {
      var A, B, C: [1..m] real;

      forall (a,b,c) in (A,B,C) do
        a = b + alpha * c;
    }
  }
```

Create a task per node

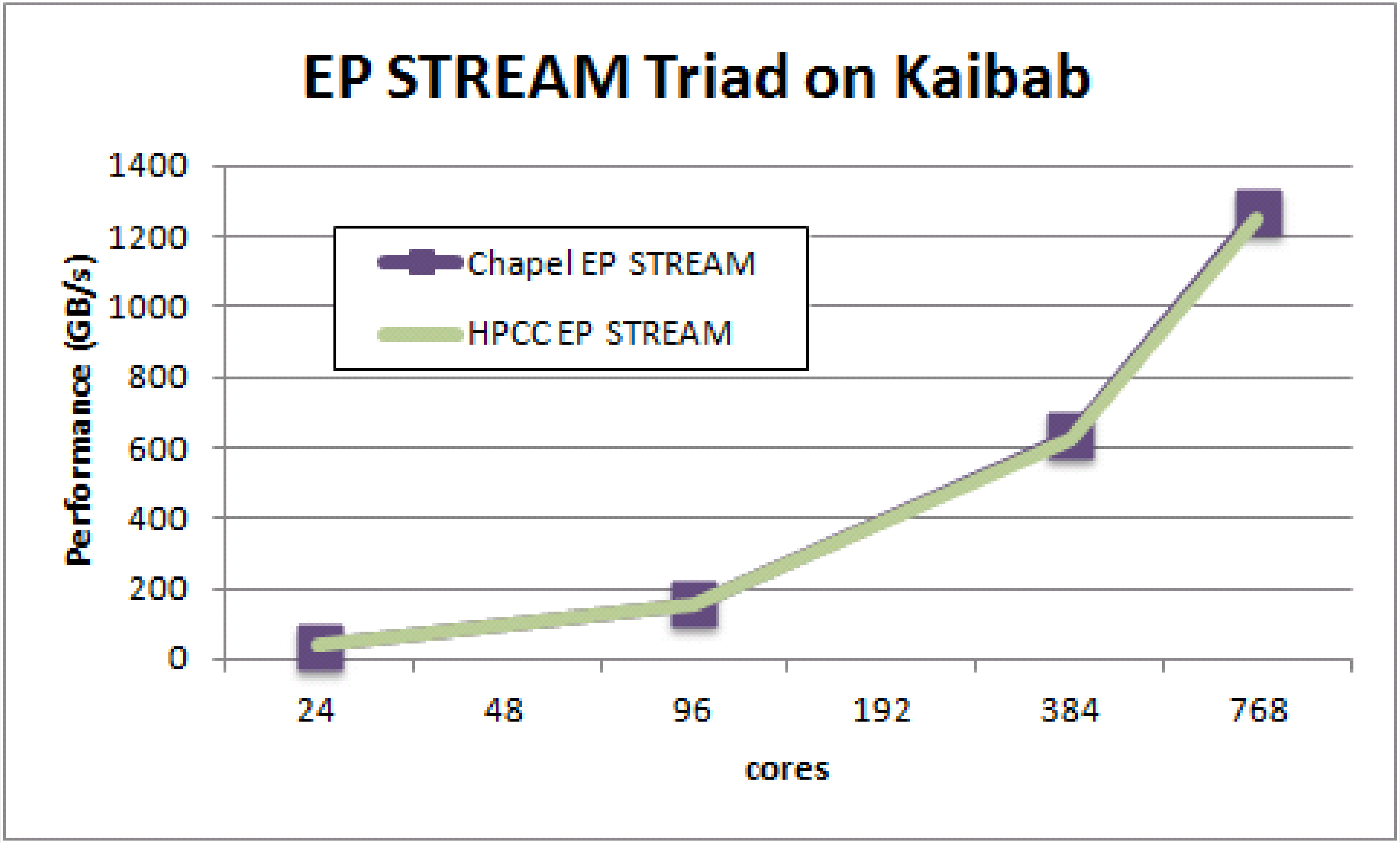
Assert this computation is local

Create 3 arrays per task

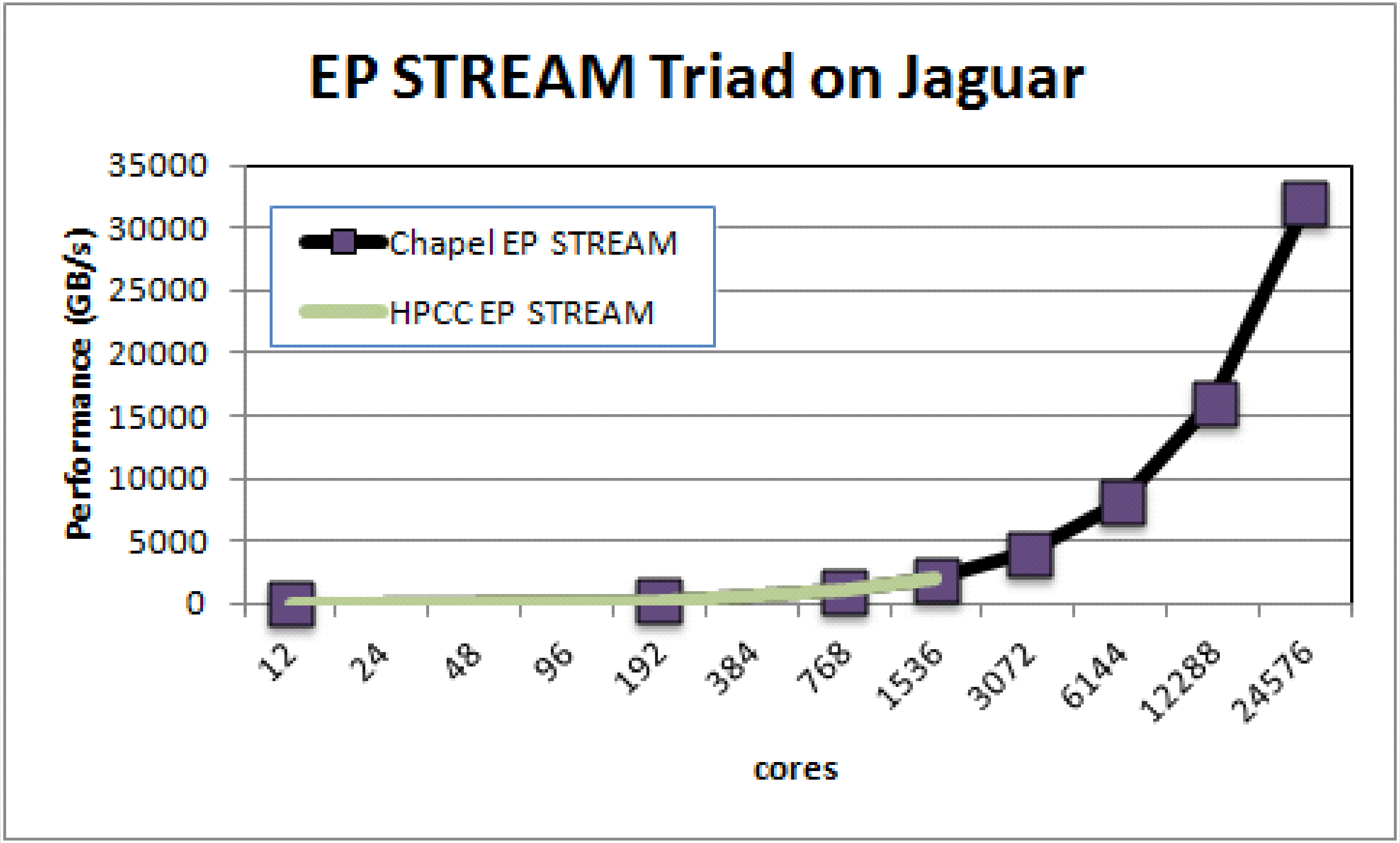
Use a data parallel forall within the node

This loop can also be written:
`A = B + alpha * C;`
(but today, performance is slightly worse)

HPCC Stream Performance on Kaibab (XE6)

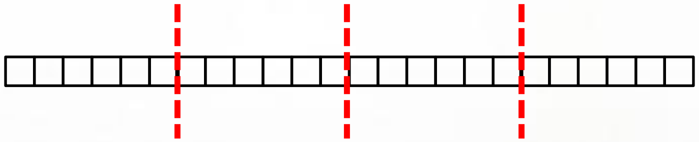


HPCC Stream Performance on Jaguar (XT5)

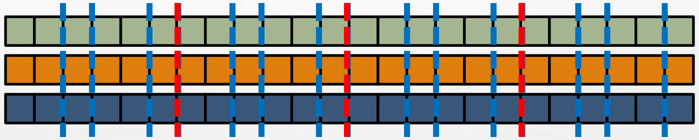


Global STREAM Triad in Chapel (Excerpts)

