State of the Chapel Project

Brad Chamberlain, Chapel Team, Cray Inc.

CHIUW 2018

May 25, 2018

What is Chapel?

Chapel: A productive parallel programming language

- portable & scalable
- open-source & collaborative

Goals:

- Support general parallel programming
  - "any parallel algorithm on any parallel hardware"
- Make parallel programming at scale far more productive

Chapel and Productivity

Chapel aims to be as…

…**programmable** as Python

…**fast** as Fortran

…**scalable** as MPI, SHMEM, or UPC

…**portable** as C

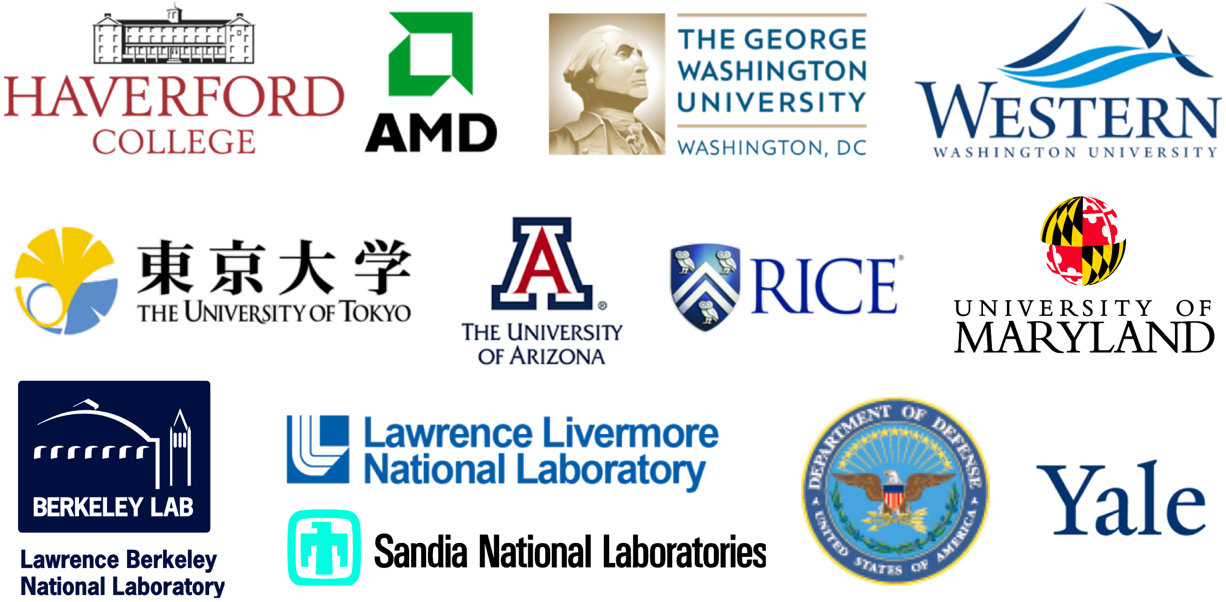
…**flexible** as C++

…**fun** as [your favorite programming language]

The Chapel Team at Cray (May 2018)

13 full-time employees + ~2 summer interns

Chapel Community Partners


HAVERFORD COLLEGE
AMD
THE GEORGE WASHINGTON UNIVERSITY WASHINGTON, DC
WESTERN WASHINGTON UNIVERSITY
東京大学 THE UNIVERSITY OF TOKYO
THE UNIVERSITY OF ARIZONA
RICE
UNIVERSITY OF MARYLAND
BERKELEY LAB
Lawrence Berkeley National Laboratory
Lawrence Livermore National Laboratory
Sandia National Laboratories
DEPARTMENT OF DEFENSE UNITED STATES OF AMERICA
Yale

(and several others…)

https://chapel-lang.org/collaborations.html

A Year in the Life of Chapel

- **Two major releases per year** (was: Apr & Oct; now: Mar & Sept)
  - **~a month later:** detailed release notes
  - **latest release:** Chapel 1.17, released April 5th 2018
- **CHIUW:** Chapel Implementers and Users Workshop (May–June)
- **SC** (November)
  - talks, tutorials, panels, BoFs, posters, exhibits, …
  - annual **CHUG (Chapel Users Group) happy hour**

- **Talks, tutorials, research visits, social media, …** (year-round)

A Year in the Life of Chapel

- **Two major releases per year** (was: Apr & Oct; now: Mar & Sept)
  - **~a month later:** detailed release notes
  - **latest release:** Chapel 1.17, released April 5th 2018
- **CHIUW:** Chapel Implementers and Users Workshop (May–June)
- **SC** (November)
  - talks, tutorials, panels, BoFs, posters, exhibits, …
  - annual **CHUG (Chapel Users Group) happy hour**

- **Talks, tutorials, research visits, social media, …** (year-round)

Welcome to CHIUW!

CHIUW 2018: Agenda (chapel-lang.org/CHIUW2018.html)

8:30:	Chapel 101 (optional)
9:00:	Welcome, State of the Project
9:30:	Break
10:00:	Talks: Applications of Chapel
11:00:	Quick Break
11:10:	Talks: Chapel Design and Evolution
12:10:	Lunch
1:40:	Keynote Talk: “Why Languages Matter”, Kathy Yelick
2:40:	Talks: Chapel Performance
3:00:	Break
3:30:	Talks: Tools for Chapel
4:30:	Lightning Talks and Flash Discussions
5:30:	Wrap-up / Head to Dinner

CHIUW 2018: Organizing Committee

General Chairs:

- **Michael Ferguson**, *Cray Inc.*
- **Nikhil Padmanabhan**, *Yale University*

Program Committee:

- **Brad Chamberlain** (chair), *Cray Inc.*
- **Aparna Chandramowlishwaran (co-chair),** *UC Irvine*
- **Mike Chu,** *AMD*
- **Anshu Dubey,** *Argonne National Laboratory*
- **Jonathan Dursi,** *The Hospital for Sick Children, Toronto*
- **Hal Finkel,** *Argonne National Laboratory*
- **Marta Garcia Gasulla,** *Barcelona Supercomputing Center*
- **Clemens Grelck,** *University of Amsterdam*
- **Jeff Hammond,** *Intel*
- **Bryce Lelbach,** *Nvidia*
- **Michelle Strout,** *University of Arizona*
- **Kenjiro Taura,** *University of Tokyo*
- **David Wonnacott,** *Haverford College*

CHIUW 2018: Keynote

Kathy Yelick, "Why Languages Matter"

Abstract: In the next few years, exascale computing systems will become available to the scientific community. These systems will require new levels of parallelization, new models of memory and storage, and a variety of node architectures for processors and accelerators. In the decade that follows, we can expect more of these changes, as well as increasing levels of hardware specialization. These systems will provide simulation and analysis capabilities at unprecedented scales, and when combined with advanced physical models, mathematical and statistical methods, and computer science and abstractions, they will lead to scientific breakthroughs. Yet the full power of these systems will only be realized if there is sufficient high-level programming support that will abstract details of the machines and give programmers a natural interface for writing new science applications.

CHIUW 2018: Keynote

Kathy Yelick, "Why Languages Matter"

Abstract: In the next few years, exascale computing systems will become available to the scientific community. These systems will require new levels of parallelization, new models of memory and storage, and a variety of node architectures for processors and accelerators. In the decade that follows, we can expect more of these changes, as well as increasing levels of hardware specialization. These systems will provide simulation and analysis capabilities at unprecedented scales, and when combined with advanced physical models, mathematical and statistical methods, and computer science and abstractions, they will lead to scientific breakthroughs. Yet the full power of these systems will only be realized if there is sufficient high-level programming support that will abstract details of the machines and give programmers a natural interface for writing new science applications.

Inspiration from Kathy Yelick (UC Berkeley, LBNL)

Why Consider New Languages at all?

- **Do we need a language? And a compiler?**
  - If higher-level syntax is needed for productivity
    - We need a language
  - If static analysis is needed to help with correctness
    - We need a compiler (front-end)
  - If static optimizations are needed to get performance
    - We need a compiler (back-end)

(Source: HPCS productivity workshop panel, ~2004?)

CHIUW 2018: Research Papers

Parallel Sparse Tensor Decomposition in Chapel

Thomas Rolinger (*University of Maryland*), Tyler Simon, and Christopher Krieger (*Laboratory for Physical Sciences*)

Iterator-Based Optimization of Imperfectly-Nested Loops

Daniel Feshbach, Mary Glaser (*Haverford College*), Michelle Strout (*University of Arizona*), and **David Wonnacott** (*Haverford College*)

Investigating Data Layout Transformations in Chapel

Apan Qasem (*Texas State University*), Ashwin AJi, and **Mike Chu** (*AMD*)

RCUArray: An RCU-like Parallel-Safe Distributed Resizable Array

Louis Jenkins (*Bloomsburg University*)

Purity: An Integrated, Fine-Grain, Data-Centric Communication Profiler for the Chapel Language

Richard Johnson and Jeffrey Hollingsworth (*University of Maryland*)

CHIUW 2018: Technical Talks

Transitioning from Constructors to Initializers in Chapel

Lydia Duncan and Michael Noakes (*Cray Inc.*)

Adding Lifetime Checking to Chapel

Michael Ferguson (*Cray Inc.*)

Tales from the Trenches: Whipping Chapel Performance Into Shape

Elliot Ronaghan, Ben Harshbarger, and Greg Titus (*Cray Inc.*)

ChplBlamer: A Data-centric and Code-centric Combined Profiler for Multi-locale Chapel Programs

Hui Zhang and Jeffrey Hollingsworth (*University of Maryland*)

Mason, Chapel's Package Manager

Ben Albrecht (*Cray Inc.*), Sam Partee (*Haverford College*), Ben Harshbarger, and Preston Sahabu (*Cray Inc.*)

CHIUW 2018: Lightning Talks & Flash Discussions

- Continuing last year's successful session
- Last session of the day!
- Goal: high-energy hot topics for low attention spans!
- Format: Short talks, Q&A, war stories, …whatever!
- Sign up for a slot!

CHIUW 2018: Lightning Talks & Flash Discussions

- Continuing last year's successful session
- Last session of the day!
- Goal: high-energy hot topics for low attention spans!
- Format: Short talks, Q&A, war stories, …whatever!
- Sign up for a slot!

CHIUW 2018: Code Camp Plans

- **Typically, we've held a code camp on day 2 of CHIUW**
  - work on questions, challenges, coding in small teams
  - takes advantage of being in one place
- **This year's advance response was a bit tepid**
- **So, taking a more *ad hoc* approach**
  - Plan is to work in pairs / small groups in common areas
  - If have a topic you're interested in partnering on, let us know
  - If there's lots of last-minute interest, we'll see about a room

CHIUW 2018: Agenda (chapel-lang.org/CHIUW2018.html)

8:30:	Chapel 101 (optional)
9:00:	Welcome, State of the Project
9:30:	Break
10:00:	Talks: Applications of Chapel
11:00:	Quick Break
11:10:	Talks: Chapel Design and Evolution
12:10:	Lunch
1:40:	Keynote Talk: “Why Languages Matter”, Kathy Yelick
2:40:	Talks: Chapel Performance
3:00:	Break
3:30:	Talks: Tools for Chapel
4:30:	Lightning Talks and Flash Discussions
5:30:	Wrap-up / Head to Dinner

A Brief History of Chapel

A Brief History of Chapel

Chapel's Infancy: DARPA HPCS (2003–2012)

- ~6–7 Chapel developers at Cray
- Research focus:
  - distinguish locality from parallelism
  - seamlessly mix data- and task-parallelism
  - support user-defined distributed arrays, parallel iterators
- Captured post-HPCS project status in CUG 2013 paper:

  The State of the Chapel Union

  Chamberlain, Choi, Dumler, Hildebrandt, Iten, Litvinov, Titus

A Brief History of Chapel

Chapel's Infancy: DARPA HPCS (2003–2012)

- ~6–7 Chapel developers at Cray
- Research
  - distinguis
  - seamless
  - support u
- Captured

The State o

Chamberla

Post-HPCS barriers to using Chapel in practice:

Performance and Scalability
Immature Language Features
Insufficient Libraries
Memory Leaks
Lack of Tools
Lack of Documentation
Fear of Being the Only User

Yet user interest in Chapel's potential was high…

A Brief History of Chapel

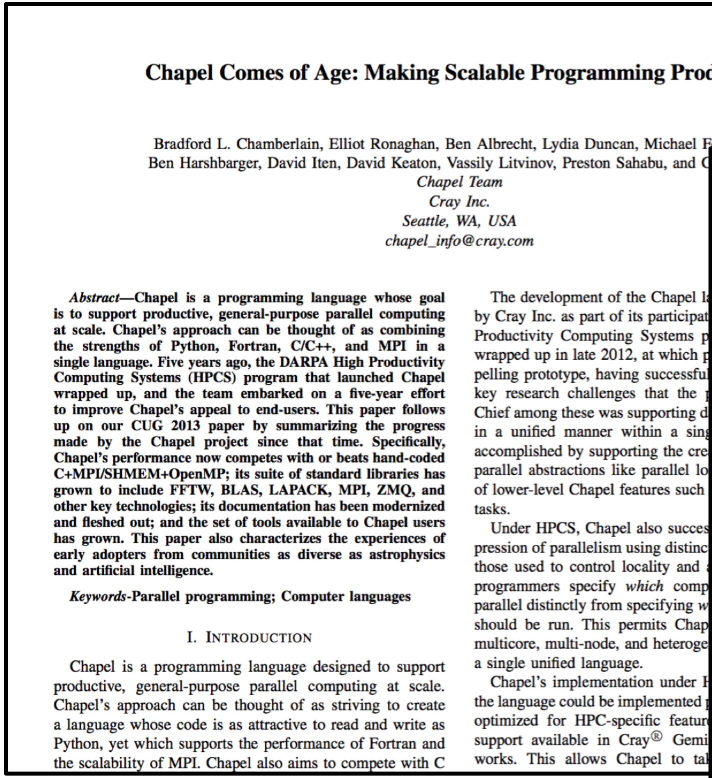
Chapel's Infancy: DARPA HPCS (2003–2012)

Chapel's Adolescence: "the five-year push" (2013–2018)

- ~13–14 Chapel developers at Cray
- Development focus
  - address weak points in HPCS prototype

CUG 2018 Paper: Summary of Five-year Push

paper and slides available at chapel-lang.org



Chapel Comes of Age: Making Scalable Programming Productive

Bradford L. Chamberlain, Elliot Ronaghan, Ben Albrecht, Lydia Duncan, Michael Ferguson,
Ben Harshbarger, David Iten, David Keaton, Vassily Litvinov, Preston Sahabu, and G
Chapel Team
Cray Inc.
Seattle, WA, USA
chapel_info@cray.com

***Abstract*—Chapel is a programming language whose goal is to support productive, general-purpose parallel computing at scale. Chapel's approach can be thought of as combining the strengths of Python, Fortran, C/C++, and MPI in a single language. Five years ago, the DARPA High Productivity Computing Systems (HPCS) program that launched Chapel wrapped up, and the team embarked on a five-year effort to improve Chapel's appeal to end-users. This paper follows up on our CUG 2013 paper by summarizing the progress made by the Chapel project since that time. Specifically, Chapel's performance now competes with or beats hand-coded C+MPI/SHMEM+OpenMP; its suite of standard libraries has grown to include FFTW, BLAS, LAPACK, MPI, ZMQ, and other key technologies; its documentation has been modernized and fleshed out; and the set of tools available to Chapel users has grown. This paper also characterizes the experiences of early adopters from communities as diverse as astrophysics and artificial intelligence.**

***Keywords*-Parallel programming; Computer languages**

I. INTRODUCTION

Chapel is a programming language designed to support productive, general-purpose parallel computing at scale. Chapel's approach can be thought of as striving to create a language whose code is as attractive to read and write as Python, yet which supports the performance of Fortran and the scalability of MPI. Chapel also aims to compete with C

The development of the Chapel l
by Cray Inc. as part of its participat
Productivity Computing Systems p
wrapped up in late 2012, at which p
pelling prototype, having successful
key research challenges that the p
Chief among these was supporting d
in a unified manner within a sing
accomplished by supporting the cre
parallel abstractions like parallel lo
of lower-level Chapel features such
tasks.

