Chapel: A Parallel Language for Productive Scalable Computing

Brad Chamberlain, Chapel Team, Cray Inc.

SeaLang Meetup

December 6, 2017
This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.
My Background

Education:
- Earned Ph.D. from University of Washington CSE in 2001
  - focused on the ZPL data-parallel array language
- Remain associated with UW CSE as an Affiliate Professor

Industry:
- Currently a Principal Engineer at Cray Inc.
- Technical lead / founding member of the Chapel project
Who are you?

- **Workplace / Role?**
- **Programming Languages?**
  - Favorites?
  - Ones you work on / in?
- **Parallel Programming Experience?**
  - On desktop? At scale?
- **Anything else?**
Who are you?  (My Answers)

● **Workplace / Role?**  Cray / Chapel Technical Lead

● **Programming Languages?**
  ● Favorites?  Pascal (sentimental), Ada (safety), C (control / speed)
  ● Ones you work on / in?  Chapel, C/C++

● **Parallel Programming Experience?**  just a tad
  ● On desktop?  At scale?

● **Anything else?**  I don’t consider myself a PL expert
  ● more of a parallel expert who works in languages/compilers
Plan for Tonight

Elements:

- prepared overview talk
- from there, whatever you like…
  …interactive Chapel programming demo?
  …more in-depth presentation of some topic?
  …Q&A / discussion?

Ground Rules:

- please ask questions anytime
- if I get too hand-wavy, feel free to ask “got a visual for that?”
What is Chapel?
What is Chapel?

Chapel: A productive parallel programming language

- portable
- open-source
- a collaborative effort

Goals:

- Support general parallel programming
  - “any parallel algorithm on any parallel hardware”
- Make parallel programming at scale far more productive
What does “Productivity” mean to you?

**Recent Graduates:**
“something similar to what I used in school: Python, Matlab, Java, …”

**Seasoned HPC Programmers:**
“that sugary stuff that I don’t need because I was born to suffer”

**Computational Scientists:**
“something that lets me express my parallel computations without having to wrestle with architecture-specific details”

**Chapel Team:**
“something that lets computational scientists express what they want, without taking away the control that HPC programmers want, implemented in a language as attractive as recent graduates want.”
Chapel and Other Languages

Chapel strives 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]
“The Audacity of Chapel”

**audacity** (according to Google):
/ɔːˈdæsɪti/

**noun**

1. a willingness to take bold risks.
   “I applaud the audacity of the Chapel team in attempting to create a new language given how hard it is for new languages to succeed.”

2. rude or disrespectful behaviour; impudence.
   “I can’t believe the Chapel team has the audacity to create a new language when we already have [ C++ | MPI | OpenCL | Python | … ]!”
Scalable Parallel Programming Concerns

Q: What do HPC programmers need from a language?
A: **Serial Code:** Software engineering and performance
   **Parallelism:** What should execute simultaneously?
   **Locality:** Where should those tasks execute?
   **Mapping:** How to map the program to the system?
   **Separation of Concerns:** Decouple these concerns

*These are first-order concerns, yet…existing languages have not treated all of them as such.*
The Challenge

Q: So why don’t we already have such a language?
A: Technical challenges?
  ● while they exist, we don’t think this is the main issue…
A: Due to a lack of…
  …long-term efforts
  …resources
  …co-design between developers and users
  …community will
  …patience

Chapel is our attempt to reverse this trend
The Chapel Team at Cray (May 2017)

14 full-time employees + 2 summer interns + 2–4 GSoC students
The Chapel Team at Cray (May 2017)

You? A friend? (hiring a manager-evangelist)

14 full-time employees + 2 summer interns + 2–4 GSoC students
Chapel Community Partners

(and several others…)

https://chapel-lang.org/collaborations.html
A Chapel Sampler
Chapel language feature areas

Chapel language concepts

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

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

Lower-level Chapel
Base Language Features, by example

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

```rust
config const n = 10;

for f in fib(n) do
    writeln(f);
```

```
0
1
1
2
3
5
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 f in fib(n) do
    writeln(f);
```

Modern iterators
```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);
```

Configuration declarations (to avoid command-line argument parsing)

./a.out --n=1000000

```
0
1
1
2
3
5
8
...
```
config const n = 10;

for f in fib(n) do writeln(f);

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

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
Base Language Features, by example

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

Zippered iteration

```
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

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

```haskell
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

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

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

Config

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

Tuples

- `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
- ...

