Chapel Comes of Age: Productive Parallelism at Scale
CUG 2018
Brad Chamberlain, Chapel Team, Cray Inc.
Or: What’s Chapel been up to since CUG 2013?

CUG 2018

Brad Chamberlain, Chapel Team, Cray Inc.
What is Chapel?

Chapel: A productive parallel programming language

- portable & scalable
- open-source & collaborative

Goals:

- Support general parallel programming
  - “any parallel algorithm on any parallel hardware”
- Make parallel programming at scale far more productive
Chapel and Productivity

Chapel aims to be as...

...programmable as Python
...fast as Fortran
...scalable as MPI, SHMEM, or UPC
...portable as C
...flexible as C++
...fun as [your favorite programming language]
CLBG Cross-Language Summary
(Oct 2017 standings)
CLBG Cross-Language Summary
(Oct 2017 standings, zoomed in)
CLBG Cross-Language Summary
(Oct 2017 standings, zoomed in)
CLBG Cross-Language Summary (Oct 2017 standings)

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)

smaller

faster
CLBG: Qualitative Code Comparisons

Can also browse program source code (but this requires actual thought!):

```
proc main()
    printColorEquations();
    const group1 = [i in .popSize1] new Chameneos(i, {(i-1)%3};Color);
    const group2 = [i in .popSize2] new Chameneos(i, colors10[i]);
    cobegin {
        holdMeetings(group1, n);
        holdMeetings(group2, n);
    }
    print(group1);
    print(group2);
    for c in group1 do delete c;
    for c in group2 do delete c;
}

// Print the results of getNewColor() for all color pairs.
// proc printColorEquations()
//     for c1 in Color do
//         for c2 in Color do
//             writeln(c1, " + ", c2, " -> ", getNewColor(c1, c2));
//             writeln();
//     //
//     // Hold meetings among the population by creating a shared meeting
//     // place, and then creating per-chameneos tasks to have meetings.
//     // proc holdMeetings(population, numMeetings)
//     //     const place = new MeetingPlace(numMeetings);
//     //     coforall c in population do
//     //         // create a task per chamenees c haveMeetings(place, population);
//     //     delete place;
// }

void get_affinity(int* ia, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
    cpu_set_t active_cpus;
    FILE* f;
    char* buf [2048];
    char const* pos;
    int cpu_idx;
    int physical_id;
    int core_id;
    int cpu_cores;
    int apic_id;
    int size_t cpu_count;
    size_t i;

    char const* processor_str = "processor";
    char const* processor_str_len = strlen(processor_str);
    char const* physical_id_str = "physical id";
    char const* physical_id_str_len = strlen(physical_id_str);
    char const* core_id_str = "core id";
    char const* core_id_str_len = strlen(core_id_str);
    char const* cpu_cores_str = "cpu cores";
    char const* cpu_cores_str_len = strlen(cpu_cores_str);

    CPU_ADDR(&active_cpus);
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
    cpu_count = 0;
    for (i = 0; i < CPU_SETSIZE; i++)
    {
        if (CPU_ISSET(i, &active_cpus))
        {
            cpu_count++;
        }
    }

    if (cpu_count == 0)
    {
        ia_samp[0] = 0;
        return;
    }

    ia_samp[0] = 1;
    CPU_ADDR(affinity1);
```

excerpt from 1210 gz Chapel entry  
excerpt from 2863 gz C gcc entry
Can also browse program source code (but this requires actual thought!):

excerpt from 1210 gz Chapel entry

```chapel
proc main()
    printColorEquations();
  end

  cobegin
      holdMeetings(group1, n);
      holdMeetings(group2, n);
  end

  for c in group1 do delete c;
  for c in group2 do delete c;

  println(group1);
  println(group2);

  // Print the results of getNewColor() for all colors.
  // PrintColorEquations() { for c in Color do
  //     writeln(c)
  //     getNewColor(c, d, e); }

  cobegin
      cobegin
          holdMeetings(group1, n);
          holdMeetings(group2, n);
      end
  end

  // Bold meetings among the population by creating a set of
  // new meetings, and then creating per-chameneos tasks to have
  // meetings

  proc holdMeetings(population, numMeetings) {
    const place = new MeetingPlace(numMeetings);
    coforall c in population do
        c.holdMeetings(place, population);
    delete place;
  }
```

excerpt from 2863 gz C gcc entry

```c
void main()
    //...

    proc holdMeetings(population, numMeetings) {
        const place = new MeetingPlace(numMeetings);
        coforall c in population do
            c.holdMeetings(place, population);
        delete place;
    }
```
CLBG: Qualitative Code Comparisons