Under HPCS, Chapel also succes
pression of parallelism using distinc
those used to control locality and a
programmers specify *which* comp
parallel distinctly from specifying *w*
should be run. This permits Chap
multicore, multi-node, and heteroge
a single unified language.

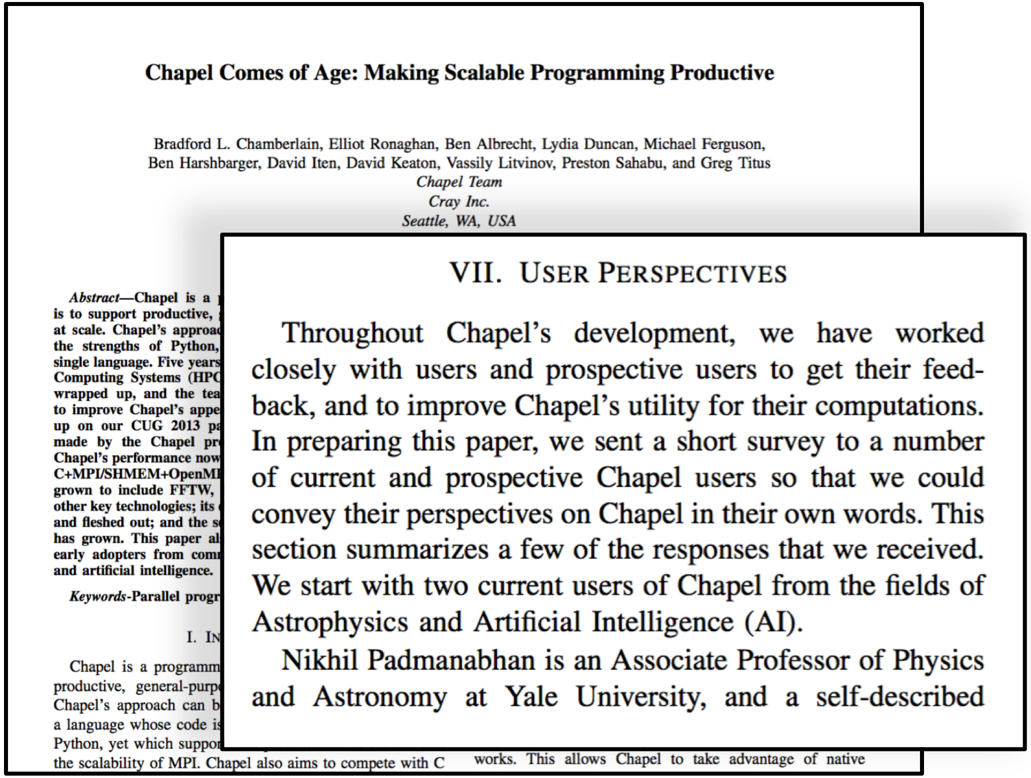
Chapel's implementation under H
the language could be implemented p
optimized for HPC-specific featur
support available in Cray® Gemi
works. This allows Chapel to tak

Chapel Comes of Age: Productive Parallelism at Scale

CUG 2018

Brad Chamberlain, Chapel Team, Cray Inc.

CUG 2018 Paper: User Perspectives



Chapel Comes of Age: Making Scalable Programming Productive

Bradford L. Chamberlain, Elliot Ronaghan, Ben Albrecht, Lydia Duncan, Michael Ferguson,
Ben Harshbarger, David Iten, David Keaton, Vassily Litvinov, Preston Sahabu, and Greg Titus
Chapel Team
Cray Inc.
Seattle, WA, USA

Abstract—Chapel is a
is to support productive,
at scale. Chapel's approac
the strengths of Python,
single language. Five years
Computing Systems (HPC
wrapped up, and the tea
to improve Chapel's appe
up on our CUG 2013 pa
made by the Chapel pr
Chapel's performance now
C+MPI/SHMEM+OpenMI
grown to include FFTW,
other key technologies; its
and fleshed out; and the s
has grown. This paper al
early adopters from comm
and artificial intelligence.

Keywords-Parallel progr

I. IN

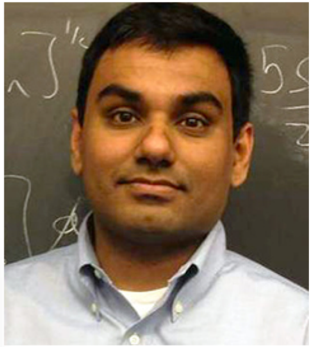
Chapel is a programm
productive, general-purp
Chapel's approach can b
a language whose code is
Python, yet which suppor
the scalability of MPI. Chapel also aims to compete with C

works. This allows Chapel to take advantage of native

VII. User Perspectives

Throughout Chapel's development, we have worked closely with users and prospective users to get their feedback, and to improve Chapel's utility for their computations. In preparing this paper, we sent a short survey to a number of current and prospective Chapel users so that we could convey their perspectives on Chapel in their own words. This section summarizes a few of the responses that we received. We start with two current users of Chapel from the fields of Astrophysics and Artificial Intelligence (AI).

Nikhil Padmanabhan is an Associate Professor of Physics and Astronomy at Yale University, and a self-described



Time-to-Science Astrophysicist

Commercial AI Scientist

Genomics Researcher

DOE Scientist

CUG 2018 Paper: User Perspectives

Notably, user responses all resonated with this goal:

Chapel aims to be as…

…**programmable** as Python

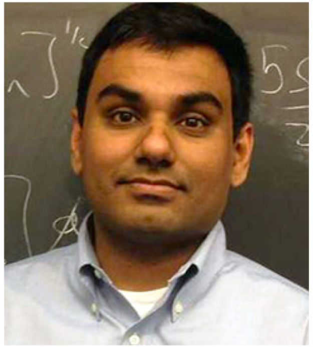
…**fast** as Fortran

…**scalable** as MPI, SHMEM, or UPC

…**portable** as C

…**flexible** as C++

…**fun** as [your favorite language]



Time-to-Science Astrophysicist

Commercial AI Scientist

Genomics Researcher

DOE Scientist

A Brief History of Chapel

Chapel's Infancy: DARPA HPCS (2003–2012)

Chapel's Adolescence: "the five-year push" (2013–2018)

- ~13–14 Chapel developers at Cray
- Development focus
  - address weak points in HPCS prototype

A Brief History of Chapel: What's Next?

Chapel's Infancy: DARPA HPCS (2003–2012)

Chapel's Adolescence: "the five-year push" (2013–2018)

Chapel's College Years: "three! more! years!" (2018-2021)

- Continue development focus:
  - **Stabilize/Harden Language Core:** "no backwards breaking changes"
  - **Interoperability / Usability:** Python, Jupyter, C++, …
  - **Portability:** Libfabric/OFI, GPUs, Cloud computing
  - **Data Structures:** Sparse, DataFrames, Distributed Associative Arrays
  - **Chapel AI, Increased Adoption**

Chapel: Highlights of the Past Year (or Five)

Chapel Language and Libraries

Language: Highlights Since CHIUW 2017

- **User-defined Initializers:** ready for use
  - constructor replacement; fix for OOP problems
  - see Lydia's talk this morning
- **Error Handling:** ready for use
- **'defer' Statement:** registers cleanup actions
- **Uninterpreted Strings:** can contain linefeeds, escapes
- **Delete-Free Programming**
  - improving 'Owned' / 'Shared' and migrating into the language
  - see Michael's talk this morning

Libraries: New Since CHIUW 2017

- **Crypto:** new module based on OpenSSL
  - developed by Sarthak Munshi, GSoC 2017
- **DistributedBag / DistributedDeque:** distributed collections
  - developed by Louis Jenkins, GSoC 2017, speaking this morning
- **DistributedIters:** distributed load-balancing iterators
- **TOML:** initial support for reading TOML files

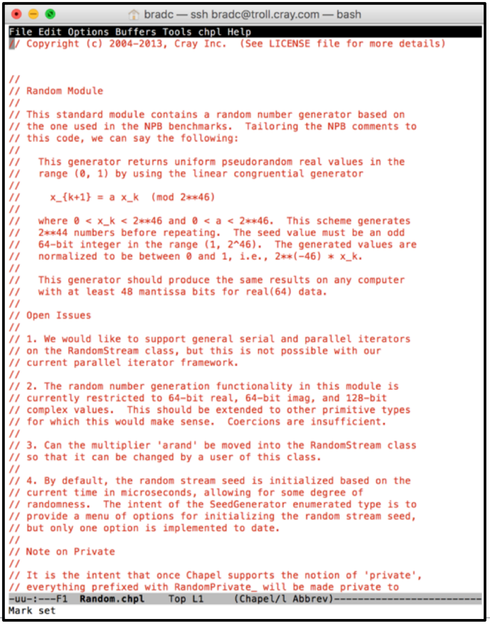
Libraries: Improved Since CHIUW 2017

- **LinearAlgebra:** various ongoing improvements
- **MPI:** improved support for mixing with various configurations
  - co-developed by Nikhil Padmanabhan
- **ZMQ:** improved interoperability with Python via ZMQ
  - developed by Nick Park
- **Path:** added missing routines
  - developed by Sarthak Munshi, Surya Priy, Unnati Parekh, Prithvi Patel, and Varsha Verma
- **Math:** added Bessel functions
  - developed by Nimit Bhardwaj

Libraries: Post-HPCS

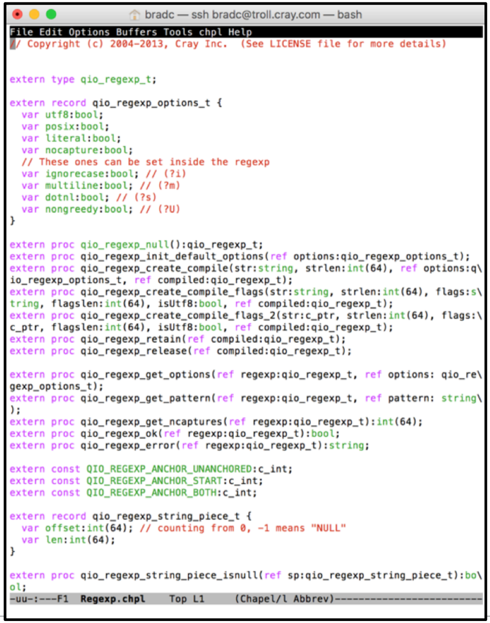
After HPCS: ~25 library modules

- documented via source comments, if at all:



bradc — ssh bradc@troll.cray.com — bash
File Edit Options Buffers Tools chpl Help

```
// Copyright (c) 2004-2013, Cray Inc.  (See LICENSE file for more details)


//
// Random Module
//
// This standard module contains a random number generator based on
// the one used in the NPB benchmarks.  Tailoring the NPB comments to
// this code, we can say the following:
//
//   This generator returns uniform pseudorandom real values in the
//   range (0, 1) by using the linear congruential generator
//
//     x_{k+1} = a x_k  (mod 2**46)
//
//   where 0 < x_k < 2**46 and 0 < a < 2**46.  This scheme generates
//   2**44 numbers before repeating.  The seed value must be an odd
//   64-bit integer in the range (1, 2^46).  The generated values are
//   normalized to be between 0 and 1, i.e., 2**(-46) * x_k.
//
//   This generator should produce the same results on any computer
//   with at least 48 mantissa bits for real(64) data.
//
// Open Issues
//
// 1. We would like to support general serial and parallel iterators
// on the RandomStream class, but this is not possible with our
// current parallel iterator framework.
//
// 2. The random number generation functionality in this module is
// currently restricted to 64-bit real, 64-bit imag, and 128-bit
// complex values.  This should be extended to other primitive types
// for which this would make sense.  Coercions are insufficient.
//
// 3. Can the multiplier 'arand' be moved into the RandomStream class
// so that it can be changed by a user of this class.
//
// 4. By default, the random stream seed is initialized based on the
// current time in microseconds, allowing for some degree of
// randomness.  The intent of the SeedGenerator enumerated type is to
// provide a menu of options for initializing the random stream seed,
// but only one option is implemented to date.
//
// Note on Private
//
// It is the intent that once Chapel supports the notion of 'private',
// everything prefixed with RandomPrivate_ will be made private to
-uu-:---F1  Random.chpl    Top L1     (Chapel/l Abbrev)------------------------
Mark set
```



