Chapel: Status/Community

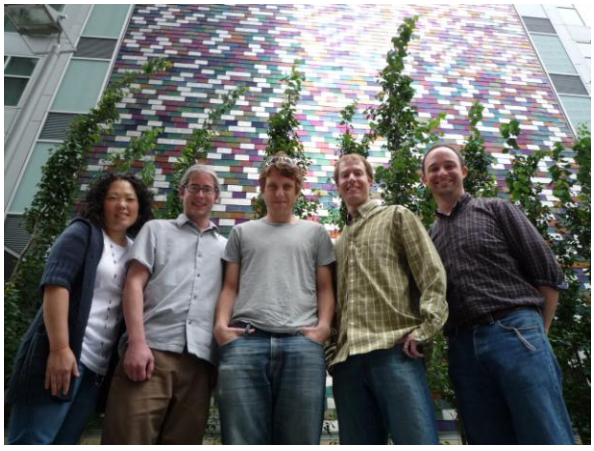
Brad Chamberlain
Cray Inc.

CSEP 524
May 20, 2010

Outline

- ✓ Chapel Context
- ✓ Global-view Programming Models
- ✓ Language Overview
- ❑ Status, Collaborations, Future Work

The Chapel Team

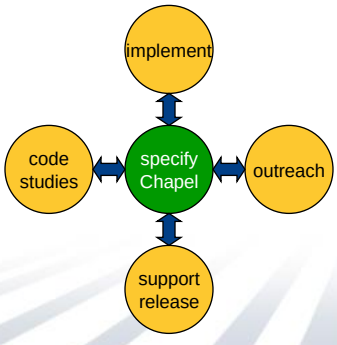


Sung-Eun Choi, David Iten, Lee Prokowich,
Steve Deitz, Brad Chamberlain,
and half of Greg Titus

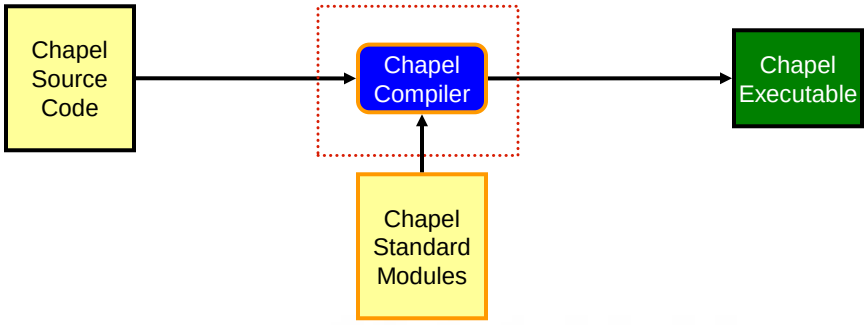
- Interns
  - Hannah Hemmaplardh (`10–UW)
  - Jonathan Turner (`10 – Boulder)
  - Jacob Nelson (`09 – UW)
  - Albert Sidelnik (`09 – UIUC)
  - Andy Stone (`08 – Colorado St)
  - James Dinan (`07 – Ohio State)
  - Robert Bocchino (`06 – UIUC)
  - Mackale Joyner (`05 – Rice)
- Alumni
  - David Callahan
  - Roxana Diaconescu
  - Samuel Figueroa
  - Shannon Hoffswell
  - Mary Beth Hribar
  - Mark James
  - John Plevyak
  - Wayne Wong
  - Hans Zima

Chapel Work

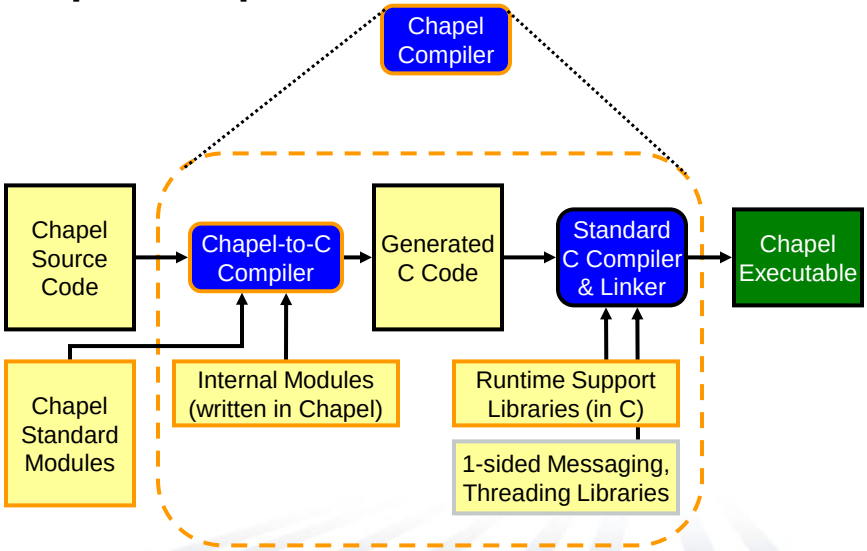
- Chapel Team's Focus:
  - specify Chapel syntax and semantics
  - implement open-source prototype compiler for Chapel
  - perform code studies of benchmarks, apps, and libraries in Chapel
  - do community outreach to inform and learn from users/researchers
  - support users of code releases
  - refine language based on all these activities



