Chapel Overview

Brad Chamberlain, Chapel Team, Cray Inc.

SC12: November 14th, 2012

What is Chapel?

- An emerging parallel programming language
  - Design and development led by Cray Inc.
    - with contributions from academics, labs, industry
  - Initiated under the DARPA HPCS program
- **Overall goal:** Improve programmer productivity
- A work-in-progress

Chapel's Implementation

- Being developed as open source at SourceForge
- Licensed as BSD software
- **Target Architectures:**
  - Cray architectures
  - multicore desktops and laptops
  - commodity clusters
  - systems from other vendors
  - (in-progress: CPU+accelerator hybrids, manycore, …)

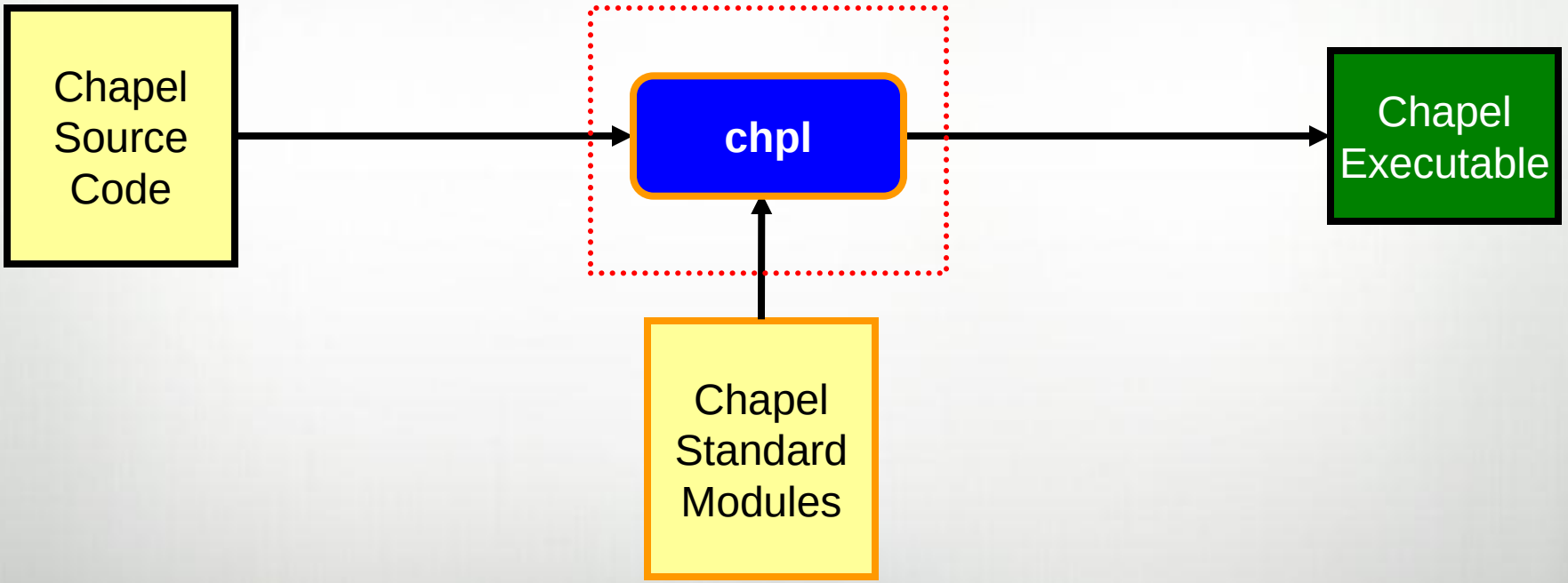
Chapel at SC12 (see chapel.cray.com/events.html for details)

- ✓ **Sun:** Chapel tutorial (8:30am)
- ✓ **Mon:** 3rd Annual Chapel Users Group (CHUG) Meeting
- ✓ **Tues:** HPC Challenge BoF (12:15pm)
- ➢ **Wed:** Chapel Lightning Talks BoF (12:15pm)
- **Wed:** Chapel talk at KISTI booth (3pm)
- **Wed:** HPCS BoF (5:30pm)
- **Wed:** Proxy Applications for Exascale BoF (5:30pm)
- **Thurs:** HPC Educators Forum on Chapel (1:30pm)

