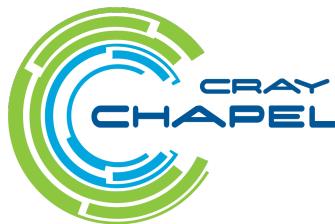


Chapel Update

Chapel Team, Cray Inc.

SC17 Briefings

November 2017



Safe Harbor Statement



This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.



COMPUTE

| STORE

| ANALYZE

What is Chapel?



Chapel: A productive parallel programming language

- portable
- open-source
- a collaborative effort



Goals:

- Support general parallel programming
 - “any parallel algorithm on any parallel hardware”
- Make parallel programming at scale 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.”



COMPUTE

| STORE

| ANALYZE

Chapel and Other Languages

Chapel strives 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 Challenge

Q: So why don't we already have such a language already?

A: ~~Technical challenges?~~

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

A: Due to a lack, in HPC, of...

- ...long-term efforts
- ...resources
- ...co-design between developers and users
- ...community will
- ...patience

Chapel is our attempt to reverse this trend



COMPUTE

| STORE

| ANALYZE

A Brief History of Chapel

CRAY®

2002–2012: DARPA HPCS

- Cray pursued a new language, Chapel
- Delivered a compelling research prototype



2013–2018: “the 5-year push”

- Based on positive user response, Cray set out to improve Chapel
 - **performance** improvements
 - fixing / improving **features**
 - maintaining / improving **portability**
 - nurturing the **community**
 - exploring **governance** models

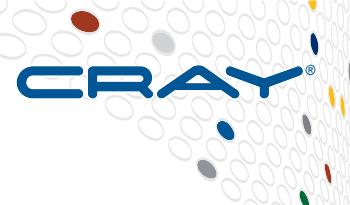


COMPUTE

STORE

ANALYZE

The Chapel Team at Cray (May 2017)



14 full-time employees + 2 summer interns + 2-4 GSoC students



COMPUTE

| STORE

| ANALYZE

Copyright 2017 Cray Inc.

Chapel Community R&D Partners



Lawrence Berkeley
National Laboratory



Sandia National Laboratories



Yale

(and several others...)

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



COMPUTE

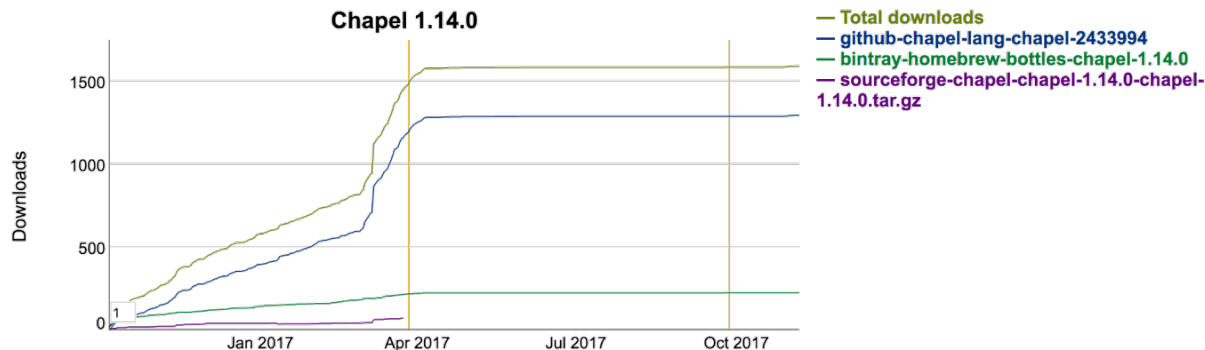
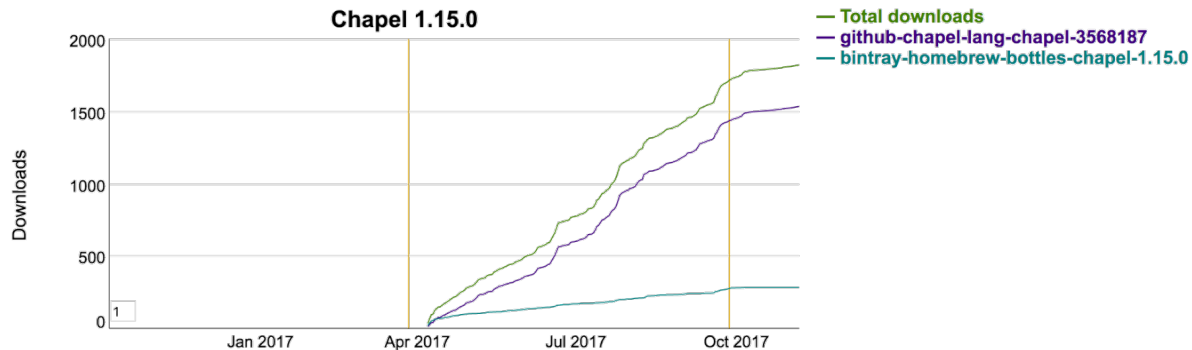
STORE

