The State of the Chapel Union

Martha Dumler
CUG 2013
May 9, 2013

What is Chapel?

- **An emerging parallel programming language**
  - Design and development led by Cray Inc.
    - in collaboration with academia, labs, industry
  - Initiated under the DARPA HPCS program
- **Overall goal: Improve programmer productivity**
  - Improve the programmability of parallel computers
  - Match or beat the performance of current programming models
  - Support better portability than current programming models
  - Improve the robustness of parallel codes
- **A work-in-progress**
  - HPCS has successfully completed; yet Chapel is active & ongoing

Chapel's Implementation

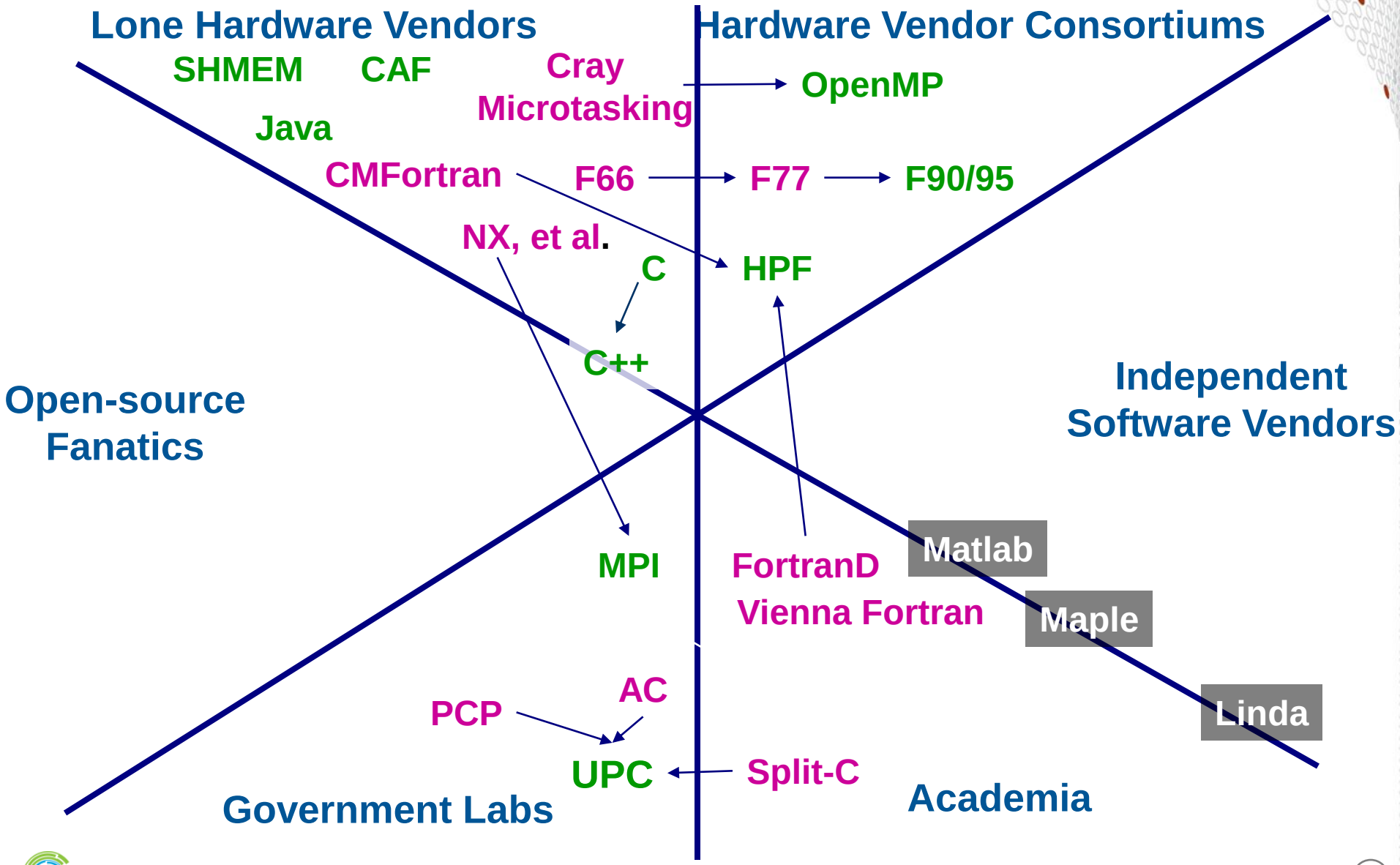
- **Being developed as open source at SourceForge**
- **Licensed as BSD software**
- **Target Architectures:**
  - Cray architectures
  - multicore desktops and laptops
  - commodity clusters
  - systems from other vendors
  - *in-progress:* accelerator-based compute nodes

