Chapel HPC Challenge Entry: 2012

SC12: November 13\textsuperscript{th}, 2012

Brad Chamberlain, Sung-Eun Choi, Martha Dumler,
Tom Hildebrandt, David Iten, Vass Litvinov, Greg Titus
Casey Battaglino, Rachel Sobel
Brandon Holt, Jeff Keasler
What is Chapel?

- An emerging parallel programming language
  - Design and development led by Cray Inc.
    - Broader community draws from academia, government, industry

- **Overall goal:** Improve programmer productivity

- A work-in-progress
Chapel's Implementation

- Being developed as open source at SourceForge
  https://sourceforge.net/projects/chapel/
- Licensed as BSD software

**Target Architectures:**
- Cray systems
- multicore desktops and laptops
- commodity clusters
- systems from other vendors
Chapel Codes for 2012

HPCC:
1. EP STREAM Triad
2. Global Random Access (RA)
3. Global HPL

Others:
4. SSCA#2, kernel 4: between-ness centrality computation
5. LULESH: LLNL Shock Hydrodynamics challenge problem
Highlights of Our 2012 Entry

- **new runtime** that leverages Cray hardware features
  - lightweight soft-threading technology
  - Gemini/Aries communication enhancements
    - lightweight puts/gets
    - network atomics
- RA, SSCA#2, HPL: significant **performance boosts**
- RA: switched to a **lossless version**
- a new benchmark: **LULESH**
- as always, **no libraries used** (as specified by the rules)
Chapel Source Code Sizes

Code Size Summary (Source Lines of Code)

- Chapel
- Reference

Chapel versions 3.4x – 13.3x shorter than reference versions

More importantly: more elegant, readable, flexible, maintainable

Stream EP: Chapel 72, Reference 329
RA: Chapel 112, Reference 938
HPL: Chapel 658, Reference N/A
SSCA #2, kernel 4: Chapel 504, Reference N/A
LULESH: Chapel 1276, Reference 4298
## Hardware Platforms

<table>
<thead>
<tr>
<th>Model</th>
<th>Name</th>
<th>Location</th>
<th># Compute Nodes</th>
<th>Processors</th>
<th>Memory / Node</th>
<th>Interconnect</th>
<th>Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray XC30™</td>
<td>Crystal</td>
<td>Cray</td>
<td>744</td>
<td>dual 16-core Intel Sandybridge (2.6/2.7 GHz)</td>
<td>32/64 GB</td>
<td>Cray Aries™</td>
<td>RA, SSCA#2,</td>
</tr>
<tr>
<td>Cray XE6™</td>
<td>Hoppe r</td>
<td>NERSC</td>
<td>6,384</td>
<td>dual 12-core AMD Magny-Cours (2.1 GHz)</td>
<td>32/64 GB</td>
<td>Cray Gemini™</td>
<td>Stream, RA</td>
</tr>
<tr>
<td>Cray XE6™</td>
<td>Hera</td>
<td>Cray</td>
<td>616</td>
<td>dual 16-core AMD Interlagos (2.1-2.5 GHz)</td>
<td>32/64 GB</td>
<td>Cray Gemini™</td>
<td>HPL, LULESH</td>
</tr>
</tbody>
</table>

Note: Performance numbers given in this talk should not be considered indicative of the hardware’s capabilities, but rather of current Chapel status.
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 zippered forall loop for the computation
last year:
  - issues due to multiple NUMA domains for first time
    - addressed by treating NUMA domains as distinct locales
      - (not the preferred Chapel model)
  - max: 32 TB/s on 2048 nodes of jaguar (Cray XT5™)

this year:
  - two approaches taken, one similar to last year:
    - max: 81.8 TB/s
    - avg: 40.0 TB/s
      - extrapolated for 2048 nodes of hopper (Cray XE6™)

improvement: 1.25 - 2.6x
  - primarily due to better hardware/larger node counts
EP STREAM Triad Chapel Performance

this year (continued):

- our second approach explicitly uses **hierarchical locales**, an emerging concept to represent vertical locality.
- an explicit hierarchical EP STREAM Triad might look like:

```
coforall loc in Locales on loc do local {
  var A, B, C: [1..m] real;
  forall numTasks = loc.numChildren();
  coforall tid in 1..numTasks do
    on loc.getChild(tid) {
      const chunk = getChunk(tid, numTasks, m);
      for i in chunk do
        A[i] = B[i] + alpha * C[i];
    }
}
```

Use on-clauses to refer to sublocales, as with traditional locales.
then, move the sublocale-aware code into the parallel iterators defined on arrays to restore the original elegance:

```chapel
coforall loc in Locales on loc do local {
    var A, B, C: [1..m] real;
    forall (a,b,c) in (A,B,C) do
    a = b + alpha * c;
}
```

not far enough along to report on performance this year, but code is working and initial results are promising
Global RA in Chapel

Declare two block-distributed index sets
- One for the table
- One for the set of updates

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

```
var T: [TableSpace] atomic uint;
```

Zipper iterate over the distributed set of updates along with an iterator generating random values.

```
forall (_, r) in zip(Updates, RAStream()) do
    T[r & indexMask].xor(r);
```

Represent table using atomic uints to guarantee a lossless implementation.