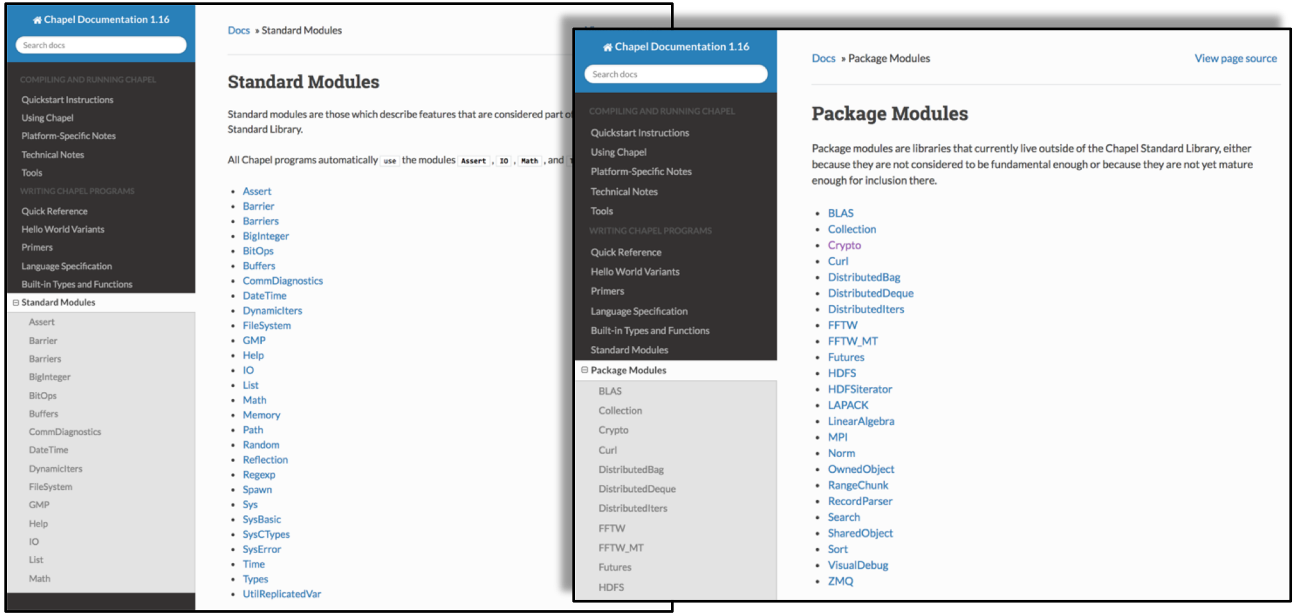
bradc — ssh bradc@troll.cray.com — bash
File Edit Options Buffers Tools chpl Help

```
// Copyright (c) 2004-2013, Cray Inc.  (See LICENSE file for more details)


extern type qio_regexp_t;

extern record qio_regexp_options_t {
  var utf8:bool;
  var posix:bool;
  var literal:bool;
  var nocapture:bool;
  // These ones can be set inside the regexp
  var ignorecase:bool; // (?i)
  var multiline:bool; // (?m)
  var dotnl:bool; // (?s)
  var nongreedy:bool; // (?U)
}

extern proc qio_regexp_null():qio_regexp_t;
extern proc qio_regexp_init_default_options(ref options:qio_regexp_options_t);
extern proc qio_regexp_create_compile(str:string, strlen:int(64), ref options:q\
io_regexp_options_t, ref compiled:qio_regexp_t);
extern proc qio_regexp_create_compile_flags(str:string, strlen:int(64), flags:s\
tring, flagslen:int(64), isUtf8:bool, ref compiled:qio_regexp_t);
extern proc qio_regexp_create_compile_flags_2(str:c_ptr, strlen:int(64), flags:\
c_ptr, flagslen:int(64), isUtf8:bool, ref compiled:qio_regexp_t);
extern proc qio_regexp_retain(ref compiled:qio_regexp_t);
extern proc qio_regexp_release(ref compiled:qio_regexp_t);

extern proc qio_regexp_get_options(ref regexp:qio_regexp_t, ref options: qio_re\
gexp_options_t);
extern proc qio_regexp_get_pattern(ref regexp:qio_regexp_t, ref pattern: string\
);
extern proc qio_regexp_get_ncaptures(ref regexp:qio_regexp_t):int(64);
extern proc qio_regexp_ok(ref regexp:qio_regexp_t):bool;
extern proc qio_regexp_error(ref regexp:qio_regexp_t):string;

extern const QIO_REGEXP_ANCHOR_UNANCHORED:c_int;
extern const QIO_REGEXP_ANCHOR_START:c_int;
extern const QIO_REGEXP_ANCHOR_BOTH:c_int;

extern record qio_regexp_string_piece_t {
  var offset:int(64); // counting from 0, -1 means "NULL"
  var len:int(64);
}

extern proc qio_regexp_string_piece_isnull(ref sp:qio_regexp_string_piece_t):bo\
ol;
-uu-:---F1  Regexp.chpl    Top L1     (Chapel/l Abbrev)------------------------
```

Libraries: Now

Now: ~60 library modules

- web-documented, many user-contributed



Standard Modules

Standard modules are those which describe features that are considered part of Standard Library.

All Chapel programs automatically `use` the modules `Assert`, `IO`, `Math`, and

- Assert
- Barrier
- Barriers
- BigInteger
- BitOps
- Buffers
- CommDiagnostics
- DateTime
- DynamicIters
- FileSystem
- GMP
- Help
- IO
- List
- Math
- Memory
- Path
- Random
- Reflection
- Regexp
- Spawn
- Sys
- SysBasic
- SysCTypes
- SysError
- Time
- Types
- UtilReplicatedVar

Package Modules

Package modules are libraries that currently live outside of the Chapel Standard Library, either because they are not considered to be fundamental enough or because they are not yet mature enough for inclusion there.

- BLAS
- Collection
- Crypto
- Curl
- DistributedBag
- DistributedDeque
- DistributedIters
- FFTW
- FFTW_MT
- Futures
- HDFS
- HDFSiterator
- LAPACK
- LinearAlgebra
- MPI
- Norm
- OwnedObject
- RangeChunk
- RecordParser
- Search
- SharedObject
- Sort
- VisualDebug
- ZMQ

Libraries: Now

Math: FFTW, BLAS, LAPACK, LinearAlgebra, Math

Inter-Process Communication: MPI, ZMQ (ZeroMQ)

Parallelism: Futures, Barrier, DynamicIters

Distributed Computing: DistributedIters, DistributedBag, DistributedDeque, Block, Cyclic, Block-Cyclic, …

File Systems: FileSystem, Path, HDFS

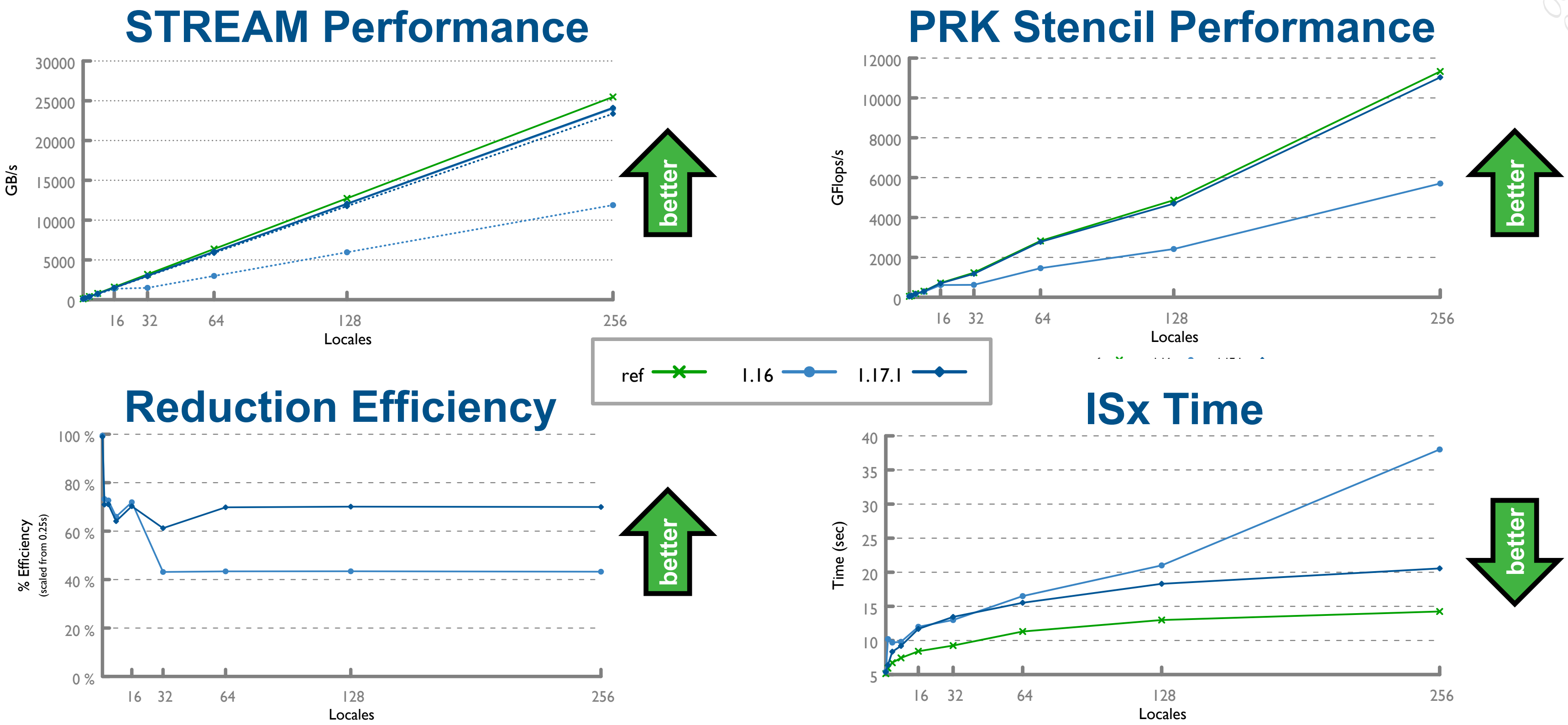
Others: BigInteger, BitOps, Crypto, Curl, DateTime, Random, Reflection, Regexp, Search, Sort, Spawn, …

Arrays, Domain Maps: New Since CHIUW 2017

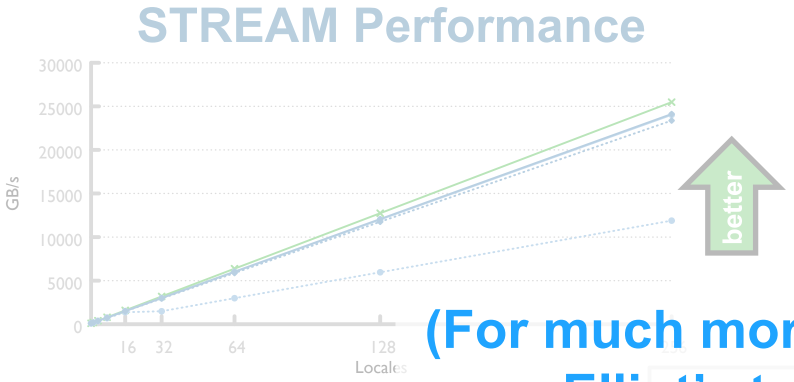
- **Sparse:**
  - Added support for CSC layouts
  - Reduced communication for Block-Sparse Arrays
- **Replicated:** Improved behavior
- **Rank Change / Reindex:** Reduced communication

Performance, Generated Code, and Memory Leaks

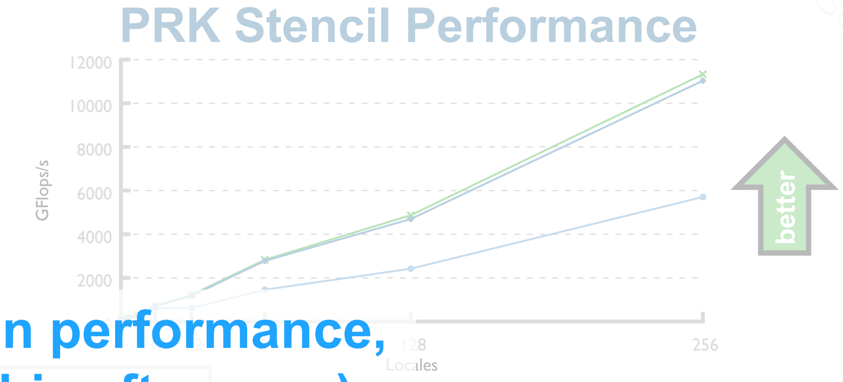
Performance: Improvements since Chapel 1.16



Performance: Improvements since Chapel 1.16

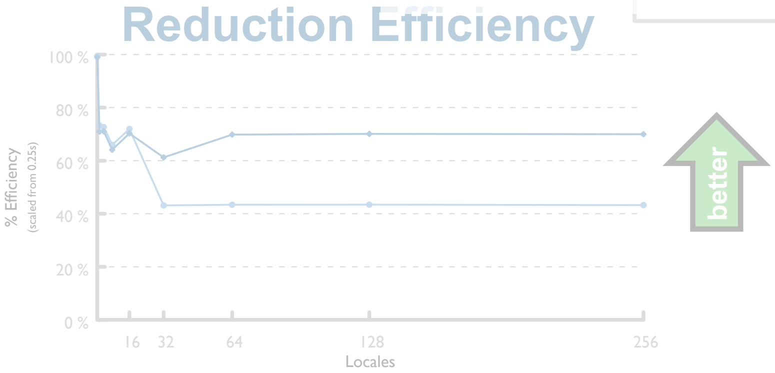


STREAM Performance

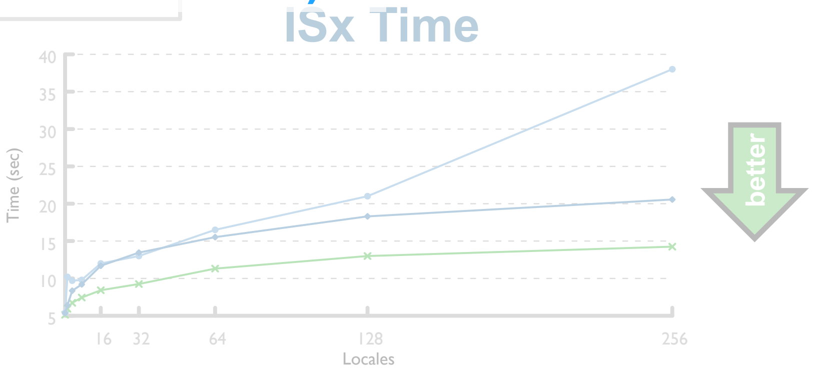


PRK Stencil Performance

(For much more on performance, see Elliot's talk this afternoon)