Past

Chapel Timeline

Jun 2002: HPCS starts

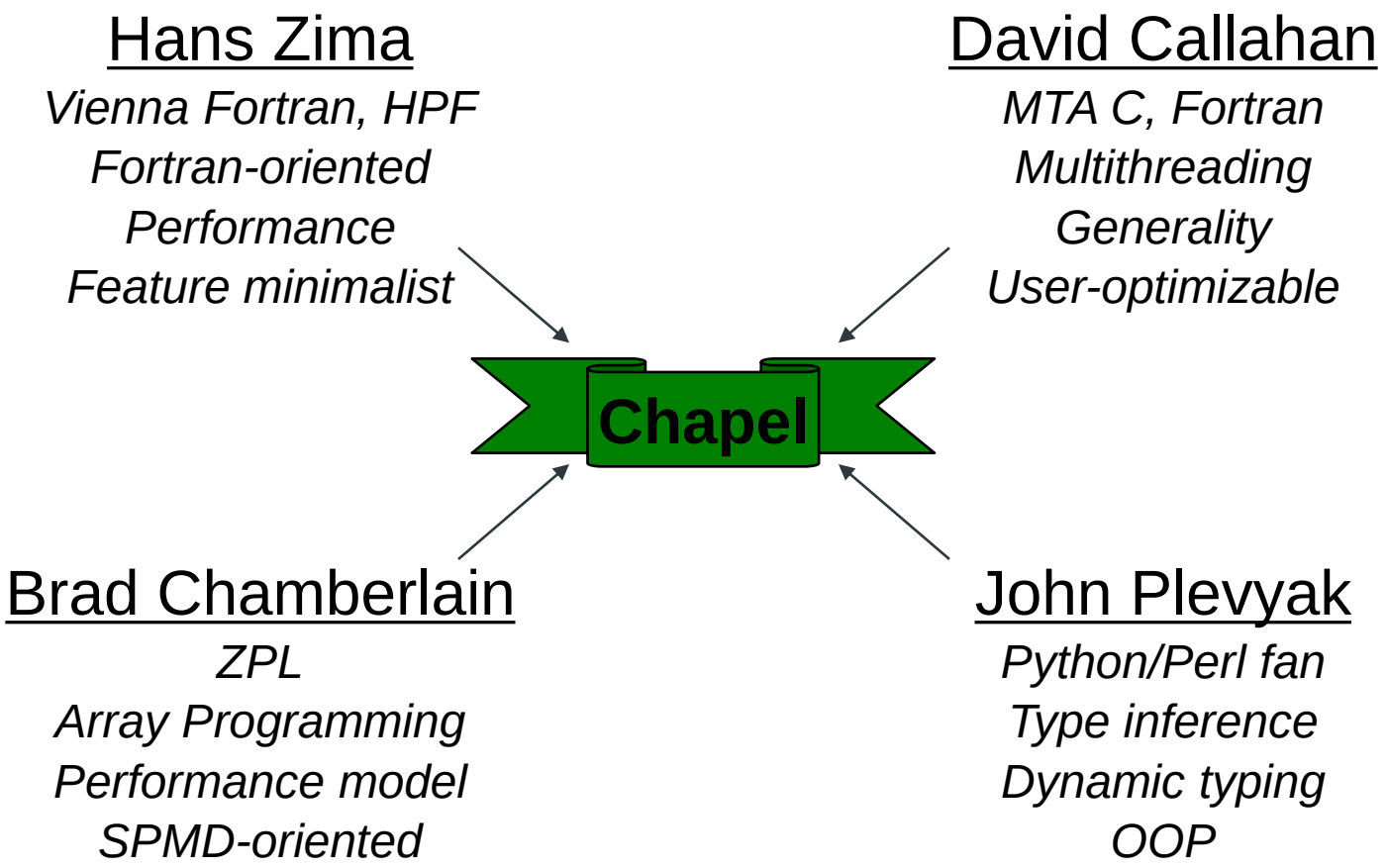
Where do Adopted Languages Come From?