Copyright 2017 Cray Inc.
Base Language Features, by example

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

```plaintext
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 |
| ...         |
Task Parallelism and Locality Control

Diagram:
- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target
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\n",
             tid, numTasks, here.name);
  }

prompt> chpl taskParallel.chpl -o taskParallel
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

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

```
prompt> chpl taskParallel.chpl -o taskParallel
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**

```chpl
taskParallel.chpl

cforall 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 -o taskParallel
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

```
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);
  }
```
Task Parallelism and Locality, by example

Abstraction of System Resources

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

```bash
prompt> chpl taskParallel.chpl -o taskParallel
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**

```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);
  }
```

Not seen here:
Data-centric task coordination via atomic and full/empty vars

```
prompt> chpl taskParallel.chpl -o taskParallel
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\n", tid, numTasks, here.name);
  }

prompt> chpl taskParallel.chpl -o taskParallel
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
Parallelism and Locality: Distinct in Chapel

● This is a **parallel**, but local program:

```chapel
coforall i in 1..msgs do
    writeln(“Hello from task ”, i);
```

● This is a **distributed**, but serial program:

```chapel
writeln(“Hello from locale 0!”);
on Locales[1] do writeln(“Hello from locale 1!”);
on Locales[2] do writeln(“Hello from locale 2!”);
```

● This is a **distributed** parallel program:

```chapel
coforall i in 1..msgs do
    on Locales[i%numLocales] do
        writeln(“Hello from task ”, i,
            “ running on locale ”, here.id);
```
Chapel: Scoping and Locality

```
var i: int;
```

**Locales** (think: “compute nodes”)
Chapel: Scoping and Locality

```chapel
var i: int;
on Locales[1] {
```

Legend:
- **0**
- **1**
- **2**
- **3**
- **4**

*Locales* (think: “compute nodes”)
```chapel
var i: int;
on Locales[1] {
    var j: int;
}
```

**Locales** (think: “compute nodes”)
Chapel: Scoping and Locality

```chapel
var i: int;
on Locales[1] {
    var j: int;
    coforall loc in Locales {
        on loc {
```

Locales (think: “compute nodes”)
var i: int;
on Locales[1] {
    var j: int;
    coforall loc in Locales {
        on loc {
            var k: int;
            ...
        }
    }
}
Chapel: Scoping and Locality

```chapel
var i: int;
on Locales[1] {
    var j: int;
    coforall loc in Locales {
        on loc {
            var k: int;
            k = 2*i + j;
        }
    }
}
```

OK to access `i`, `j`, and `k` wherever they live

$k = 2*i + j$
Chapel: Scoping and Locality

```chapel
def 

var i: int; 
on Locales[1] {
    var j: int;
    coforall loc in Locales {
        on loc {
            var k: int;
            k = 2*i + j;
        }
    }
}

here, i and j are remote, so the compiler + runtime will transfer their values
```
Chapel: Locality queries

```chapel
var i: int;

on Locales[1] {
    var j: int;

    coforall loc in Locales {
        on loc {
            var k: int;

            ...here... // query the locale on which this task is running
            ...j.locale... // query the locale on which j is stored
        }
    }
}
```

Locales (think: “compute nodes”)
Higher-Level Features

Chapel language concepts

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

Higher-level Chapel
Data Parallelism, by example

```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 -o dataParallel
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 -o dataParallel
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 -o dataParallel
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

Data-Parallel Forall Loops

```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 -o dataParallel
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);
```

```
> chpl dataParallel.chpl -o dataParallel
> ./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
```
Chapel Evaluations
Website supporting cross-language comparisons

- 13 toy benchmark programs x ~28 languages x many implementations
- exercise key computational idioms
- specific approach prescribed

Take results with a grain of salt
- your mileage may vary

That said, it is one of the only such games in town…
Chapel’s approach to the CLBG:

- striving for elegance over heroism
- ideally: “Want to learn how program xyz works? Read the Chapel version.”
CLBG: Fast-faster-fastest graph (Sep 2016)