Can also browse program source code *(but this requires actual thought!)*:

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
  cpu_set_t active_cpus;
  FILE* f;
  char buf [2048];
  char const* pos;
  int cpu_idx;
  int physical_id;
  int core_id;
  int cpu_cores;
  int apic_id;
  size_t cpu_count;
  size_t i;

  char const* processor_str = "processor";
  size_t processor_str_len = strlen(processor_str);
  char const* physical_id_str = "physical id";
  size_t physical_id_str_len = strlen(physical_id_str);
  char const* core_id_str = "core id";
  size_t core_id_str_len = strlen(core_id_str);
  char const* cpu_cores_str = "cpu cores";
  size_t cpu_cores_str_len = strlen(cpu_cores_str);

  CPUZERO(&active_cpus);
  sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
  cpu_count = 0;
  for (i = 0; i != CPU_SETSIZE; i++)
  {
    if (CPU_ISSET(i, &active_cpus))
    {
      cpu_count += 1;
    }
  }
  if (cpu_count == 1)
  {
    is_smp[0] = 0;
    return;
  }

  void CPUZERO(void);
  sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
  cpu_count = 0;
  for (i = 0; i != CPU_SETSIZE; i++)
  {
    if (CPU_ISSET(i, &active_cpus))
    {
      cpu_count += 1;
    }
  }
  if (cpu_count == 1)
  {
    is_smp[0] = 1;
    CPUZERO(affinity1);
  }
```

excerpt from 1210 gz Chapel entry

```
char const* core_id_str = "core id";
size_t core_id_str_len = strlen(core_id_str);
char const* cpu_cores_str = "cpu cores";
size_t cpu_cores_str_len = strlen(cpu_cores_str);
```

excerpt from 2863 gz C gcc entry
13 full-time employees + ~2 summer interns
Chapel Community Partners

(and several others…)

https://chapel-lang.org/collaborations.html
Outline

✓ What is Chapel?

➢ Chapel Overview
  ● Chapel: Then vs. Now
  ● Chapel User Profiles
  ● What’s Next?
Chapel language feature areas

Chapel language concepts

- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target
Base Language
iter fib(n) {
    var current = 0,
        next = 1;

    for i in 1..n {
        yield current;
        current += next;
        current <=> next;
    }
}

config const n = 10;
for f in fib(n) do
    writeln(f);

0
1
1
2
3
5
8
...
Base Language Features, by example

Iterators

```fib(n) {
  var current = 0,
  next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <= next;
  }
}
```

```config const n = 10;
for f in fib(n) do
  writeln(f);
```

0
1
1
2
3
5
8
...
Base Language Features, by example

Static type inference for:
- arguments
- return types
- variables

iter fib(n) {
  var current = 0,
         next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}

config const n = 10;

for f in fib(n) do
  writeln(f);
Base Language Features, by example

```plaintext
iter fib(n : int): int {
    var current: int = 0,
    next: int = 1;

    for i in 1..n {
        yield current;
        current += next;
        current <=> next;
    }
}
```

```plaintext
config const n: int = 10;
for f: int in fib(n) do
    writeln(f);
```

| 0 | 1 | 1 | 2 | 3 | 5 | 8 | ... |
```

Explicit types also permitted

Static Type Inference for:
• arguments
• return types
• variables

Explicit types also permitted

Static Type Inference for:
• variables
• arguments
• return types

Copyright 2018 Cray Inc.
```plaintext
iter fib(n) {
    var current = 0,
        next = 1;

    for i in 1..n {
        yield current;
        current += next;
        current <=< next;
    }
}

config const n = 10;
for f in fib(n) do
    writeln(f);
```

0 1 1 2 3 5 8 ...