Chapel Timeline

Jun 2002: HPCS starts
Jan 2003: Cray expresses interest in pursuing a new language
Jul 2003: Chapel design and implementation starts

Chapel Influences



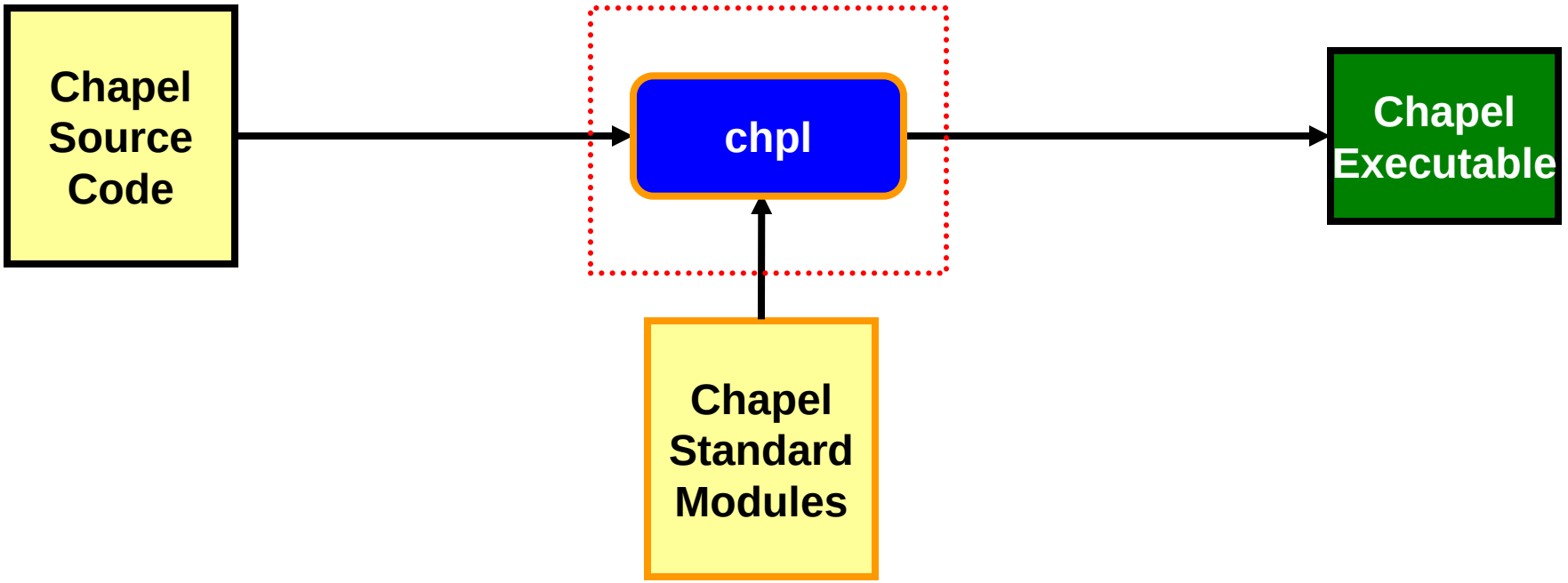
Chapel Timeline

Jun 2002: HPCS starts
Jan 2003: Cray expresses interest in pursuing a new language
Jul 2003: Chapel design and implementation starts
Jun 2006: current language and compiler architecture gel