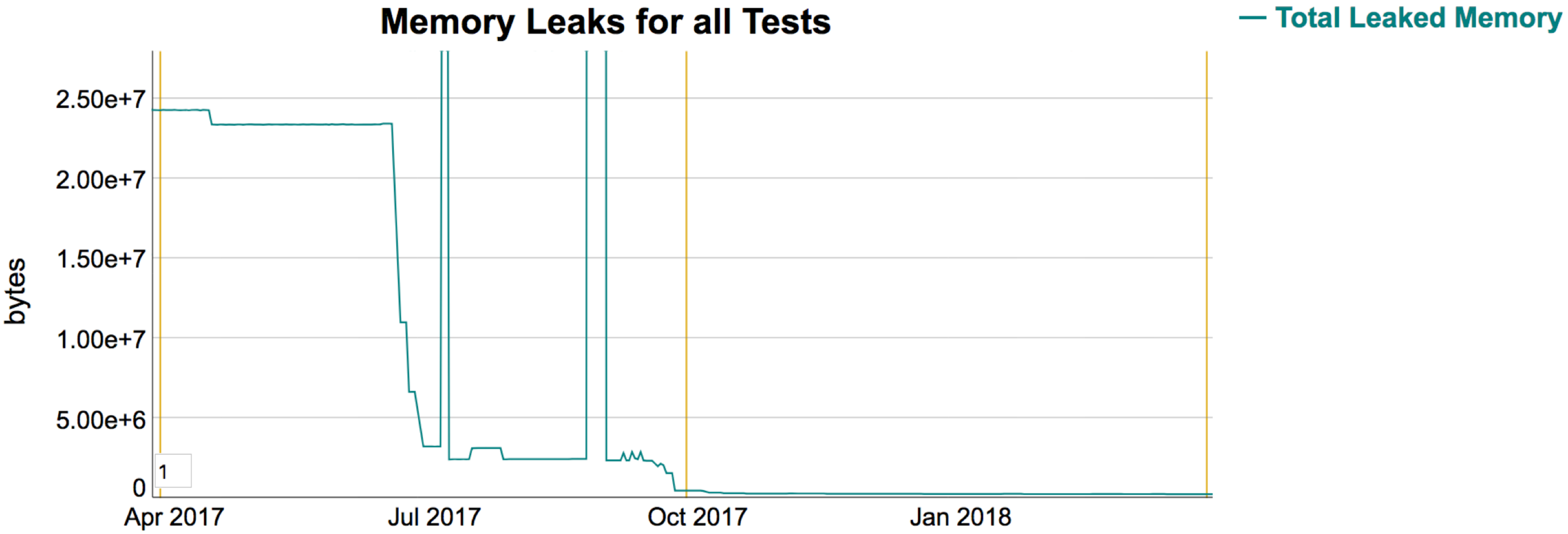
Reduction Efficiency



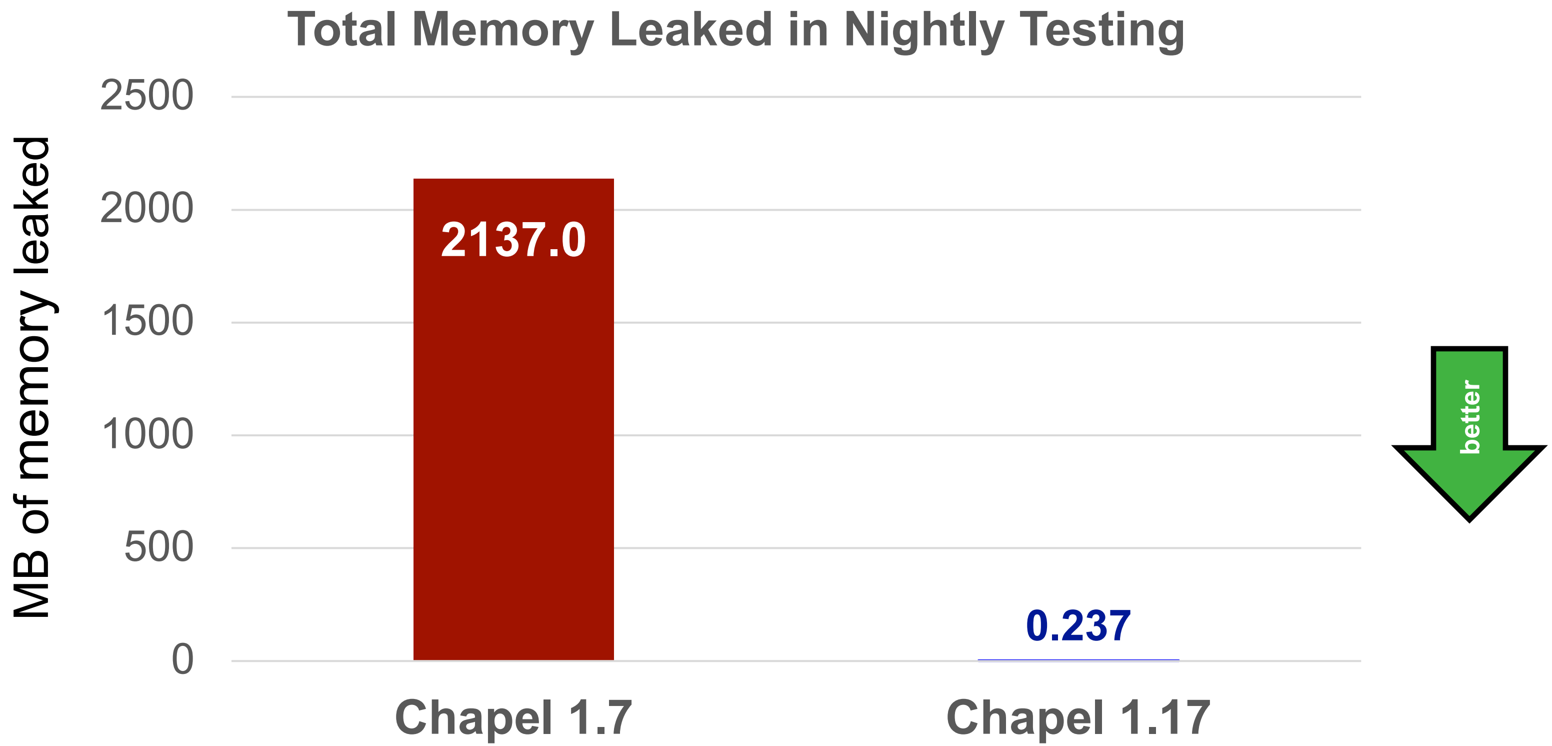
ISx Time

Memory Leaks: Since CHIUW 2017

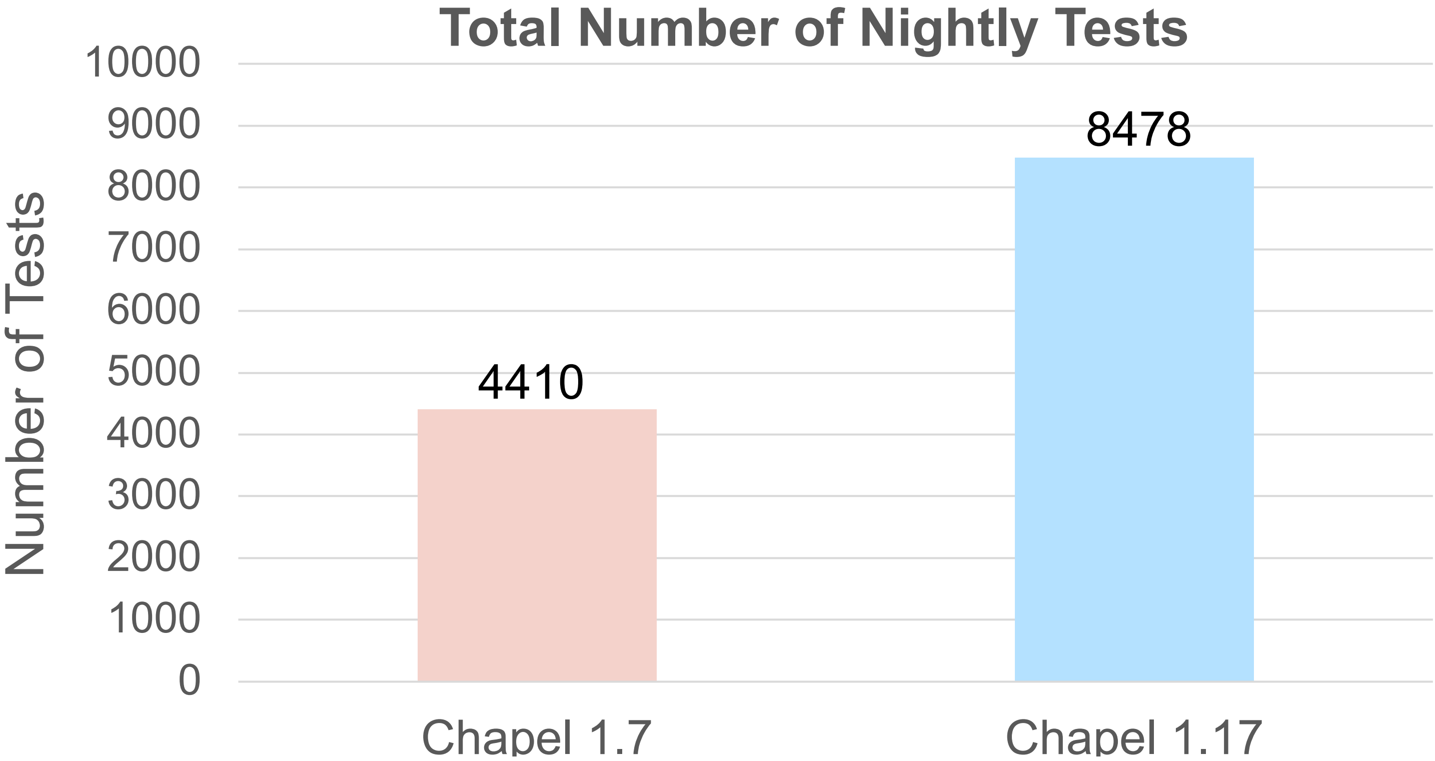
Memory leaks in testing reduced ~100x from 1.15 to 1.17:



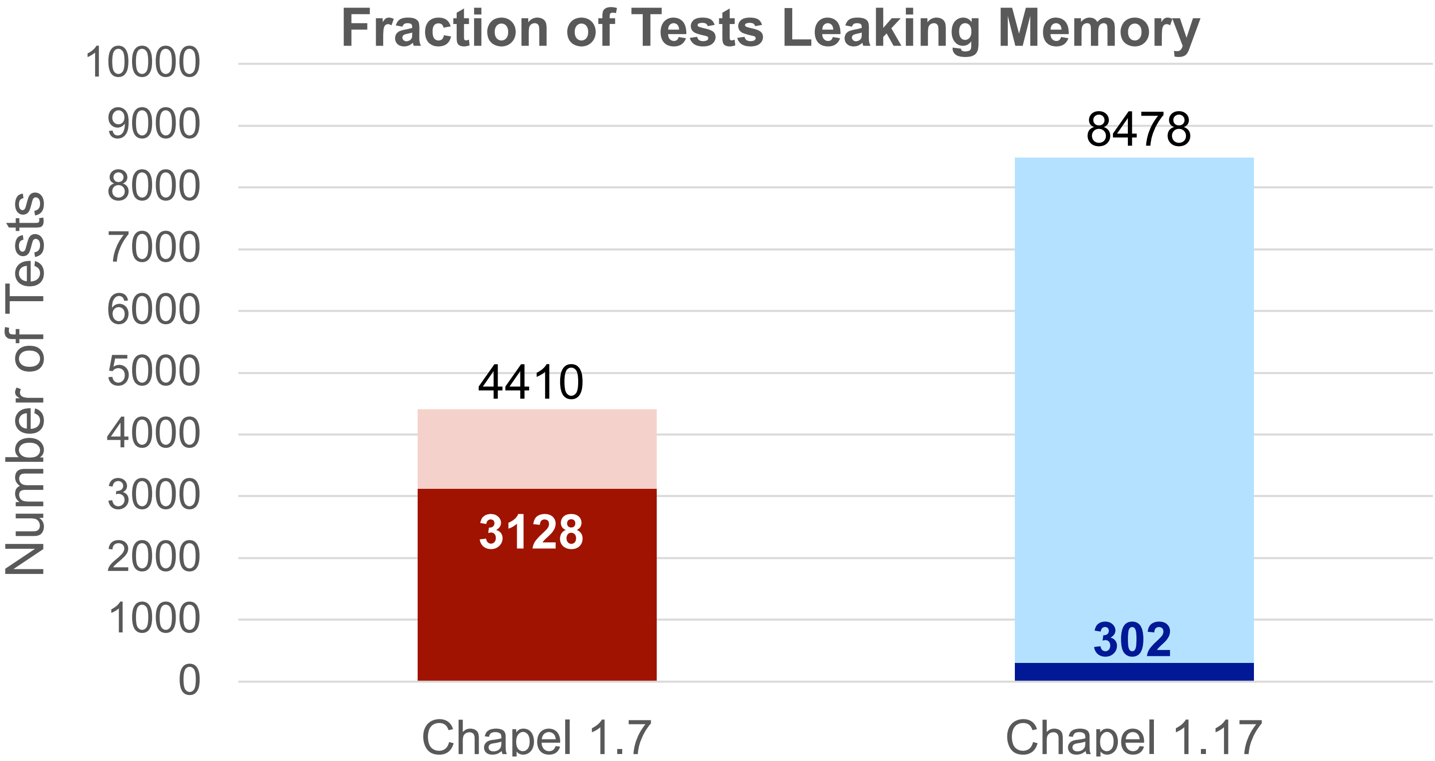
Memory Leaks: Post-HPCS vs. Now



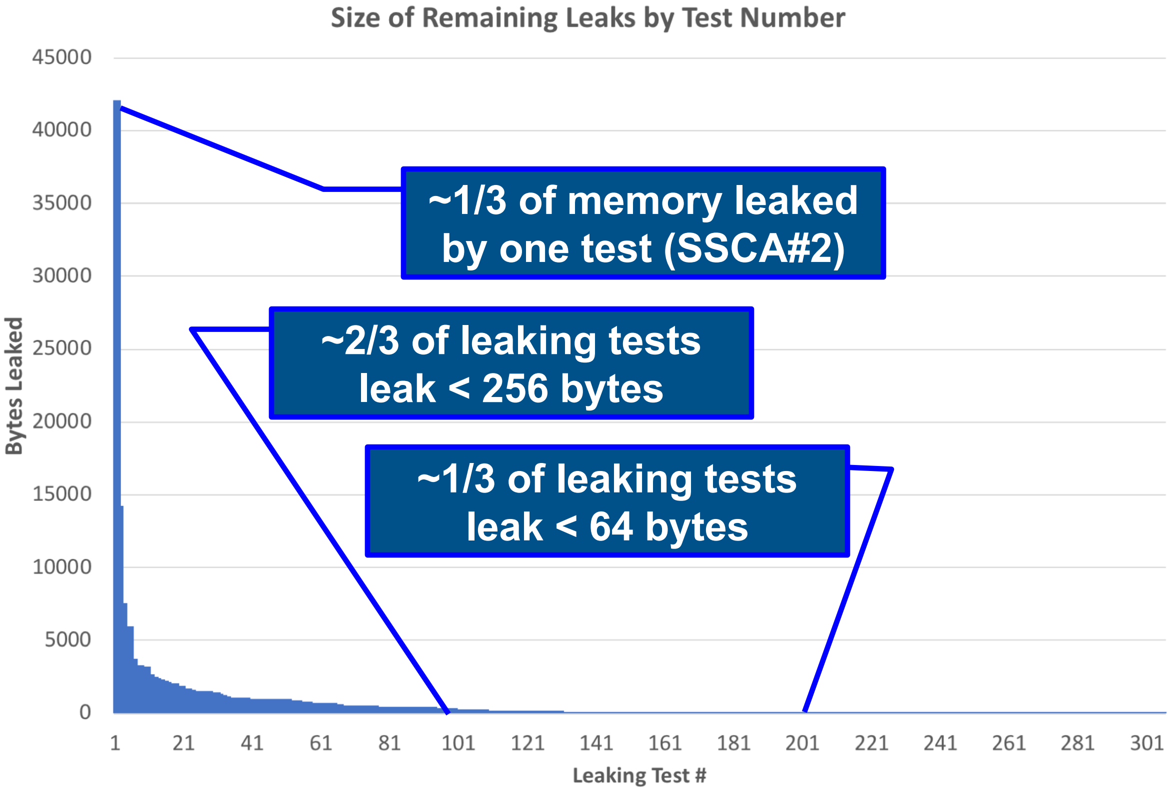
Memory Leaks: Post-HPCS vs. Now



Memory Leaks: Post-HPCS vs. Now



Memory Leaks: Remaining Leaks



~1/3 of memory leaked by one test (SSCA#2)

~2/3 of leaking tests leak < 256 bytes

~1/3 of leaking tests leak < 64 bytes

Portability: Highlights Since CHIUW 2017

- **ARM:** Chapel support for Cray XC50 with ARM processors
- **FreeBSD, PowerPC:** Improved portability
- **OmniPath:** Added support
- **gcc:** Improved portability to new versions

Chapel Ecosystem

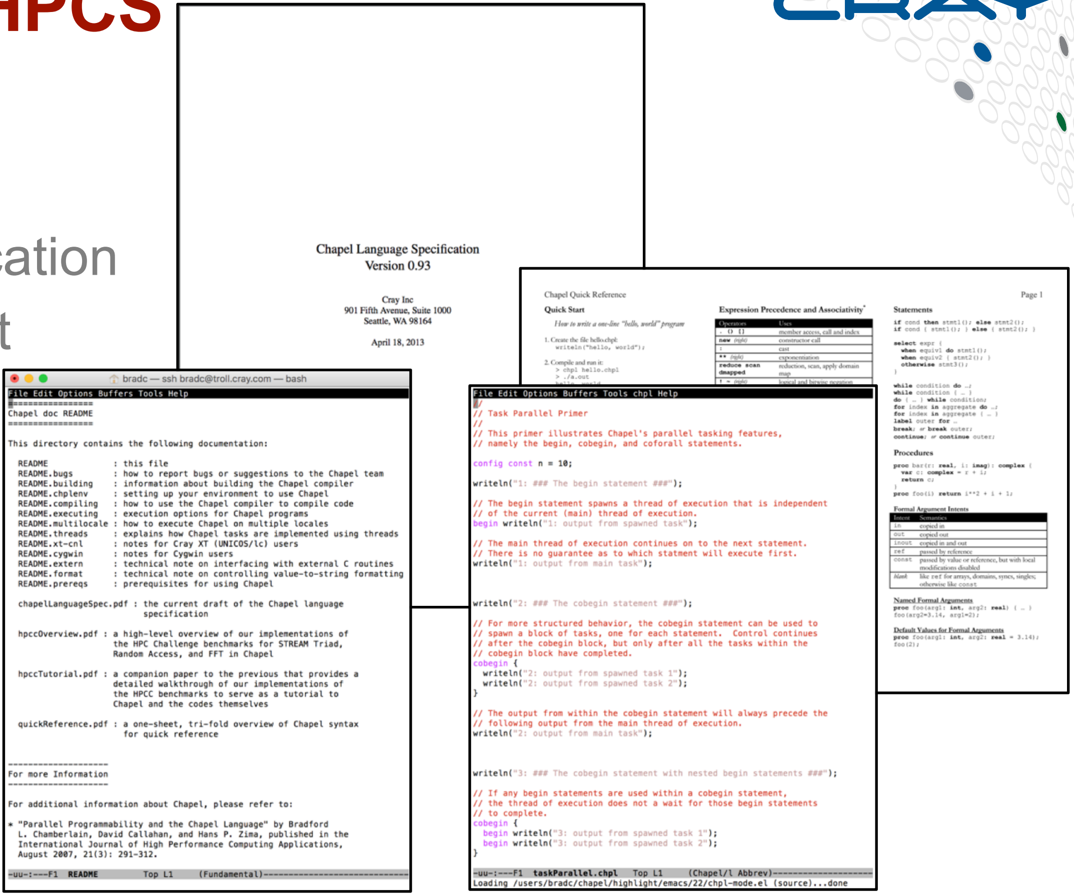
Tools: Highlights Since CHIUW 2017

- **mason: package manager**
  - see Ben Albrecht’s talk this afternoon
- **c2chapel:** convert C header files to ‘extern’ declarations
- **bash tab completion:** command-line help for ‘chpl’ args
- **chpl:**
  - now names executable after main file rather than ‘a.out’
  - now offers suggestions for unfamiliar flags
  - improved support for LLVM back-end
- **configure + make install:** added familiar ways to build

Documentation: Post-HPCS

After HPCS:

- a PDF language specification
- a Quick Reference sheet
- a number of READMEs
- ~22 primer examples



Chapel Language Specification
Version 0.93

Cray Inc
901 Fifth Avenue, Suite 1000
Seattle, WA 98164

April 18, 2013

Chapel Quick Reference

Page 1

Quick Start

How to write a one-line "hello, world" program

1. Create the file hello.chpl:

```
writeln("hello, world");
```

2. Compile and run it:

```
> chpl hello.chpl
> ./a.out
```

Expression Precedence and Associativity*

Operators	Uses
. () []	member access, call and index
new *(right)*	constructor call
:	cast
** *(right)*	exponentiation
reduce scan **dmapped**	reduction, scan, apply domain map
! ~ *(right)*	logical and bitwise negation

Statements

```
if cond then stmt1(); else stmt2();
if cond { stmt1(); } else { stmt2(); }

select expr {
  when equiv1 do stmt1();
  when equiv2 { stmt2(); }
  otherwise stmt3();
}

while condition do …;
while condition { … }
do { … } while condition;
for index in aggregate do …;
for index in aggregate { … }
label outer for …
break; or break outer;
continue; or continue outer;
```

Procedures

```
proc bar(r: real, i: imag): complex {
  var c: complex = r + i;
  return c;
}
proc foo(i) return i**2 + i + 1;
```

Formal Argument Intents

Intent	Semantics
in	copied in
out	copied out
inout	copied in and out
ref	passed by reference
const	passed by value or reference, but with local modifications disabled
blank	like ref for arrays, domains, syncs, singles; otherwise like const

Named Formal Arguments

```
proc foo(arg1: int, arg2: real) { … }
foo(arg2=3.14, arg1=2);
```

Default Values for Formal Arguments

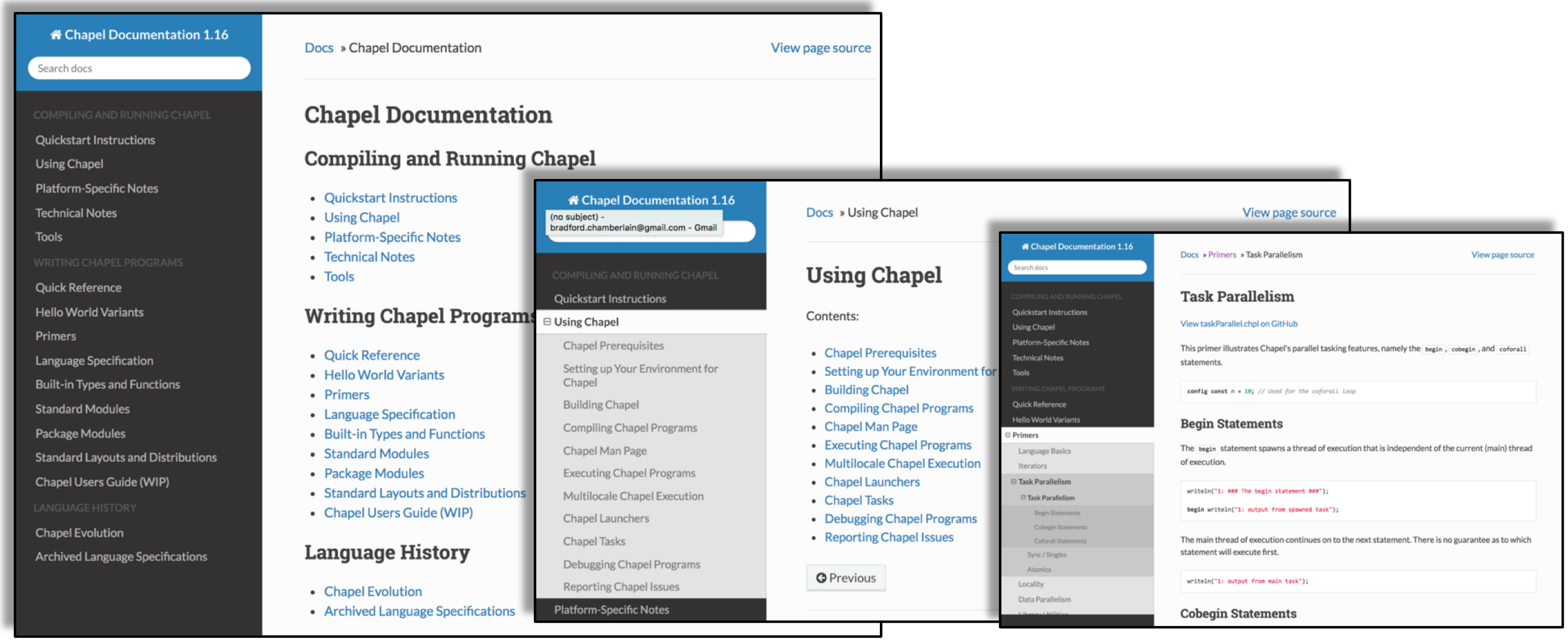
```
proc foo(arg1: int, arg2: real = 3.14);
foo(2);
```

```
File Edit Options Buffers Tools Help
==================
Chapel doc README
==================

This directory contains the following documentation:

  README             : this file
  README.bugs        : how to report bugs or suggestions to the Chapel team
  README.building    : information about building the Chapel compiler
  README.chplenv     : setting up your environment to use Chapel
  README.compiling   : how to use the Chapel compiler to compile code
  README.executing   : execution options for Chapel programs
  README.multilocale : how to execute Chapel on multiple locales
  README.threads     : explains how Chapel tasks are implemented using threads
  README.xt-cnl      : notes for Cray XT (UNICOS/lc) users
  README.cygwin      : notes for Cygwin users
  README.extern      : technical note on interfacing with external C routines
  README.format      : technical note on controlling value-to-string formatting
  README.prereqs     : prerequisites for using Chapel

  chapelLanguageSpec.pdf : the current draft of the Chapel language
                           specification

  hpccOverview.pdf : a high-level overview of our implementations of
                     the HPC Challenge benchmarks for STREAM Triad,
                     Random Access, and FFT in Chapel

  hpccTutorial.pdf : a companion paper to the previous that provides a
                     detailed walkthrough of our implementations of
                     the HPCC benchmarks to serve as a tutorial to
                     Chapel and the codes themselves

  quickReference.pdf : a one-sheet, tri-fold overview of Chapel syntax
                       for quick reference


--------------------
For more Information
--------------------

For additional information about Chapel, please refer to:

* "Parallel Programmability and the Chapel Language" by Bradford
  L. Chamberlain, David Callahan, and Hans P. Zima, published in the
  International Journal of High Performance Computing Applications,
  August 2007, 21(3): 291-312.

-uu-:---F1  README          Top L1     (Fundamental)----------------------------
```

```
File Edit Options Buffers Tools chpl Help
//
// Task Parallel Primer
//
// This primer illustrates Chapel's parallel tasking features,
// namely the begin, cobegin, and coforall statements.

config const n = 10;

writeln("1: ### The begin statement ###");

// The begin statement spawns a thread of execution that is independent
// of the current (main) thread of execution.
begin writeln("1: output from spawned task");

// The main thread of execution continues on to the next statement.
// There is no guarantee as to which statment will execute first.
writeln("1: output from main task");



writeln("2: ### The cobegin statement ###");

// For more structured behavior, the cobegin statement can be used to
// spawn a block of tasks, one for each statement.  Control continues
// after the cobegin block, but only after all the tasks within the
// cobegin block have completed.
cobegin {
  writeln("2: output from spawned task 1");
  writeln("2: output from spawned task 2");
}

// The output from within the cobegin statement will always precede the
// following output from the main thread of execution.
writeln("2: output from main task");



writeln("3: ### The cobegin statement with nested begin statements ###");

// If any begin statements are used within a cobegin statement,
// the thread of execution does not a wait for those begin statements
// to complete.
cobegin {
  begin writeln("3: output from spawned task 1");
  begin writeln("3: output from spawned task 2");
}

-uu-:---F1  taskParallel.chpl   Top L1     (Chapel/l Abbrev)-------------------
Loading /users/bradc/chapel/highlight/emacs/22/chpl-mode.el (source)...done
```

