Chapel Motivation

Q: Why doesn’t parallel programming have an equivalent to Python / Matlab / Java / C++ / (your favorite programming language here)?
   ● one that makes it easy to quickly get codes up and running
   ● one that is portable across system architectures and scales
   ● one that bridges the HPC, data analysis, and mainstream communities

A: We believe this is due not to any particular technical challenge, but rather a lack of sufficient…
   …long-term efforts
   …resources
   …community will
   …co-design between developers and users
   …patience

Chapel is an attempt to break this trend
What is Chapel?

● **An emerging parallel programming language**
  ● Design and development led by Cray Inc.
    ● in collaboration with academia, labs, industry; domestically & internationally

● **A work-in-progress**

● **Goal**: Improve productivity of parallel programming
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”
want full control
to ensure performance”

**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 need, implemented in a language as attractive as recent graduates want.”
Chapel's Implementation

- **Being developed as open source at GitHub**
  - Licensed as Apache v2.0 software

- **Portable design and implementation, targeting:**
  - multicore desktops and laptops
  - commodity clusters and the cloud
  - HPC systems from Cray and other vendors
  - *in-progress*: manycore processors, CPU+accelerator hybrids, …
Outline

✔ Chapel Motivation and Background

➢ Chapel in a Nutshell

● Chapel Project: Past, Present, Future

● Chapel Resources
Multiresolution Design: Support multiple tiers of features

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

Chapel language concepts

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

- build the higher-level concepts in terms of the lower
- permit the user to intermix layers arbitrarily
Lower-Level Features

Chapel language concepts

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

Lower-level Chapel
Chapel in a Nutshell: Base Language

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

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

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
...

CLU-style iterators

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

```
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
...
```
Chapel in a Nutshell: Base Language

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

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

```chapel
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
...
```
Chapel in a Nutshell: Base Language

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

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

```chapel
defib = fib(10);
```

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

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

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
...

swap operator
Chapel in a Nutshell: Base Language

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

```chapel
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
...```
Chapel in a Nutshell: Base Language

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

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

<table>
<thead>
<tr>
<th>fib #</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#0</td>
<td>0</td>
</tr>
<tr>
<td>#1</td>
<td>1</td>
</tr>
<tr>
<td>#2</td>
<td>1</td>
</tr>
<tr>
<td>#3</td>
<td>2</td>
</tr>
<tr>
<td>#4</td>
<td>3</td>
</tr>
<tr>
<td>#5</td>
<td>5</td>
</tr>
<tr>
<td>#6</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Copyright 2015 Cray Inc.
Chapel in a Nutshell: Task Parallelism, Locality

```chapel
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      writef("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
```
Chapel in a Nutshell: Task Parallelism, Locality

High-Level Task Parallelism

```
taskParallel.chpl

coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    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
```
Chapel in a Nutshell: Task Parallelism, Locality

Abstraction of System Resources

taskParallel.chpl

coforall loc in Locales do
on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
        writesf("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
Chapel in a Nutshell: Task Parallelism, Locality

```
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    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
```
Chapel in a Nutshell: Task Parallelism, Locality

taskParallel.chpl

```chapel
coforall loc in Locales do
    on loc {
        const numTasks = here.maxTaskPar;
        coforall tid in 1..numTasks do
            printff("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
```
Higher-Level Features

Chapel language concepts

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

Higher-level Chapel
Chapel in a Nutshell: Data Parallelism

```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 --numLocales=4 --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
```
Chapel in a Nutshell: Data Parallelism

```
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 --numLocales=4 --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
```
Chapel in a Nutshell: Data Parallelism

```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 --numLocales=4 --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
```
Chapel in a Nutshell: Data Parallelism

Data-Parallel Forall Loops

```chapel
use CyclicDist;
cfg 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 --numLocales=4 --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
```
Chapel in a Nutshell: Data Parallelism

```
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 --numLocales=4 --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
Chapel in a Nutshell: Data Parallelism

```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 --numLocales=4 --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
```
Parallelism and Locality: Orthogonal in Chapel

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

```chapel
begin writeln("Hello world!");
writeln("Goodbye!");
```
Parallelism and Locality: Orthogonal in Chapel

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

  ```chapel
  begin writeln("Hello world!");
  writeln("Goodbye!");
  ```

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

  ```chapel
  writeln("Hello from locale 0!");
  on Locales[1] do writeln("Hello from locale 1!");
  writeln("Goodbye from locale 0!");
  ```
Parallelism and Locality: Orthogonal in Chapel

- This is a **parallel**, but local program:
  ```chapel
  begin writeln(“Hello world!”);
  writeln(“Goodbye!”);
  ```

- This is a **distributed**, but serial program:
  ```chapel
  writeln(“Hello from locale 0!”);
  on Locales[1] do writeln(“Hello from locale 1!”);
  writeln(“Goodbye from locale 0!”);
  ```

