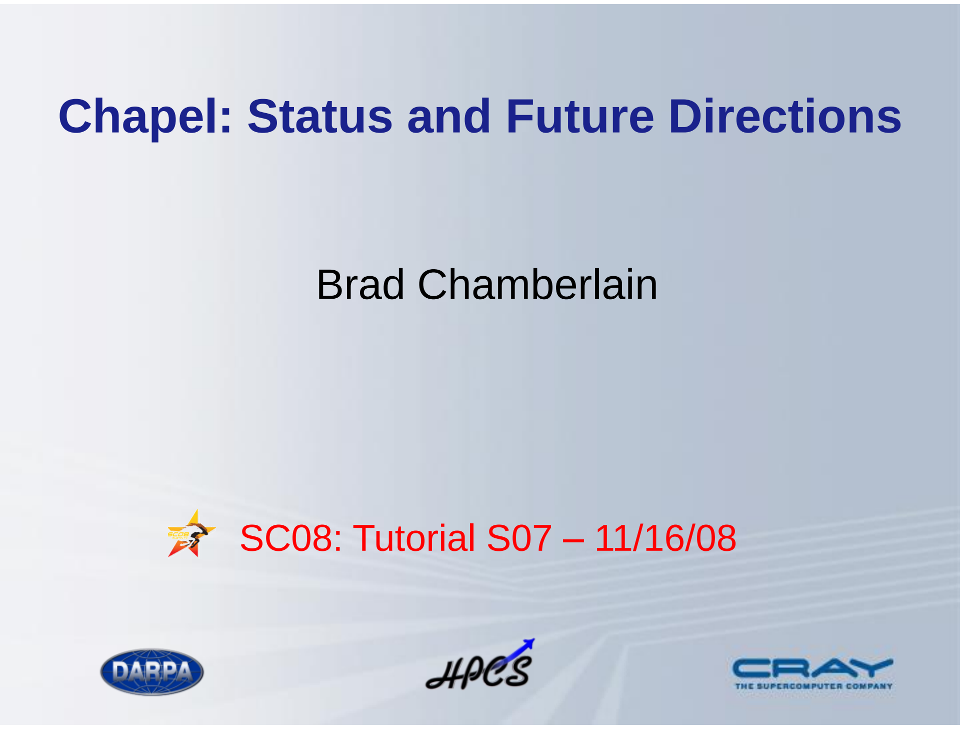
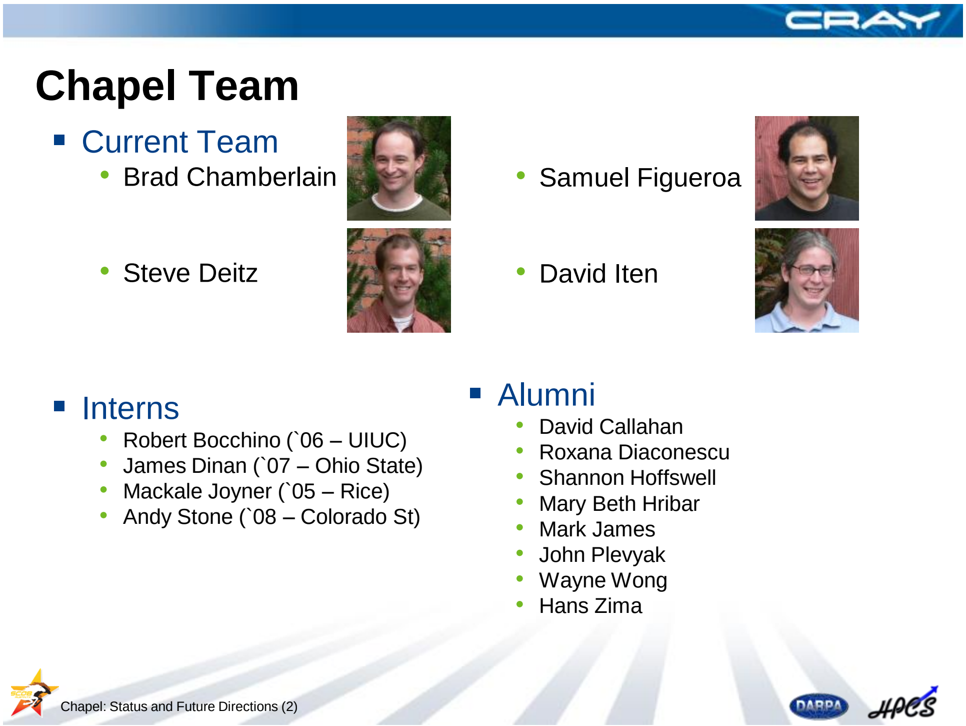


Chapel: Status and Future Directions

Brad Chamberlain

SC08: Tutorial S07 – 11/16/08

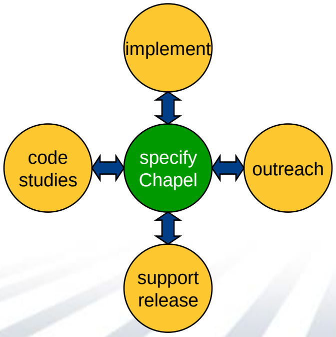


Chapel Team

- Current Team
  - Brad Chamberlain
  - Samuel Figueroa
  - Steve Deitz
  - David Iten
- Interns
  - Robert Bocchino (`06 – UIUC)
  - James Dinan (`07 – Ohio State)
  - Mackale Joyner (`05 – Rice)
  - Andy Stone (`08 – Colorado St)
- Alumni
  - David Callahan
  - Roxana Diaconescu
  - 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



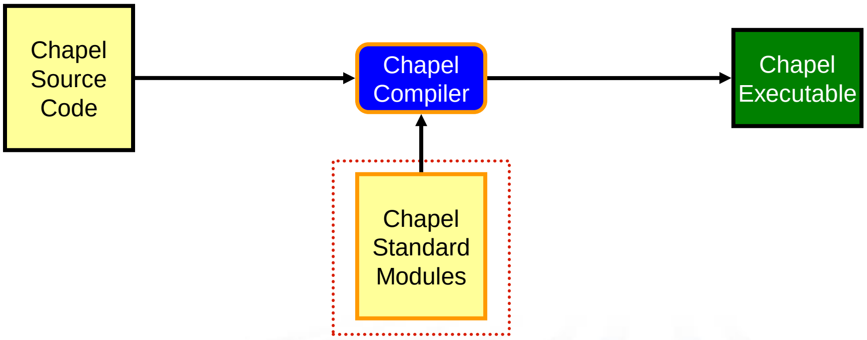
Outline

- Who we are and what we do
- Chapel prototype compiler
  - compiler architecture
  - implementation status
- Chapel and the broader community
- Wrap-up