Relative performance, sorted by geometric mean

How many times slower?

program time / fastest program time

benchmarks game 09 Sep 2016 u64q
CLBG: Fast-faster-fastest graph (May 2017)

Relative performance, sorted by geometric mean

How many times slower?

benchmarks game
09 May 2017 u64q
Relative performance, sorted by geometric mean
Can sort results by execution time, code size, memory or CPU use:

<table>
<thead>
<tr>
<th>pidigits</th>
<th>description</th>
<th>program source code, command-line and measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td>× source</td>
<td>secs</td>
<td>mem</td>
</tr>
<tr>
<td>1.0</td>
<td>Chapel #2</td>
<td>1.62</td>
</tr>
<tr>
<td>1.0</td>
<td>Chapel</td>
<td>1.62</td>
</tr>
<tr>
<td>1.1</td>
<td>Pascal Free Pascal #3</td>
<td>1.73</td>
</tr>
<tr>
<td>1.1</td>
<td>C gcc</td>
<td>1.73</td>
</tr>
<tr>
<td>1.1</td>
<td>Ada 2005 GNAT #2</td>
<td>1.74</td>
</tr>
<tr>
<td>1.1</td>
<td>Rust #2</td>
<td>1.74</td>
</tr>
<tr>
<td>1.1</td>
<td>Rust</td>
<td>1.74</td>
</tr>
<tr>
<td>1.1</td>
<td>Swift #2</td>
<td>1.75</td>
</tr>
<tr>
<td>1.1</td>
<td>Lisp SBCL #4</td>
<td>1.79</td>
</tr>
<tr>
<td>1.2</td>
<td>C++ g++ #4</td>
<td>1.89</td>
</tr>
<tr>
<td>1.2</td>
<td>Lua #5</td>
<td>1.94</td>
</tr>
<tr>
<td>1.2</td>
<td>Go #3</td>
<td>2.02</td>
</tr>
<tr>
<td>1.3</td>
<td>PHP #5</td>
<td>2.15</td>
</tr>
<tr>
<td>1.3</td>
<td>PHP #4</td>
<td>2.16</td>
</tr>
<tr>
<td>1.3</td>
<td>Racket #2</td>
<td>2.17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pidigits</th>
<th>description</th>
<th>program source code, command-line and measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td>× source</td>
<td>secs</td>
<td>mem</td>
</tr>
<tr>
<td>1.0</td>
<td>Perl #4</td>
<td>3.53</td>
</tr>
<tr>
<td>1.5</td>
<td>Python 3 #2</td>
<td>3.51</td>
</tr>
<tr>
<td>1.5</td>
<td>PHP #4</td>
<td>2.16</td>
</tr>
<tr>
<td>1.5</td>
<td>Perl #2</td>
<td>3.92</td>
</tr>
<tr>
<td>1.5</td>
<td>PHP #5</td>
<td>2.15</td>
</tr>
<tr>
<td>1.6</td>
<td>Chapel #2</td>
<td>1.62</td>
</tr>
<tr>
<td>1.7</td>
<td>C gcc</td>
<td>1.73</td>
</tr>
<tr>
<td>1.7</td>
<td>Perl</td>
<td>15.87</td>
</tr>
<tr>
<td>1.8</td>
<td>Racket</td>
<td>25.63</td>
</tr>
<tr>
<td>1.8</td>
<td>Lua #7</td>
<td>3.76</td>
</tr>
<tr>
<td>1.8</td>
<td>Ruby #5</td>
<td>3.14</td>
</tr>
<tr>
<td>1.8</td>
<td>Lua #5</td>
<td>5.18</td>
</tr>
<tr>
<td>1.8</td>
<td>Pascal Free Pascal</td>
<td>5.27</td>
</tr>
<tr>
<td>1.9</td>
<td>Lisp SBCL #3</td>
<td>3.14</td>
</tr>
<tr>
<td>1.9</td>
<td>PHP #3</td>
<td>5.27</td>
</tr>
</tbody>
</table>

gz == code size metric
strip comments and extra whitespace, then gzip
Can also compare languages pair-wise:

- but only sorted by execution speed...