```
const ProblemSpace: domain(1, int(64))
                    dmapped Block([1..m])
                  = [1..m];
```

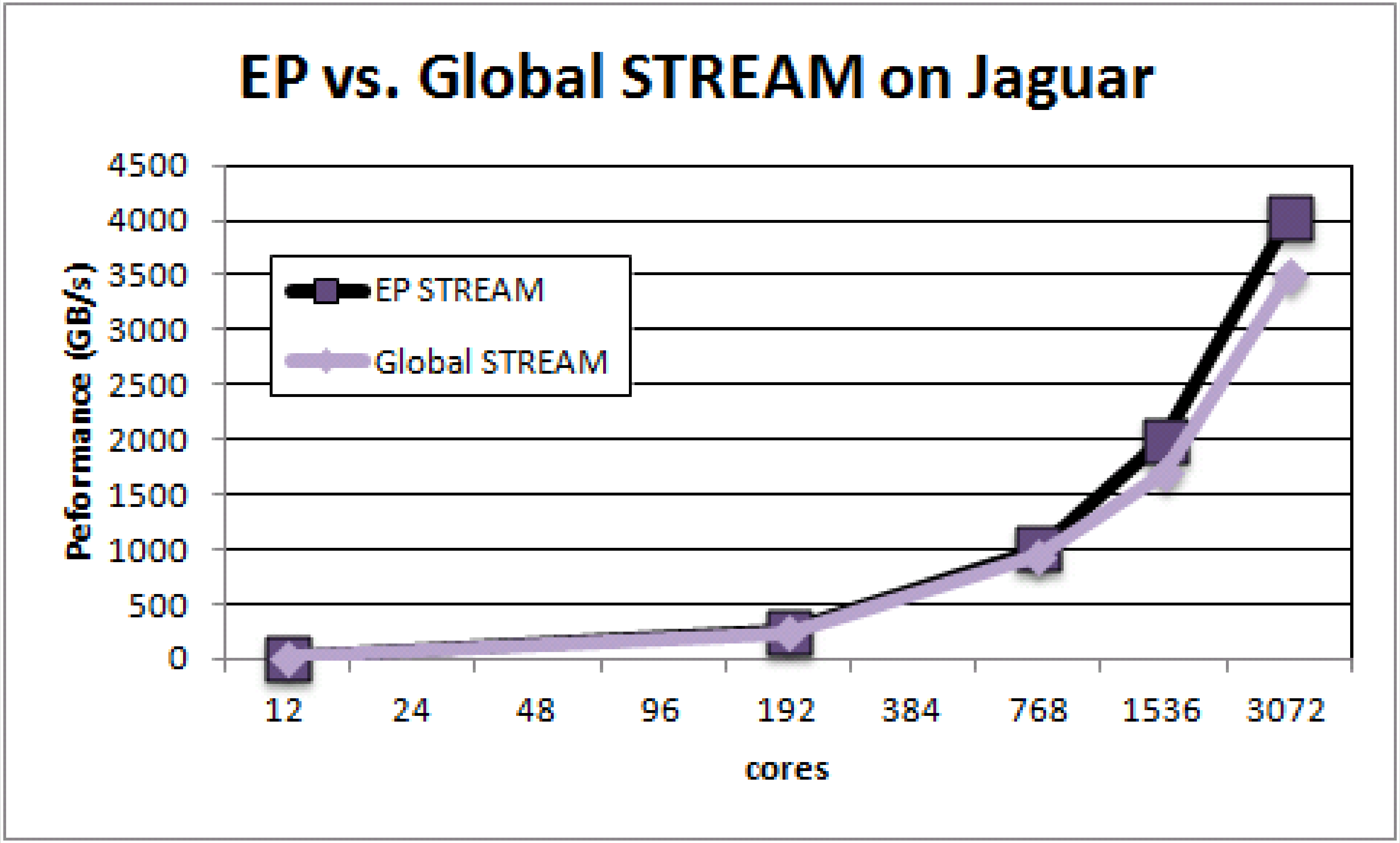