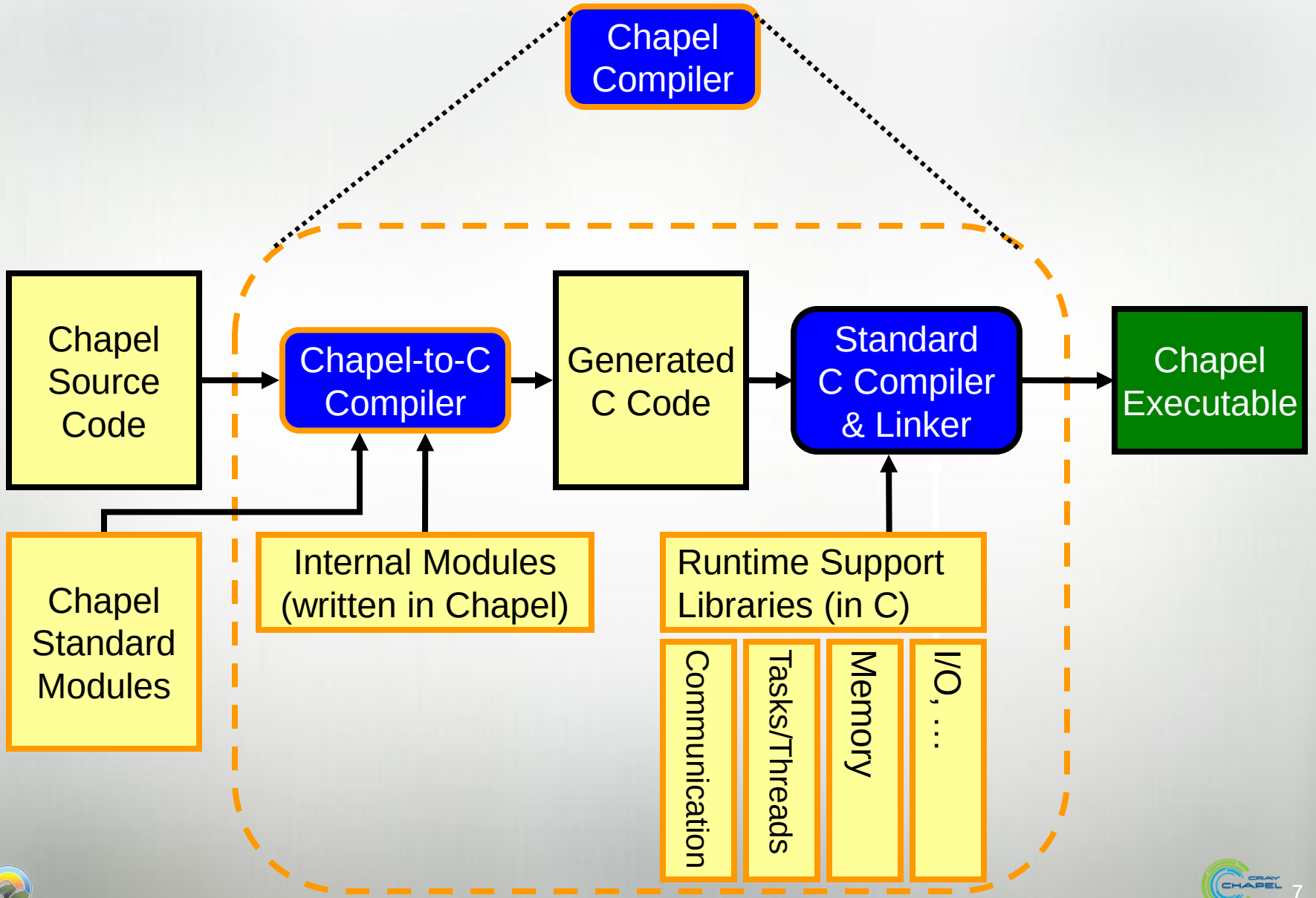
Outline

- ✓ Chapel Context
- ➢ Chapel Background for today's talks
- • Project Information

Compiling Chapel



Chapel Compiler Architecture



Declaring stuff

Declaring procedures:

```
proc foo(x = 0.0, y: real) {
  writeln("In foo, x and y are: ", (x,y));
  return x+y;
}

foo(y=pi);  // uses default value for x
```

Declaring variables, constants, types:

```
var total: real;          // variable
const pi = 3.14;          // run-time constant
param verbose = false;    // compile-time constant
type eltType = real;      // type alias
```

Configuration variables

config declarations: support command-line overrides

```
config var total: real;
config const pi = 3.14;
config param verbose = false;
config type eltType = real;
```

```
# override params and types at compile time
> chpl foo.chpl -sverbose=true -seltType=complex

# override consts and vars at execution time
> ./a.out --total=100.0 --pi=3.1415926
```

Task Parallelism Concepts

begin:

```
begin foo();    // create a task to run foo
bar();          // original task continues on
```

cobegin:

```
cobegin {
  foo();    // one task runs foo()
  bar();    // one task runs bar()
}           // join on both tasks here
```

coforall:

```
coforall tid in 1..numTasks {
  foo();    // each iteration is a foo() task
}           // join on all iteration tasks here
```

