



The Chapel Language: Background and Status

Brad Chamberlain, Chapel Team, Cray Inc.

Customer Briefing, SC16

November 14–17, 2016



COMPUTE

|
STORE

|
ANALYZE

The Chapel Team's Vision



To create a programming language that is...

- ...as productive as Python
- ...as fast as Fortran
- ...as portable as C
- ...as scalable as MPI / UPC / SHMEM
- ...as fun as <your favorite language here>



The Challenge



Q: Why don't we have such languages already?

A: ~~Technical challenges?~~

- while they exist, we don't think this is the primary issue...

A: Due to a lack of...

...long-term efforts

...resources

...community will

...co-design between developers and users

...patience

Chapel is our effort to reverse this trend



What is Chapel?



Chapel: A productive parallel programming language

Characteristics:

- portable
- open-source
- a collaborative effort
- a work-in-progress

Goals:

- Support general parallel programming
 - “any parallel algorithm on any parallel hardware”
- Make parallel programming far more productive



What does “Productivity” mean to you?



Recent Graduate:

“something similar to what I used in school: Python, Matlab, Java, ...”

Seasoned HPC Programmer:

“that sugary stuff that I can’t use because I need full control to ensure good performance”

Computational Scientist:

“something that lets me express my parallel computations without requiring me to wrestle with architecture-specific details”

Chapel Team:

“something that lets the computational scientist express what they want, without taking away the control the HPC programmer needs, implemented in a language as attractive as recent graduates would like.”



Chapel is Portable



- Chapel is designed to be hardware-independent
- The current implementation requires:
 - a C/C++ compiler
 - a *NIX environment (Linux, OS X, BSD, Cygwin, ...)
 - POSIX threads
 - UDP, MPI, or RDMA (for distributed memory execution)
- As a result, Chapel can run on...
 - ...laptops and workstations
 - ...commodity clusters
 - ...the cloud
 - ...HPC systems from Cray and other vendors



Chapel is Open-Source



- Chapel's development is hosted at GitHub
 - <https://github.com/chapel-lang>
- Chapel is licensed as Apache v2.0 software
- Instructions for download + install are online
 - see <http://chapel.cray.com/download.html> to get started



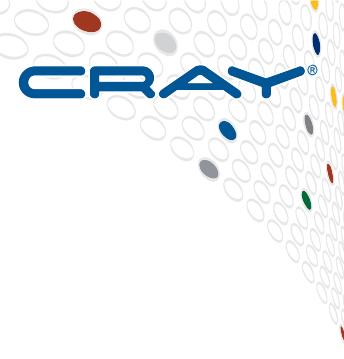
COMPUTE

|

STORE

|

ANALYZE



Chapel History and Characterization



COMPUTE

STORE

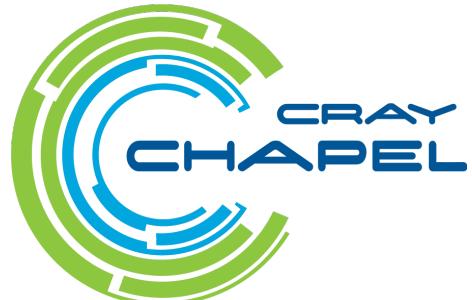
ANALYZE

Chapel's Origins: HPCS



DARPA HPCS: High Productivity Computing Systems

- **Goal:** improve productivity by a factor of 10x
- **Timeframe:** Summer 2002 – Fall 2012
- Cray developed a new system architecture, network, software stack...
 - this became the very successful Cray XC30™ Supercomputer Series



...and a new programming language: Chapel



COMPUTE

STORE

ANALYZE

Chapel's 5-year push



- Based on positive user response to Chapel under HPCS, Cray undertook a five-year effort to improve it
 - we're currently ~3.5 years in
- Focus Areas:
 1. Improving **performance** and scaling
 2. **Fixing** immature aspects of the language and implementation
 - e.g., strings, memory management, error handling, ...
 3. **Porting** to emerging architectures
 - Intel Xeon Phi, accelerators, heterogeneous processors and memories, ...
 4. Improving **interoperability**
 5. Growing the Chapel user and developer **community**
 - including non-scientific computing communities
 6. Exploring transition of Chapel **governance** to a neutral, external body



The Chapel Team at Cray (Summer 2016)



14 full-time employees + 2 summer interns + 1 visiting professor
(one of each started after this photo was taken)



COMPUTE

STORE

ANALYZE

Chapel is a Collaborative Effort



Lawrence Berkeley
National Laboratory



Yale

(and several others...)