```
var A, B, C: [ProblemSpace] real;
```



```
forall (a,b,c) in (A,B,C) do
  a = b + alpha * c;
```

HPCC Global STREAM on Jaguar (XT5)



Global Random Access in Chapel (Excerpts)

Declare two blocked index sets
- One for the table
- One for the iteration space

```
const TableSpace = [0..m-1]   dmapped Block([0..m-1]),
      Updates    = [0..N_U-1] dmapped Block([0..N_U-1]);

var T: [TableSpace] uint(64);

forall (_,r) in (Updates, RAStream()) do
  on TableSpace.idxToLocale(r & indexMask) {
    const myR = r;
    local T[myR & indexMask] ^= myR;
  }
```

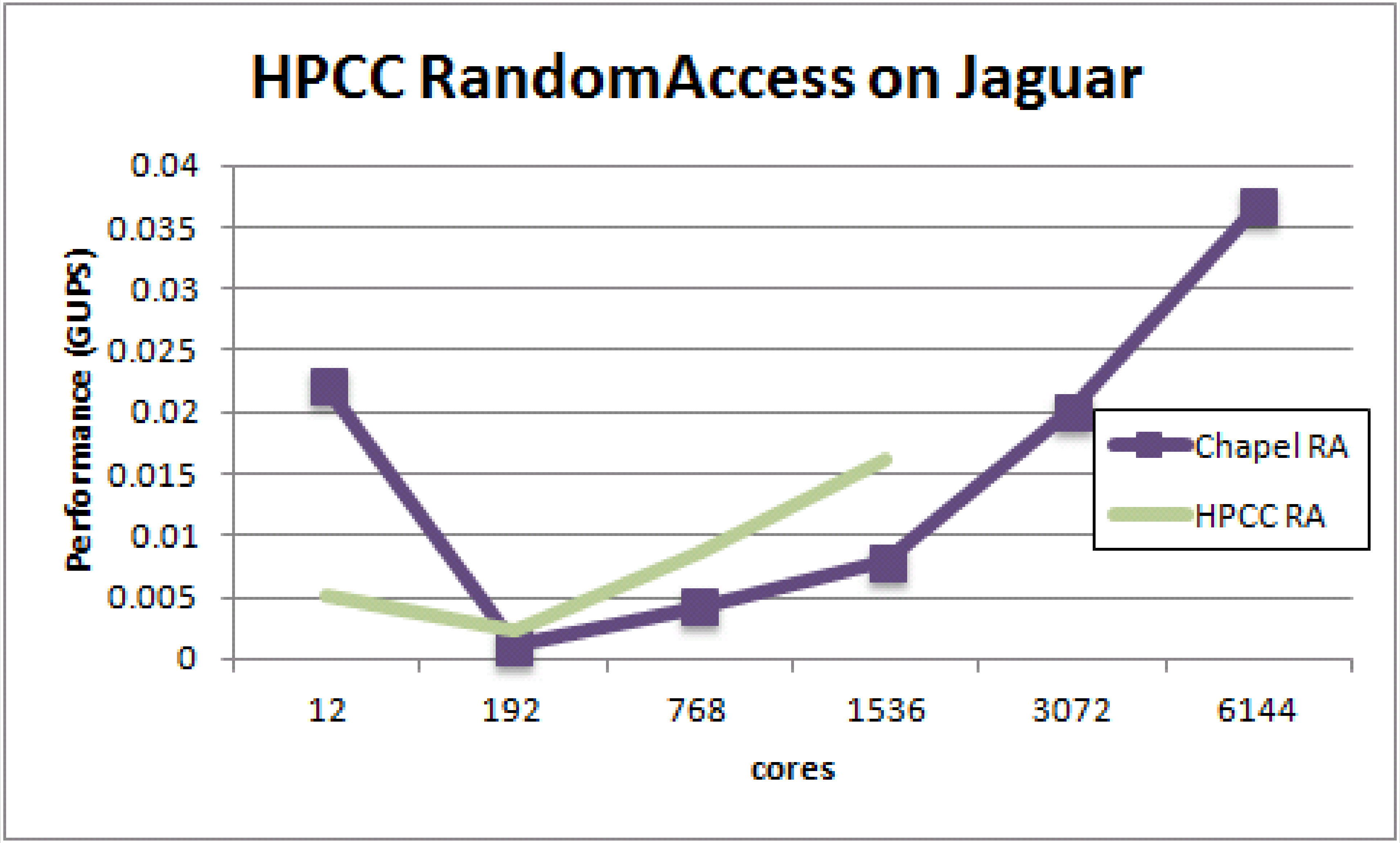
Compute the random updates to the table using a zippered forall loop

This can also be written:

```
    on T[r&indexMask] do
      T[r&indexMask] ^= r;
```

(but again, performance is slightly worse today)

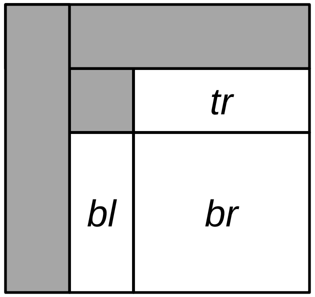
HPCC RandomAccess on Jaguar (XT5)



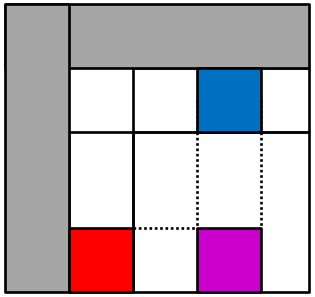
HPL

- Since 2009 entry, we have taken HPL from a shared memory implementation to distributed memory
- Have only started studying performance tuning in past month
  - Focusing initially on Schur Complement phase

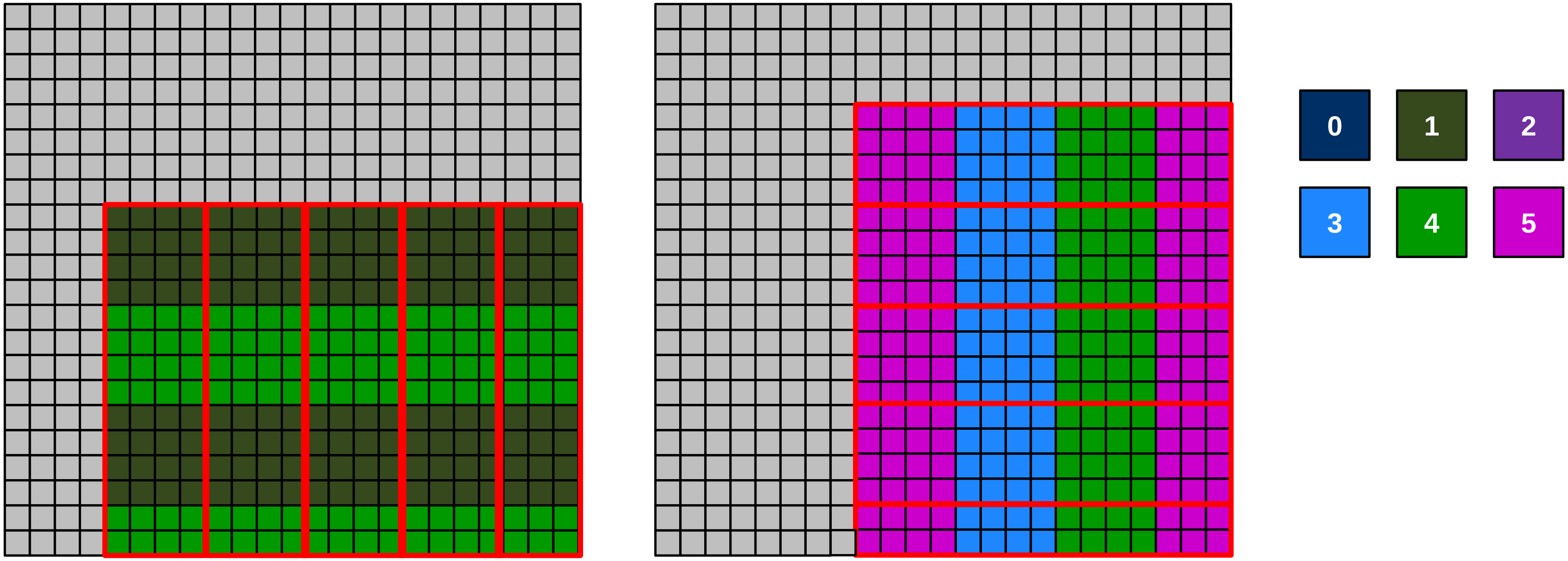
Schur Complement



- accumulate into each block in br the product of its corresponding blocks from bl and tr



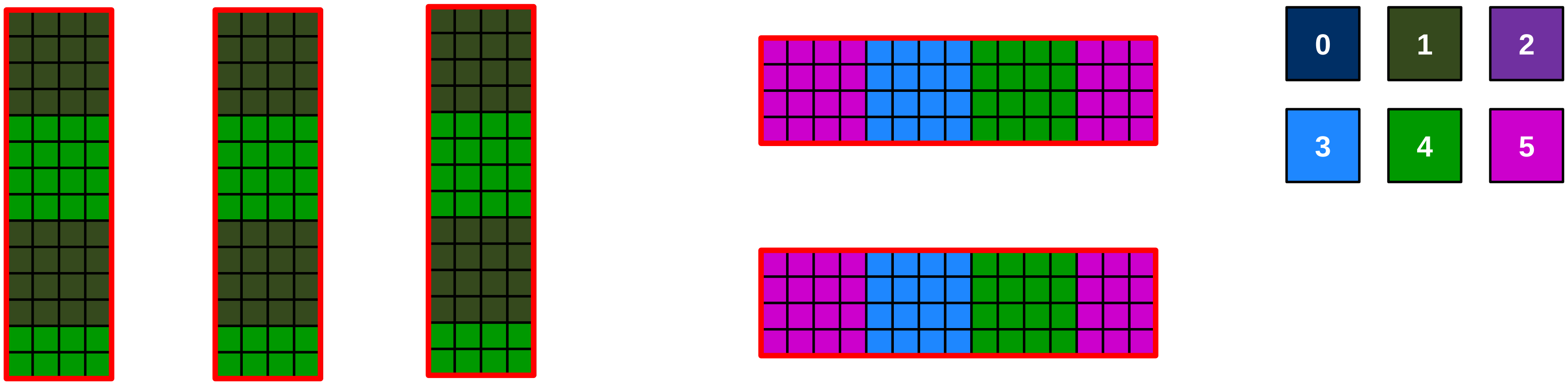
Schur Complement w/ distribution



replicated col, logical view

replicated row, logical view

Schur Complement w/ distribution



replicated column, physical view

replicated row, physical view

- For scalability, we had to add distributions beyond Block:
  - Block-Cyclic
  - Replicated
  - Dimensional: meta-distribution that takes a distribution per dimension

Schur Complement (elegant version)