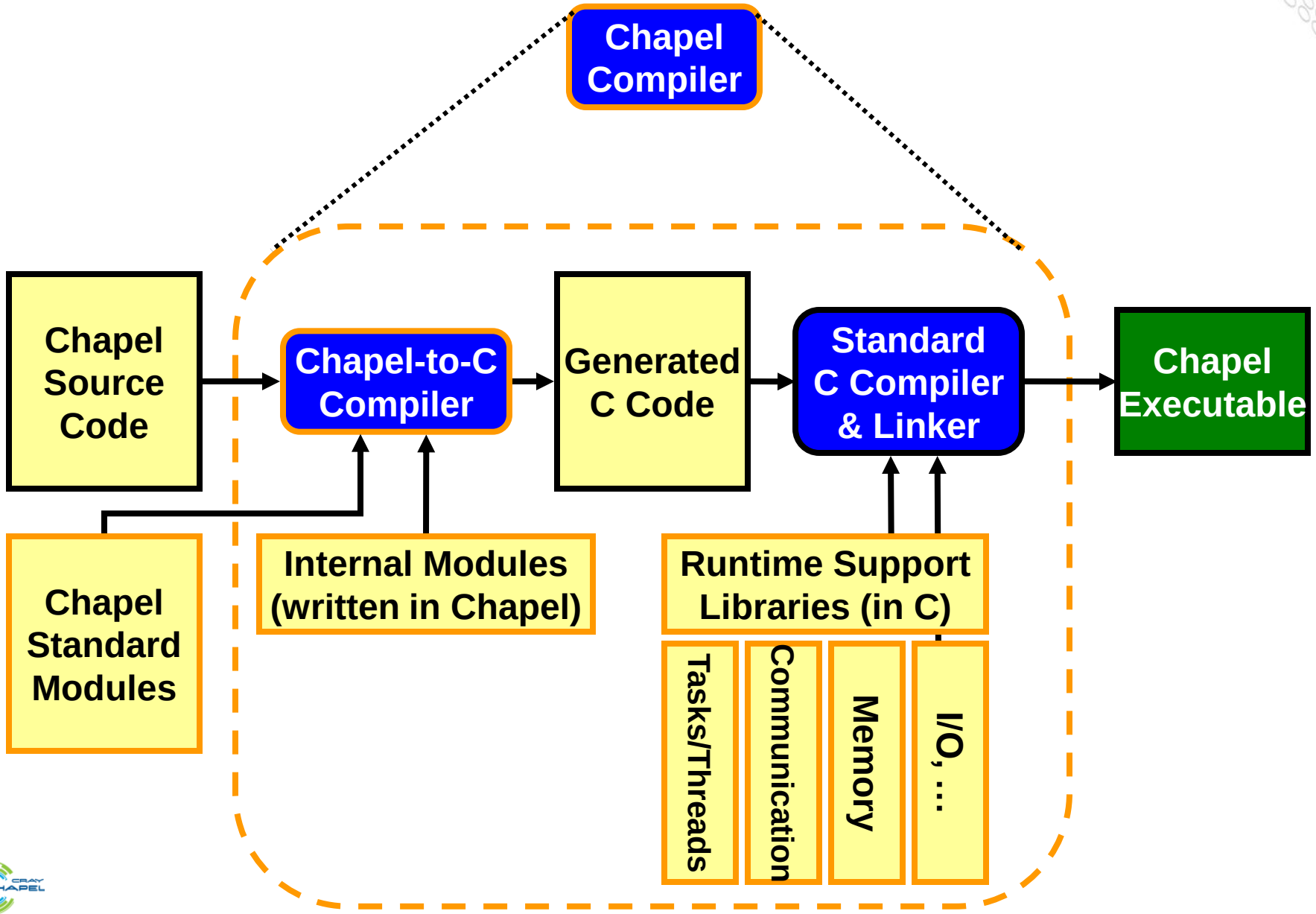
Chapel in a Nutshell

- **Productive Base Language:**
  - type inference, iterators, ranges, generics, optional OOP, …
- **Task Parallel Features:**
  - structured and unstructured task creation
  - synchronization variables for producer-consumer coordination
  - atomic variables supporting common HW atomic operations
- **Data Parallel Features:**
  - rich array types, forall loops, first-class index sets, reductions, …
- **Locality/Affinity Features**
  - locale types/values for reasoning about machine resources
  - on-clauses for data-driven placement of tasks
  - distributions for mapping data/work to locales

Compiling Chapel



Chapel Compiler Architecture



Chapel Timeline

Jun 2002: HPCS starts
Jan 2003: Cray expresses interest in pursuing a new language
Jul 2003: Chapel design and implementation starts
Jun 2006: current language and compiler architecture gel
Apr 2006: first task-parallel programs (shared-memory only)
Dec 2006: first request-only release made available (v0.4)
Jul 2007: first distributed memory task parallel programs
Oct 2008: first distributed memory data parallel programs
Nov 2008: first public release (v0.8), first tutorial at SC08

Chapel Timeline

Jun 2002: HPCS starts
Jan 2003: Cray expresses interest in pursuing a new language
Jul 2003: Chapel design and implementation starts
Jun 2006: current language and compiler architecture gel
Apr 2006: first task-parallel programs (shared-memory only)
Dec 2006: first request-only release made available (v0.4)
Jul 2007: first distributed memory task parallel programs
Oct 2008: first distributed memory data parallel programs
Nov 2008: first public release (v0.8), first tutorial at SC08
Apr 2009: Chapel repository moves to SourceForge
Jul 2009: http://chapel.cray.com launched

Some sample collaborations during HPCS

- **Lightweight Multithreading**
  - Qthreads (SNL), Nanos++ (BSC), MassiveThreads (U. Tokyo)
- **Performance of bulk-synchronous style codes**
  - Bulk and bulk-strided transfers (University of Malaga)
