CRAY
THE SUPERCOMPUTER COMPANY

Chapel: Project Overview

CRAY CHAPEL

Outline

- Who we are
- What we do
- What's next?

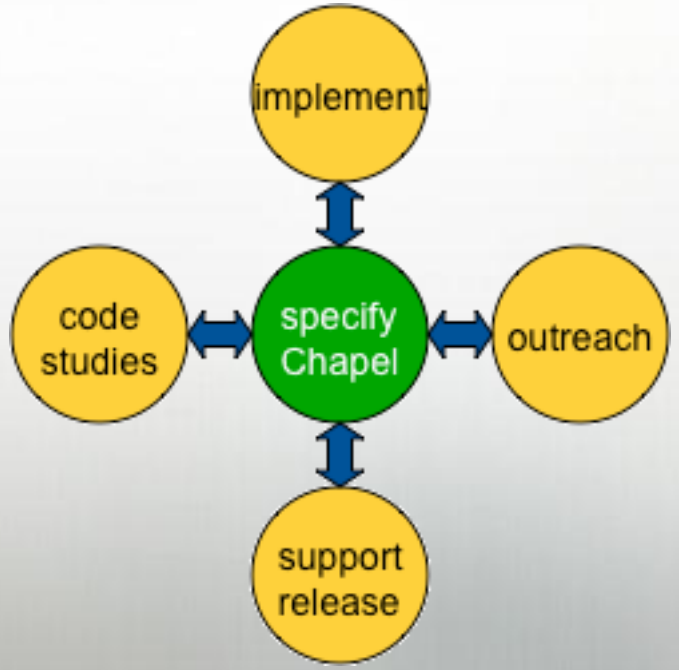
The Cray Chapel Team (Summer 2012)

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**
- **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.)
- **Interoperability via Babel/BRAID:** LLNL/Rice (Tom Epperly, Adrian Prantl, Shams Imam)
  - **paper at PGAS, Oct 2011**
- **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.)
- **Lightweight Tasking using MassiveThreads:** U Tokyo (Kenjiro Taura, Jun Nakashima)
- **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)

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 collaborators and users of code releases
  - refine the language based on all these activities



Implementation Status -- Version 1.7.0

In a nutshell:

- Most features work at a functional level
- Many performance optimizations remain

This is a good time to:

- Try out the language and compiler
- Give us feedback to improve Chapel
- Use Chapel for parallel programming education
- Use Chapel for non-performance-critical projects

In evaluating the language:

- Try to judge it by how it should *ultimately* perform rather than how it does today
  - lots of low-hanging fruit remains, as well as some challenges

Chapel and Education

- In teaching parallel programming, I like to cover:
  - data parallelism
  - task parallelism
  - concurrency
  - synchronization
  - locality/affinity
  - deadlock, livelock, and other pitfalls
  - performance tuning
  - (see for example http://www.cs.washington.edu/education/courses/csep524/13wi/)
- I don't think there's a good language out there…
  - for teaching *all* of these things
  - for teaching some of these things well at all
  - ***until now:*** I believe Chapel can potentially fill a crucial gap here

“I Like Chapel, how can I help?”

- **Let people know that you like it and why**
  - your colleagues
  - your employer/institution
  - Cray leadership
- **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?”

Next Steps

- Continue to improve performance
- Continue to add missing features
- Grow the set of codes that we are evaluating
- Grow the set of architectures that we can target effectively
- Support Chapel users and developers
- Continue to support collaborations and seek out new ones
- Evolve the Chapel project for the post-HPCS timeframe
  - e.g., begin transitioning governance to an external group

Questions?

- What we do
- Who we are
- What's next?