```
proc schurComplement(AD, BD, Rest) {

  replA[AD] = Ab[AD];
  replB[BD] = Ab[BD];

  forall (row,col) in Rest by (blkSize, blkSize) do
    local dgemm(replA[row..#blkSize, ..],
                replB[.., col..#blkSize],
                Ab[row..#blkSize, col..#blkSize]);
}

proc dgemm(A: [1.., 1..] elemType,
           B: [1.., 1..] elemType,
           C: [1.., 1..] elemType) {
  for i in C.domain.dim(1) do
    for j in C.domain.dim(2) do
      for k in A.domain.dim(2) do
        C[i,j] -= A[i, k] * B[k, j];
}
```

Parallel loop over all result blocks

Dgemm of slices from replicated row/col blocks

Schur Complement (manually tuned version)

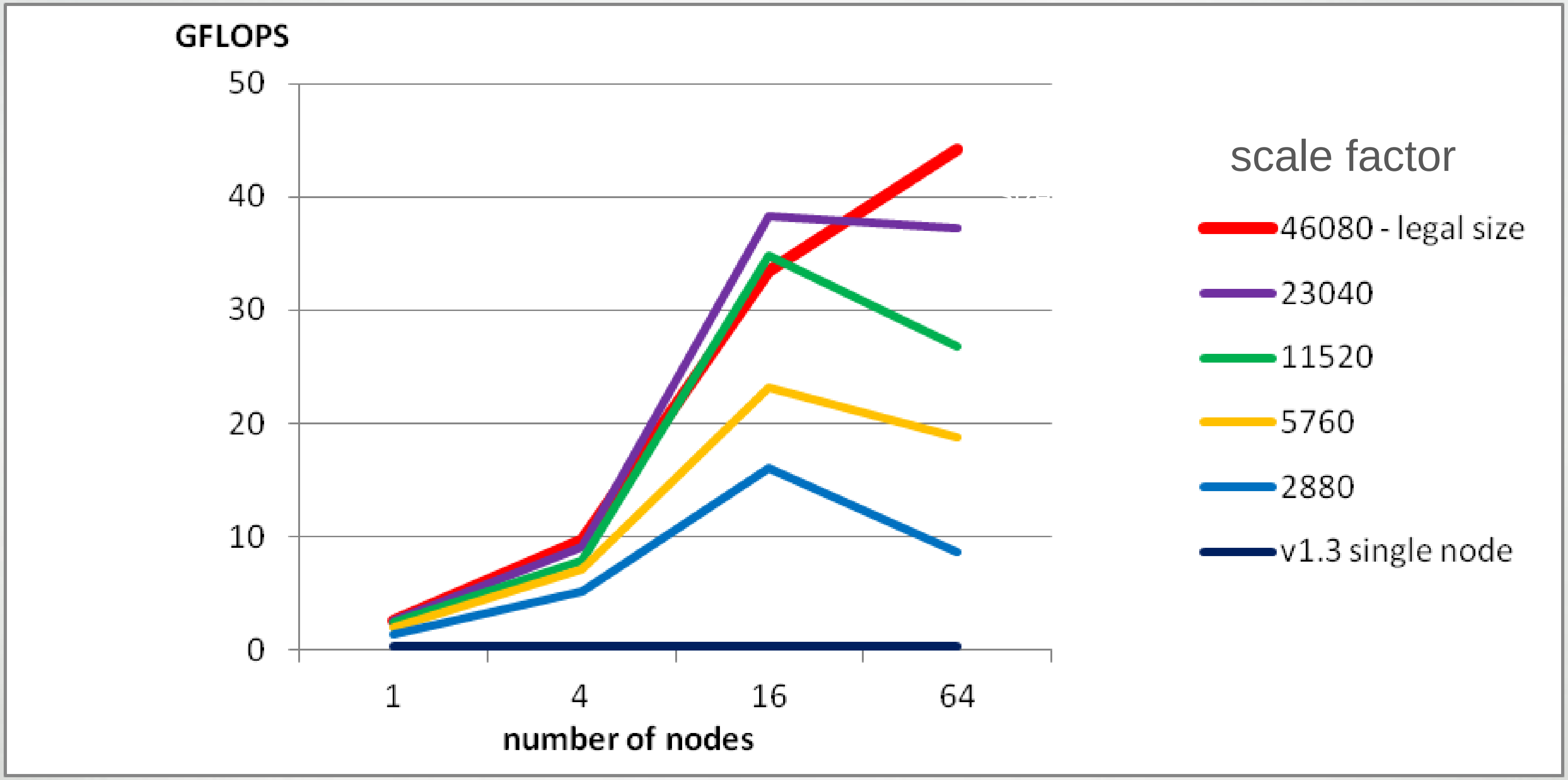
```
proc schurComplement(blk, const slct) {
  const AD_dim2 = MatVectSpace.dim(2)[blk..#blkSize];
  const BD_dim1 = MatVectSpace.dim(1)[blk..#blkSize];

  fillReplicants(blk, AD_dim2, BD_dim1);

  const RestLocal = MatVectSpace[blk+blkSize.., blk+blkSize..];
  Rest = RestLocal;
  RestByBlkSize = RestLocal by (blkSize, blkSize);

  if Rest.numIndices == 0 then
    return;

