Chapel:
Productive Parallel Programming
from the Pacific Northwest

Brad Chamberlain, Cray Inc. / UW CS&E
Pacific Northwest Prog. Languages and Software Eng. Meeting
March 15th, 2016
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 / (your favorite programming language here) ?

A: We believe this is due less to technical challenges, and more because of insufficient…

…long-term efforts
…resources
…community will
…co-design between developers and users
…patience

Chapel is our attempt to change this
What is Chapel?

**Chapel**: An emerging parallel programming language

- extensible
- 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.”
Note: We currently have full-time, intern, and Google SoC opportunities available
The Broader Chapel Community

(And many others as well...)

http://chapel.cray.com/collaborations.html
Introduction to Chapel by Example
Chapel’s Multiresolution Philosophy

**Multiresolution Design:** Support multiple tiers of features

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

*Chapel language concepts*

```
<table>
<thead>
<tr>
<th>Domain Maps</th>
<th>Data Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Parallelism</td>
<td></td>
</tr>
<tr>
<td>Base Language</td>
<td></td>
</tr>
<tr>
<td>Locality Control</td>
<td></td>
</tr>
<tr>
<td>Target Machine</td>
<td></td>
</tr>
</tbody>
</table>
```

- 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
Base Language Features, by example

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
  ...

Copyright 2016 Cray Inc.
Base Language Features, by example

CLU-style iterators

```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);
```

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 2016 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
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
...

Copyright 2016 Cray Inc.
Base Language Features, by example

```plaintext
ter 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);
```

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
...

zippered iteration
Base Language Features, by example

```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 |
```
...
Base Language Features, by example

```
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, by example

```chapel
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
...```
Task Parallelism, Locality Control, by example

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
Task Parallelism, Locality Control, by example

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
Task Parallelism, Locality Control, by example

Abstraction of System Resources

taskParallel.chpl

```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
Task Parallelism, Locality Control, by example

```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);
  }
```

Control of Locality/Affinity

```
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, by example

```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
```
High-Level Task Parallelism

```chapel
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, by example

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, by example

Data-centric task coordination via atomic and F/E variables (not seen here)

```
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
```
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
Chapel by Example: Data Parallelism

```chapel
use CyclicDist;
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);
```

### Output

```
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
```
Chapel by Example: Data Parallelism

Domains (Index Sets)

```chapel
use CyclicDist;
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
```
Chapel by Example: Data Parallelism

```
use CyclicDist;
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
```
Chapel by Example: Data Parallelism

Data-Parallel Forall Loops

```chapl
use CyclicDist;
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 -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
```
Chapel by Example: 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);
```

Data Parallelism

```
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
```
Chapel by Example: 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 --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
```
LULESH: a DOE Proxy Application

**Goal:** Solve one octant of the spherical Sedov problem (blast wave) using Lagrangian hydrodynamics for a single material

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

1288 lines of source code
plus 266 lines of comments
487 blank lines

(the corresponding C+MPI+OpenMP version is nearly 4x bigger)

This can be found in the Chapel release in examples/benchmarks/lulesh/
This is the only representation-dependent code. It specifies:

- data structure choices:
  - structured vs. unstructured mesh
  - local vs. distributed data
  - sparse vs. dense materials arrays
- a few supporting iterators

Domain maps insulate the rest of the application from these choices.
Chapel Characterizations
Chapel is Extensible

Advanced users can create their own...
  ...array layouts and distributions (domain maps)...
  ...scheduling policies for forall loops...
  ...architectural models and mappings...

...as Chapel code, without modifying the compiler.

Why? To make the language future-proof.

This is our main research challenge: How to create a language that does not lock these policies into its definition while obtaining competitive performance?
Chapel is a Work-in-Progress

● Currently being picked up by early adopters
  ● Users who try it typically like what they see
  ● Last release got 1400+ downloads over six months

● Most features are functional and working well
  ● some areas need further attention: object-oriented features, strings

● Performance is improving, but not yet optimal
  ● shared memory performance is typically competitive with C+OpenMP
  ● distributed memory performance can be hit-or-miss

● Our current work is focused on addressing these lacks
Chapel is Portable

- Chapel’s design is hardware-independent

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

- 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.

* = academic work only; not yet supported in the official release
Chapel is Open-Source

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

- Chapel is licensed as Apache v2.0 software

- Instructions for download + install are online
  - see http://chapel.cray.com/download.html
Chapel: For More Information
Chapel Websites

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

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

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

Twitter: [https://twitter.com/ChapelLanguage](https://twitter.com/ChapelLanguage)
Suggested Reading

Chapel chapter from *Programming Models for Parallel Computing*
- a detailed overview of Chapel’s history, motivating themes, features
- edited by Pavan Balaji, published by MIT Press
- an early draft is available online, entitled *A Brief Overview of Chapel*

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

**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 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
- chapel-bugs@lists.sourceforge.net: public bug forum

*(subscribe at SourceForge: [http://sourceforge.net/p/chapel/mailman/](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
Get Involved!

Attend CHIUW 2016 at IPDPS (Chicago, May 27-28)
- 3rd annual Chapel Implementers and Users Workshop
- May 27th: mini-conference day
  - keynote: Nikhil Padmanabhan, Astrophysics, Yale University
  - 4 research paper talks, 10 short talks, “state of the project” talk, discussion
- May 28th: code camp day

Send us your students! / Join us!
- as Google Summer of Coders, interns, full-time employees

Propose a research collaboration
- join the growing Chapel community!
Questions?
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, URIKA, and YARCDATA. 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.

Copyright 2016 Cray Inc.