<table>
<thead>
<tr>
<th>k-nucleotide</th>
<th>secs</th>
<th>mem</th>
<th>gz</th>
<th>cpu</th>
<th>cpu load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapel</td>
<td>16.69</td>
<td>350,432</td>
<td>1063</td>
<td>62.96</td>
<td>100% 92% 93% 93%</td>
</tr>
<tr>
<td>Fortran Intel</td>
<td>87.62</td>
<td>203,604</td>
<td>2238</td>
<td>87.57</td>
<td>1% 0% 100% 0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fasta</th>
<th>secs</th>
<th>mem</th>
<th>gz</th>
<th>cpu</th>
<th>cpu load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapel</td>
<td>1.71</td>
<td>52,184</td>
<td>1392</td>
<td>5.90</td>
<td>99% 82% 83% 82%</td>
</tr>
<tr>
<td>Fortran Intel</td>
<td>2.53</td>
<td>8</td>
<td>1327</td>
<td>2.53</td>
<td>0% 1% 0% 100%</td>
</tr>
</tbody>
</table>
Scatter plots of CLBG code size x speed

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)
CLBG Cross-Language Summary
(Oct 2017 standings)

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)

smaller

faster

Copyright 2017 Cray Inc.
CLBG Cross-Language Summary
(Oct 2017 standings, zoomed in)

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)

smaller

faster

Copyright 2017 Cray Inc.
CLBG Cross-Language Summary
(Oct 2017 standings, zoomed in)
CLBG: Qualitative Comparisons

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

```c
void get_affinity(int* is_cmp, cpu_set_t* affinity1, cpu_set_t* affinity2) {
  cpu_set_t active_cpus;
  Fills;
  char buf[2048];
  char const* cpu_id;
  int physical_id;
  int core_id;
  int cpu_cores;
  int apic_id;
  int size_t;
  size_t;
  char const* processor_str = "processor";
  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);

  CPU_ZERO(&active_cpus);
  sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
  cpu_count = 0;
  for (i = 0; i < CPU_GETSIZE; i++) {
    if (CPU_ISSET(i, &active_cpus)) {
      cpu_count++;
    }
  }
  if (cpu_count == 1) {
    is_cmp[0] = 0;
    return;
  }
  is_cmp[0] = 1;
  CPU_ZERO(affinity1);
}
```

excerpt from 2863 gz C gcc entry

```chapel
proc main() {
  procColorEquations();
  const group1 = [i in 1..popSize] new Chameneos{i, ((i-1)*3).Color};
  const group2 = [i in 1..popSize] new Chameneos{i, colors[i][i]};
  cobegin {
    holdMeetings(group1, n);
    holdMeetings(group2, n);
  }
  for c in group1 do delete c;
  for c in group2 do delete c;

  // Print the results of getNewColor() for all color pairs.
  // proc procColorEquations() {
  //   for cl in Color do
  //     for c2 in Color do
  //       writeln(cl, ' + ', c2, ' -> ', getNewColor(cl, 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);
  //   forall c in population do // create a task per chameneos
  //     c.haveMeetings(place, population);
  //   delete place;
  // }
```

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

```c
void get_affinity(int *mp, cpu_set_t *affinity1, cpu_set_t *affinity2)
    active_cpus;
    f;
    buf [0x100];
    pos;
    cpu_idx;
    physical_id;
    core_id;
    cpu_cores;
    apic_id;
    cpu_count;
    i;

    processor_str = "processor";
    processor_str_len = strlen(processor_str);
    physical_id = "physical_id";
    physical_id_str_len = strlen(physical_id);
    core_str = "core_id";

    get_processor_str();
    get_physical_id();
    get_core_id();

    taskbar[
```

```chapel
proc main()
    printColorEquations();
    const group1 = {{i in [1..popSize]} new Chameneos[i,}.
    const group2 = {{i in [1..popSize]} new Chameneos[i,}.

    cobegin {
        holdMeetings(group1, n);
        cobegin {
            holdMeetings(group1, n);
            cobegin {
                holdMeetings(group2, n);
            }
        }
    }

    p.println("all");
    print(group2);
    for c in group1 do delete c;
    for c in group2 do delete c;

    // Print the results of getColor() for all colors.
    proc printColorEquations() {
        for cl in Color do
            writeln(cl, " + ", cl, " = ", getColor(cl, c), " ");
    }

    // Hold meetings among the population by creating a shared
    // place, and then creating per-chameleo tasks to have
    proc holdMeetings(population, numMeetings) {
        const place = new MeetingPlace(numMeetings);

        coforall c in population do
            c.meetingPlace(place, population);

        delete place;
    }
```