<http://chapel.cray.com/collaborations.html>



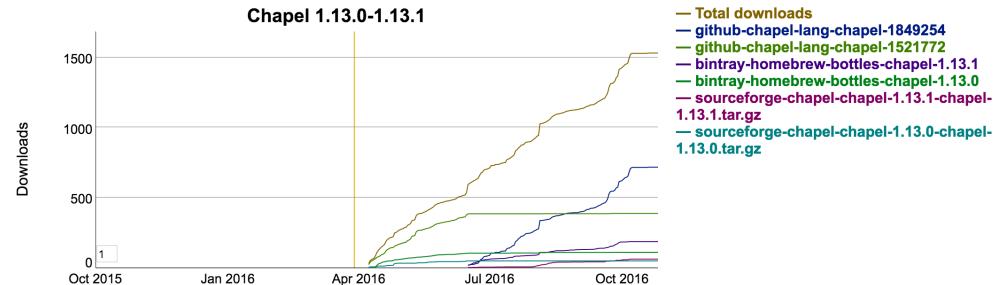
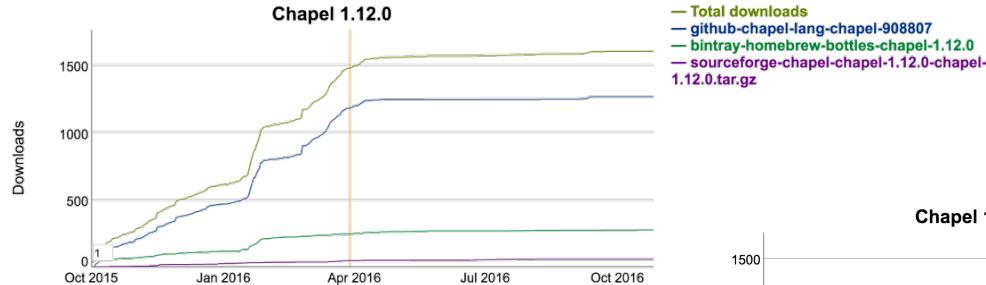
COMPUTE

STORE

ANALYZE

Chapel is a Work-in-Progress

- Currently being picked up by early adopters
 - 3000+ downloads per year across two releases



- Users who try it generally like what they see

A notable early adopter



Chapel in the (Cosmological) Wild

1:00 – 2:00

Nikhil Padmanabhan, Yale University Professor, Physics & Astronomy

Abstract: This talk aims to present my personal experiences using Chapel in my research. My research interests are in observational cosmology; more specifically, I use large surveys of galaxies to constrain the evolution of the

Universe
spatial struc-
universes
our appro-
quickly at
Chapel b-
interoper-
to switch

ber of
simulated
light and
orithms
le of
ke for me

The image shows a YouTube video thumbnail for a Chapel keynote. The thumbnail features a man in a white shirt and khaki pants standing in front of a projection screen displaying a map of the universe. The video title is "CHIUW 2016 keynote: "Chapel in the (Cosmological) Wild", Nikhil Padmanabhan". Below the title, it says "Chapel Parallel Programming Language", "Videos", "Playlists", and "Channels". A timestamp "56:14" is visible in the bottom right corner of the thumbnail image.



COMPUTE

STORE

ANALYZE

Highlights of the first 3-1/2 years



The Standard Library has Grown Significantly



- Many new modules:
 - Bigints
 - Bit Operations
 - Spawn
 - PCG Random Number Generation
 - File System / Path utilities
 - HDFS / cURL
 - LAPACK
 - Reflection
 - Barrier
 - ...
- Several from users:
 - FFTW
 - BLAS
 - MPI
 - ZeroMQ
 - Matrix Market
 - JAMA
 - ...



The Language and Compiler are Improving



- Interoperability
- Strings
- Namespace features
- Semantic changes to reduce race conditions
- Improved set / vector operations
- Numerous bug fixes
- ...