Base Language Features, by example
Base Language Features, by example

```javascript
iter fib(n) {
  var current = 0,
      next = 1;
  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```javascript
config const n = 10;
for (i, f) in zip(0..#n, fib(n)) do
  writeln("fib #", i, " is ", f);
```

- fib #0 is 0
- fib #1 is 1
- fib #2 is 1
- fib #3 is 2
- fib #4 is 3
- fib #5 is 5
- fib #6 is 8
...
Base Language Features, by example

```
iter fib(n) {
    var current = 0,
        next = 1;
    for i in 1..n {
        yield current;
        current += next;
        current <=> next;
    }
}

config const n = 10;
for (i, f) in zip(0..#n, fib(n)) do
    writeln("fib #", i, " is ", f);
```

```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
...```
Base Language Features, by example

iter fib(n) {
    var current = 0,
        next = 1;

    for i in 1..n {
        yield current;
        current += next;
        current <=> next;
    }
}

config const n = 10;

for (i, f) in zip(0..#n, fib(n)) do
    writeln("fib #", i, " is ", f);

fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
...
Base Language Features, by example

```perl
iter fib(n) {
    var current = 0,
          next = 1;

    for i in 1..n {
        yield current;
        current += next;
        current <=> next;
    }
}
```

```perl
config const n = 10;

for (i, f) in zip(0..<n, fib(n)) do
    writeln("fib #", i, " is ", f);

fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
...
```
Other Base Language Features

- Object-oriented features
- Generic programming / polymorphism
- Procedure overloading / filtering
- Default args, arg intents, keyword-based arg passing
- Argument type queries / pattern-matching
- Compile-time meta-programming
- Modules (namespaces)
- Error-handling
- and more…
Locales, briefly

- Locales can run tasks and store variables
  - Think “compute node”

Locales:

- User’s main() executes on locale #0
Task Parallelism and Locality, by example

```
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
    printf("Hello from task \%n of \%n " +
            "running on \%s\n",
        tid, numTasks, here.name);
```

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```
Task Parallelism and Locality, by example

```
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  printf("Hello from task %n of %n "+
         "running on %s\n", tid, numTasks, here.name);
```

Abstraction of System Resources

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```
Task Parallelism and Locality, by example

High-Level Task Parallelism

```
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  wprintf("Hello from task %n of %n "+
          "running on %s", tid, numTasks, here.name);
```
Task Parallelism and Locality, by example

This is a shared memory program
Nothing has referred to remote locales, explicitly or implicitly

```chpl
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  printf("Hello from task %n of %n +
         "running on %s\n",
         tid, numTasks, here.name);
```

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```
Task Parallelism and Locality, by example

```chpl
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      printf("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

Prompt> chpl taskParallel.chpl
Prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
Task Parallelism and Locality, by example

```
taskParallel.chpl

coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      printf("Hello from task %n of %n "+
              "running on %s
",
             tid, numTasks, here.name);
  }
```

prompt> chpl taskParallel.chpl
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1033
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032

Abstraction of System Resources
Task Parallelism and Locality, by example

```chpl
taskParallel.chpl

coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      printf("Hello from task %n of %n "+
        "running on %s\n", tid, numTasks, here.name);
  }

prompt> chpl taskParallel.chpl
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```

Control of Locality/Affinity
Task Parallelism and Locality, by example

```chpl
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      printf("Hello from task %n of %n " +
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```
Data Parallelism in Chapel

Chapel language concepts

- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target

Higher-level Chapel
Data Parallelism, by example

dataParallel.chpl

```chpl
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
write(A);
```

prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5

1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
Data Parallelism, by example

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i, j) in D do
    A[i, j] = i + (j - 0.5)/n;
writeln(A);
```

prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
```
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```
Data Parallelism, by example

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i, j) in D do
    A[i, j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```
Data Parallelism, by example

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i, j) in D do
    A[i, j] = i + (j - 0.5)/n;
writeln(A);
```

```bash
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```
This is a shared memory program
Nothing has referred to remote locales, explicitly or implicitly

```chpl
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```
Distributed Data Parallelism, by example

```
use CyclicDist;
config const n = 1000;
var D = {1..n, 1..n}
dmapped Cyclic(startIdx = (1,1));
var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