  forall (row,col) in RestByBlkSize do
    dgdriver(row, col, slct);
}
```

Most changes involve inserting temporaries to localize data or optimize copies until the compiler can do it automatically

Schur Complement (manually tuned version) 2

```
proc dgdriver(row, col, slct) {
  local {
    const rng1 = Rest.dim(1)[row..#blkSize],
          rng2 = Rest.dim(2)[col..#blkSize],
          slctLoc = slct;
    dgemm(replA.localSlice(rng1, ..),
          replB.localSlice(.., rng2),
          Ab.localSlice(rng1, rng2),
          slctLoc);
} }
proc dgemm(LreplA, LreplB, LAb, slct) {
  const r1 = LAb.domain.dim(1), r2 = LAb.domain.dim(2),
        rB = 1..blkSize;
for i in r1 do
    for k in r2 do {
      var acc: elemType = 0;
      for j in rB do
        acc -= LreplA[i,j] * LreplB[j,k];
      LAb[i,k] += acc;
}   }
```

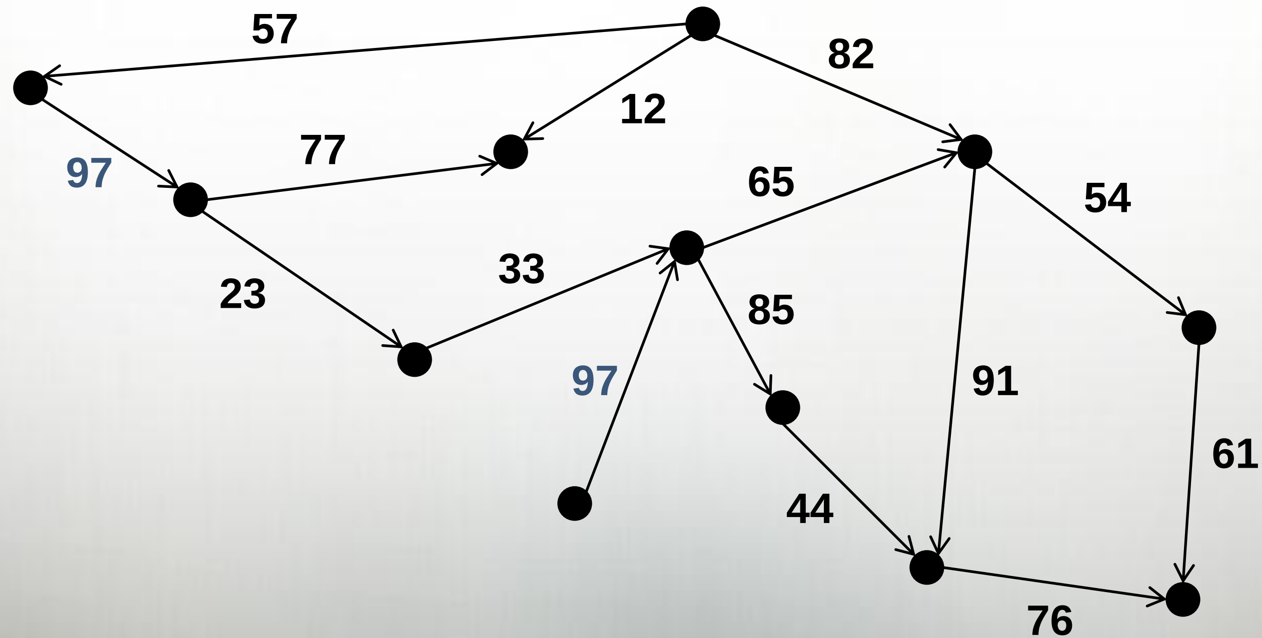
Schur Complement Performance (XE)



SSCA#2 Kernel 4

Goal: Compute Betweenness Centrality for graph

Our approach: Madduri et al.'s modification of Brandes's 2001 algorithm



SSCA#2, kernel 4 excerpts