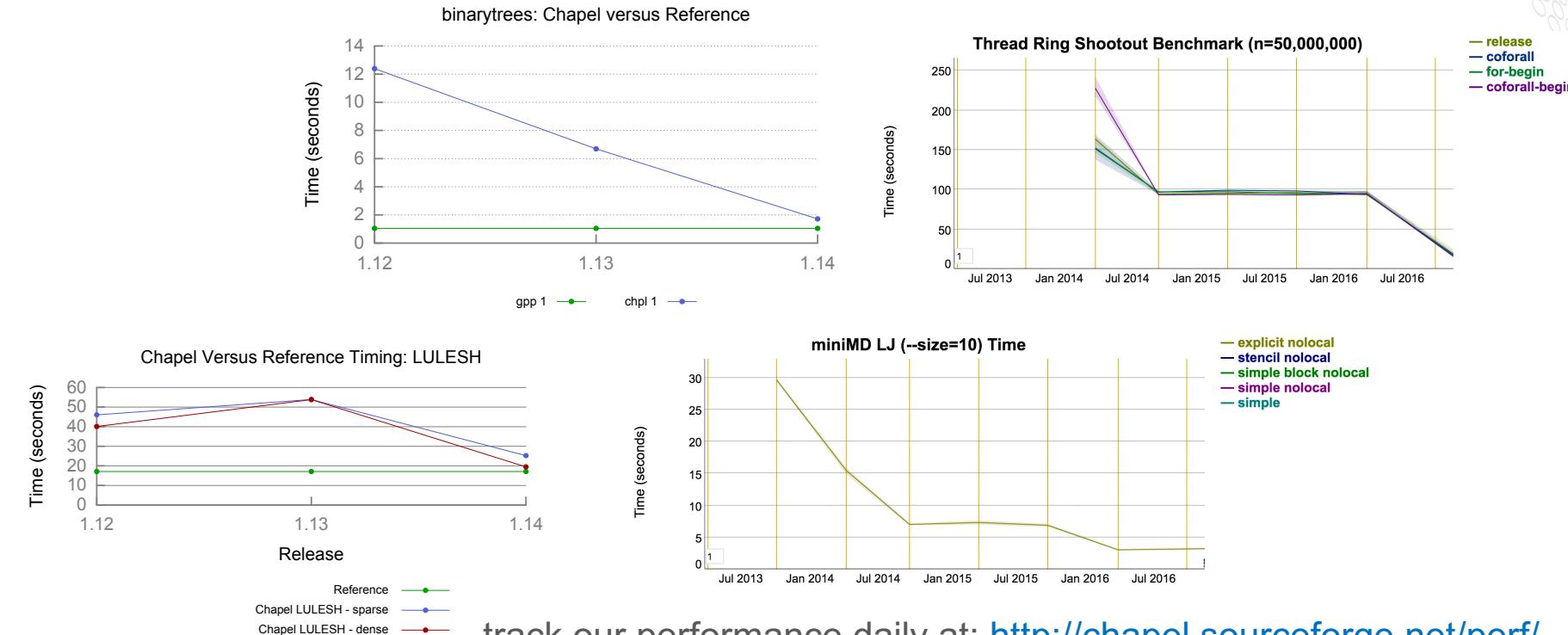
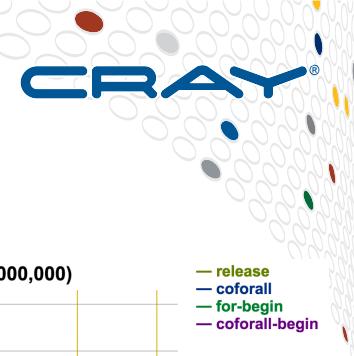


COMPUTE

STORE

ANALYZE

Shared-Memory Performance is Improving...