Prototype Compiler Development

- Development Strategy:
  - start by developing and nurturing within Cray under HPCS
  - initial releases to small sets of “friendly” users for past few years
    - ~45 users at ~30 sites (academic, government, industry)
  - public release scheduled for SC08 timeframe
  - turn over to community when it’s ready to stand on its own
- Compilation approach:
  - source-to-source compiler for portability (Chapel-to-C)
  - link against runtime libraries to hide machine details
    - threading layer currently implemented using pthreads
    - communication currently implemented using Berkeley’s GASNet

Compiling Chapel



Chapel Source Code

Chapel Compiler

Chapel Executable

Chapel Standard Modules

Chapel Standard Modules

Standard Modules: implement standard library support

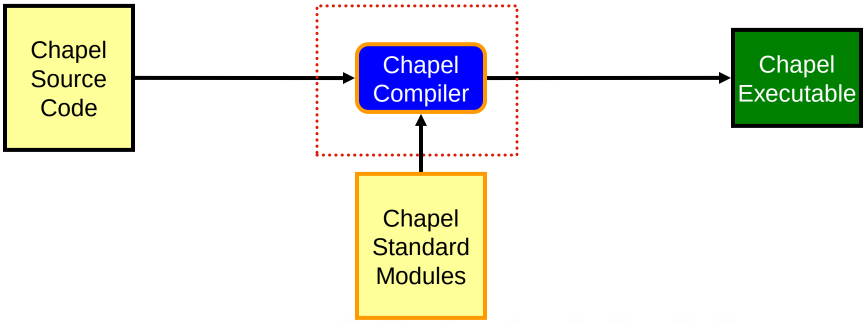
- explicitly imported by user code:

```
use Random;
use Time;
```

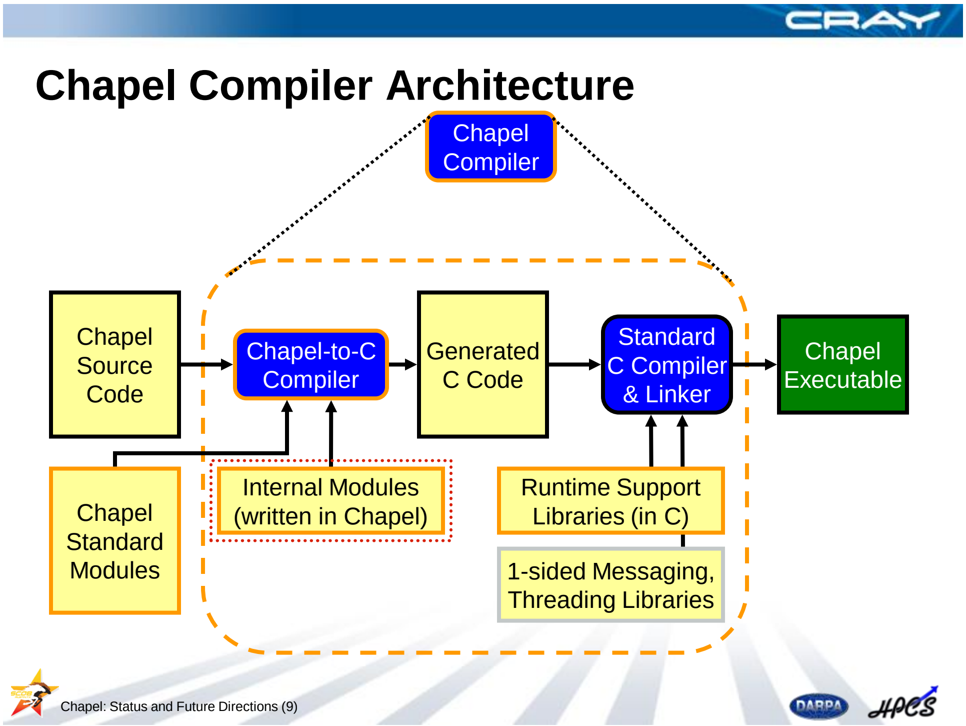
- current release contains rough sketch of anticipated support:
  - machine resource queries
  - timer and time-of-day support
  - random number generators
  - advanced bit operations
  - more to come…

Chapel: Status and Future Directions (7)

Compiling Chapel

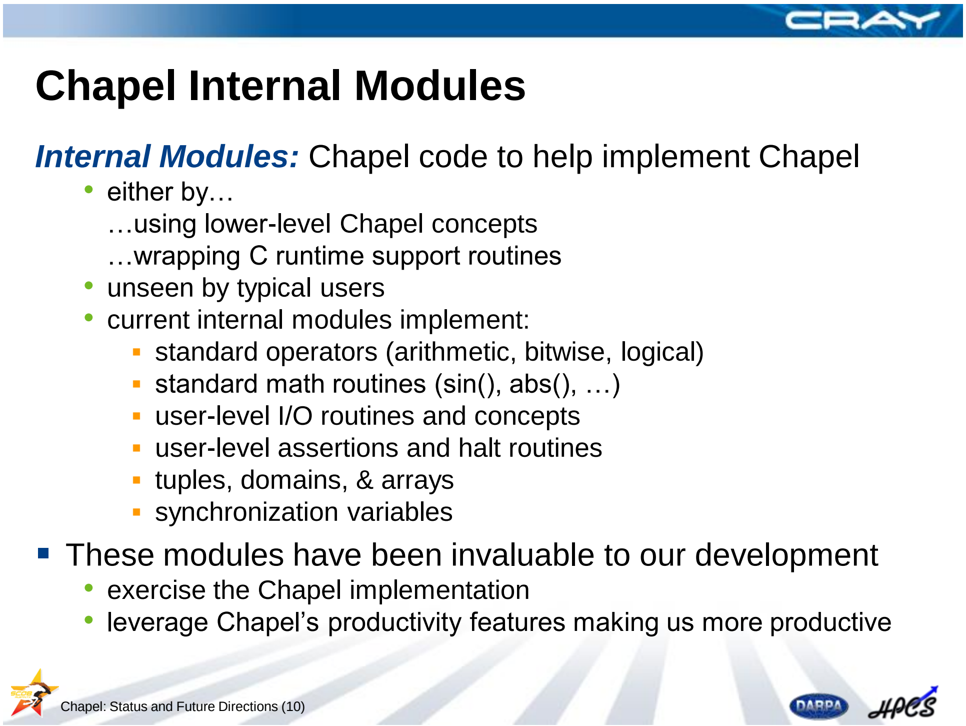


Chapel: Status and Future Directions (8)