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

```
#include "sys/param.h"  // Chapel

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

CPU_ZERO(&active_cpus);
sched_getaffinity(0, sizeof(active_cpus), &active_cpus);

for (i = 0; i != CPU_SETSIZE; i += 1)
{
  if (CPU_ISSET(i, &active_cpus))
  {
    cpu_count += 1;
  }
}

if (cpu_count == 1)
{
  is_smp[0] = 0;
  return;
}
```

```
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 core:
  size_t cpu_cores_str_len = strlen(cpu_cores_str);

  CPU_ZERO(&active_cpus);
sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
  cpu_count = 0;
  for (i = 0; i != CPU_SETSIZE; i += 1)
  {
    if (CPU_ISSET(i, &active_cpus))
    {
      cpu_count += 1;
    }
  }

  if (cpu_count == 1)
  {
    is_smp[0] = 1;
    CPU_ZERO(affinity1);
  
    is_smp[0] = 1;
    CPU_ZERO(affinity2);
  }
```

excerpt from 1210 gz Chapel entry  
excerpt from 2863 gz C gcc entry
Chapel Performance: HPC Benchmarks

**LCALS: Chapel vs. C + OpenMP**
Shared memory performance competitive with hand-coded

**HPCC RA**
Performance of RA (atomics)

**HPCC Stream Triad: Chapel vs. MPI+OpenMP**

**Isx Peformance: Chapel vs. MPI, SHMEM**

**Stencil PRK Scalability**

Nightly performance graphs online at: https://chapel-lang.org/perf
**LCALS**: Chapel vs. C + OpenMP

Shared memory performance competitive with hand-coded

**Serial LCALS kernels: Chapel vs. g++**

**Parallel LCALS kernels: Chapel vs g++ w/ OMP**
HPCC Stream Triad: Chapel vs. MPI+OpenMP

Performance of STREAM
(GASNet/mpi+qthreads)

GB/s

Locales

Reference
1.11 EP
1.12 EP
1.11 Global
1.12 Global

16 32 64 128 256

better
HPCC RA Performance: Chapel vs. MPI

Performance of RA (atomics)

Locales (x 36 cores per locale)
- ref MPI no-bucketing
- ref MPI bucketing
- 1.15 u+q
- 1.15 u+q oversubscribed

better
ISx Performance: Chapel vs. MPI, SHMEM

ISx weakISO Total Time

Nodes (x 36 cores per node)

Time (seconds)

SHMEM
Chapel
MPI

better
Stencil PRK Scalability

Stencil PRK Performance (weak scaling)

GFlops/s vs Locales

MPI+OpenMP | Chapel

better
Chapel’s Multiresolution Features
**Multiresolution Design**: Support multiple tiers of features

- higher levels for programmability, productivity
- lower levels for greater degrees of control

- build the higher-level concepts in terms of the lower
- permit users to intermix layers arbitrarily
Domain Maps: A Multiresolution Feature

Domain maps are “recipes” that instruct the compiler how to map the global view of a computation…

\[ A = B + \alpha \cdot C; \]

…to the target locales’ memory and processors:
Sample Domain Maps: Block and Cyclic

```
var Dom = {1..4, 1..8} dmapped Block( {1..4, 1..8} );
```

```
var Dom = {1..4, 1..8} dmapped Cyclic( startIdx=(1,1) );
```
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);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
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
```
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 -o dataParallel
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
Distributed Data Parallelism, by example

Chapel’s prescriptive approach:

```chapel
forall (i, j) in D do...
```

⇒ invoke D’s default parallel iterator
• defined by D’s type / domain map

```
dataParallel.chpl
forall (i, j) in D do ...
⇒ invoke D’s default parallel iterator
• defined by D’s type / domain map
```

```
default domain map
• create a task per local core
• chunk indices across tasks
```

```
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 -o dataParallel
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
```
Distributed Data Parallelism, by example

Chapel’s prescriptive approach:

```chapel
forall (i,j) in D do...
```

⇒ invoke and inline D’s default parallel iterator

- defined by D’s type / domain map

```chapel
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 -o dataParallel
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
```
Distributed Data Parallelism, by example

Chapel’s prescriptive approach:

```chapel
forall (i,j) in D do...
```

What if I don’t like D’s iteration strategy?

- Write and call your own parallel iterator:
  ```chapel
  forall (i,j) in myParIter(D) do...
  ```
- Or, use a different domain map:
  ```chapel
  var D = {1..n, 1..n} dmapped Block(...);
  ```
- Or, write your own domain map and use it:
  ```chapel
  var D = {1..n, 1..n} dmapped MyDomMap(...);
  ```

Domain Maps specify...

- ...mapping of indices to locales
- ...layout of domains / arrays in memory
- ...parallel iteration strategies
- ...core operations on arrays / domains
Chapel and Performance Portability

- Avoid locking key policy decisions into the language
  - Array memory layout?
  - Sparse storage format?
  - Parallel loop policies?
Chapel and Performance Portability

- Avoid locking key policy decisions into the language
  - Array memory layout? not defined by Chapel
  - Sparse storage format? not defined by Chapel
  - Parallel loop policies? not defined by Chapel

- Instead, permit users to specify these in Chapel itself
  - goal: to make Chapel a future-proof language
Another Key Multiresolution Feature

**locale models:** User-specified locale types for new node architectures
- how do I allocate memory, create tasks, communicate, …

Like domain maps, these are…
- written in Chapel by expert users using lower-level features
- targeted by the compiler as it lowers code
- available to the end-user via higher-level abstractions
Wrapping Up
What’s Next? (Big Ticket Items)

● LLVM back-end as the default
● Work towards Chapel 2.0 release
  ● goal: no changes thereafter that break backwards compatibility
● Support for delete-free computation
● GPU support
● Application studies / application partnerships
Crossing the Stream of Adoption

Research Prototype
Adopted in Production
Next MET Office model
Next DOE app
[your production app here]

MiniMD
ISx
RA
CLBG
PRK Stencil
CoMD
LULESH
LCALS
Stream

What are the next stepping stones?
Who’s interested in meeting us partway?

Codes from startups
Time-to-science academic codes

[Your production app here]
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 for solving scientific problems 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

Chapel Resources
The Chapel Parallel Programming Language

What is Chapel?

Chapel is a modern programming language that is...

- **parallel:** contains first-class concepts for concurrent and parallel computation
- **productive:** designed with programmability and performance in mind
- **portable:** runs on laptops, clusters, the cloud, and HPC systems
- **scalable:** supports locality-oriented features for distributed memory systems
- **open-source:** hosted on GitHub, permissively licensed

New to Chapel?

As an introduction to Chapel, you may want to...

- read a blog article or book chapter
- watch an overview talk or browse its slides
- download the release
- browse sample programs
- view other resources to learn how to trivially write distributed programs like this:

```
use CyclicDist; // use the cyclic distribution library
config const n = 100;
// use .as.out --> (real) to override this default
forall i in 1:n (i=0) do print("Hello from iteration ", i, " of ", n, " running on node ", here.id);
```

What’s Hot?

- Chapel 1.16 is now available—download a copy today!
- The CHIUW 2018 call for participation is now available!
- A recent Cray blog post reports on highlights from CHIUW 2017.
- Chapel is now one of the supported languages on Try It Online!
- Watch talks from ACCU 2017, CHIUW 2017, and ATPESC 2016 on YouTube.
- Browse slides from PADAL, EAGE, EMBRACE, ACCU, and other recent talks.
- See also: What’s New?
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 now also available online

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

- a run-down of recent events

- 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

- a series of technical opinion pieces designed to argue against standard reasons given for not developing high-level parallel languages
Chapel StackOverflow and GitHub Issues
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
#chapel-users (irc.freenode.net): user-oriented IRC channel
chapel-users@lists.sourceforge.net: user discussions

Discuss Chapel development
chapel-developers@lists.sourceforge.net: developer discussions
#chapel-developers (irc.freenode.net): developer-oriented IRC channel

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 publically 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, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. 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 in this document are the property of their respective owners.