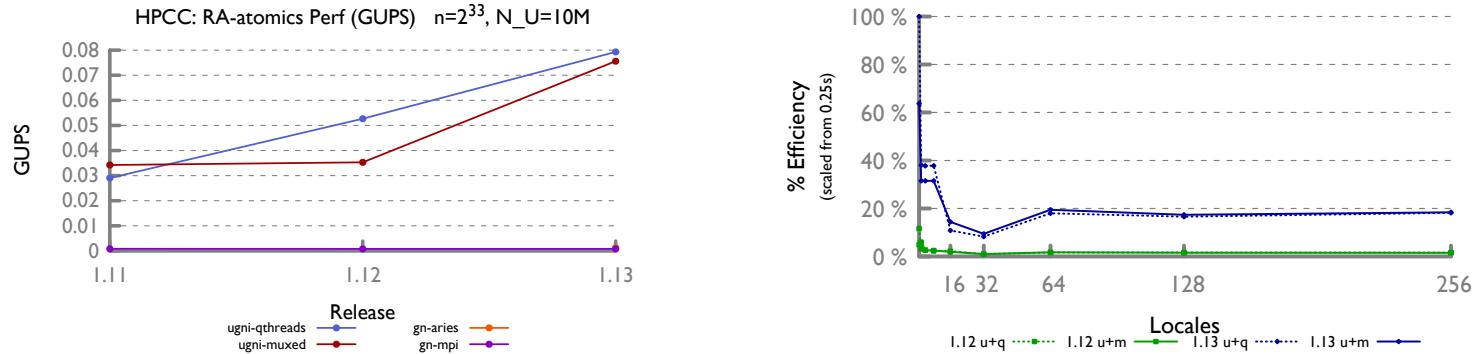
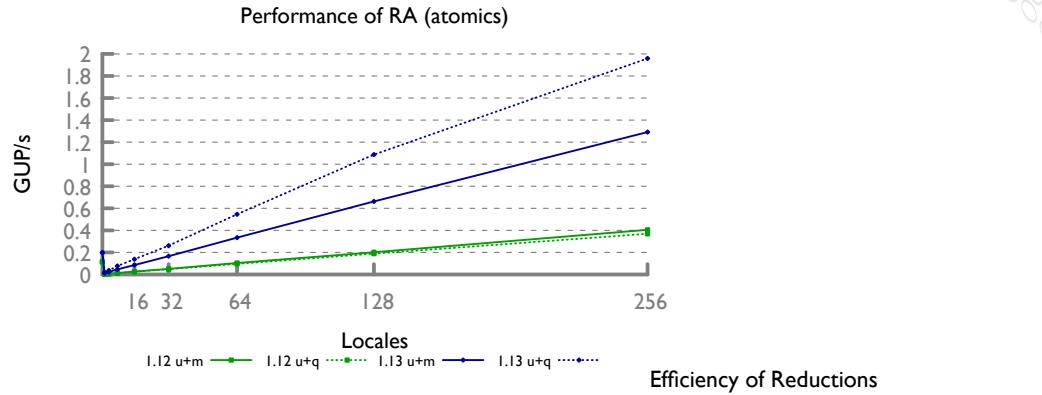
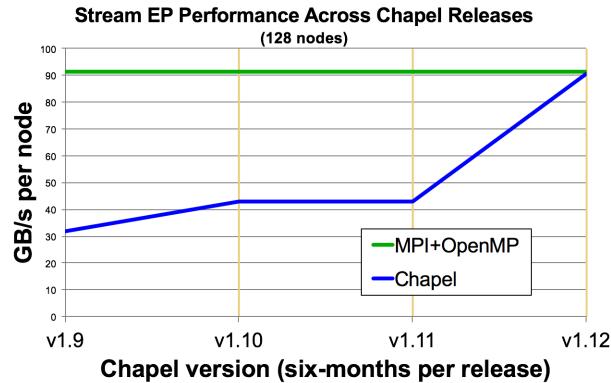