Chapel Compiler Architecture

Chapel Compiler
Chapel Source Code
Chapel-to-C Compiler
Generated C Code
Standard C Compiler & Linker
Chapel Executable
Chapel Standard Modules
Internal Modules (written in Chapel)
Runtime Support Libraries (in C)
1-sided Messaging, Threading Libraries



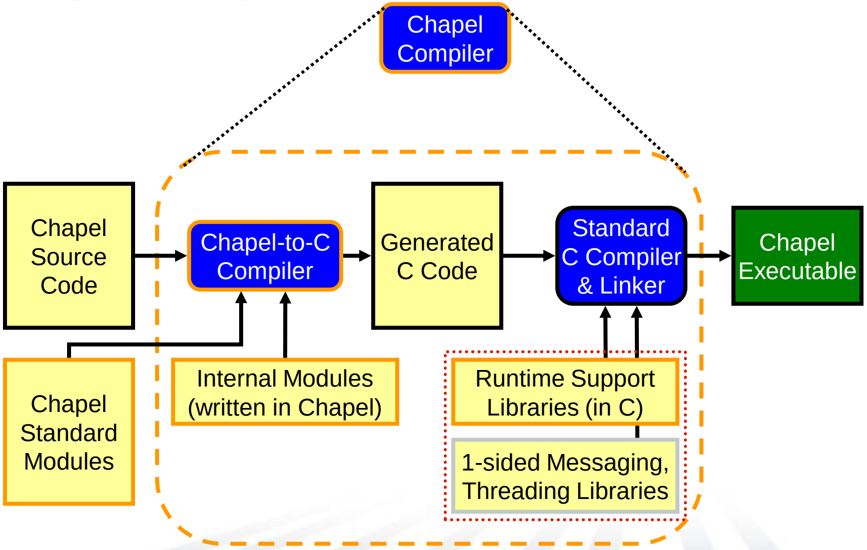
Chapel Internal Modules

Internal Modules: Chapel code to help implement Chapel

- either by…
  …using lower-level Chapel concepts
  …wrapping C runtime support routines
- unseen by typical users
- current internal modules implement:
  - standard operators (arithmetic, bitwise, logical)
  - standard math routines (sin(), abs(), …)
  - user-level I/O routines and concepts
  - user-level assertions and halt routines
  - tuples, domains, & arrays
  - synchronization variables

- These modules have been invaluable to our development
  - exercise the Chapel implementation
  - leverage Chapel’s productivity features making us more productive

Chapel Compiler Architecture

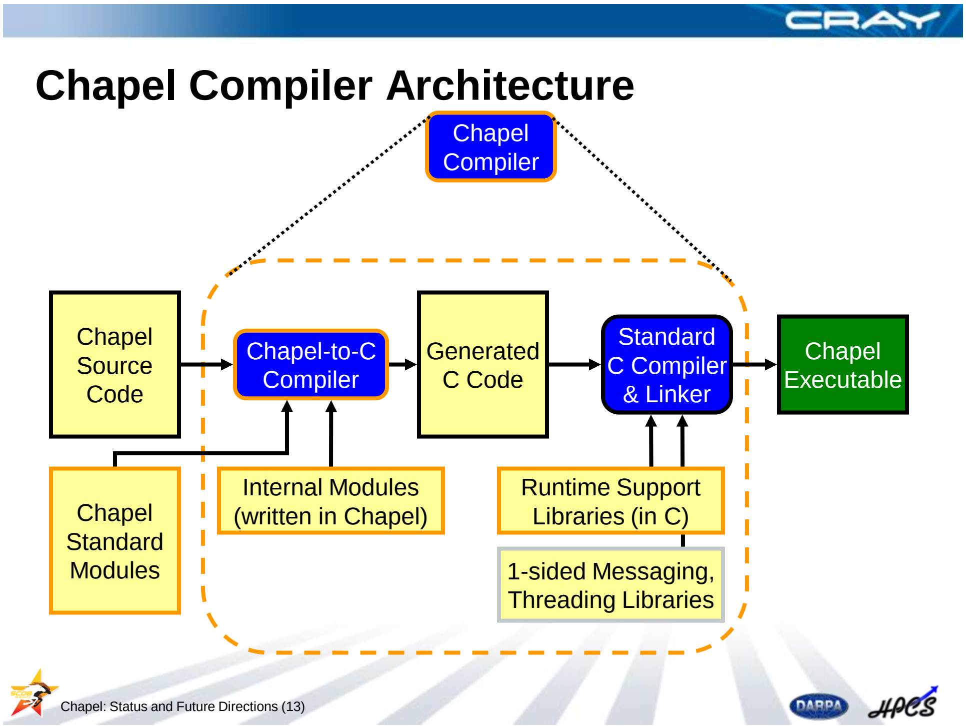

Chapel Compiler
Chapel Source Code
Chapel-to-C Compiler
Generated C Code
Standard C Compiler & Linker
Chapel Executable
Chapel Standard Modules
Internal Modules (written in Chapel)
Runtime Support Libraries (in C)
1-sided Messaging, Threading Libraries

Chapel Runtime Support Libraries

Runtime Support Libraries: C code to help implement Chapel

- for features that are too low-level to implement in Chapel
- can be thought of as helping bootstrap the language
- current support libraries implement:
  - command-line argument parsing
  - console and file I/O primitives
  - error handling
  - memory management and tracking
  - timing/time-of-day primitives
  - type conversions
  - thread creation and management
  - inter-process communication and coordination

- As Chapel matures, functionality tends to migrate from the C runtime support libraries to the Chapel internal modules



