Chapel Boot Camp

Ben Albrecht
Chapel Team, Cray Inc.
June 2, 2017
Safe Harbor Statement

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.
Motivation for Chapel

Q: Why doesn’t HPC programming have an equivalent to Python / Matlab / Java / C++ / ___(your favorite programming language here)__?  
  ● one that makes it easy to get programs up and running quickly  
  ● 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  
  …patience

Chapel is our attempt to reverse this trend!
What is Chapel?

**Chapel**: An emerging parallel programming language
- portable
- open-source
- a collaborative effort
- a work-in-progress

**Goals:**
- Support general parallel programming
  - “any parallel algorithm on any parallel hardware”
- Make parallel programming 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”
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 want,
implemented in a language as attractive as recent graduates want.”
Chapel is Portable

- Chapel is designed to be hardware-independent

- The current release requires:
  - a C/C++ compiler
  - a *NIX environment (Linux, OS X, BSD, Cygwin, WSL, …)
  - POSIX threads
  - RDMA, MPI, or UDP (for distributed memory execution)

- Chapel can run on…
  - laptops and workstations
  - commodity clusters
  - the cloud
  - HPC systems from Cray and other vendors
  - modern processors like Intel Xeon Phi, GPUs*, etc.
  - * = not yet supported in the official release
Chapel is Open-Source

- Chapel’s development is hosted at GitHub
  - [https://github.com/chapel-lang](https://github.com/chapel-lang)

- Chapel is licensed as Apache v2.0 software

- Instructions for download + install are online
  - [http://chapel.cray.com/download.html](http://chapel.cray.com/download.html)
Chapel Community R&D Efforts

(and several others, some of whom you will hear from today…)

http://chapel.cray.com/collaborations.html
Outline

✅ Chapel Motivation and Background
➢ Chapel in a Nutshell
   ● Chapel Project: Past, Present, Future
   ● Chapel Resources
Chapel’s Multiresolution Philosophy

**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 the user to intermix layers arbitrarily

-Chapel language concepts-
Lower-Level Features

Chapel language concepts

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

Lower-level Chapel

Target Machine
Lower-Level Features

Chapel language concepts

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

Lower-level Chapel
Base Language Features: Fibonacci Example

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

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

```haskell
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: Fibonacci Example

**Iterators**

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

```CLU
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: Fibonacci Example

```plaintext
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
...
```
Base Language Features: Fibonacci Example

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

```python
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: Fibonacci Example

```javascript
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);
```

Copyright 2017 Cray Inc.
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
...
Base Language Features: Fibonacci Example

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

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

```lang
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: Fibonacci Example

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

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

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>fib</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fib</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>fib</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>fib</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>fib</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>fib</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>fib</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Lower-Level Features

Chapel language concepts

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

Lower-level Chapel

Target Machine
Task Parallelism

beginTask.chpl

begin writeln("Hello!");
writeln("Goodbye...");

prompt> chpl beginTask.chpl -o beginTask
prompt> ./beginTask
Hello!
Goodbye...
prompt> ./beginTask
Goodbye...
Hello!
Task Parallelism

beginTask.chpl

```
begin writeln("Hello!");
writeln("Goodbye...");
```

`prompt> chpl beginTask.chpl -o beginTask`

`prompt> ./beginTask`

Hello!
Goodbye...

`prompt> ./beginTask`

Goodbye...
Hello!
Lower-Level Features

Chapel language concepts

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

Lower-level Chapel
**Task Parallelism & Locality Control**

```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
```
Task Parallelism & Locality Control

High-Level Task Parallelism

```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
```
Task Parallelism & Locality Control

Abstraction of System Resources

```chpl
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);
  }
```
Task Parallelism & Locality Control

**Control of Locality/Affinity**

```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
```
**Task Parallelism & Locality Control**

Abstraction of System Resources

```chpl
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
```
Task Parallelism & Locality Control

High-Level Task Parallelism

taskParallel.chpl

```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
Task Parallelism & Locality Control

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

taskParallel.chpl

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

Copyright 2017 Cray Inc.
Parallelism and Locality: Orthogonal 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);
  ```
Higher-Level Features

Chapel language concepts

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

Higher-level Chapel
Data Parallelism

dataParallel.chpl

```chapl
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

Domains (Index Sets)

```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

**Arrays**

```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

```
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

dataParallel,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);
```

Domain Maps
(Map Data Parallelism to the System)
Distributed Data Parallelism

Distributions
- BlockCycDist
- BlockDist
- CyclicDist
- DimensionalDist2D
- PrivateDist
- ReplicatedDist
- SparseBlockDist
- StencilDist

DataParallel.chpl

```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);
```

Layouts
- CSR

Prompt>
```sh
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
```
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’s focus areas

- Based on positive user response to Chapel under HPCS, Cray undertook a longer-term effort to improve it
  - we’ve just completed our fourth year of this effort

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
Chapel is a Work-in-Progress

● **Currently being picked up by early adopters**
  ● Users who try it generally like what they see

● **Most current features are functional and working well**
  ● some areas under active development, particularly:
    ● Initializers
    ● Error handling

● **Performance is improving, but remains hit-or-miss**
  ● shared memory performance is often competitive with C+OpenMP
  ● distributed memory performance continues to need more work
Outline

✓ Chapel Motivation and Background
✓ Chapel in a Nutshell
✓ Chapel Project: Past, Present, Future
➢ Chapel Resources
Chapel Websites

Project page: http://chapel.cray.com
  ● overview, papers, presentations, language spec, ...

GitHub: https://github.com/chapel-lang
  ● download Chapel; browse source repository; contribute code

Facebook: https://www.facebook.com/ChapelLanguage

Twitter: https://twitter.com/ChapelLanguage
Suggested Reading

Chapel chapter from *Programming Models for Parallel Computing*
- a detailed overview of Chapel’s history, motivating themes, features
- chapter is now also available online

Other Chapel papers/publications available at [http://chapel.cray.com/papers.html](http://chapel.cray.com/papers.html)
Chapel Blog Articles

- a short-and-sweet introduction to Chapel

**Chapel Springs into a Summer of Code**, Cray Blog, April 2016.
- a run-down of some current events

**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
Mailing Lists

low-traffic (read-only):
  chapel-announce@lists.sourceforge.net: announcements about Chapel

community lists:
  chapel-users@lists.sourceforge.net: user-oriented discussion list
  chapel-developers@lists.sourceforge.net: developer discussions
  chapel-education@lists.sourceforge.net: educator discussions

(subscribe at SourceForge: http://sourceforge.net/p/chapel/mailman/)

To contact the Cray team:
  chapel_info@cray.com: contact the team at Cray
  chapel_bugs@cray.com: for reporting non-public bugs
Other Community Resources

IRC channels (freenode.net):
  #chapel: user-oriented discussions
  #chapel-developers: developer discussions

Stack Overflow
  stackoverflow.com: [chapel] tag monitored by core team

GitHub Issues:
  github.com/chapel-lang/chapel/issues: bug reports & feature requests
Questions about Chapel?
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.