Domain Maps (Map Data Parallelism to the System)
Distributed Data Parallelism, by example

```chpl
use CyclicDist;
config const n = 1000;
var D = {1..n, 1..n}
    dmapped Cyclic(startIdx = (1,1));
var A: [D] real;
forall (i,j) in D do
    A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5 --numLocales=4

1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
A Brief History of Chapel
A Brief History of Chapel: Infancy

Chapel’s Infancy: DARPA HPCS (2003–2012)

- ~6–7 FTEs
- Research focus:
  - distinguish locality from parallelism
  - seamlessly mix data- and task-parallelism
  - support user-defined distributed arrays, parallel iterators
- CUG 2013 paper captured post-HPCS project status:
  
  *The State of the Chapel Union*
  
  Chamberlain, Choi, Dumler, Hildebrandt, Iten, Litvinov, Titus
Crossing the Stream of Adoption: Post-HPCS Barriers

- Research Prototype
- Adopted in Production
- Performance & Scalability
- Immature Language Features
- Insufficient Libraries
- Memory Leaks
- Lack of Tools
- Lack of Documentation
- Fear of Being the Only User

Next DOE app
Next weather / climate model
[your production app here]
A Brief History of Chapel: Adolescence

Chapel’s Adolescence: “the five-year push” (2013–2018)

- Motivated by user enthusiasm for Chapel
- Development focus:
  - address weak points in HPCS prototype
  - support and grow the Chapel community
- ~13–14 FTEs
- This CUG 2018 talk & paper reports on progress during this time
Chapel Performance: Then vs. Now vs. Reference
Performance Focus Areas (during 5-year push)

Array Optimizations:
- shifted data optimization (eliminates arbitrary indexing overhead)
- loop-invariant code motion (eliminates meta-data overhead)
- eliminated multiply in indexing for 1D (and innermost dim of 2D+) arrays

Runtime Library Improvements:
- scalable parallel memory allocator
- tasks mapped to affinity aware user-level threads
- native/optimized comm with RDMA and limited software overhead

Optimized Communication:
- compiler locality analysis improvements
- bulk array assignments
- remote-value-forwarding, new distributions, fast-ons, …
Experimental Methodology

Methodology for the next several slides:

- Resurrected a copy of Chapel 1.7
  - updated it to build with current versions of gcc/g++
- Compared it to Chapel 1.17, released April 2018
- Used today’s Cray systems
- Used today’s benchmark codes
  - with modest edits for 1.7 in response to language changes
LCALS Serial Kernel

- Chapel source:

```chapel
for i in 0..#len do
    bvc[i] = cls * (compression[i] + 1.0);
```
LCALS Serial Kernel: Chapel Then vs. Now

Copyright 2018 Cray Inc.
LCALS Serial Kernels: Chapel Now vs. Ref

LCALS Serial Kernels (Normalized to Ref)

0.2
0.4
0.6
0.8
1
1.2
1.4
Normalized Time

pressure_calc
vol3d_calc
couple
init3
if_quad
hydro_1d
inner_prod
tridiag_elim
adi
diff_predict
first_diff
Pic_1d
gen_lin_recur
mat_x_mat
imp_hydro_2d

1 Locale (x 28 cores)

faster
LCALS Parallel Kernel: Chapel Then vs. Now

LCALS Parallel Time (seconds)

Time (sec)

0 10 20 30 40 50 60 70 80 90

Chapel 1.7
Chapel 1.17

faster

Copyright 2018 Cray Inc.
LCALS Parallel Kernels: Chapel Now vs. Ref

LCALS Parallel Kernels (Normalized to Ref)

Normalized Time

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Faster</th>
</tr>
</thead>
<tbody>
<tr>
<td>pressure_calc</td>
<td></td>
</tr>
<tr>
<td>energy_calc</td>
<td></td>
</tr>
<tr>
<td>vol3d_calc</td>
<td></td>
</tr>
<tr>
<td>del_dot_vec_2d</td>
<td></td>
</tr>
<tr>
<td>couple</td>
<td></td>
</tr>
<tr>
<td>fir</td>
<td></td>
</tr>
<tr>
<td>init3</td>
<td></td>
</tr>
<tr>
<td>muladdsub</td>
<td></td>
</tr>
<tr>
<td>if_quad</td>
<td></td>
</tr>
<tr>
<td>trap_int</td>
<td></td>
</tr>
<tr>
<td>pic_2d</td>
<td></td>
</tr>
</tbody>
</table>

1 Locale (x 28 cores)
HPCC STREAM Triad: Chapel Then

STREAM Performance (GB/s)

GB/s

Locales (x 28 cores / locale)

Chapel 1.7

better
HPCC STREAM Triad: Chapel Then vs. Now

STREAM Performance (GB/s)

Locales (x 28 cores / locale)

0 500 1000 1500 2000 2500 3000

0 1 2 4 8 16 32

Chapel 1.17
Chapel 1.7

better
HPCC STREAM Triad: Chapel Now vs. Ref

STREAM Performance (GB/s)

Locales (x 36 cores / locale)

GB/s

Reference
Chapel 1.17

Copyright 2018 Cray Inc.
PRK Stencil: Chapel Then

PRK Stencil Performance (Gflop/s)

Locales (x 28 cores / locale)
PRK Stencil: Chapel Then vs. Now

PRK Stencil Performance (Gflop/s)

Locales (x 28 cores / locale)

Gflop/s

Chapel 1.17
Chapel 1.7

better
PRK Stencil: Chapel Now vs. Ref

PRK Stencil Performance (Gflop/s)

Locales (x 36 cores / locale)
HPC Patterns: Chapel Now vs. reference

LCALS: Chapel 1.17 vs. Reference

HPCC RA: Chapel 1.17 vs. Reference

STREAM Triad vs. Reference

ISx: Chapel Now vs. Reference

PRK Stencil: Chapel Now vs. Reference

Nightly performance tickers online at: https://chapel-lang.org/perf-nightly.html
HPC Patterns: Chapel Now vs. reference

**LCALS**: Chapel 1.17 vs. Reference

![Local loop kernels](image)

**STREAM**

![Embarrassing/Pleasing Parallelism](image)

**ISx**: Chapel Now vs. Reference

![Bucket-Exchange Pattern](image)

**PRK**: Chapel Now vs. Reference

![Stencil Boundary Exchanges](image)

**HPCC RA**: Chapel 1.17 vs. Reference

![Global Random Updates](image)

---

Nightly performance tickers online at: [https://chapel-lang.org/perf-nightly.html](https://chapel-lang.org/perf-nightly.html)
HPC Patterns: Chapel Now vs. reference

LCALS: Chapel 1.17 vs. Reference
Local loop kernels

STREAM Triad: Chapel 1.17 vs. Reference
Embarrassing/Pleasing Parallelism

ISx: Chapel Now vs. Reference
Bucket-Exchange Pattern

PRK Stencil: Chapel Now vs. Reference
Stencil Boundary Exchanges

HPCC RA: Chapel 1.17 vs. Reference
Global Random Updates

Nightly performance tickers online at: https://chapel-lang.org/perf-nightly.html

Copyright 2018 Cray Inc.
/* Perform updates to main table. The scalar equivalent is: */

HPCC Random Access Kernel: MPI

while (i = SendCnt) {
    /* receive messages */
    MPI_Test(inreg, &have_done, &status);
    if (have_done) {
        if (status.MPI_TAG == UPDATE_TAG) {
            HPCC_Table[LocalOffset] = hpcc_inmsg;
            HPCC_Wait[(Buckets, &status) = inmsg);
        } else if (status.MPI_TAG == FINISHED_TAG) {
            MPI_Abort(MPI_COMM_WORLD, tparams.MyProc);
        } else {
            HPCC_InsertUpdate(Ran, WhichBuckets, status); pendingUpdates++;
        }
    } else {
        if (have_done) {
            MPI_Isend(outreq, &have_done, MPI_STATUS_IGNORE);
            outreq = MPI_REQUEST_NULL;
            if (have_done) {
                MPI_Abort(MPI_COMM_WORLD, tparams.MyProc);
            }
        } else {
            if (have_done) {
                MPI_Test(outreq, &have_done, MPI_STATUS_IGNORE);
                outreq = MPI_REQUEST_NULL;
            } else if (have_done) {
                MPI_Irecv(LocalRecvBuffer, localBufferSize, tparams.dtype64, (int) tparams.UPDATE_TAG, MPI_COMM_WORLD, &inmsg);
                pendingUpdates += peUpdates;
                while (pendingUpdates > 0) {
                    /* send our done messages */
                    for (proc_count = tparams.NumProcs; proc_count > 0) {
                        if (proc_count = tparams.MyProc) {