Chapel Compiler Architecture

Chapel Compiler

Chapel Source Code

Chapel-to-C Compiler

Generated C Code

Standard C Compiler & Linker

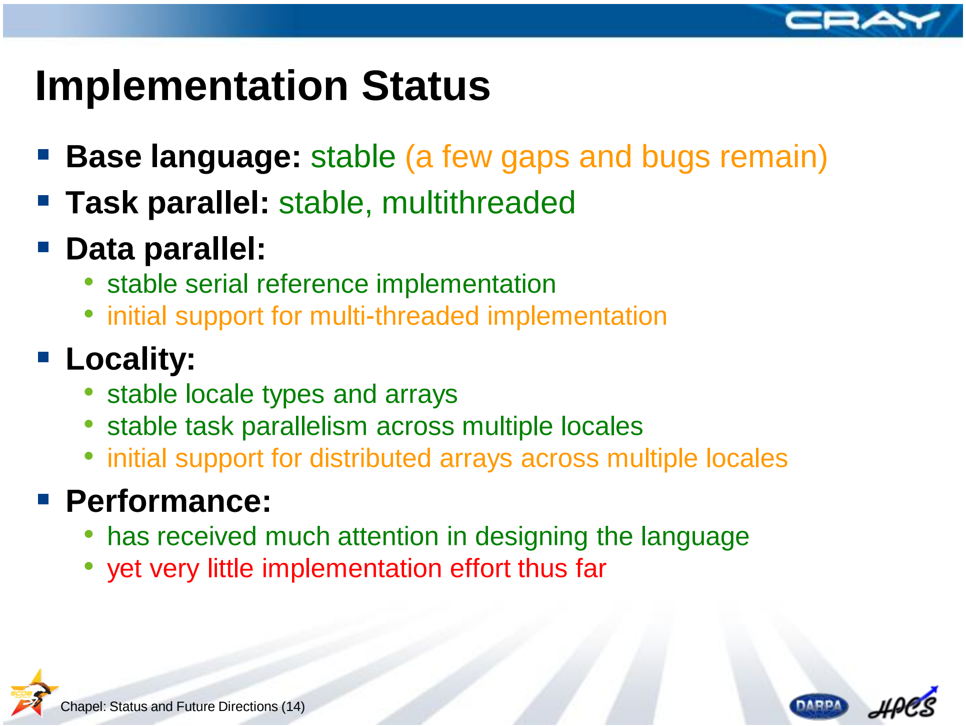
Chapel Executable

Chapel Standard Modules

Internal Modules (written in Chapel)

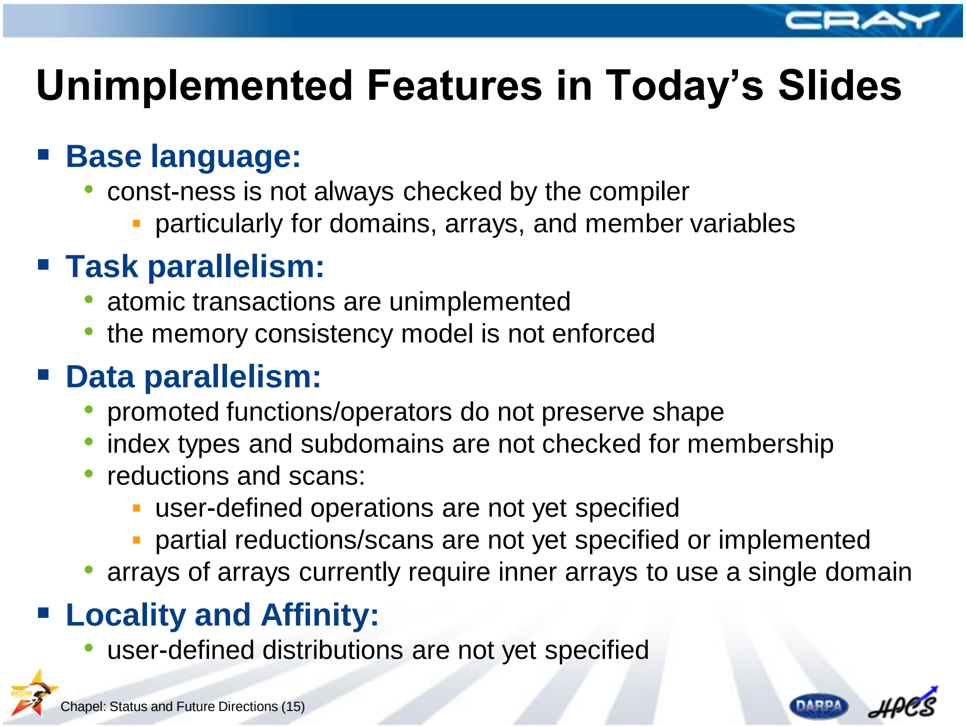
Runtime Support Libraries (in C)