```
do {
  coforall loc in Locales {
    on loc {
      forall u in rcLocal[Active_Level].Members {

        forall v in G.Neighbors(u) {
          on v {

            if  min_distance$[v].readXX() < 0  {
              if  min_distance$[v] < 0  {
                min_distance$[v] = current_distance_c;
                rcLocal[Next_Level].Members.add(v);
              } else {
                min_distance$[v] = current_distance_c;
              }
            }
```

Create a task per locale

Loop over all of the active vertices

On the locale owning the neighbor…

Update the minimum distance

SSCA#2, kernel 4 excerpts

```
          if  min_distance$[v].readFF() == current_distance_c  {
            path_count$[v] += path_count$[u].readFF();
            children_list[u].add_child(v);
          }
        }
      }}  }  }
      rcLocal[Next_Level] = new Level_Set(Sparse_Vertex_List);
      rcLocal[Next_Level].previous = rcLocal[Active_Level];

      Active_Remaining[here.id] =
        rcLocal[Active_Level].Members.numIndices:bool;
    }
  }
  remaining = || reduce Active_Remaining;
} while remaining;
```

Update our paths/children

Swap our current level set for a fresh one

See how much more local work we have

See if any work remains globally…

and continue if there is

SSCA#2 Kernel 4

- Performance is measured in Traversed Edges Per Second (TEPS)

  February 2011: 22 TEPS

  November 2011: 264,569 TEPS

- Improvement due to a combination of algorithmic changes and optimizations in Chapel

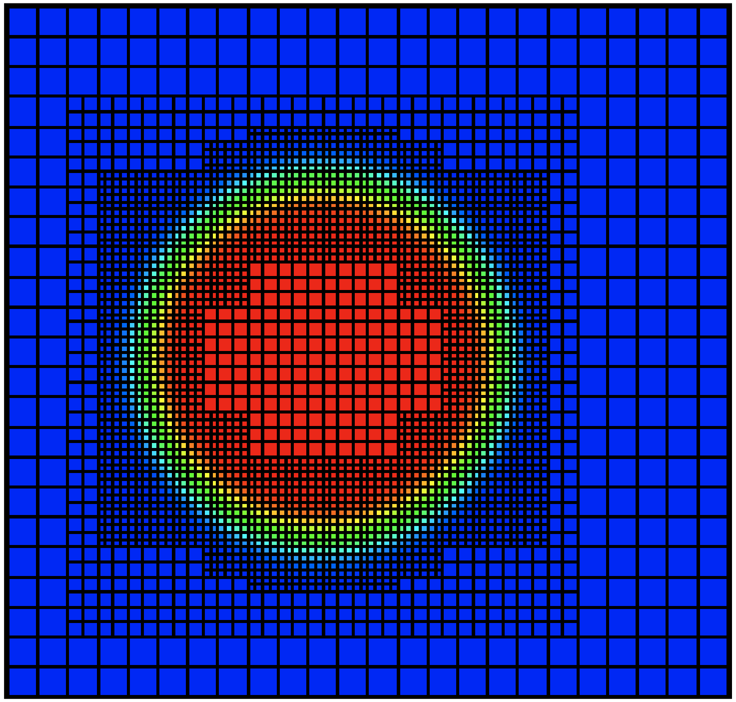
Adaptive Mesh Refinement Framework

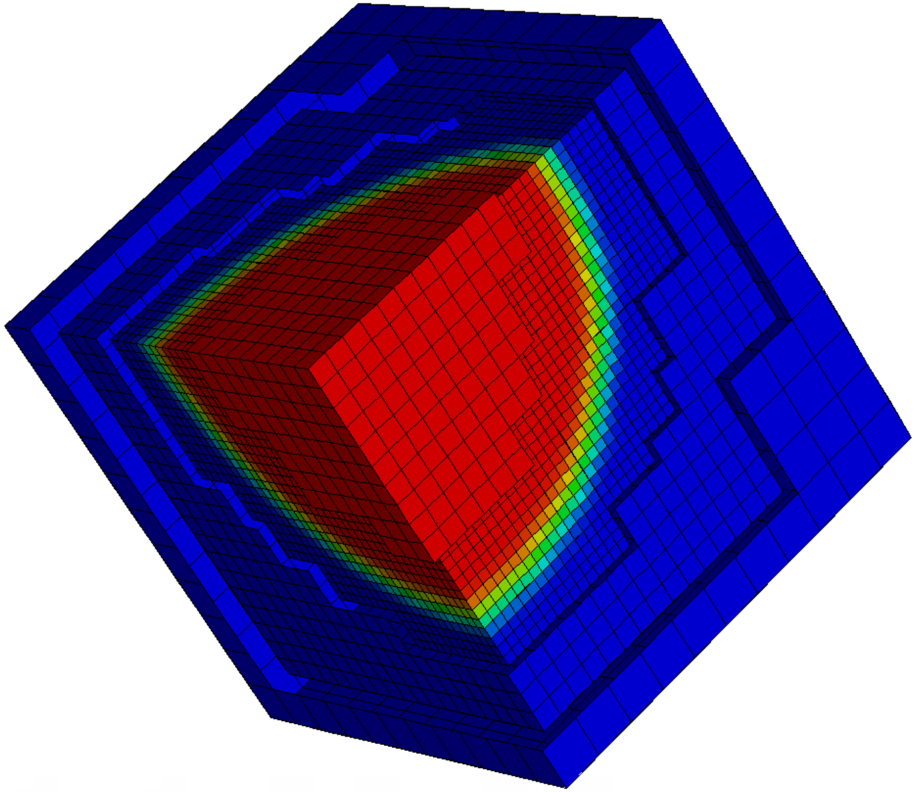
Setting: Had UW Applied Math PhD Student (Jonathan Claridge) write an AMR framework in Chapel

- expertise with AMR
- no prior use of Chapel
- little-to-no experience parallel programming

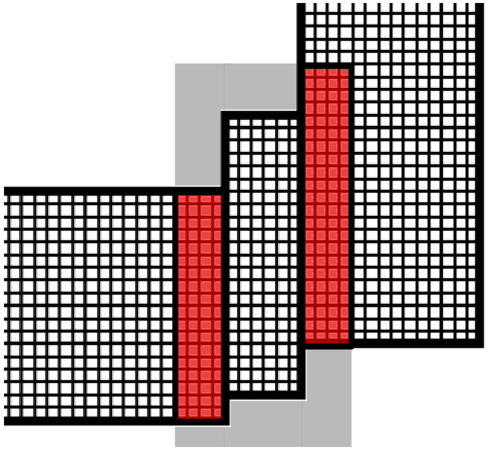
One Key Feature: Rank-independent

Uses the same code to produce results in 2D, 3D, 6D, 17D…





Levels: Sibling overlaps