Perform updates using atomic xor; implemented using network AMOs on Gemini/Aries systems.
Global RA Chapel Performance

last year:
- 0.0368 GUPS on 512 nodes of jaguar (Cray XT5™)

this year:
- 2.7 GUPS on 512 nodes of crystal (Cray XC30™)
- 3.8 GUPS on 2048 nodes of hopper (Cray XE6™)

improvement: 103x
- primarily due to Chapel runtime improvements
- better hardware/larger node counts also helped
Chapel sketch of schurComplement:

```chapel
proc schurComplement(blk, AD, BD, Rest) {
    if Rest.numIndices == 0 then return; // Prevent replication of unequal-sized slices
    replicateA(blk);
    replicateB(blk);
    forall (row, col) in Rest by (blkSize, blkSize) {
        const outerRange = Rest.dim(1)(row..#blkSize),
            innerRange = Rest.dim(2)(col..#blkSize),
            blkRange = 1..blkSize;
        local {
            for a in outerRange do
                for w in blkRange do
                    for b in innerRange do
                        Ab[a,b] -= replA[a,w] * replB[w,b];
                } // local
        } // forall
    }
}
```

Triply nested loop for matrix multiply

Explicitly localized/hoisted values to work around lack of compiler optimizations at present time
HPL in Chapel

Code used in practice:

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

    if Rest.numIndices == 0 then return;

    replicateA(blk, AD.dim(2));
    replicateB(blk, BD.dim(1));

    const low1 = Rest.dim(1).low,
    low2 = Rest.dim(2).low;

coforall lid1 in 0..#numTargetLocalesDim1 do

coforall lid2 in 0..#numTargetLocalesDim2 do
    on targetLocales[lid1, lid2] do
        local {
            const myStarts1 = low1..n
                by blkSize*t11
                align 1+blkSize*lid1;
            const myStarts2 = low2..n+1
                by blkSize*t12
                align 1+blkSize*lid2;
            const blkRange = 1..blkSize;

            forall j1 in myStarts1 {
                const outerRange = j1..min(j1+blkSize-1, n);
                var h2 => replA._value.dsiLocalSlice1((outerRange, blkRange));

                forall j2 in myStarts2 {
                    const innerRange = j2..min(j2+blkSize-1, n+1);
                    var h1 => Ab._value.dsiLocalSlice1((outerRange, innerRange)),
                    h3 => replB._value.dsiLocalSlice1((blkRange, innerRange));

                    for a in outerRange {
                        const
                        h2dd = h2._value.data,
                        h2off = hoistOffset(h2, a, blkRange);
                    for w in blkRange {
                        const h2aw = h2dd(h2off+w); // h2[a,w];
                        const
                        h1dd = h1._value.data,
                        h1off = hoistOffset(h1, a, innerRange),
                        h3dd = h3._value.data,
                        h3off = hoistOffset(h3, w, innerRange);
                        for b in innerRange do
                            // Ab[a,b] -= replA[a,w] * replB[w,b];
                            h1dd(h1off+b) -= h2aw * h3dd(h3off+b);
                        } // for w
                    } // for a
                } // forall j2
            } // forall j1
        } // local
```

last year:
- only ran schurComplement, ignored other phases
- 4.42 GFLOPs on 64 nodes of kaibab (Cray XE6™)

this year:
- tuned other phases and ran entire benchmark
- 2031 GFLOPs on 64 nodes of hera (Cray XE6™)
  - 511,999 x 511,999 (official problem size), blocksize = 200
- 7833 GFLOPs on 576 nodes of hera (Cray XE6™)
  - 479,999 x 479,999 (smaller problem size), blocksize = 200

improvement: 451-1741x

first-order bottlenecks:
- quality/optimizability of generated code
- leaked memory (still diagnosing source)
HPL in Chapel: Life of a Chapel Benchmark

- **2011**: focused on correctness, elegance
- **2012**: focused on performance with sacrifices to elegance
- **next**: add optimizations, abstractions to restore elegance

*Source: Deutsche Bank*
SSCA#2 Kernel 4

**SSCA#2:**

- Unstructured graph benchmark
- Kernel 4: computes between-ness centrality
- Emerged from DARPA HPCS program
- Representative of big data analytics problems
var curr_Level = Active_Level[here.id].previous;

for current_distance in 2 .. graph_diameter by -1 {
    curr_Level = curr_Level.previous;

    for u in curr_Level.Members do
        on vertex_domain.dist.idxToLocale[u] do
            f4(BCaux, Between_Cent, u);

    barrier.barrier();
}

inline proc f4(BCaux, Between_Cent, u) {
    BCaux[u].depend =
        + reduce
            forall v in BCaux[u].children_list.
                Row_Children[1..BCaux[u].children_list.child_count.read()]
            do
                ( BCaux[u].path_count.read() / BCaux[v].path_count.read() ) *
                ( 1.0 + BCaux[v].depend );
    Between_Cent[u].add(BCaux[u].depend);
}

---

Loop over frontiers of the graph

Entire SSCA#2 benchmark in Chapel is generic w.r.t. graph representation:
- nD torus
- 1D edge lists
- associative domain edge lists
- ...

User can select between representations via a compile-time flag

Atomic operations used to avoid conflicting modifications
SSCA#2, kernel 4 Chapel Performance

last year:
- 264 TEPS on ~24 nodes of kaibab (Cray XE6™)
  - problem size: $2^{14}$ vertices

this year:
- 158 MTEPs on 744 locales of crystal (Cray XC30™)
  - problem size: $2^{28}$ vertices

improvement: 598,484x on 16384x bigger graph
- primarily due to improved runtime, compiler, and SSCA#2

first-order bottlenecks:
- memory utilization and leaks (still diagnosing source)
LULESH:

- Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics challenge problem
- developed by LLNL under DARPA UHPC
- serves as a proxy app for key computation patterns
- [https://computation.llnl.gov/casc/ShockHydro/](https://computation.llnl.gov/casc/ShockHydro/)

pictures courtesy of Rob Neely, Bert Still, Jeff Keasler, LLNL
LULESH in Chapel

- Chapel version of LULESH:
  - developed as a co-design exercise between LLNL and Cray
  - physics code (all but ~25 lines) unchanged when switching...
    - ...from 3D regular- vs. 1D irregular-mesh
    - ...from dense vs. sparse materials elements representation
  - great demonstration of domain maps, rank independent syntax
  - LLNL application scientists notably impressed