1-sided Messaging, Threading Libraries



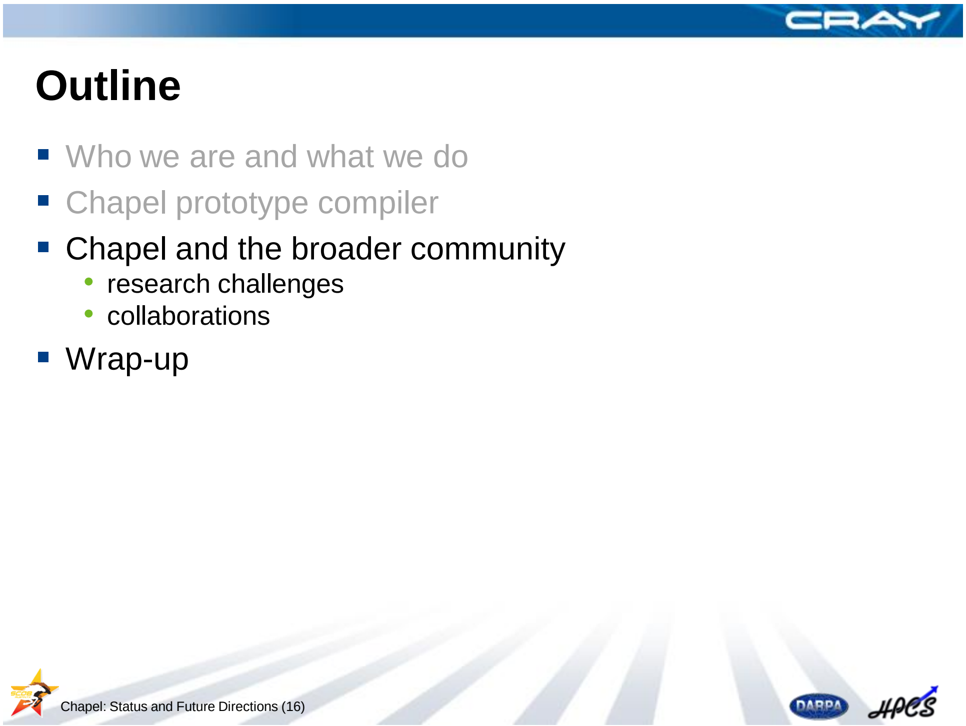
Implementation Status

- **Base language:** stable (a few gaps and bugs remain)
- **Task parallel:** stable, multithreaded
- **Data parallel:**
  - stable serial reference implementation
  - initial support for multi-threaded implementation
- **Locality:**
  - stable locale types and arrays
  - stable task parallelism across multiple locales
  - initial support for distributed arrays across multiple locales
- **Performance:**
  - has received much attention in designing the language
  - yet very little implementation effort thus far



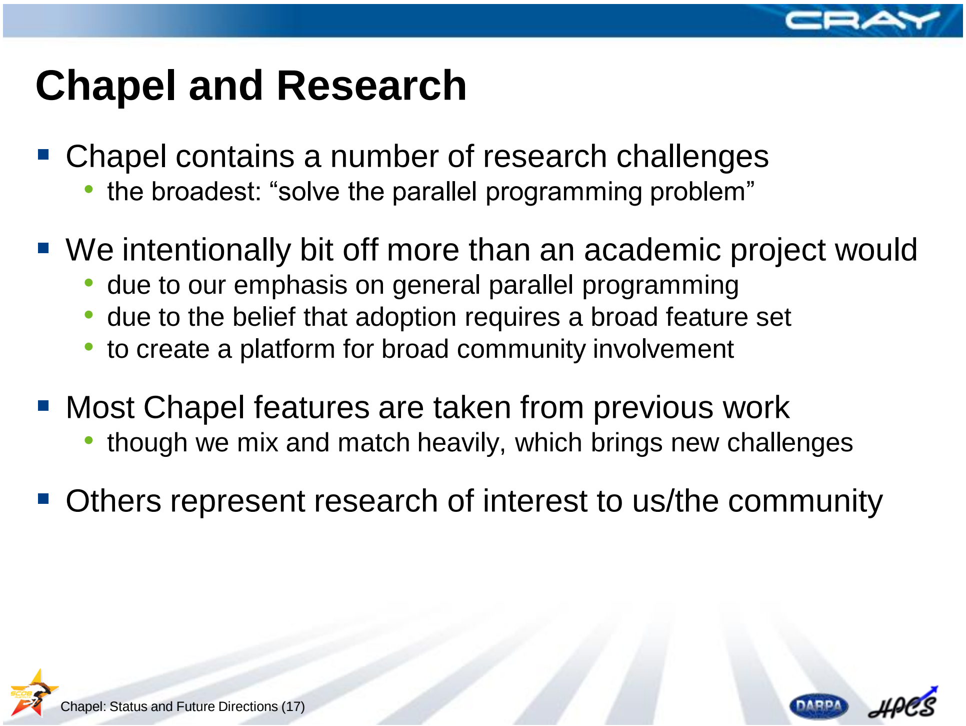
Unimplemented Features in Today’s Slides

- **Base language:**
  - const-ness is not always checked by the compiler
    - particularly for domains, arrays, and member variables
- **Task parallelism:**
  - atomic transactions are unimplemented
  - the memory consistency model is not enforced
- **Data parallelism:**
  - promoted functions/operators do not preserve shape
  - index types and subdomains are not checked for membership
  - reductions and scans:
    - user-defined operations are not yet specified
    - partial reductions/scans are not yet specified or implemented
  - arrays of arrays currently require inner arrays to use a single domain
- **Locality and Affinity:**
  - user-defined distributions are not yet specified



Outline

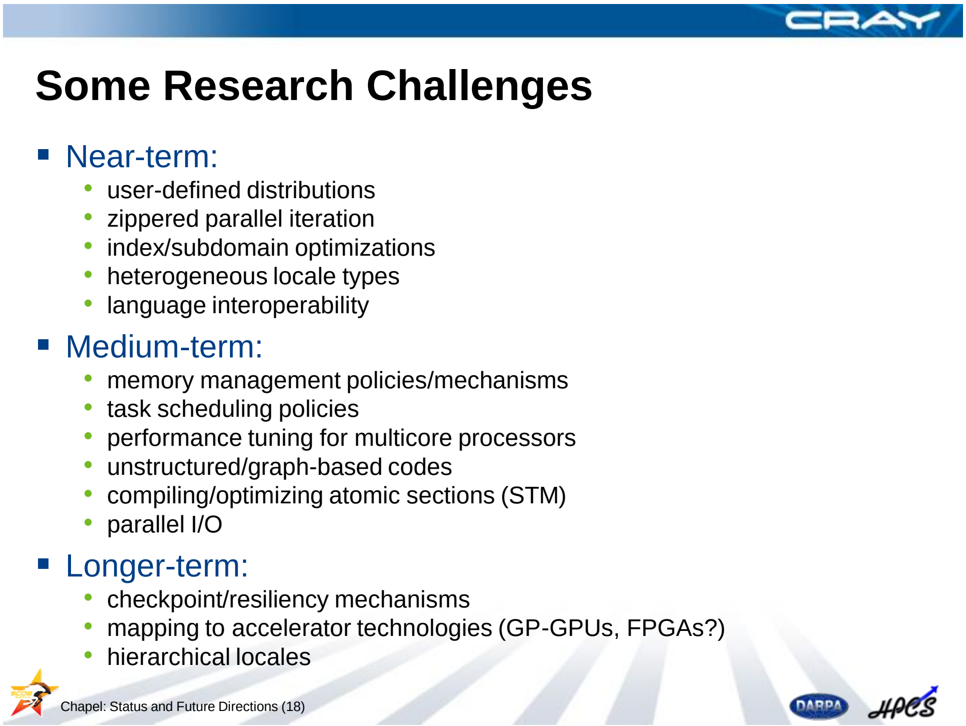
- Who we are and what we do
- Chapel prototype compiler
- Chapel and the broader community
  - research challenges
  - collaborations
- Wrap-up



