Chapel in Ten* Minutes

Brad Chamberlain, Chapel Team, Cray Inc.
NII Shonan Meeting: Putting Heterogeneous High-Performance Computing at the Fingertips of Domain Experts
November 17th, 2015

* mmmmaybe…
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.
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 “Productive” mean?

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

Seasoned HPC Programmers:
“that sugary stuff that I can’t afford to use because I need full control to ensure optimal 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.”
Introduction to Chapel by Example
Lower-Level Features

Chapel language concepts

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

Lower-level Chapel

Target Machine
Chapel by Example: 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
...

Copyright 2015 Cray Inc.
Chapel by Example: Base Language

CLU-style iterators

```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 by Example: 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
...
```
Chapel by Example: Base Language

```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
...
```
Chapel by Example: Base Language

```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
...
```
Chapel by Example: 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
...
```
Chapel by Example: 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 by Example: Task Parallelism, Locality

taskParallel.chpl

```plaintext
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 by Example: Task Parallelism, Locality

High-Level Task Parallelism

```
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 by Example: Task Parallelism, Locality

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
",
               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 by Example: Task Parallelism, Locality

Control of Locality/Affinity

```chapel
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      writeln("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 by Example: Task Parallelism, Locality

```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

Abstraction of System Resources
Chapel by Example: 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
      wwritef("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 by Example: Task Parallelism, Locality

```
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
```
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);
```

```
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

```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

```chpl
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);
```

Data-Parallel Forall Loops

```
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}
    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)

```bash
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/*\.chpl
This is all of the 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
Chapel Characterizations
Chapel is Extensible

Advanced users can create their own…
  …forall-loop schedules…
  …array layouts and distributions…
  …abstract system models…

…as Chapel code, without modifying the compiler.

Why? Make a future-proof language.

This has been our main research challenge: How to create a language that does not lock these policies into the implementation without sacrificing performance?
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*

- 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
Chapel is Portable

- Chapel’s design is intended to be hardware-independent

- The current release requires:
  - a C compiler
  - a *NIX environment
  - 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 by the official release
Chapel is Open-Source

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

● Chapel is licensed as Apache v2.0 software

● Download/install online
  ● see http://chapel.cray.com/download.html for instructions
Chapel is a Collaborative Effort

(and many others as well…)

http://chapel.cray.com/collaborations.html
Chapel is a Work-in-Progress

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

- Most features are functional and working well
  - Some areas need improvements: strings, object-oriented features

- Performance is hit-or-miss
  - Shared memory performance is often competitive with C+OpenMP
  - Distributed memory performance needs more work

- We are actively working to address these lacks
Stream-EP Performance Over Time

Stream EP Performance Across Chapel Releases (128 nodes)

Chapel version (six-months per release)

GB/s per node

v1.9  v1.10  v1.11  v1.12

MPI+OpenMP
Chapel
Chapel: For More Information
Chapel Websites

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

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

Facebook page: https://www.facebook.com/ChapelLanguage
Suggested Chapel Reading

**A Brief Overview of Chapel**

- a detailed overview of Chapel’s history, motivating themes, features
- early draft chapter for *Programming Models for Parallel Computing*, edited by Pavan Balaji, MIT Press
- buy the book to get the updated version

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 Aliases

**read-only:**
- chapel-announce@lists.sourceforge.net: announcements about Chapel

**read/write:**
- 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

**write-only:**
- chapel_info@cray.com: contact the team at Cray
- chapel_bugs@cray.com: for reporting non-public bugs

Subscribe at SourceForge: [http://sourceforge.net/p/chapel/mailman/](http://sourceforge.net/p/chapel/mailman/)
- (also serves as an alternate release download site to GitHub)
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 2015 Cray Inc.