```chapel
proc CalcKinematicsForElems(dxx, dyy, dzz, const dt: real) {
  // loop over all elements
  forall k in Elems {
    var b_x, b_y, b_z: 8*real,
    d: 6*real,
    detJ: real;

    //get nodal coordinates from global arrays and copy into local arrays
    var x_local, y_local, z_local: 8*real;
    localizeNeighborNodes(k, x, x_local, y, y_local, z, z_local);

    //get nodal velocities from global arrays and copy into local arrays
    var xd_local, yd_local, zd_local: 8*real;
    localizeNeighborNodes(k, xd, xd_local, yd, yd_local, zd, zd_local);
    var dt2 = 0.5 * dt; //wish this was local, too...

    local {
      //volume calculations
      const volume = CalcElemVolume(x_local, y_local, z_local);
      const relativeVolume = volume / volo.localAccess[k];
      vnew.localAccess[k] = relativeVolume;
      delv.localAccess[k] = relativeVolume - v.localAccess[k];

      //set characteristic length
      arealg.localAccess[k] = CalcElemCharacteristicLength(x_local, y_local, z_local, volume);

      for param i in 1..8 {
        x_local[i] = dt2 * xd_local[i];
        y_local[i] = dt2 * yd_local[i];
        z_local[i] = dt2 * zd_local[i];
      }

      CalcElemShapeFunctionDerivatives(x_local, y_local, z_local, b_x, b_y, b_z, detJ);
      CalcElemVelocityGradient(xd_local, yd_local, zd_local, b_x, b_y, b_z, detJ, d);
    }

    // put velocity gradient quantities into their global arrays.
    dxx.localAccess[k] = d[1];
    dyy.localAccess[k] = d[2];
    dzz.localAccess[k] = d[3];
  }
}
```

Representation-Independent Physics!
last year: N/A

this year:
  • 442 seconds/cycle on 64 nodes of hera (Cray XE6™)
    • problem size: $192^3$
    • num cycles: 50

first-order bottlenecks:
  • reductions
  • atomic updates
  • communication optimizations (aggregation, overlap)
Benchmark Sources

STREAM:

RA:

HPL:

SSCA#2:
https://chapel.svn.sourceforge.net/svnroot/chapel/trunk/test/release/examples/benchmarks/ssca2

LULESH:
(Recipes for compiler/execution/environment options for our performance results available by request)
The Cray Chapel Team (Summer 2012)
Sun: Chapel tutorial (8:30am)

Mon: 3rd Annual Chapel Users Group (CHUG) Meeting

Tues: HPC Challenge BoF (12:15pm)

Wed: Chapel Lightning Talks BoF (12:15pm)

Wed: Chapel talk at KISTI booth (~3-4pm)

Wed: HPCS BoF (5:30pm)

Wed: Proxy Applications for Exascale BoF (5:30pm)

Thurs: HPC Educators Forum on Chapel (1:30pm)
Resources For After Today

Chapel project page: http://chapel.cray.com
- overview, papers, presentations, language spec, ...

Chapel SourceForge page: https://sourceforge.net/projects/chapel/
- release downloads, public mailing lists, code repository, ...

IEEE TCSC Blog Series:
- Myths About Scalable Parallel Programming Languages

Mailing Lists:
- chapel_info@cray.com: contact the team
- chapel-users@lists.sourceforge.net: user-oriented discussion list
- chapel-developers@lists.sourceforge.net: dev.-oriented discussion
- chapel-education@lists.sourceforge.net: educator-oriented discussion
- chapel-bugs@lists.sourceforge.net/chapel_bugs@cray.com: public/private bug forum