Chapel and Research

- Chapel contains a number of research challenges
  - the broadest: “solve the parallel programming problem”
- We intentionally bit off more than an academic project would
  - due to our emphasis on general parallel programming
  - due to the belief that adoption requires a broad feature set
  - to create a platform for broad community involvement
- Most Chapel features are taken from previous work
  - though we mix and match heavily, which brings new challenges
- Others represent research of interest to us/the community

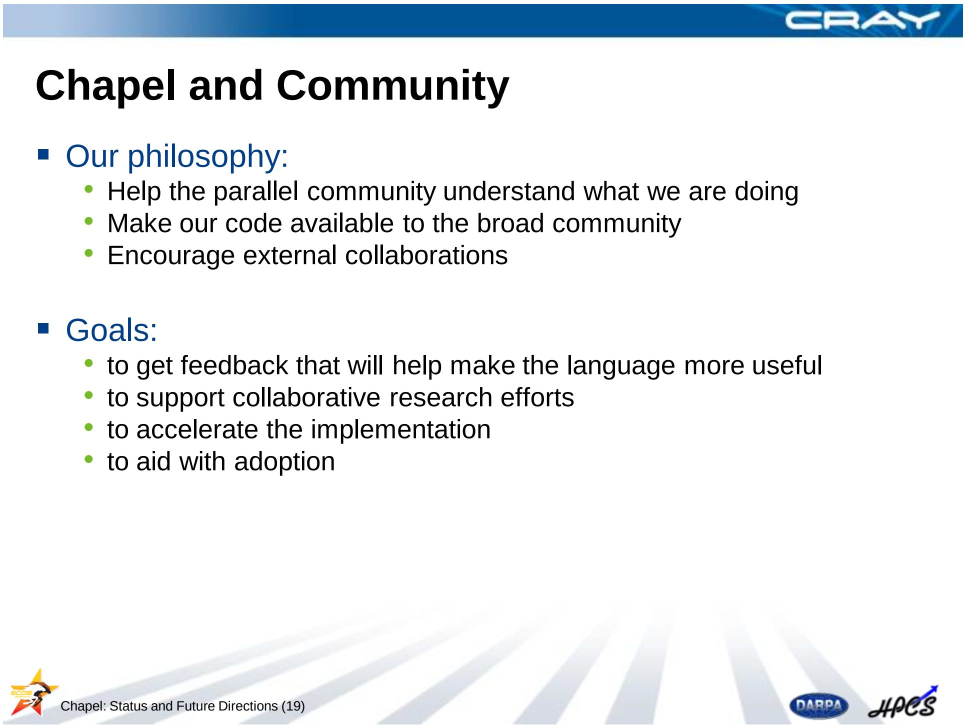
Chapel: Status and Future Directions (17)



Some Research Challenges

- Near-term:
  - user-defined distributions
  - zippered parallel iteration
  - index/subdomain optimizations
  - heterogeneous locale types
  - language interoperability
- Medium-term:
  - memory management policies/mechanisms
  - task scheduling policies
  - performance tuning for multicore processors
  - unstructured/graph-based codes
  - compiling/optimizing atomic sections (STM)
  - parallel I/O
- Longer-term:
  - checkpoint/resiliency mechanisms
  - mapping to accelerator technologies (GP-GPUs, FPGAs?)
  - hierarchical locales

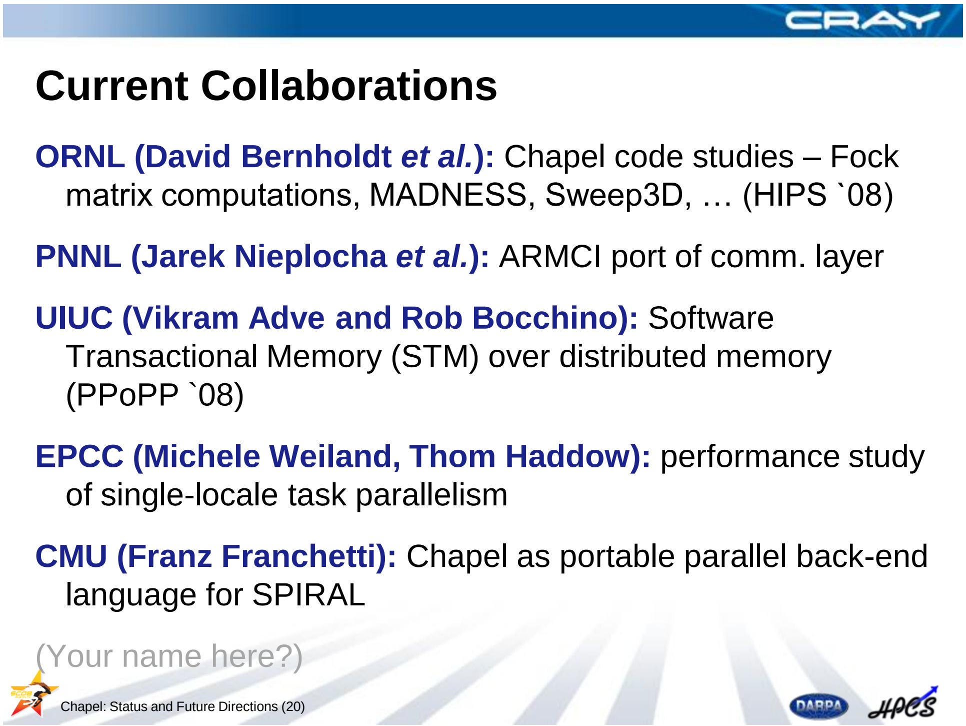
Chapel: Status and Future Directions (18)



Chapel and Community

- Our philosophy:
  - Help the parallel community understand what we are doing
  - Make our code available to the broad community
  - Encourage external collaborations

- 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



Current Collaborations

ORNL (David Bernholdt *et al.*): Chapel code studies – Fock matrix computations, MADNESS, Sweep3D, … (HIPS `08)