- This is a **distributed**, **parallel** program:
  ```chapel
  begin on Locales[1] do writeln(“Hello from locale 1!”);
  on Locales[2] do begin writeln(“Hello from locale 2!”);
  writeln(“Goodbye from locale 0!”);
  ```
Outline

✔ Chapel Motivation and Background
✔ Chapel in a Nutshell
➢ Chapel Project: Past, Present, Future
• Chapel Resources
Chapel’s Origins: HPCS

DARPA HPCS: High Productivity Computing Systems

● **Goal:** improve productivity by a factor of 10x
● **Timeframe:** Summer 2002 – Fall 2012
● Cray developed a new system architecture, network, software stack…
  ● this became the very successful Cray XC30™ Supercomputer Series

...and a new programming language: Chapel
Chapel under HPCS: Major Successes

Clean, general parallel language design
- unified data-, task-, concurrent-, nested-parallelism
- distinct concepts for parallelism and locality
- multiresolution language design philosophy

SSCA#2 demonstration on the prototype Cray XC30
- unstructured graph compact application
- clean separation of computation from data structure choices
- fine-grain latency-hiding runtime
- use of Cray XC30™ network AMOs via Chapel’s ‘atomic’ types
- ran on full-scale demo system for significant amount of time

Portable design and implementation
- while still being able to take advantage of Cray-specific features

Revitalization of Community Interest in Parallel Languages
- HPF-disenchantment became interest, cautious optimism, enthusiasm
Chapel under HPCS: Shortcomings

Performance was hit-or-miss (and mostly “miss” at scale)
  ● a litmus test for the HPC community

Focused on a narrow set of benchmarks (mostly SSCA#2)
  ● several key idioms and language features were neglected

Contract milestones were set too far in advance
  ● unable to respond effectively to needs of real users
  ● changes required contract renegotiations

Insufficient focus on emerging node architectures
  ● unable to effectively leverage NUMA nodes, GPUs

Didn’t get over the tipping point of adoption
  ● but, we got far enough to make it to the next level…
Chapel’s 5-year push

- Based on positive user response to Chapel under HPCS, Cray undertook a five-year effort to improve it
  - we’ve just completed our second year

- **Focus Areas:**
  1. Improving *performance* and scaling
  2. Fixing immature aspects of the language and implementation
     - e.g., strings, memory management, error handling, …
  3. Porting to emerging architectures
     - Intel Xeon Phi, accelerators, heterogeneous processors and memories, …
  4. Improving *interoperability*
  5. Growing the Chapel user and developer *community*
     - including non-scientific computing communities
  6. Exploring transition of Chapel *governance* to a neutral, external body
The Chapel Team at Cray
Chapel is a Collaborative, Community Effort

(and many others as well, some of which you will hear from today…)

http://chapel.cray.com/collaborations.html
A Year in the Life of Chapel

- **Two major releases per year** (April / October)
  - ~a month later: detailed release notes

- **SC** (Nov)
  - annual Lightning Talks BoF featuring talks from the community
  - annual CHUG happy hour
  - plus tutorials, panels, BoFs, posters, educator sessions, exhibits, …

- **CHIUW**: Chapel Implementers and Users Workshop (May/June)
  - kicked off May 2014 at IPDPS

- **Talks, tutorials, research visits, blogs, …** (year-round)
Outline

- Chapel Motivation and Background
- Chapel in a Nutshell
- Chapel Project: Past, Present, Future
- Chapel Resources
Suggested Reading

Overview Papers:

  - a detailed overview of Chapel’s history, motivating themes, features

  - a higher-level overview of the project, summarizing the HPCS period
Blog Articles:

  - a short-and-sweet introduction to Chapel

- **Why Chapel?** (part 1, part 2, part 3), Cray Blog, June-October 2014.
  - a recent 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 combat standard arguments against the development of high-level parallel languages
Online Resources

Project page: [http://chapel.cray.com](http://chapel.cray.com)
- overview, papers, presentations, language spec, …

GitHub page: [https://github.com/chapel-lang](https://github.com/chapel-lang)
- download Chapel; browse source repository; contribute code

Facebook page: [https://www.facebook.com/ChapelLanguage](https://www.facebook.com/ChapelLanguage)
Community Resources

SourceForge page: https://sourceforge.net/projects/chapel/
- hosts community mailing lists
  (also serves as an alternate release download site to GitHub)

Mailing Aliases:
- chapel_info@cray.com: contact the team at Cray
- chapel-announce@lists.sourceforge.net: read-only announcement list
- chapel-users@lists.sourceforge.net: user-oriented discussion list
- chapel-developers@lists.sourceforge.net: developer discussion
- chapel-education@lists.sourceforge.net: educator discussion
- chapel-bugs@lists.sourceforge.net: public bug forum