- A grid's layer of ghost cells will, in general, overlap some of its siblings. Data will be copied into these overlapped ghost cells prior to mathematical operations.

- Calculating the **overlaps** between siblings:

```
var neighbors: domain(Grid);
var overlaps:  [neighbors] domain(dimension,stridable=true);

for sibling in parent_level.grids {
  var overlap = extended_cells[sibling.cells];

  if overlap.numIndices > 0 && sibling != this {
    neighbors.add(sibling);
    overlaps[sibling] = overlap;
  }
}
```

AMR Summary

- Chapel domains made many fundamental AMR calculations very easy, even in a dimension-independent setting

- Difficult to do apples-to-apples comparisons (in terms of size or performance) with other AMR codes
  - Or even to know what would be a meaningful use of the framework to time
  - Community is generally lacking standardized AMR benchmarks

Pointers to code

Stream, RA, HPL:

- In release: $CHPL_HOME/examples/hpcc
- Under SVN:
  https://chapel.svn.sourceforge.net/svnroot/chapel/trunk/test/release/examples/hpcc/
  https://chapel.svn.sourceforge.net/svnroot/chapel/trunk/test/studies/hpcc/HPL

SSCA #2:

- In release: $CHPL_HOME/examples/ssca2
- Under SVN:
  https://chapel.svn.sourceforge.net/svnroot/chapel/trunk/test/release/examples/ssca2/

AMR:

- Under SVN:
  https://chapel.svn.sourceforge.net/svnroot/chapel/trunk/test/studies/amr/

Summary

- Nodes are getting harder to optimize for
- Chapel compiler continues to improve
  - new optimizations
  - new distributions
  - new runtime capabilities
- Lots of work remains
  - but issues continue to seem tractable

Chapel at SC11 (see chapel.cray.com/events.html for details)

- **Mon:** full-day tutorial
- **Mon:** 2nd annual CHUG happy hour/meet-up
- **Tues:** "HPC Challenge" BoF (12:15-1:15)
- **Wed:** "Chapel Lightning Talks" BoF (12:15-1:15)
  - I/O, education, tasking, GPUs, interoperability
- **Thurs:** "Punctuated Equilibrium at Exascale" Panel (5:30-7:00)
- **Fri:** half-day tutorial

- **T-Th:** Chapel posters in PGAS booth, Chapel team members staffing (T 2-4, W 10-12, W 4-6, Th 10-12)