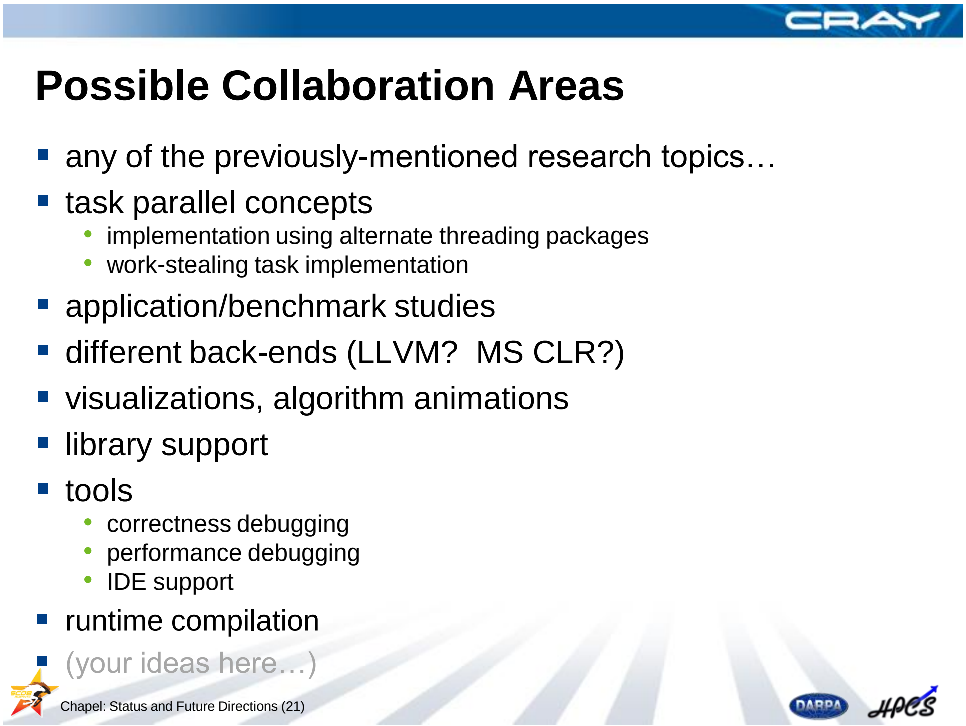
PNNL (Jarek Nieplocha *et al.*): ARMCI port of comm. layer

UIUC (Vikram Adve and Rob Bocchino): Software Transactional Memory (STM) over distributed memory (PPoPP `08)

EPCC (Michele Weiland, Thom Haddow): performance study of single-locale task parallelism

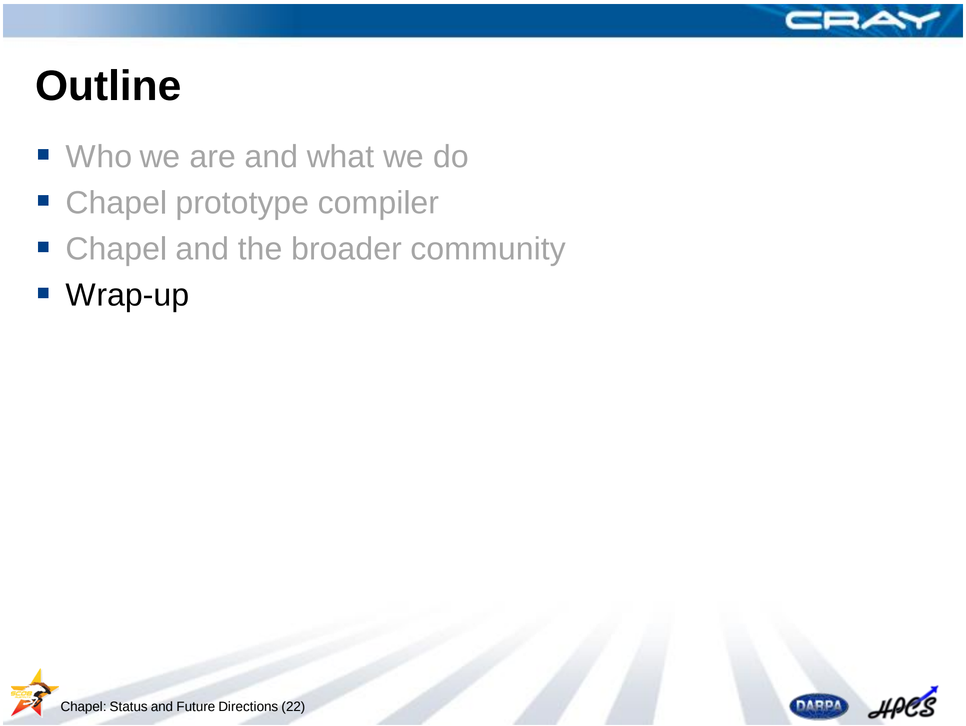
CMU (Franz Franchetti): Chapel as portable parallel back-end language for SPIRAL

(Your name here?)