COMPUTE

STORE

ANALYZE

...as is Distributed Memory Perf. and Scaling

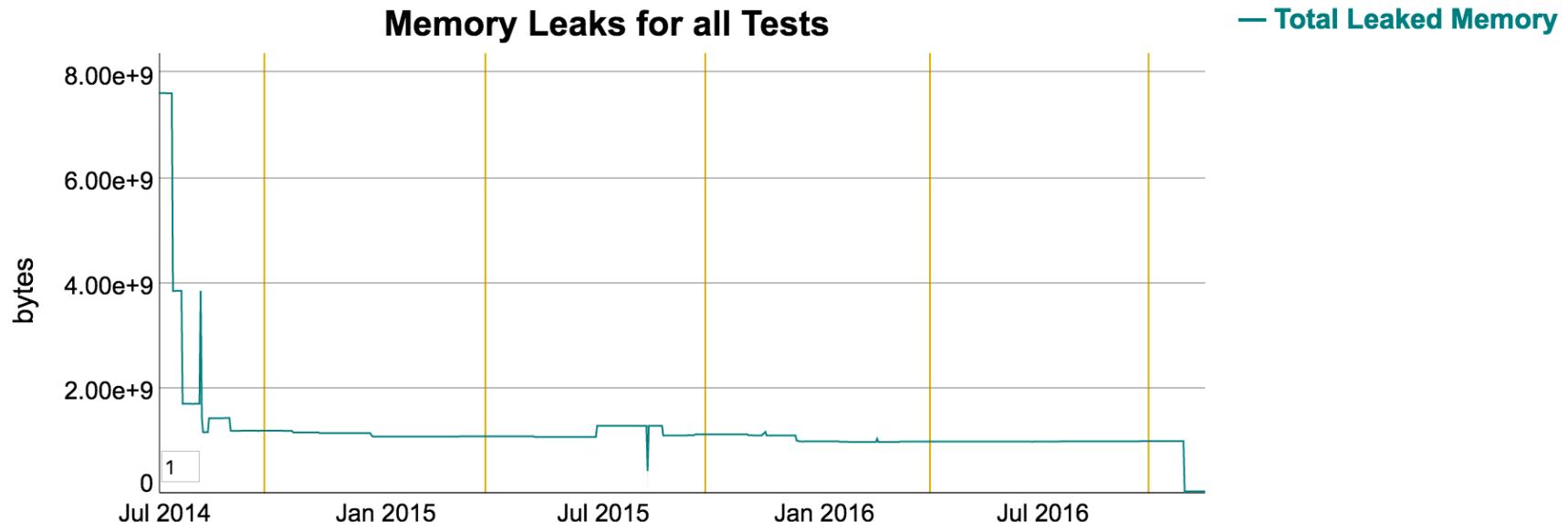


COMPUTE

STORE

ANALYZE

Memory Leaks Are Being Plugged



COMPUTE

STORE

ANALYZE

Documentation Is Now Online / Modern



The screenshot displays the Chapel Documentation website with three main sections:

- Chapel Documentation**: This page includes sections for **COMPILING AND RUNNING CHAPEL** (Quickstart Instructions, Using Chapel, Platform-Specific Notes, Technical Notes, Tools) and **WRITING CHAPEL PROGRAMS** (Quick Reference, Hello World Variants, Primers, Language Specification, Built-in Types and Functions, Standard Modules, Package Modules, Standard Layouts and Distributions, Chapel Users Guide (WIP)). It also features a sidebar for **Language History** (Chapel Evolution, Archived Language Specifications).
- Primers**: This page contains a section for **Language Basics** (Variables, Procedures, Classes, Generic Classes, Variadic Arguments (var args), Modules) and a section for **Iterators** (Iterators, Parallel Iterators). A sidebar lists various **Standard Modules** such as Assert, Barrier, BigInteger, BitOps, Buffers, CommDiagnostics, DynamicIntegers, Error, FileSystem, GMP, Help, IO, List, Math, Memory, Path, Random, Reflection, Regexp, and Spawn.
- Standard Modules**: This page provides a brief description stating "Standard modules are those which describe features of the Standard Library." It lists all the standard modules mentioned in the sidebar of the Primers page.