Compiling Chapel



Chapel Compiler Architecture



Chapel and the Community

- Our philosophy:
  - help the parallel community understand what we are doing
  - develop Chapel as an open-source project
  - encourage external collaborations
  - over time, turn language over to the community
- Goals:
  - to get feedback that will help make the language more useful
  - to support collaborative research efforts
  - to accelerate the implementation
  - to aid with adoption

Chapel Release

- **Current release:** version 1.1 (April 15th, 2010)
- Supported environments: UNIX/Linux, Mac OS X, Cygwin
- How to get started:
  1. Download from: http://sourceforge.net/projects/chapel
  2. Unpack tar.gz file
  3. See top-level README
     - for quick-start instructions
     - for pointers to next steps with the release
- Your feedback desired!
- Remember: a work-in-progress
  - ⇒ it's likely that you will find problems with the implementation
  - ⇒ this is still a good time to influence the language's design

Implementation Status (v1.1)

- **Base language:** stable (some gaps and bugs remain)
- **Task parallel:**
  - stable multi-threaded implementation of tasks, sync variables
  - atomic sections are an area of ongoing research with U. Notre Dame
- **Data parallel:**
  - stable multi-threaded data parallelism for dense domains/arrays
  - other domain types have a single-threaded reference implementation
- **Locality:**
  - stable locale types and arrays
  - stable task parallelism across multiple locales
  - initial support for some distributions: Block, Cyclic, Block-Cyclic
- **Performance:**
  - has received much attention in designing the language
  - yet minimal implementation effort to date

Selected Collaborations (see chapel.cray.com for complete list)

Notre Dame/ORNL (Peter Kogge, Srinivas Sridharan, Jeff Vetter):
Asynchronous Software Transactional Memory over distributed memory

UIUC (David Padua, Albert Sidelnik):
Chapel for hybrid CPU-GPU computing

BSC/UPC (Alex Duran):
Chapel over Nanos++ user-level tasking

U/Malaga (Rafa Asenjo, Maria Gonzales, Rafael Larossa):
Parallel file I/O for whole-array reads/writes

University of Colorado, Boulder (Jeremy Siek, Jonathan Turner):
Concepts/interfaces for improved support for generic programming

PNNL/CASS-MT (John Feo, Daniel Chavarria):
Hybrid computing in Chapel; performance tuning for the Cray XMT; ARMCI port

ORNL (David Bernholdt *et al.*; Steve Poole *et al.*):
Chapel code studies – Fock matrices, MADNESS, Sweep3D, coupled models, …

U Oregon, Paratools Inc.:
Chapel performance analysis using Tau

(Your name here?)

Collaboration Opportunities (see chapel.cray.com for more details)

- memory management policies/mechanisms
- dynamic load balancing: task throttling and stealing
- parallel I/O and checkpointing
- exceptions; resiliency
- language interoperability
- application studies and performance optimizations
- index/subdomain semantics and optimizations
- targeting different back-ends (LLVM, MS CLR, …)
- runtime compilation
- library support
- tools
  - debuggers, performance analysis, IDEs, interpreters, visualizers
- database-style programming
- (your ideas here…)

Chapel and Education

- If I were to offer a parallel programming class, I'd want to teach about:
  - data parallelism
  - task parallelism
  - concurrency
  - synchronization
  - locality/affinity
  - deadlock, livelock, and other pitfalls
  - performance tuning
  - …
- I don't think there's a good language out there…

  …for teaching *all* of these things

  …for teaching some of these things at all

  …until now: I think Chapel has the potential to play a crucial role here

Our Next Steps

- Expand our set of supported distributions
- Continue to improve performance
- Continue to add missing features
- Expand the set of codes that we are studying
- Expand the set of architectures that we are targeting
- Support the public release
- Continue to support collaborations and seek out new ones
- Continue to expand our team

Summary

Chapel strives to greatly improve Parallel Productivity

via its support for…

…general parallel programming
…global-view abstractions
…control over locality
…multiresolution features
…modern language concepts and themes