- **Base language features**
  - Native processor atomics, externs, I/O, regexp, … (LTS)
- **GPU Computing**
  - GPU-oriented domain maps (UIUC)
- **Benchmark/Proxy App Studies**
  - Language shootout (LTS + interns), MADNESS (OSU, ORNL), AMR (UW + interns), LULESH (LLNL)

Proxy Application Study: LULESH in Chapel

Apr 2011: LLNL expresses interest in Chapel at Salishan
- made us aware of the LULESH benchmark from DARPA UHPC

Aug 2011: Cray intern ports LULESH to Chapel
- *caveat:* used structured mesh to represent data arrays

Dec 2011: Chapel team visits LLNL (talk, tutorial, 1-on-1 sessions)

Mar 2012: Jeff Keasler (LLNL) visits Cray to pair-program
- in one afternoon, converted from structured to unstructured mesh
- impact on code minimal (mostly in declarations) due to:
  - domains/arrays/iterators
  - rank-independent features

Apr 2012: LLNL reports on collaboration at Salishan

Apr 2012: Chapel 1.5.0 release includes a version of LULESH

Jan 2013: LLNL paper on LULESH named "best software paper" for IPDPS'13

Apr 2013: Chapel 1.7.0 contains the latest and greatest Chapel LULESH
- compile-time flags to select between sparse/dense, 1D/3D, local/dist. data structures
- in 1288 lines of code, only a few domain declarations are affected, not physics code

Next steps: distributed sparse domains, improved scalability

Chapel Timeline

Jun 2002: HPCS starts
Jan 2003: Cray expresses interest in pursuing a new language
Jul 2003: Chapel design and implementation starts
Jun 2006: current language and compiler architecture gel
Apr 2006: first task-parallel programs (shared-memory only)
Dec 2006: first request-only release made available (v0.4)
Jul 2007: first distributed memory task parallel programs
Oct 2008: first distributed memory data parallel programs
Nov 2008: first public release (v0.8), first tutorial at SC08
Apr 2009: Chapel repository moves to SourceForge
Jul 2009: http://chapel.cray.com launched
Nov 2009: Chapel named “most elegant” at HPC Challenge
Oct 2012: final Chapel deliverables completed for HPCS