Documentation: **Now**

Now: 200+ modern, hyperlinked, web-based documentation pages



Chapel Documentation 1.16

Search docs

COMPILING AND RUNNING CHAPEL

- Quickstart Instructions
- Using Chapel
- Platform-Specific Notes
- Technical Notes
- Tools

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)

LANGUAGE HISTORY

- Chapel Evolution
- Archived Language Specifications

Docs » Chapel Documentation   View page source

Chapel Documentation

Compiling and Running Chapel

- Quickstart Instructions
- Using Chapel
- Platform-Specific Notes
- Technical Notes
- Tools

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)

Language History

- Chapel Evolution
- Archived Language Specifications

Chapel Documentation 1.16

(no subject) - bradford.chamberlain@gmail.com - Gmail

COMPILING AND RUNNING CHAPEL

- Quickstart Instructions
- ⊟ Using Chapel
  - Chapel Prerequisites
  - Setting up Your Environment for Chapel
  - Building Chapel
  - Compiling Chapel Programs
  - Chapel Man Page
  - Executing Chapel Programs
  - Multilocale Chapel Execution
  - Chapel Launchers
  - Chapel Tasks
  - Debugging Chapel Programs
  - Reporting Chapel Issues
- Platform-Specific Notes

Docs » Using Chapel   View page source

Using Chapel

Contents:

- Chapel Prerequisites
- Setting up Your Environment for
- Building Chapel
- Compiling Chapel Programs
- Chapel Man Page
- Executing Chapel Programs
- Multilocale Chapel Execution
- Chapel Launchers
- Chapel Tasks
- Debugging Chapel Programs
- Reporting Chapel Issues

Previous

Chapel Documentation 1.16

Search docs

COMPILING AND RUNNING CHAPEL

- Quickstart Instructions
- Using Chapel
- Platform-Specific Notes
- Technical Notes
- Tools

WRITING CHAPEL PROGRAMS

- Quick Reference
- Hello World Variants
- ⊟ Primers
  - Language Basics
  - Iterators
  - ⊟ Task Parallelism
    - ⊟ Task Parallelism
      - Begin Statements
      - Cobegin Statements
      - Coforall Statements
    - Sync / Singles
    - Atomics
  - Locality
  - Data Parallelism

Docs » Primers » Task Parallelism   View page source

Task Parallelism

View taskParallel.chpl on GitHub

This primer illustrates Chapel's parallel tasking features, namely the `begin`, `cobegin`, and `coforall` statements.

```
config const n = 10; // Used for the coforall loop
```

Begin Statements

The `begin` statement spawns a thread of execution that is independent of the current (main) thread of execution.

```
writeln("1: ### The begin statement ###");

begin writeln("1: output from spawned task");
```

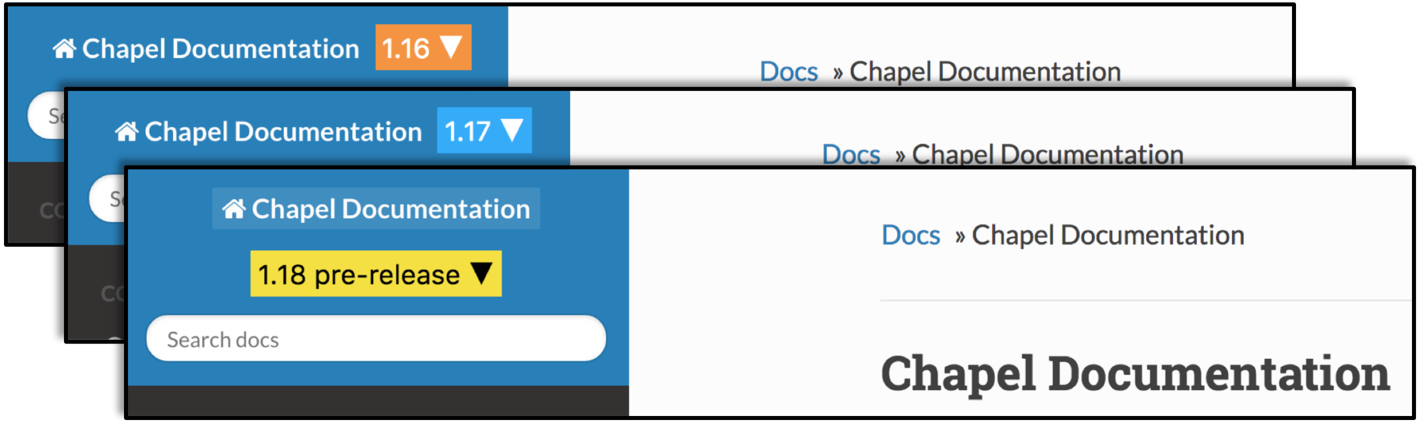
The main thread of execution continues on to the next statement. There is no guarantee as to which statement will execute first.

```
writeln("1: output from main task");
```

Cobegin Statements

Website: Highlights Since CHIUW 2017

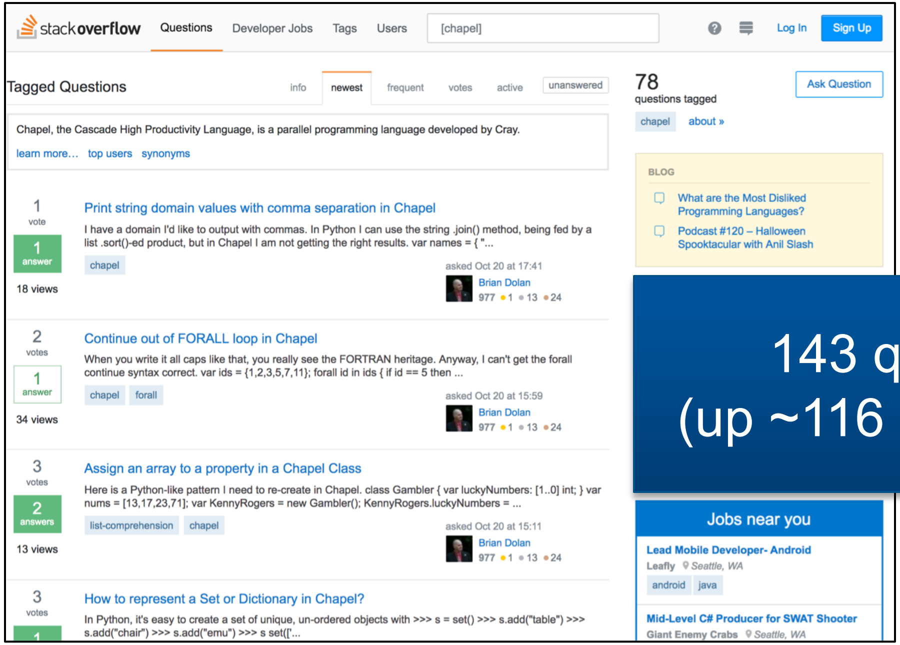
- Added color-coded documentation version menu



- Moved http://chapel.cray.com to https://chapel-lang.org

Chapel on StackOverflow

- StackOverflow ‘chapel’ questions are on the rise



Tagged Questions

Chapel, the Cascade High Productivity Language, is a parallel programming language developed by Cray.

learn more… top users synonyms

1 vote | 1 answer | 18 views

Print string domain values with comma separation in Chapel

I have a domain I'd like to output with commas. In Python I can use the string .join() method, being fed by a list .sort()-ed product, but in Chapel I am not getting the right results. var names = { "...

chapel

asked Oct 20 at 17:41 Brian Dolan 977 ●1 ●13 ●24

2 votes | 1 answer | 34 views

Continue out of FORALL loop in Chapel

When you write it all caps like that, you really see the FORTRAN heritage. Anyway, I can't get the forall continue syntax correct. var ids = {1,2,3,5,7,11}; forall id in ids { if id == 5 then ...

chapel forall

asked Oct 20 at 15:59 Brian Dolan 977 ●1 ●13 ●24

3 votes | 2 answers | 13 views

Assign an array to a property in a Chapel Class

Here is a Python-like pattern I need to re-create in Chapel. class Gambler { var luckyNumbers: [1..0] int; } var nums = [13,17,23,71]; var KennyRogers = new Gambler(); KennyRogers.luckyNumbers = ...

list-comprehension chapel

asked Oct 20 at 15:11 Brian Dolan 977 ●1 ●13 ●24

3 votes | 1

How to represent a Set or Dictionary in Chapel?

In Python, it's easy to create a set of unique, un-ordered objects with >>> s = set() >>> s.add("table") >>> s.add("chair") >>> s.add("emu") >>> s set(['...

78 questions tagged

chapel about »

BLOG

- What are the Most Disliked Programming Languages?
- Podcast #120 – Halloween Spooktacular with Anil Slash

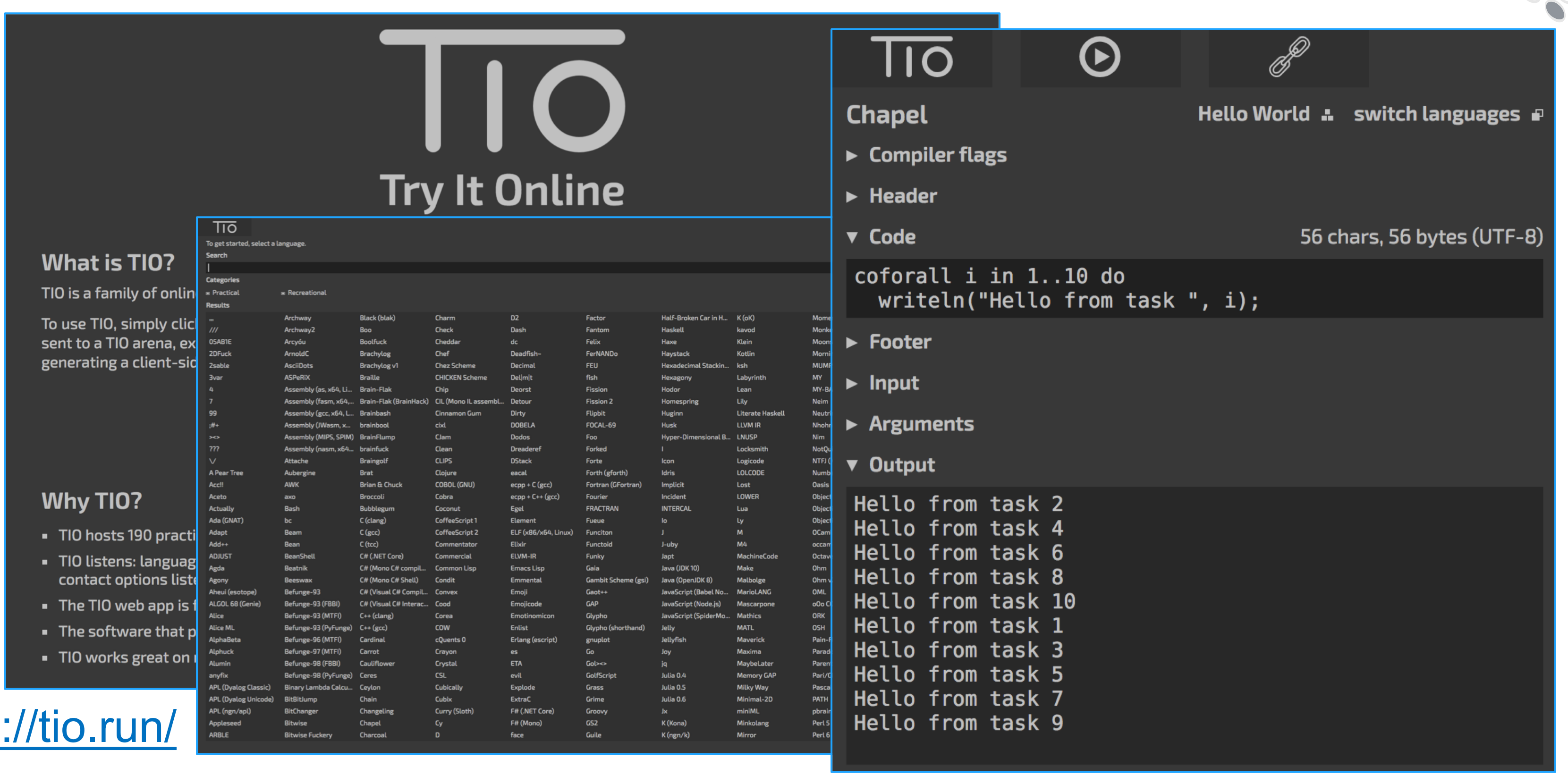
Jobs near you

Lead Mobile Developer- Android
Leafly Seattle, WA
android java

Mid-Level C# Producer for SWAT Shooter
Giant Enemy Crabs Seattle, WA

143 questions tagged
(up ~116 since CHIUW 2017)

Try It Online (TIO): now supports Chapel



What is TIO?

TIO is a family of onlin

To use TIO, simply clic
sent to a TIO arena, ex
generating a client-sid

Why TIO?

- TIO hosts 190 practi
- TIO listens: languag
  contact options liste
- The TIO web app is f
- The software that p
- TIO works great on

To get started, select a language.