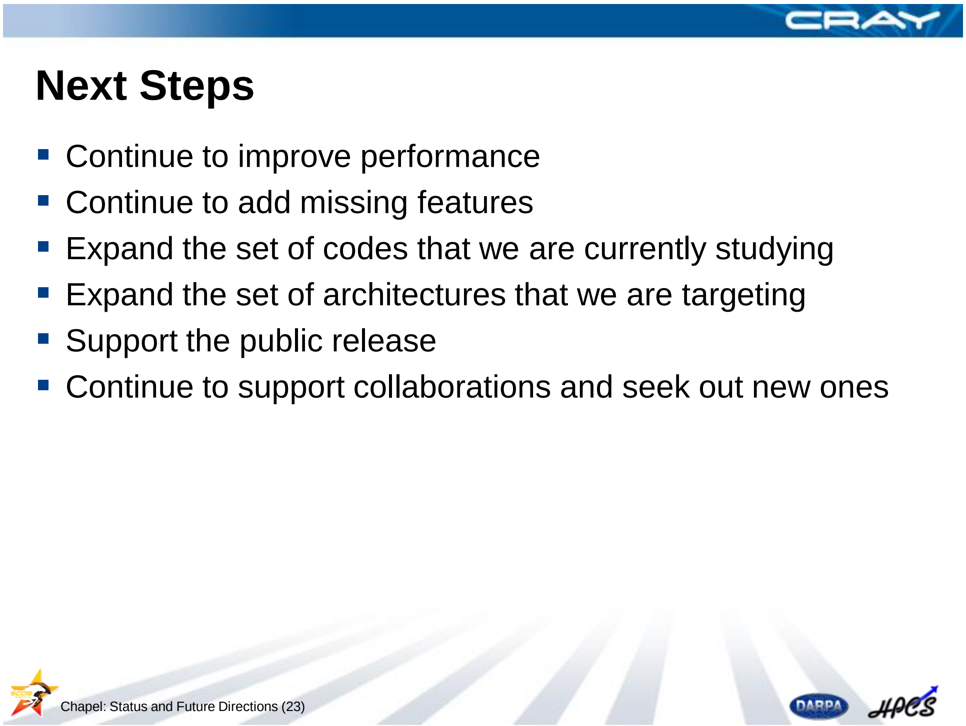
Possible Collaboration Areas

- any of the previously-mentioned research topics…
- task parallel concepts
  - implementation using alternate threading packages
  - work-stealing task implementation
- application/benchmark studies
- different back-ends (LLVM? MS CLR?)
- visualizations, algorithm animations
- library support
- tools
  - correctness debugging
  - performance debugging
  - IDE support
- runtime compilation
- (your ideas here…)



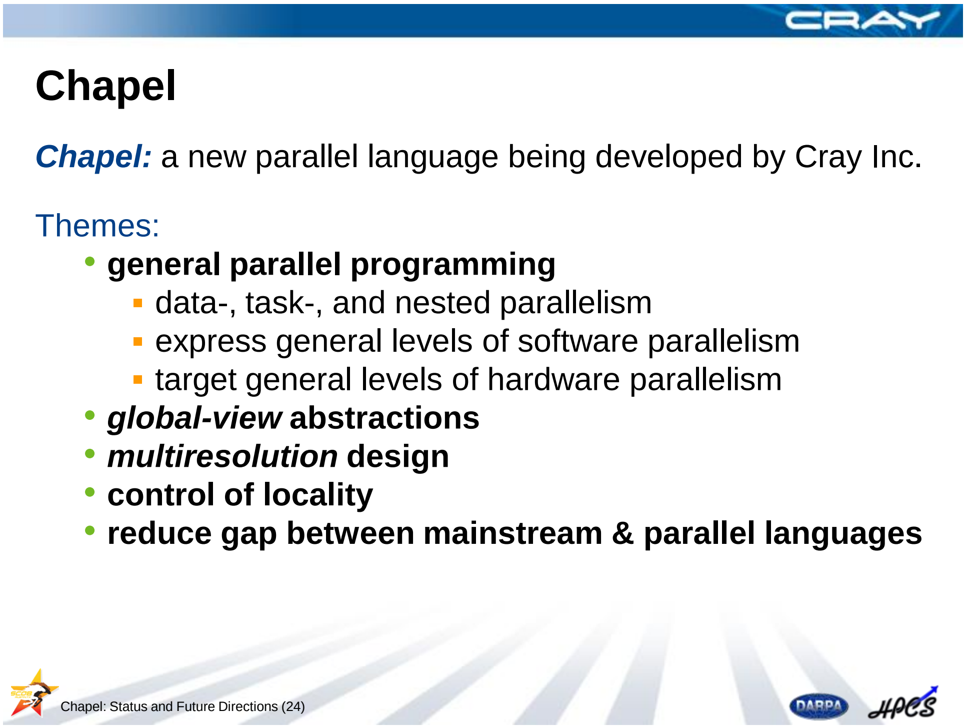
Outline

- Who we are and what we do
- Chapel prototype compiler
- Chapel and the broader community
- Wrap-up



Next Steps

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

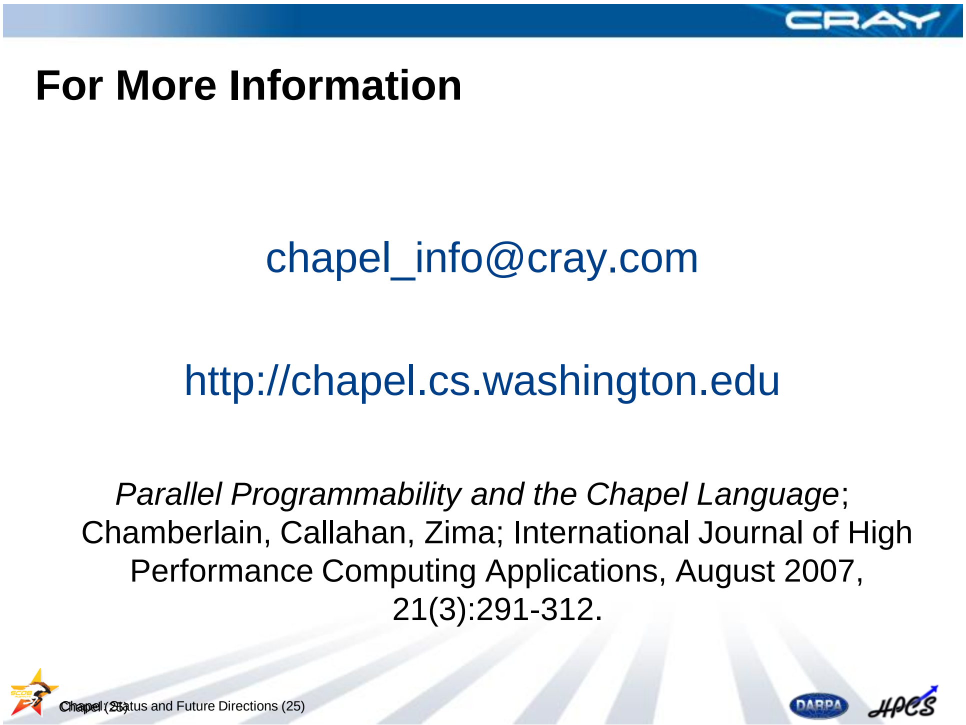


Chapel

Chapel: a new parallel language being developed by Cray Inc.

Themes:

- **general parallel programming**
  - data-, task-, and nested parallelism
  - express general levels of software parallelism
  - target general levels of hardware parallelism
- ***global-view* abstractions**
- ***multiresolution* design**
- **control of locality**
- **reduce gap between mainstream & parallel languages**



For More Information

chapel_info@cray.com

http://chapel.cs.washington.edu

Parallel Programmability and the Chapel Language; Chamberlain, Callahan, Zima; International Journal of High Performance Computing Applications, August 2007, 21(3):291-312.

Chapel: Status and Future Directions (25)

Questions?