Chapel's final HPCS milestone: SSCA#2

- **Demonstration of SSCA#2 (kernel 4) on a Cascade system**
  - Complete Chapel implementation of SSCA#2 benchmark
  - Run largest problem size that completes in a fixed time

- **Scalable Synthetic Compact Application #2**
  - Unstructured graph analysis benchmark
    - Kernel 4 computes betweenness centrality
  - Representative of large data analytics problems
  - http://www.graphanalysis.org/benchmark/

Chapel's HPCS Scorecard

- ✓ **Representation-independent implementation of SSCA#2 (four working representations)**
  - generics, iterators, base language
- ✓ **Latency hiding for fine-grained communication**
  - Custom Cray tasking/threading layers
  - Cray Gemini/Aries-optimized communication layers
- ✓ **Optimized remote memory operations**
  - Use of network AMOs
- ✓ **Stable, scalable execution on prototype XC30**
  - Successful multi-hour run utilizing large fraction of memory, all nodes

Chapel Timeline

Jun 2002: HPCS starts
Jan 2003: Cray expresses interest in pursuing a new language
Jul 2003: Chapel design and implementation starts
Jun 2006: current language and compiler architecture gel
Apr 2006: first task-parallel programs (shared-memory only)
Dec 2006: first request-only release made available (v0.4)
Jul 2007: first distributed memory task parallel programs
Oct 2008: first distributed memory data parallel programs
Nov 2008: first public release (v0.8), first tutorial at SC08
Apr 2009: Chapel repository moves to SourceForge
Jul 2009: http://chapel.cray.com launched
Nov 2009: Chapel named “most elegant” at HPC Challenge
Oct 2012: final Chapel deliverables completed for HPCS
Apr 2013: tenth major public release (v1.7.0)

Present

Chapel: By the Numbers

21,334: commits against the repository

6116: downloads of public Chapel releases

1024: messages sent to chapel-users mailing list

192: unique non-Cray mailing list subscribers

~154: Chapel talks given during the HPCS program
(80 workshops/conferences, 32 milestone reviews, 18 academic, 16 government, 8 industry)

~24: notable collaborations
(10 government labs, 10 academic, 4 international)

17: Chapel tutorials
(6 SC, 5 European, 3 government labs, 3 CUG)

14: major releases
(4 request-only, 10 public)

0: language mods due to Cascade architectural changes

Chapel Status Today

- **Most features work as designed**
  - in some cases, corners were cut to hit milestones
- **Profile of today's users**
  - Chapel developers: implementation contains ~21,000 SLOC of Chapel
  - parallel educators: a growing community
  - HPC end-users: interested, but need the implementation to mature
- **The general sentiment among current end-users is that Chapel is very close to what they want**
  - We've achieved acceptance without adoption
- **To gain adoption we need to provide**
  - Better performance
  - Hardened implementation
  - Assurance of longevity

The Elephant in the Room: Performance

- **Chapel performance is not yet where it needs to be to serve as a viable alternative to MPI or UPC**
- **Chapel was designed for performance, but our focus under HPCS has been on features/correctness**
  - Performance in Chapel is a question of "when", not "if"
- **Making Chapel perform better is simpler than making a conventional language significantly more productive**

Chapel Performance: steady progress

Chapel performance and scalability *are* improving

- e.g., 2012 HPC Challenge entry improved performance over 2011:
  - EP STREAM Triad: 2.6x
  - Global RA: 103x
  - HPL: 1741x
  - SSCA#2: ~600,000x for a 16,384x larger graph
- we're not done, but not stuck either