Synchronizing

sync variables: store full/empty state with value

- useful for producer/consumer synchronization

```
var buffer$: sync int;

begin buffer$ = 1; // block til empty, leave full
begin x = buffer$; // block til full, leave empty
```

other forms of synchronization:

- single-assignment variables
- atomic variables
- sync statements (join on all dynamically contained tasks)

The Locale Type

Definition:

- Abstract unit of target architecture
- Supports reasoning about locality
- Capable of running tasks and storing variables
  - i.e., has processors and memory

Typically: A compute node (multi-core processor or SMP node)

Defining Locales

- Specify # of locales when running Chapel programs

```
% a.out --numLocales=8
```

```
% a.out -nl 8
```

- Chapel provides built-in locale variables

```
config const numLocales: int = …;
const Locales: [0..#numLocales] locale = …;
```

Locale Operations

- Locale methods support queries about target system:

```
proc locale.physicalMemory(…) { … }
proc locale.numCores { … }
proc locale.id { … }
proc locale.name { … }
```

- *On-clauses* support placement of computations:

```
writeln("on locale 0");

on Locales[1] do
  writeln("now on locale 1");

writeln("on locale 0 again");
```

```
cobegin {
  on A[i,j] do
    bigComputation(A);

  on node.left do
    search(node.left);
}
```

Data Parallelism

```
const D = [1..n] dmapped Cyclic(startIdx=1);
var A, B, C: [D] real;
forall (a,b,c) in (A,B,C) do
  a = b + alpha * c;
```

High-level features implemented…
- in Chapel
- using lower-level features
- by end-users

```
var buffer$: [0..numElts] sync real;
cobegin {
  on Locales[1] do producer(buffer$);
  on A[i]       do consumer(buffer$);
}
```

Chapel language concepts

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

Target Machine

Outline

- ✓ Chapel Context
- ✓ Chapel Background for today's talks
- ➢ Project Information

The Cray Chapel Team (Summer 2012)

Chapel Community (see chapel.cray.com/collaborations.html for further details)

- **Lightweight Tasking using Qthreads:** Sandia (Kyle Wheeler, Dylan Stark, Rich Murphy)
  - **paper at CUG, May 2011**
- **Lightweight Tasking using MassiveThreads:** U Tokyo (Kenjiro Taura, Jun Nakashima)
- **Interoperability via Babel/BRAID:** LLNL/Rice (Tom Epperly, Adrian Prantl, Shams Imam)
  - **paper at PGAS, Oct 2011**
- **Parallel File I/O, Bulk-Copy Opt:** U Malaga (Rafael Asenjo, Maria Angeles Navarro, et al.)
  - **papers at ParCo, Aug 2011; SBAC-PAD, Oct 2012**
- **I/O, LLVM back-end, etc.:** LTS (Michael Ferguson, Matthew Lentz, Joe Yan, et al.)
- **Application Studies:** LLNL (Rob Neely, Bert Still, Jeff Keasler)
- **Interfaces/Generics/OOP:** CU Boulder (Jeremy Siek, Jonathan Turner, et al.)
- **Futures/Task-based Parallelism:** Rice (Vivek Sarkar, Shams Imam, Sagnak Tasirlar, et al.)
- **CPU-accelerator Computing:** UIUC (David Padua, Albert Sidelnik, Maria Garzarán)
  - **paper at IPDPS, May 2012**
- **Model Checking and Verification:** U Delaware (Stephen Siegel, T. Zirkel, T. McClory)
- **Chapel-MPI Compatibility:** Argonne (Pavan Balaji, Rajeev Thakur, Rusty Lusk, Jim Dinan)

“I Like Chapel, how can I help?”

- **Let people know that you like it and why**
  - your colleagues
  - your employer/institution
  - Cray leadership (e.g., mention it at the Cray booth this week)

- **Help us evolve it from prototype to production**
  - contribute back to the source base
  - collaborate with us
  - help fund the effort
  - help us transition from “How will Cray make Chapel succeed?” to “How can we as a community make Chapel succeed?”

Resources For After Today

Chapel project page: http://chapel.cray.com

- papers, presentations, tutorials, language spec, …

Chapel SourceForge page: https://sourceforge.net/projects/chapel/

- release downloads, code repository, public mailing lists, …

IEEE TCSC Blog Series:

- *Myths About Scalable Parallel Programming Languages*

Mailing Lists:

- chapel_info@cray.com: contact the team
- chapel-users@lists.sourceforge.net: user-oriented discussion list
- chapel-developers@lists.sourceforge.net: dev.-oriented discussion
- chapel-education@lists.sourceforge.net: educator-oriented discussion
- chapel-bugs@lists.sourceforge.net/chapel_bugs@cray.com : public/private bug forum