                            tparams.finish_req[tparams.MyProc] = MPI_REQUEST_NULL;
                        } else
                            MPI_Abort(MPI_COMM_WORLD, tparams.MyProc);
                    }
                    /* Finish everyone else up... */
                    while (NumberReceiving > 0) {
                        MPI_Waitall(&have_done, &status);
                    }
                    /* we got a done message. Thanks for playing... */
                    NumberReceiving = 0;
                    NumberReceiving = 0;
                    if (have_done) {
                        MPI_Waitall(&proc_count, &status);
                    } else
                        MPI_Abort(MPI_COMM_WORLD, tparams.MyProc);
                }
            }
        }
    }
}

Return = (Ran <= tparams.TableSize)
if (GlobalOffset < tparams.top) WhichBuckets = (GlobalOffset / tparams.MinLocalTableSize + 1));
else
WhichBuckets = (GlobalOffset - tparams.Remainder) / tparams.MinLocalTableSize);
if (WhichBuckets == tparams.MyProc) LocalOffset = (Ran & (tparams.TableSize - 1)) - tparams.GlobalStartMyProc;
HPCC_Table[LocalOffset] = Ran;
/* Perform updates to main table. The scalar equivalent is:

for (i=0; i<NUPDATE; i++) {
    Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
    Table[Ran & (TABSIZE-1)] ^= Ran;
} */

forall (_, r) in zip(Updates, RAStream()) do
    T[r & indexMask] ^= r;

MPI Comment

Chapel Kernel
HPCC RA: Chapel Now vs. Ref

RA Performance (GUPS)

Reference (bucketing)
Reference (no bucketing)
Chapel 1.17

Locales (x 36 cores / locale)

GUPS
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
16
32
64
128
256
better
Memory Leaks: Then vs. Now

(skipped at CUG due to time constraints)
Memory Leaks: Chapel Then vs. Now

Total Number of Nightly Tests

Number of Tests

<table>
<thead>
<tr>
<th>Chapel 1.7</th>
<th>Chapel 1.17</th>
</tr>
</thead>
<tbody>
<tr>
<td>4410</td>
<td>8478</td>
</tr>
</tbody>
</table>
Memory Leaks: Chapel Then vs. Now

Fraction of Tests Leaking Memory

Number of Tests

Chapel 1.7

- 4410
- 3128

Chapel 1.17

- 8478
- 302
Memory Leaks: Chapel Then vs. Now

Total Memory Leaked in Nightly Testing

MB of memory leaked

Chapel 1.7: 2137.0 MB
Chapel 1.17: 0.237 MB

Better
Memory Leaks: Remaining Leaks

~1/3 of memory leaked by one test (SSCA#2)

~2/3 of leaking tests leak < 256 bytes

~1/3 of leaking tests leak < 64 bytes
Chapel Language: Then vs. Now
Language: Then

Parallelism and Locality: Generally in good shape
- not many changes here since HPCS

Base Language: Left much to be desired
- lots of focus here since HPCS
Language: Now

Parallelism and Locality

● introduced task intents to reduce chances of race conditions
● and user-defined locale models to support new node architectures

Base Language

● fixed a number of problems with object-oriented programming
  ● records: poor memory management discipline
  ● classes: problems with generic classes, class hierarchies
● made strings usable
● added error-handling features
● made namespace improvements (and much more…)
Chapel Ecosystem: Then vs. Now
Documentation: Then

After HPCS:

- a PDF language specification
- a Quick Reference sheet
- a number of READMEs
- ~22 primer examples
Documentation: Now

Now: 200+ modern, hyperlinked, web-based documentation pages
After HPCS: ~25 library modules

- documented via source comments, if at all:
Libraries: Now

Now: ~60 library modules
  • web-documented, many user-contributed
Libraries: Now

**Math:** FFTW, BLAS, LAPACK, LinearAlgebra, Math

**Inter-Process Communication:** MPI, ZMQ (ZeroMQ)

**Parallelism:** Futures, Barrier, DynamicIters

**Distributed Computing:** DistributedIters, DistributedBag, DistributedDeque, Block, Cyclic, Block-Cyclic, …

**File Systems:** FileSystem, Path, HDFS

**Others:** BigInteger, BitOps, Crypto, Curl, DateTime, Random, Reflection, Regexp, Search, Sort, Spawn, …
Tools: Then

After HPCS:

- highlighting modes for emacs and vim
- chpIdoc: documentation tool (early draft)
Tools: Now

Now:

- highlighting modes for emacs, vim, atom, ...
- chpldoc: documentation tool
- mason: package manager
- c2chapel: interoperability aid
- bash tab completion: command-line help
- chplvis: performance visualizer / debugger
Then vs. Now: And so much more…

**Interoperability:**
- passing arrays & functions to C, working with C pointers, …

**Development process:**
- GitHub, Jenkins, Travis, interactive nightly performance graphs…

**Social media:** Twitter, Facebook, YouTube

**User support:** GitHub issues, StackOverflow, Gitter, email

**Web presence:** CLBG, Try It Online, CyberDojo, …

**Memory Leaks:** significantly reduced

**CHIUW:** annual community workshop
Chapel User Profiles
Chapel User Profiles

Current Users:
- Time-to-science Cosmologist

Potential Users:
- Commercial AI Scientist
- Genomic Researcher
- DOE Scientist
Chapel User Profiles

Current Users:
- Time-to-science Cosmologist
- Commercial AI Scientist

Potential Users:
- Genomic Researcher
- DOE Scientist

skipped at CUG due to time constraints
User Profile: Time-to-Science Cosmologist

Name: Nikhil Padmanabhan  
Title: Associate Professor of Physics and Astronomy, Yale University

Computations: Surveys of galaxies to constrain cosmological models, n-body simulations of gravity

Why Chapel?  “My interests in Chapel developed from a desire to have a lower barrier to writing parallel codes. In particular, I often find myself writing prototype codes (often serial), but then need to scale these codes to run on large numbers of simulations/datasets. Chapel allows me to smoothly transition from serial to parallel codes with a minimal number of changes.

“Another important issue for me is "my time to solution" (some measure of productivity vs performance). Raw performance is rarely the only consideration.”
Why Chapel? “I have used Fortran, R, Java and Python extensively. If I had to give up Chapel, I would probably move to C++. I prefer Chapel due to the extreme legibility and performance. We have abandoned Python on large problems for performance reasons. “We’ve now developed thousands of lines of Chapel code and a half dozen open source libraries for things like database connectivity, numerical libraries, graph processing, and even a REST framework. We’ve done this because AI is about to face an HPC crisis, and the folks at Chapel understand the intersection of usability and scalability.”
Potential User Profile: Genomic Researcher

Name: Jonathan Dursi  
Title: Senior Research Associate, The Hospital for Sick Children, Toronto  
Computations: Human genomics, bioinformatics, and medical informatics  

Why Chapel? “My interest in Chapel lies in its potential for bioinformatics tools that are currently either written in elaborately crafted, threaded but single node, C++ code, or in Python. Either has advantages and disadvantages (performance vs rapid development cycles), but neither has a clear path to cross-node computation, for performance as well as larger memory and memory bandwidth. Chapel has the potential to have some of the best of both worlds in terms of C++ and Python, as well as having a path to distributed memory.”
Why Chapel? “In Multiphysics applications separation of concerns and use of high level abstractions is critical for sustainable software. Chapel combines language features that would enable this for clean implementation.

“HPC Scientific software is made more complex than it needs to be because the only language designed for scientific work, Fortran, is losing ground for various reasons. Its object oriented features are clunky and make it nearly as unsuitable as other languages for scientific work. **Chapel appears to be parallel and modern Fortran done better, therefore has the potential to become a more suitable language.**”
Chapel and Productivity

Chapel aims to be as...

...programmable as Python
...fast as Fortran
...scalable as MPI, SHMEM, or UPC
...portable as C
...flexible as C++
...fun as [your favorite language]
What’s Next?
Crossing the Stream of Adoption

Research Prototype

Performance & Scalability

Immature Language Features

Insufficient Libraries

Memory Leaks

Lack of Tools

Lack of Documentation

Fear of Being the Only User

Adopted in Production

Next DOE app

Next weather / climate model

[your production app here]
Crossing the Stream of Adoption

Research Prototype

- Performance & Scalability
- Immature Language Features
- Insufficient Libraries
- Lack of Tools
- Fear of Being the Only User

Adopted in Production

- Next DOE app
- Next weather / climate model

[Your production app here]
Crossing the Stream of Adoption

Research Prototype
- MiniMD
- ISx
- CoMD
- CLBG
- PRK Stencil
- RA
- LULESH
- Stream
- LCALS

Adopted in Production
- What are the next stepping stones?
- Codes from startups
- Time-to-science academic codes

Next DOE app
- Where can Chapel help your workflow’s productivity?
- Next weather / climate model
- [your production app here]
Discovery Roadblocks

Data Science Pain Points

**SELECTING FEATURES**
Which features should be used for accurate predictions?

**DATA EXPLORATION**
Do I fully understand my data? Does it need to be cleaned?

**HYPER PARAMETERS**
What are the correct values to set the variables to before training?

**MODEL ENSEMBLES**
Which ensemble of AI/ML models will be more performant?

**MODEL RATIONALE**
Do I trust my model? Why does it predict that way?
Chapel AI Ecosystem
Sample Chapel AI Workflow

- User works from within a Jupyter notebook
- Uses Chapel to ingest large HDF5 data files