Chapel's Greatest Hits

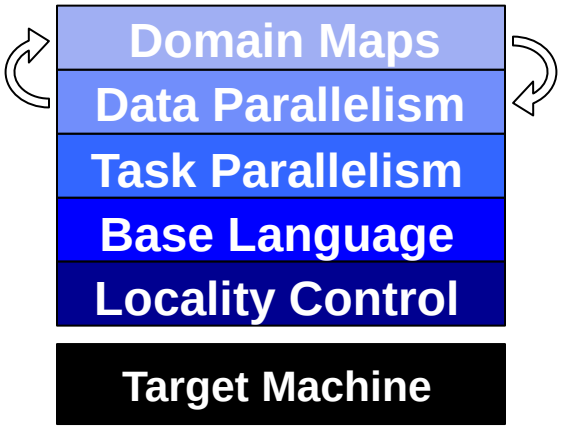
- **Multiresolution Language Design**
  - user-defined layouts, distributions, and parallel loop schedules
  - unified data- and task-parallelism
- **Distinct Concepts for Parallelism vs. Locality**
- **Portable Design and Implementation**
- **User Interactions**

Multiresolution Design

Multiresolution Design: **Support multiple tiers of features**

- higher levels for programmability, productivity
- lower levels for greater degrees of control

Chapel language concepts



- build the higher-level concepts in terms of the lower
  - examples: array distributions and layouts; forall loop implementations
- permit the user to intermix layers arbitrarily

Chapel: Lessons Learned

- If we, as language developers want to do something, eventually an end-user will want to as well
- Multicore NUMA nodes: bigger performance impact than anticipated
- Good research prototypes should anticipate productization
- Don't put off performance/scalability work too long
- Don't underestimate effort required to support early users and collaborators

Chapel Testimonials

"Chapel is a maintainable future-proof language. With additional back-end performance enhancements, we would be using it to develop science codes, with an eye towards multiphysics production codes."

- Jeff Keasler, ASC code developer, LLNL

"After 8 years of observing the Chapel project from the outside while working with X10, I've now had the opportunity to start working with Chapel this year. The experience has been very positive for my research group. have been able to come to up to speed with the Chapel compiler and runtime infrastructure very quickly. On the language front, we've found Chapel's data-parallel constructs to be extremely elegant. However, we see opportunities to broaden the task-parallel constructs, which is the focus of our current research related to Chapel."

- Vivek Sarkar, Professor, Rice University

"I have been closely following the evolution of the X10 and Chapel programming languages since the inception of the DARPA HPCS program. Chapel has been successful in attracting users and has emerged as the victor in the HPCS language wars. It supports multiple productivity features (distributions, locales) in a portable implementation that exhibits good performance."

- Costin Iancu, LBNL

Chapel Testimonials

"Chapel's well-thought-out language design and its modular implementation made it an ideal target for plugging in our task parallel library work. Chapel turned out to be not only an excellent standalone effort, but also a valuable platform for world-wide collaboration."

- Kenjiro Taura, Associate Professor, University of Tokyo

"In association with the department of Pre-College Programs at the University of Notre Dame, the Center for Research Computing (CRC) runs a two week summer school in high-performance computing for high-school students from across the nation. For the last two years we have introduced the high-school students to parallel programming (and for a lot of students an introduction to programming in general) via the Chapel programming language from Cray Inc. After just 5 hours of instruction, the majority of students had not only understood the basics of Chapel but also the important concepts of task and data parallelism to the point were they are able to implement their own parallel solutions to a set of challenge questions. Furthermore, the Chapel session ends with a benchmarking competition were teams are to parallelize a vector-vector addition problem from a serial template solution. In less than one hour all teams were able to submit two sets of solutions and timings corresponding to single node parallelization using the Chapel data-parallel 'forall' construct as well as multi-node parallelization using Chapel's data-distribution features. I believe it would be impossible for a group of inexperienced high-school students to achieve this rapid progress and understanding without the benefit of Chapel language's expressibility and productivity strengths. Having taught programming at college level for over 10 years, Chapel is at the top of my list as a language for introducing students to the art of both sequential and parallel programming. In all seriousness, it should be the first language used to introduce programming on every computing curriculum!"