""
///
05AB1E
2DFuck
2sable
3var
4
7
99
;#+
><>
???
\/
A Pear Tree
Acc!!
Aceto
Actually
Ada (GNAT)
Adapt
Add++
ADJUST
Agda
Agony
Aheui (esotope)
ALGOL 68 (Genie)
Alice
Alice ML
AlphaBeta
Alphuck
Alumin
anyfix
APL (Dyalog Classic)
APL (Dyalog Unicode)
APL (ngn/apl)
Appleseed
ARBLE
Archway
Archway2
Arcyóu
ArnoldC
AsciiDots
ASPeRiX
Assembly (as, x64, Li...
Assembly (fasm, x64,...
Assembly (gcc, x64, L...
Assembly (JWasm, x...
Assembly (MIPS, SPIM)
Assembly (nasm, x64...
Attache
Aubergine
AWK
axo
Bash
bc
Beam
Bean
BeanShell
Beatnik
Beeswax
Befunge-93
Befunge-93 (FBBI)
Befunge-93 (MTFI)
Befunge-93 (PyFunge)
Befunge-96 (MTFI)
Befunge-97 (MTFI)
Befunge-98 (FBBI)
Befunge-98 (PyFunge)
Binary Lambda Calcu...
BitBitJump
BitChanger
Bitwise
Bitwise Fuckery
Black (blak)
Boo
Boolfuck
Brachylog
Brachylog v1
Braille
Brain-Flak
Brain-Flak (BrainHack)
Brainbash
brainbool
BrainFlump
brainfuck
Braingolf
Brat
Brian & Chuck
Broccoli
Bubblegum
C (clang)
C (gcc)
C (tcc)
C# (.NET Core)
C# (Mono C# compil...
C# (Mono C# Shell)
C# (Visual C# Compil...
C# (Visual C# Interac...
C++ (clang)
C++ (gcc)
Cardinal
Carrot
Cauliflower
Ceres
Ceylon
Chain
Changeling
Chapel
Charcoal
Charm
Check
Cheddar
Chef
Chez Scheme
CHICKEN Scheme
Chip
CIL (Mono IL assembl...
Cinnamon Gum
cixl
CJam
Clean
CLIPS
Clojure
COBOL (GNU)
Cobra
Coconut
CoffeeScript 1
CoffeeScript 2
Commentator
Commercial
Common Lisp
Condit
Convex
Cood
Corea
COW
cQuents 0
Crayon
Crystal
CSL
Cubically
Cubix
Curry (Sloth)
Cy
D
D2
Dash
dc
Deadfish~
Decimal
Del|m|t
Deorst
Detour
Dirty
DOBELA
Dodos
Dreaderef
DStack
eacal
ecpp + C (gcc)
ecpp + C++ (gcc)
Egel
Element
ELF (x86/x64, Linux)
Elixir
ELVM-IR
Emacs Lisp
Emmental
Emoji
Emojicode
Emotinomicon
Enlist
Erlang (escript)
es
ETA
evil
Explode
ExtraC
F# (.NET Core)
F# (Mono)
face
Factor
Fantom
Felix
FerNANDo
FEU
fish
Fission
Fission 2
Flipbit
FOCAL-69
Foo
Forked
Forte
Forth (gforth)
Fortran (GFortran)
Fourier
FRACTRAN
Fueue
Funciton
Functoid
Funky
Gaia
Gambit Scheme (gsi)
Gaot++
GAP
Glypho
Glypho (shorthand)
gnuplot
Go
Gol><>
GolfScript
Grass
Grime
Groovy
GS2
Guile
Half-Broken Car in H...
Haskell
Haxe
Haystack
Hexadecimal Stackin...
Hexagony
Hodor
Homespring
Huginn
Husk
Hyper-Dimensional B...
I
Icon
Idris
Implicit
Incident
INTERCAL
Io
J
J-uby
Japt
Java (JDK 10)
Java (OpenJDK 8)
JavaScript (Babel No...
JavaScript (Node.js)
JavaScript (SpiderMo...
Jelly
Jellyfish
Joy
jq
Julia 0.4
Julia 0.5
Julia 0.6
Jx
K (Kona)
K (ngn/k)
K (oK)
kavod
Klein
Kotlin
ksh
Labyrinth
Lean
Lily
Literate Haskell
LLVM IR
LNUSP
Locksmith
Logicode
LOLCODE
Lost
LOWER
Lua
Ly
M
M4
MachineCode
Make
Malbolge
MarioLANG
Mascarpone
Mathics
MATL
Maverick
Maxima
MaybeLater
Memory GAP
Milky Way
Minimal-2D
miniML
Minkolang
Mirror
Mome
Monk
Moon
Morni
MUMP
MY
MY-BA
Neim
Neutri
Nhohr
Nim
NotQu
NTFJ (
Numb
Oasis
Object
Object
Object
OCam
occam
Octav
Ohm
Ohm v
OML
oOo C
ORK
OSH
Pain-F
Parad
Paren
Pari/C
Pasca
PATH
pbrain
Perl 5
Perl 6

56 chars, 56 bytes (UTF-8)

```
coforall i in 1..10 do
  writeln("Hello from task ", i);
```

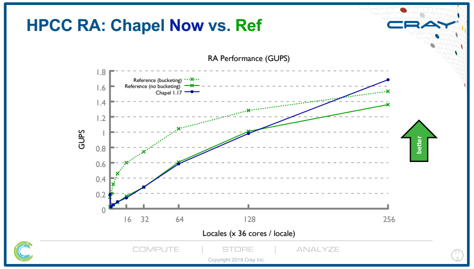
```
Hello from task 2
Hello from task 4
Hello from task 6
Hello from task 8
Hello from task 10
Hello from task 1
Hello from task 3
Hello from task 5
Hello from task 7
Hello from task 9
```

https://tio.run/

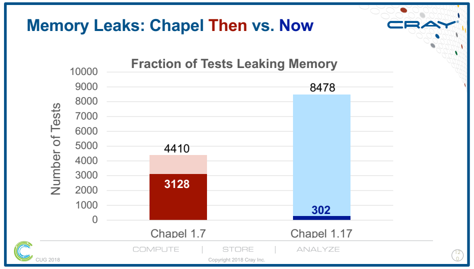
Summary

Chapel has made huge strides over the past year/5 years

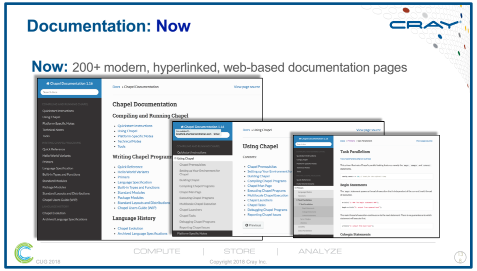
We've addressed many historical barriers to using Chapel



HPCC RA: Chapel Now vs. Ref



Memory Leaks: Chapel Then vs. Now



Documentation: Now

Now: 200+ modern, hyperlinked, web-based documentation pages

Chapel Documentation

Compiling and Running Chapel

- Quickstart Instructions
- Using Chapel
- Platform-Specific Notes
- Technical Notes
- Tools

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)

Language History

- Chapel Evolution
- Archived Language Specifications

Using Chapel

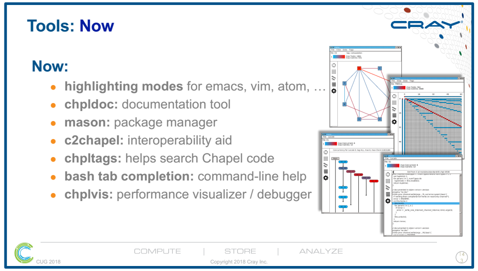
- Chapel Prerequisites
- Setting up Your Environment for Chapel
- Building Chapel
- Compiling Chapel Programs
- Chapel Man Page
- Executing Chapel Programs
- Multilocale Chapel Execution
- Chapel Launchers
- Chapel Tasks
- Debugging Chapel Programs
- Reporting Chapel Issues

Task Parallelism

Begin Statements

Cobegin Statements

Chapel Documentation 1.16
Docs » Chapel Documentation
View page source
Docs » Using Chapel
Docs » Primers » Task Parallelism
Previous



Tools: Now

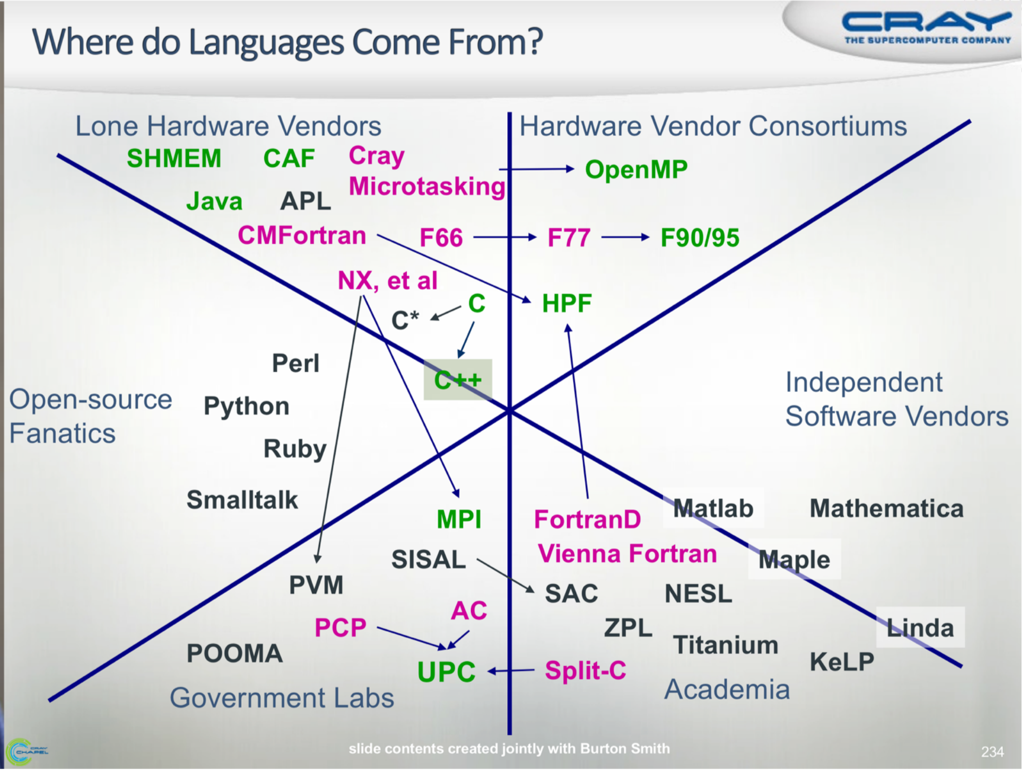
Now:

- **highlighting modes** for emacs, vim, atom, …
- **chpldoc:** documentation tool
- **mason:** package manager
- **c2chapel:** interoperability aid
- **chpltags:** helps search Chapel code
- **bash tab completion:** command-line help
- **chplvis:** performance visualizer / debugger

We're continuing our work to support and improve Chapel

We're looking for the next generation of Chapel users, as well as concrete use cases for AI / ML

In Memory of Burton Smith



THE SUPERCOMPUTER COMPANY

Where do Languages Come From?

Lone Hardware Vendors

SHMEM CAF Cray Microtasking

Java APL

CMFortran

NX, et al

Hardware Vendor Consortiums

OpenMP

F66 → F77 → F90/95

HPF

C C* C++

Open-source Fanatics

Perl

Python

Ruby

Smalltalk

Independent Software Vendors

Matlab Mathematica

Maple

Linda

KeLP

MPI

SISAL

PVM

AC

PCP

POOMA

UPC

Government Labs

FortranD

Vienna Fortran

SAC NESL

ZPL

Titanium

Split-C

Academia

slide contents created jointly with Burton Smith

234

CHIUW 2018: Agenda (chapel-lang.org/CHIUW2018.html)

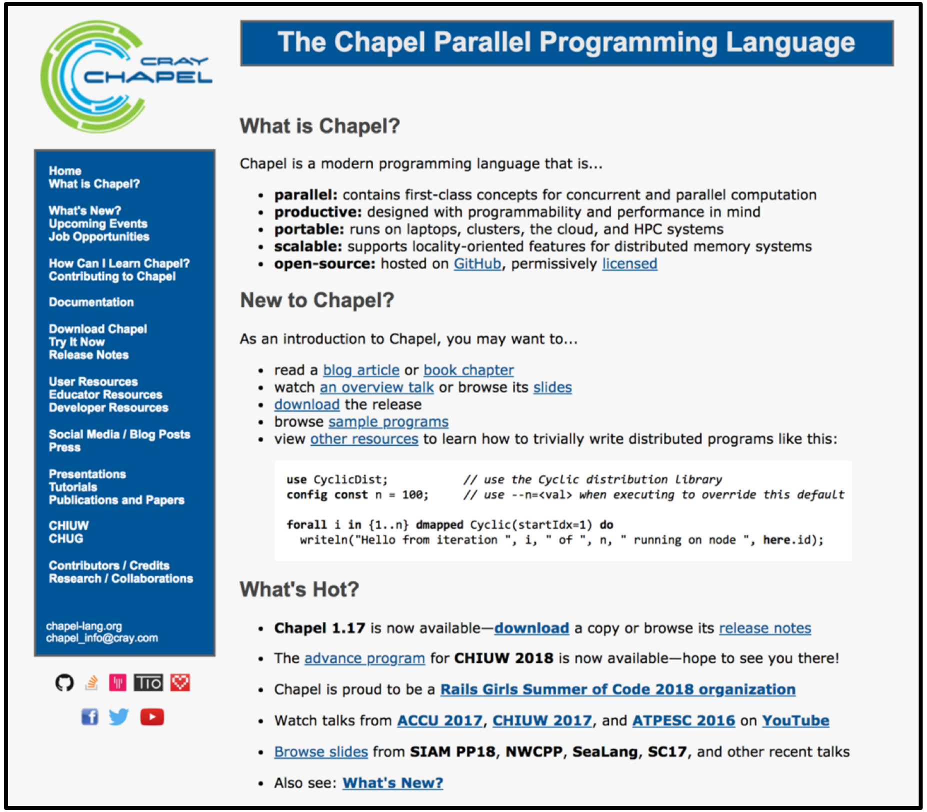
8:30:	Chapel 101 (optional)
9:00:	Welcome, State of the Project
9:30:	Break
10:00:	Talks: Applications of Chapel
11:00:	Quick Break
11:10:	Talks: Chapel Design and Evolution
12:10:	Lunch
1:40:	Keynote Talk: “Why Languages Matter”, Kathy Yelick
2:40:	Talks: Chapel Performance
3:00:	Break
3:30:	Talks: Tools for Chapel
4:30:	Lightning Talks and Flash Discussions
5:30:	Wrap-up / Head to Dinner

Chapel Resources

Chapel Central

https://chapel-lang.org

- downloads
- documentation
- resources
- presentations
- papers



The Chapel Parallel Programming Language

Home
What is Chapel?

What's New?
Upcoming Events
Job Opportunities

How Can I Learn Chapel?
Contributing to Chapel

Documentation

Download Chapel
Try It Now
Release Notes

User Resources
Educator Resources
Developer Resources

Social Media / Blog Posts
Press

Presentations
Tutorials
Publications and Papers

CHIUW
CHUG

Contributors / Credits
Research / Collaborations

chapel-lang.org
chapel_info@cray.com

What is Chapel?

Chapel is a modern programming language that is...

- **parallel:** contains first-class concepts for concurrent and parallel computation
- **productive:** designed with programmability and performance in mind
- **portable:** runs on laptops, clusters, the cloud, and HPC systems
- **scalable:** supports locality-oriented features for distributed memory systems
- **open-source:** hosted on GitHub, permissively licensed

New to Chapel?