<http://chapel.cray.com/docs/latest/>

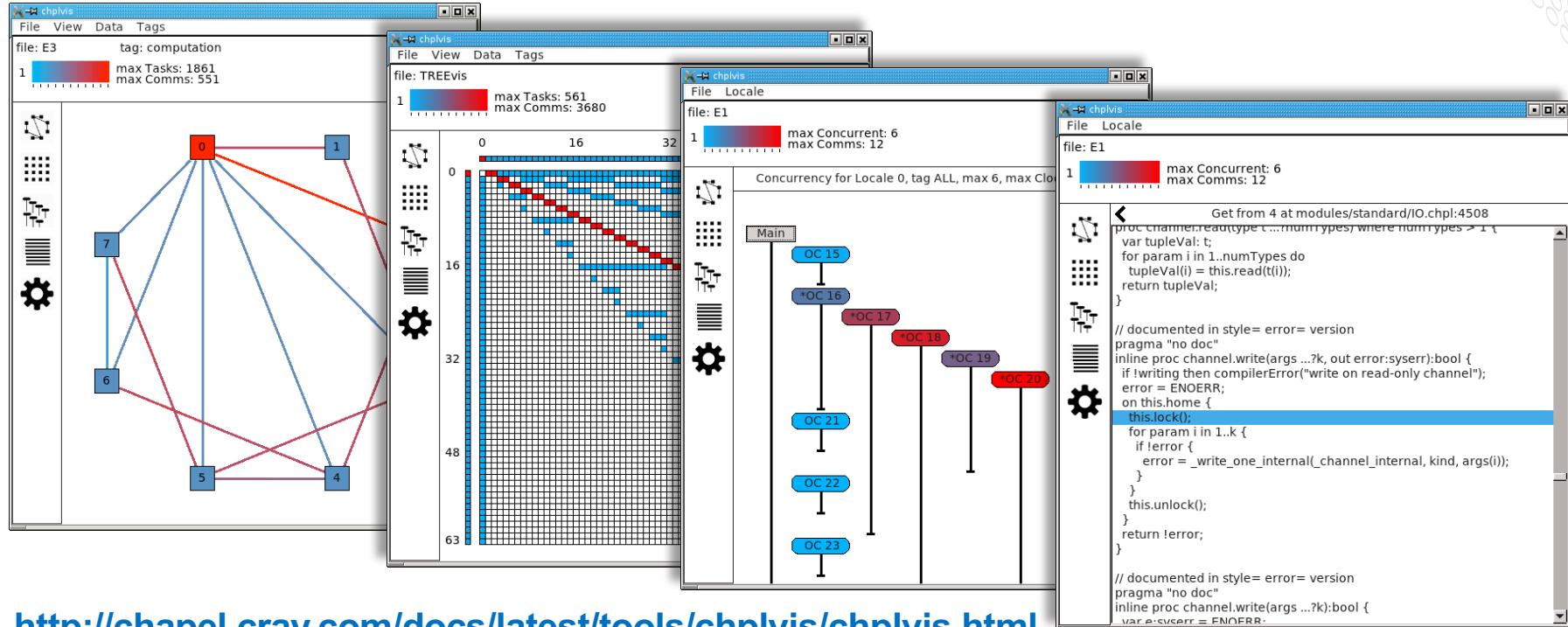
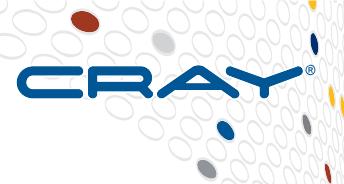


COMPUTE

STORE

ANALYZE

chplvis: Chapel Execution Visualization Tool



<http://chapel.cray.com/docs/latest/tools/chplvis/chplvis.html>



COMPUTE

STORE

ANALYZE

Computer Language Benchmarks Game



The Computer Language Benchmarks Game

64-bit quad core data set

Will your toy benchmark program be faster if you write it in a different programming language? It depends how you write it!

Which programs are fast?

Which are succinct? Which are efficient?

<u>Ada</u>	<u>C</u>	<u>Chapel</u>	<u>Clojure</u>	<u>C#</u>	<u>C++</u>
<u>Dart</u>	<u>Erlang</u>	<u>F#</u>	<u>Fortran</u>	<u>Go</u>	<u>Hack</u>
<u>Haskell</u>	<u>Java</u>	<u>JavaScript</u>	<u>Lisp</u>	<u>Lua</u>	
<u>OCaml</u>	<u>Pascal</u>	<u>Perl</u>	<u>PHP</u>	<u>Python</u>	
<u>Racket</u>	<u>Ruby</u>	<u>JRuby</u>	<u>Rust</u>	<u>Scala</u>	
<u>Smalltalk</u>	<u>Swift</u>	<u>TypeScript</u>			

- **performance rankings:**

- 1 top entry
- + 2 other top-5 entries
- + 2 other top-10 entries
- + 3 other top-20 entries

- **code compactness rankings:**

- 2 top entries
- + 2 other top-5 entries
- + 4 other top-20 entries

We want **easy answers**, but easy answers are often incomplete or wrong. You and I know, there's more we should understand:

stories details fast-faster-fastest
conclusions { for researchers }

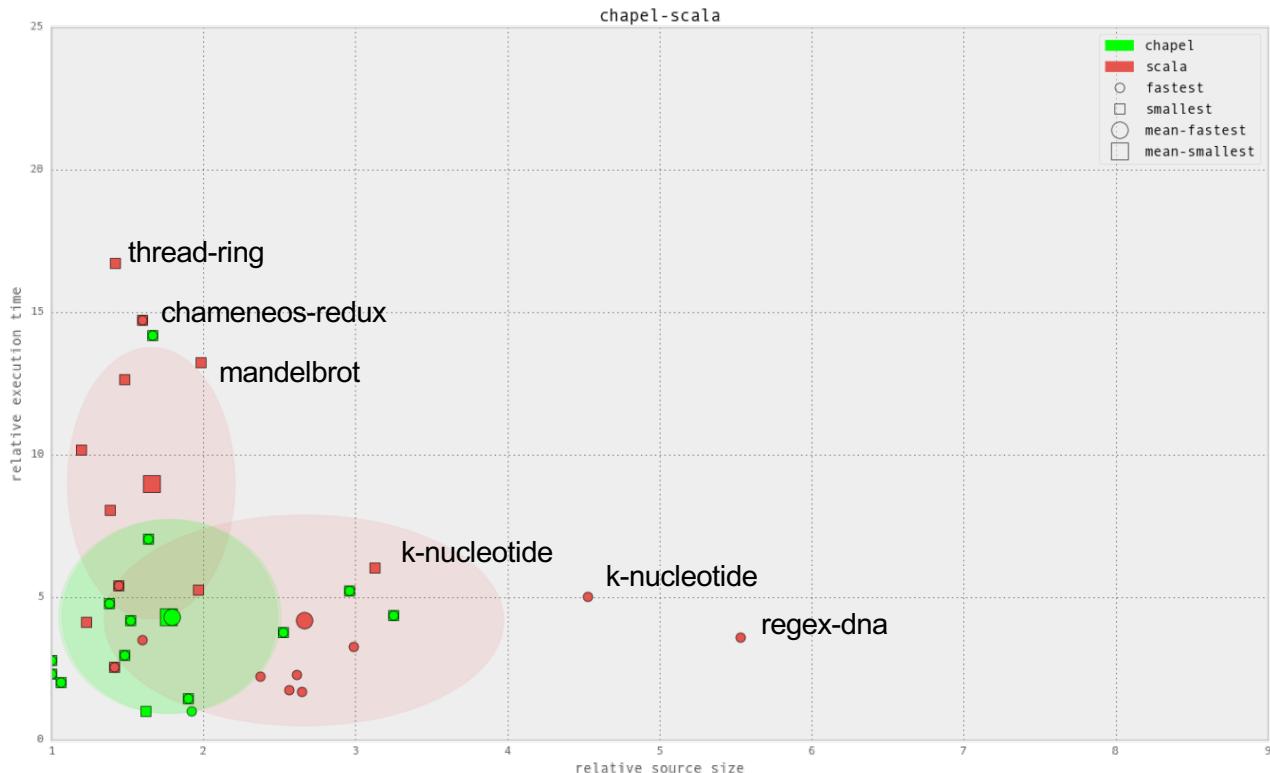
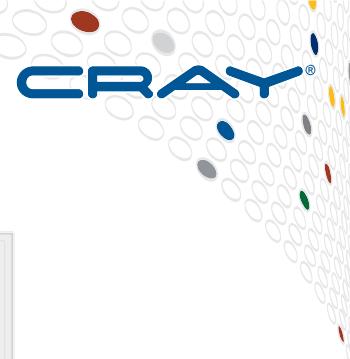


COMPUTE

STORE

ANALYZE

CLBG Scatter Plots: Chapel vs. Scala



COMPUTE

STORE

ANALYZE

There's never been a better time to try Chapel



- **Lots of great progress in the past few years**
 - Like buying a laptop, waiting will always get you more...
...but this is a great time to weigh in with your requests / concerns
- **Chapel trainings / talks / help available on request**
 - I'm available for discussions this week, as are other developers
- **Direct us to killer apps / demos that would help you**



COMPUTE

|

STORE

|

ANALYZE



What's Next?



COMPUTE

STORE

ANALYZE

Top Priorities for Spring 2017 Release



- **Language Features**
 - complete initializers, copy constructors
 - complete draft of error-handling
 - address array rewrite performance regressions
 - implement array views
- **Locality/Memory Improvements**
 - NUMA-aware domains and arrays
 - NUMA-aware memory allocation (including for ugni)
 - HBM support
- **Performance Improvements**
 - single-locale: get LCALS to parity with reference, improve shootouts
 - multi-locale: close in on ref versions of ISx, MiniMD/CoMD, LULESH
 - continue closing significant memory leaks
- **Refocus effort on IPE (REPL) and compiler v2 strategy**



Bigger Picture Chapel Challenges



- **Getting the snowball of adoption rolling**
 - including outreach to non-HPC parallel programmers
- **Amassing a large suite of library code**
- **No interpreter / REPL (yet)**
- **Separate / Incremental compilation**
- **Overhaul of current compiler source code**



COMPUTE

STORE

ANALYZE



Questions?



COMPUTE

| STORE

| ANALYZE