  - read in parallel
  - transformed / analyzed during ingestion
  - stored in a distributed Dataframe
- Starts working on model locally on laptop
- As confidence in model grows, tunes it at scale
  - feature selection
  - hyperparameter optimization
What’s Next?

Chapel’s college years: plans for 2018-2021

- Language Core
- Interoperability / Usability
- Portability
- Data Ingestion
- Chapel AI
What’s Next?

Chapel’s college years: plans for 2018-2021

- **Language Core**
  - Language stabilization: avoid backward-breaking changes
  - Sparse array improvements, partial reductions, delete-free features, …
  - Additional performance and scalability improvements

- **Interoperability / Usability**

- **Portability**

- **Data Ingestion**

- **Chapel AI**
What’s Next?

Chapel’s college years: plans for 2018-2021

- Language Core
- Interoperability / Usability
  - Python / C++ interoperability
  - Support for Jupyter notebooks / REPL
- Portability
- Data Ingestion
- Chapel AI
What’s Next?

Chapel’s college years: plans for 2018-2021

- Language Core
- Interoperability / Usability
- Portability
  - LLVM back-end
  - Target Libfabric/OFI
  - Target GPUs
  - Cloud computing support
- Data Ingestion
- Chapel AI
What’s Next?

Chapel’s college years: plans for 2018-2021

- Language Core
- Interoperability / Usability
- Portability
- Data Ingestion
  - Support HDF5, NetCDF, CSV, …
  - Transform-on-ingest
  - Distributed DataFrames support
- Chapel AI
What’s Next?

Chapel’s college years: plans for 2018-2021

- Language Core
- Interoperability / Usability
- Portability
- Data Ingestion
- Chapel AI
  - Hyperparameter optimization
  - Deep Learning
  - …
Summary

Chapel has made huge strides over the past five years

We’ve addressed many historical barriers to using Chapel

We’re continuing our work to support and improve Chapel

We’re looking for the next generation of Chapel users, as well as concrete use cases for AI / ML
Chapel’s Home in the Landscape of New Scientific Computing Languages (and what it can learn from the neighbours)

Jonathan Dursi, The Hospital for Sick Children, Toronto
“My opinion as an outsider...is that Chapel is important, Chapel is mature, and Chapel is just getting started.

“If the scientific community is going to have frameworks...that are actually designed for our problems, they’re going to come from a project like Chapel.

“And the thing about Chapel is that the set of all things that are ‘projects like Chapel’ is ‘Chapel.’”

—Jonathan Dursi

Chapel’s Home in the New Landscape of Scientific Frameworks
(and what it can learn from the neighbours)

CHIUW 2017 keynote

Dedicated to the Memory of Burton Smith
Chapel Central

https://chapel-lang.org
- downloads
- documentation
- resources
- presentations
- papers
Chapel Community

https://stackoverflow.com/questions/tagged/chapel
https://github.com/chapel-lang/chapel/issues
https://gitter.im/chapel-lang/chapel
chapel-announce@lists.sourceforge.net
Suggested Reading (healthy attention spans)

Chapel chapter from *Programming Models for Parallel Computing*

- a detailed overview of Chapel’s history, motivating themes, features
- published by MIT Press, November 2015
- edited by Pavan Balaji (Argonne)
- chapter is also available online

Other Chapel papers/publications available at https://chapel-lang.org/papers.html
Suggested Reading (short attention spans)

- a run-down of recent events (as of 2017)

- a short-and-sweet introduction to Chapel

**Six Ways to Say “Hello” in Chapel** (parts 1, 2, 3), Cray Blog, Sep-Oct 2015.
- a series of articles illustrating the basics of parallelism and locality in Chapel

**Why Chapel?** (parts 1, 2, 3), Cray Blog, Jun-Oct 2014.
- a series of articles answering common questions about why we are pursuing Chapel in spite of the inherent challenges

(index available on chapel-lang.org “blog posts” page), Apr-Nov 2012.
- a series of technical opinion pieces designed to argue against standard reasons given for not developing high-level parallel languages
Where to..

Submit bug reports:
- GitHub issues for chapel-lang/chapel: public bug forum
- chapel_bugs@cray.com: for reporting non-public bugs

Ask User-Oriented Questions:
- StackOverflow: when appropriate / other users might care
- Gitter (chapel-lang/chapel): community chat with archives
- chapel-users@lists.sourceforge.net: user discussions

Discuss Chapel development
- chapel-developers@lists.sourceforge.net: developer discussions
- GitHub issues for chapel-lang/chapel: for feature requests, design discussions

Discuss Chapel’s use in education
- chapel-education@lists.sourceforge.net: educator discussions

Directly contact Chapel team at Cray: chapel_info@cray.com
Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.