As an introduction to Chapel, you may want to...

- read a blog article or book chapter
- watch an overview talk or browse its slides
- download the release
- browse sample programs
- view other resources to learn how to trivially write distributed programs like this:

```
use CyclicDist;           // use the Cyclic distribution Library
config const n = 100;     // use --n=<val> when executing to override this default

forall i in {1..n} dmapped Cyclic(startIdx=1) do
  writeln("Hello from iteration ", i, " of ", n, " running on node ", here.id);
```

What's Hot?

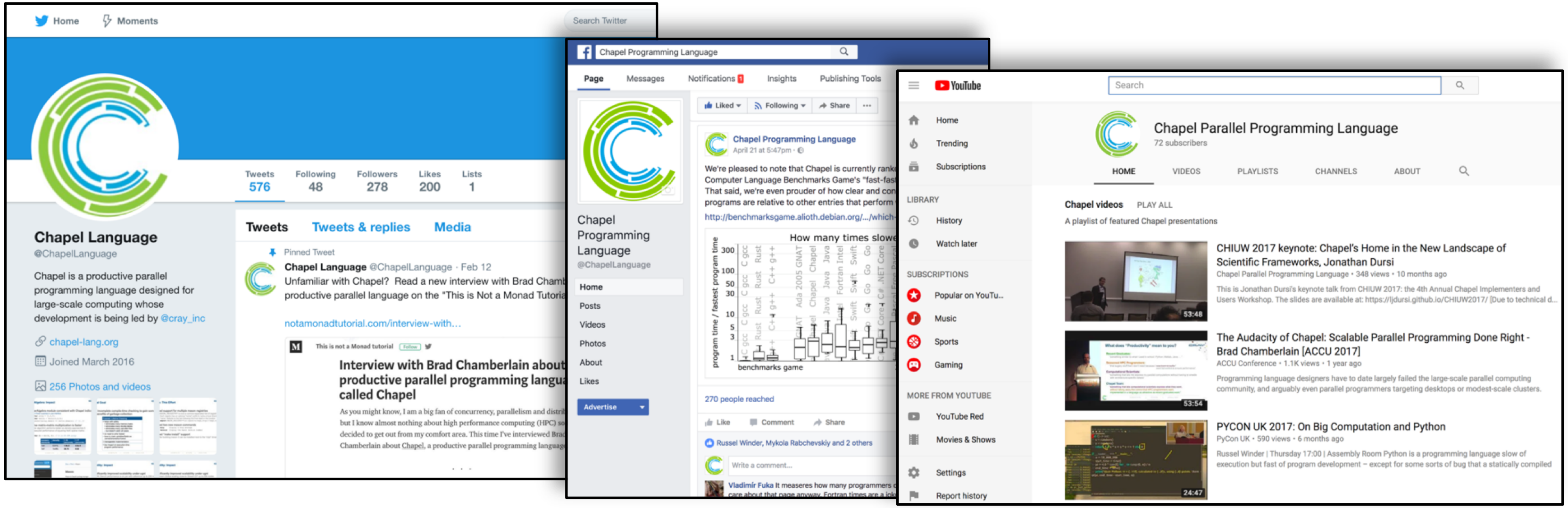
- **Chapel 1.17** is now available—**download** a copy or browse its release notes
- The advance program for **CHIUW 2018** is now available—hope to see you there!
- Chapel is proud to be a **Rails Girls Summer of Code 2018 organization**
- Watch talks from **ACCU 2017**, **CHIUW 2017**, and **ATPESC 2016** on **YouTube**
- Browse slides from **SIAM PP18**, **NWCPP**, **SeaLang**, **SC17**, and other recent talks
- Also see: **What's New?**

Chapel Social Media (no account required)

http://twitter.com/ChapelLanguage

http://facebook.com/ChapelLanguage

https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/



Chapel Language
@ChapelLanguage

Chapel is a productive parallel programming language designed for large-scale computing whose development is being led by @cray_inc

chapel-lang.org

Joined March 2016

256 Photos and videos

Chapel Language @ChapelLanguage · Feb 12
Unfamiliar with Chapel? Read a new interview with Brad Chamb
productive parallel language on the "This is Not a Monad Tutoria

notamonadtutorial.com/interview-with…

This is not a Monad tutorial Follow

Interview with Brad Chamberlain about productive parallel programming langua called Chapel

As you might know, I am a big fan of concurrency, parallelism and distri but I know almost nothing about high performance computing (HPC) so decided to get out from my comfort area. This time I've interviewed Brad Chamberlain about Chapel, a productive parallel programming languag

Chapel Programming Language
@ChapelLanguage

Chapel Programming Language
April 21 at 5:47pm ·

We're pleased to note that Chapel is currently ranke
Computer Language Benchmarks Game's "fast-fast
That said, we're even prouder of how clear and con
programs are relative to other entries that perform

http://benchmarksgame.alioth.debian.org/…/which-

Russel Winder, Mykola Rabchevskiy and 2 others

Vladimír Fuka It measeres how many programmers c
care about that page anyway. Fortran times are a joke

Chapel Parallel Programming Language
72 subscribers

Chapel videos
A playlist of featured Chapel presentations

CHIUW 2017 keynote: Chapel's Home in the New Landscape of Scientific Frameworks, Jonathan Dursi
Chapel Parallel Programming Language • 348 views • 10 months ago
This is Jonathan Dursi's keynote talk from CHIUW 2017: the 4th Annual Chapel Implementers and Users Workshop. The slides are available at: https://ljdursi.github.io/CHIUW2017/ [Due to technical d...

The Audacity of Chapel: Scalable Parallel Programming Done Right - Brad Chamberlain [ACCU 2017]
ACCU Conference • 1.1K views • 1 year ago
Programming language designers have to date largely failed the large-scale parallel computing community, and arguably even parallel programmers targeting desktops or modest-scale clusters.

PYCON UK 2017: On Big Computation and Python
PyCon UK • 590 views • 6 months ago
Russel Winder | Thursday 17:00 | Assembly Room Python is a programming language slow of execution but fast of program development – except for some sorts of bug that a statically compiled

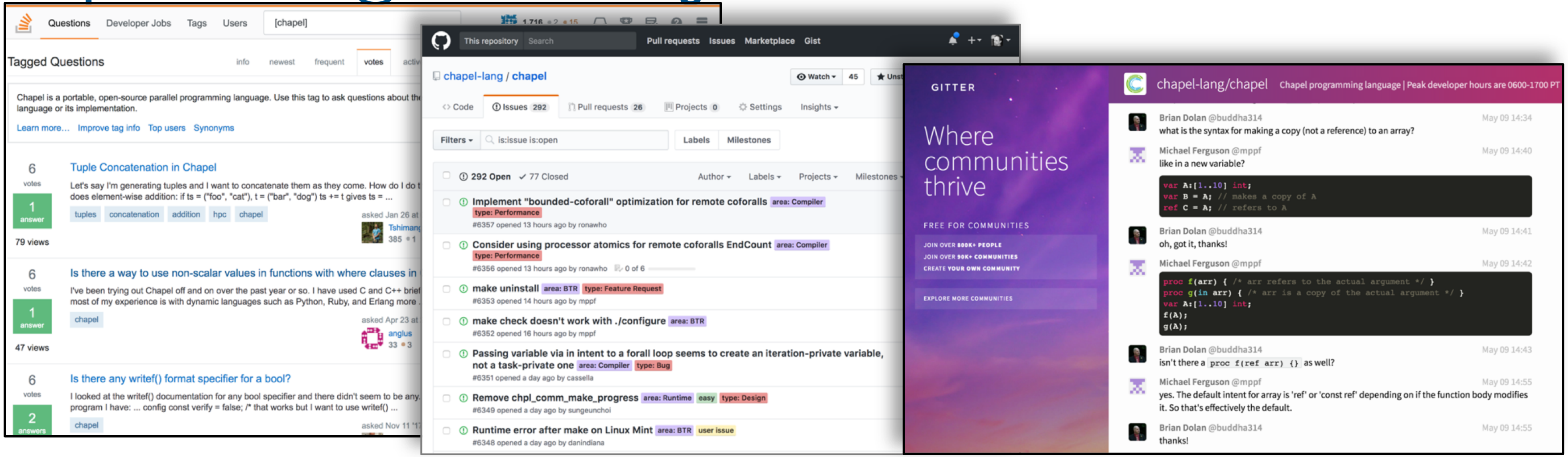
Chapel Community

https://stackoverflow.com/questions/tagged/chapel

https://github.com/chapel-lang/chapel/issues

https://gitter.im/chapel-lang/chapel

chapel-announce@lists.sourceforge.net



Tagged Questions

Chapel is a portable, open-source parallel programming language. Use this tag to ask questions about the language or its implementation.

Learn more… Improve tag info Top users Synonyms

6 votes, 1 answer, 79 views

Tuple Concatenation in Chapel

Let's say I'm generating tuples and I want to concatenate them as they come. How do I do t does element-wise addition: if ts = ("foo", "cat"), t = ("bar", "dog") ts += t gives ts = ...

tuples concatenation addition hpc chapel

asked Jan 26 at Tshimang 385 • 1

6 votes, 1 answer, 47 views

Is there a way to use non-scalar values in functions with where clauses in

I've been trying out Chapel off and on over the past year or so. I have used C and C++ brief most of my experience is with dynamic languages such as Python, Ruby, and Erlang more .

chapel

asked Apr 23 at anglus 33 • 3

6 votes, 2 answers

Is there any writef() format specifier for a bool?

I looked at the writef() documentation for any bool specifier and there didn't seem to be any. program I have: ... config const verify = false; /* that works but I want to use writef() ...

chapel

asked Nov 11 '17

chapel-lang / chapel

292 Open ✓ 77 Closed

- Implement "bounded-coforall" optimization for remote coforalls area: Compiler type: Performance
  #6357 opened 13 hours ago by ronawho
- Consider using processor atomics for remote coforalls EndCount area: Compiler type: Performance
  #6356 opened 13 hours ago by ronawho 0 of 6
- make uninstall area: BTR type: Feature Request
  #6353 opened 14 hours ago by mppf
- make check doesn't work with ./configure area: BTR
  #6352 opened 16 hours ago by mppf
- Passing variable via in intent to a forall loop seems to create an iteration-private variable, not a task-private one area: Compiler type: Bug
  #6351 opened a day ago by cassella
- Remove chpl_comm_make_progress area: Runtime easy type: Design
  #6349 opened a day ago by sungeunchoi
- Runtime error after make on Linux Mint area: BTR user issue
  #6348 opened a day ago by danindiana

chapel-lang/chapel Chapel programming language | Peak developer hours are 0600-1700 PT

Brian Dolan @buddha314 May 09 14:34
what is the syntax for making a copy (not a reference) to an array?

Michael Ferguson @mppf May 09 14:40
like in a new variable?

```
var A:[1..10] int;
var B = A; // makes a copy of A
ref C = A; // refers to A
```

Brian Dolan @buddha314 May 09 14:41
oh, got it, thanks!

Michael Ferguson @mppf May 09 14:42

```
proc f(arr) { /* arr refers to the actual argument */ }
proc g(in arr) { /* arr is a copy of the actual argument */ }
var A:[1..10] int;
f(A);
g(A);
```

Brian Dolan @buddha314 May 09 14:43
isn't there a `proc f(ref arr) {}` as well?

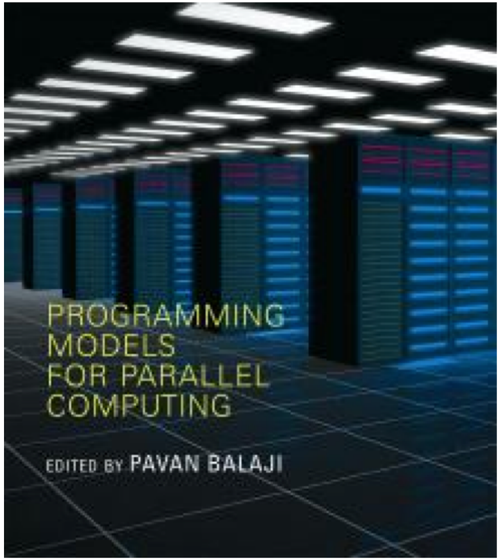
Michael Ferguson @mppf May 09 14:55
yes. The default intent for array is 'ref' or 'const ref' depending on if the function body modifies it. So that's effectively the default.

Brian Dolan @buddha314 May 09 14:55
thanks!

Suggested Reading (healthy attention spans)

Chapel chapter from ***Programming Models for Parallel Computing***

- a detailed overview of Chapel's history, motivating themes, features
- published by MIT Press, November 2015
- edited by Pavan Balaji (Argonne)
- chapter is also available online



Other Chapel papers/publications available at https://chapel-lang.org/papers.html

Suggested Reading (short attention spans)

CHIUW 2017: Surveying the Chapel Landscape, Cray Blog, July 2017.

- *a run-down of recent events (as of 2017)*

Chapel: Productive Parallel Programming, Cray Blog, May 2013.

- *a short-and-sweet introduction to Chapel*

Six Ways to Say "Hello" in Chapel **(parts 1, 2, 3)**, Cray Blog, Sep-Oct 2015.

- *a series of articles illustrating the basics of parallelism and locality in Chapel*

Why Chapel? **(parts 1, 2, 3)**, Cray Blog, Jun-Oct 2014.

- *a series of articles answering common questions about why we are pursuing Chapel in spite of the inherent challenges*

[Ten] Myths About Scalable Programming Languages, IEEE TCSC Blog (index available on chapel-lang.org "blog posts" page), Apr-Nov 2012.

- *a series of technical opinion pieces designed to argue against standard reasons given for not developing high-level parallel languages*

Where to..

Submit bug reports:

GitHub issues for chapel-lang/chapel: public bug forum
chapel_bugs@cray.com: for reporting non-public bugs

Ask User-Oriented Questions:

StackOverflow: when appropriate / other users might care
Gitter (chapel-lang/chapel): community chat with archives
chapel-users@lists.sourceforge.net: user discussions

Discuss Chapel development

chapel-developers@lists.sourceforge.net: developer discussions
GitHub issues for chapel-lang/chapel: for feature requests, design discussions

Discuss Chapel's use in education

chapel-education@lists.sourceforge.net: educator discussions

Directly contact Chapel team at Cray: chapel_info@cray.com

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

CRAY CHAPEL

CRAY

THE SUPERCOMPUTER COMPANY