- Tim Stitt, Research Assistant Professor, University of Notre Dame

I remain consistently optimistic about Chapel. Over time, the quality of the language and the implementation has been steadily increasing. […] In my opinion, the current prototype for Chapel is at the forefront of parallel programming languages. At the same time, Chapel is an ambitious effort that will require more work."

- Michael Ferguson, Researcher, LTS

Future

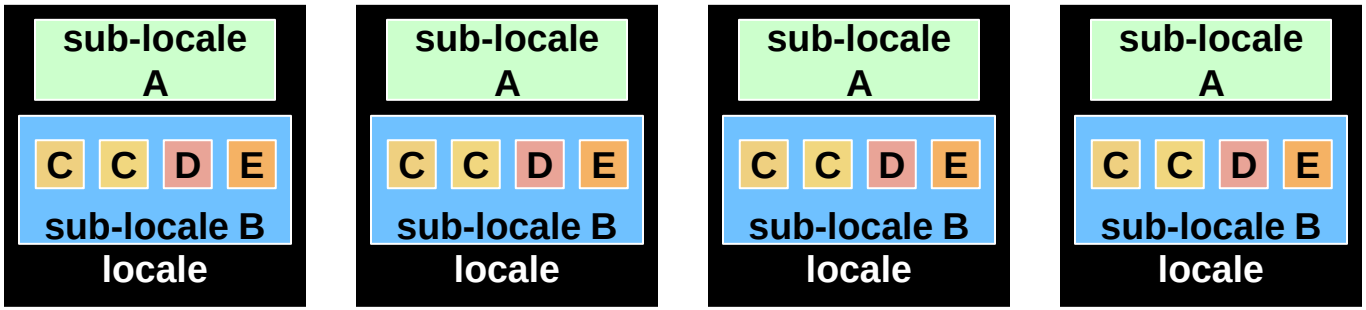
Chapel: The next five (or so) years

- **Ramp up staffing**
- **Fill gaps in the language design**
  - exception handling, task teams, interoperability, RAII, OOP, etc.
- **Address heterogeneity within compute nodes**
  - Hierarchical locales support for GPUs/Intel MIC
- **Benchmark-driven Performance Improvements**
  - Scalar idioms, communication optimizations, memory leaks
- **Work on transitioning governance to an external entity**
  - e.g., “The Chapel Foundation”

Current Work: Hierarchical Locales

Concept:

- Extend locales to support locales within locales to describe architectural sub-structures within a node



- As with traditional locales, *on-clauses* and *domain maps* will be used to map tasks and variables to a sub-locale's memory and processors
- Locale structure is defined as Chapel code
  - permits implementation policies to be specified in-language
  - introduces a new Chapel role: *architectural modeler*

Summary

- **The end of an era**
  - DARPA HPCS program did a nice job of setting up the Chapel project for success beyond the program itself
    - Good demonstrations and achievements
    - Strong fan base
- **The start of a new era**
  - Transition from prototype to product
    - Performance and general hardening
  - Set it free
    - Hand over governance

The Chapel Team (Summer 2012)

Heartfelt thanks!

- To the HPCS program for providing the very unique opportunity to work on an incredibly rewarding project
- To the members of the Chapel team, past and present, within Cray and externally
- To the broader Chapel community for their support and enthusiasm

For More Information

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

- papers, presentations, tutorials, language spec, …

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

- download releases, join mailing lists, browse code repository, …

Mailing Lists:

- chapel_info@cray.com: to contact us
- chapel-users@lists.sourceforge.net: user-oriented discussion list
- chapel-developers@lists.sourceforge.net: developer discussion list
- chapel-education@lists.sourceforge.net: educator discussion list
- …

CRAY
THE SUPERCOMPUTER COMPANY

http://chapel.cray.com chapel_info@cray.com http://sourceforge.net/projects/chapel/