ANALYZE

Highlights of the Past Year or 4½



The Year in Downloads (~3400 total, a record)



COMPUTE

STORE

ANALYZE

Copyright 2017 Cray Inc.

Computer Language Benchmarks Game (CLBG)

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?

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

{ for researchers } fast-faster-fastest
stories

Website supporting cross-language comparisons

- 13 toy benchmark programs x ~28 languages x many implementations
- exercise key computational idioms
- specific approach prescribed



COMPUTE

STORE

ANALYZE

Copyright 2017 Cray Inc.

Computer Language Benchmarks Game (CLBG)

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?

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

{ for researchers } fast-faster-fastest
 stories

Chapel's approach to the CLBG:

- striving for elegance over heroism
- ideally: “Want to learn how program xyz works? Read the Chapel version.”



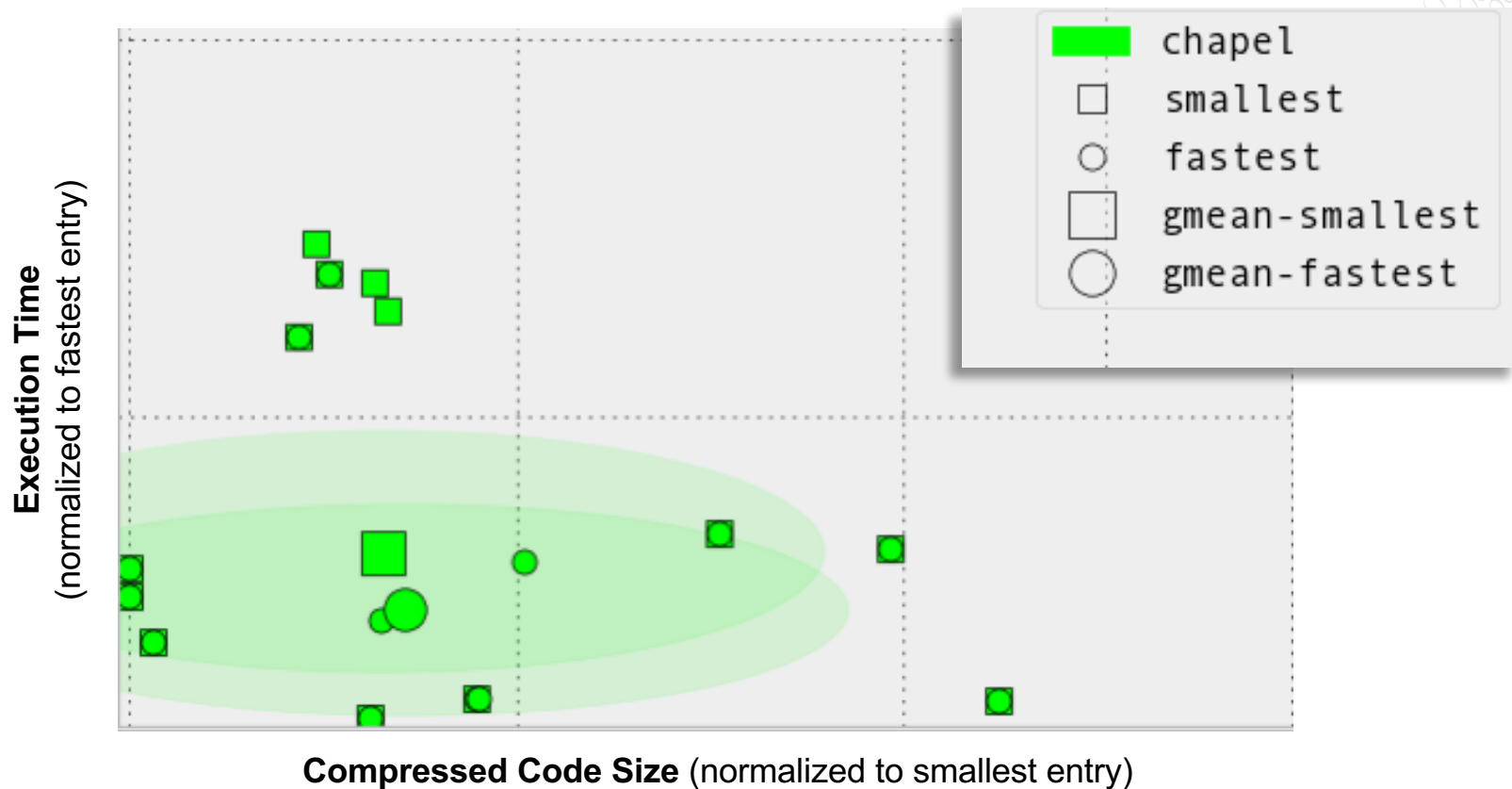
COMPUTE

STORE

ANALYZE

Copyright 2017 Cray Inc.

Scatter plots of CLBG code size x speed



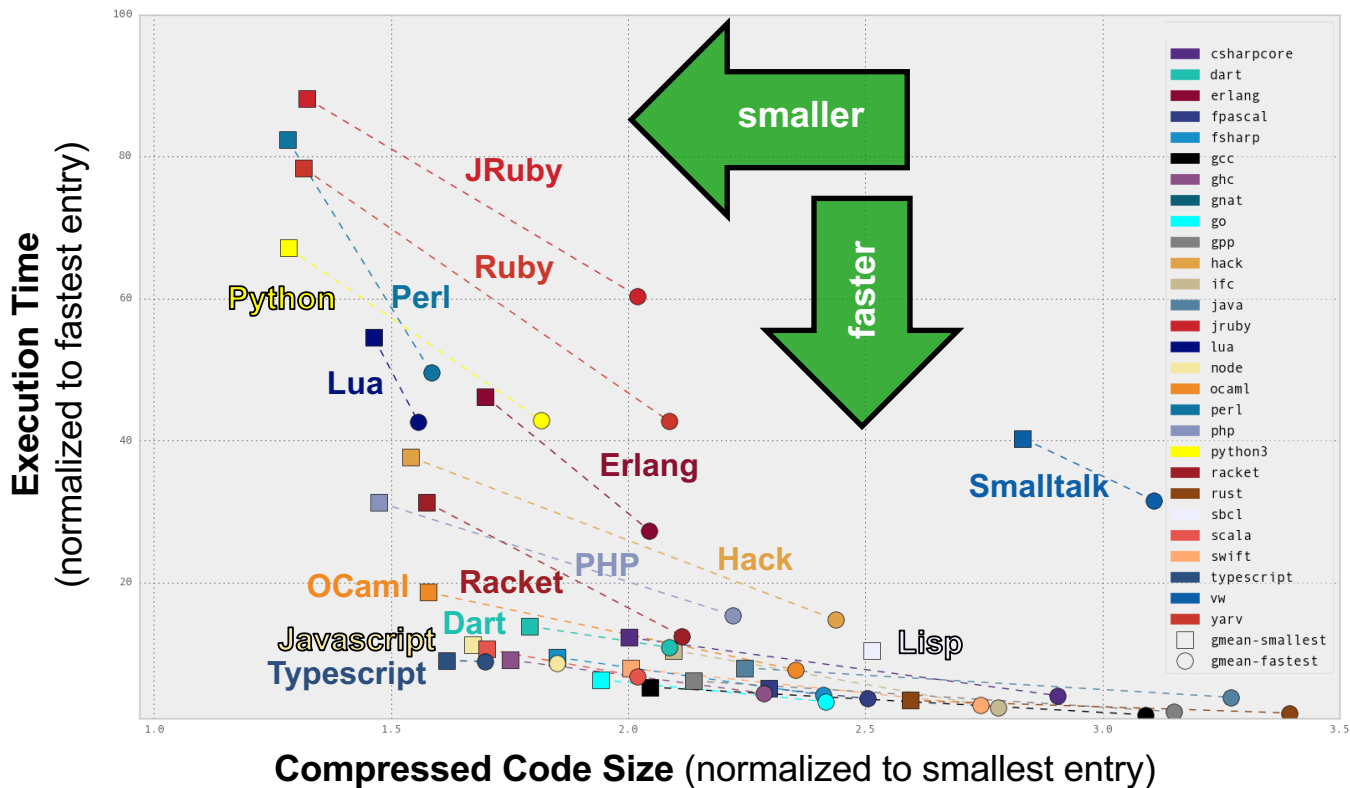
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(Oct 2017 standings)



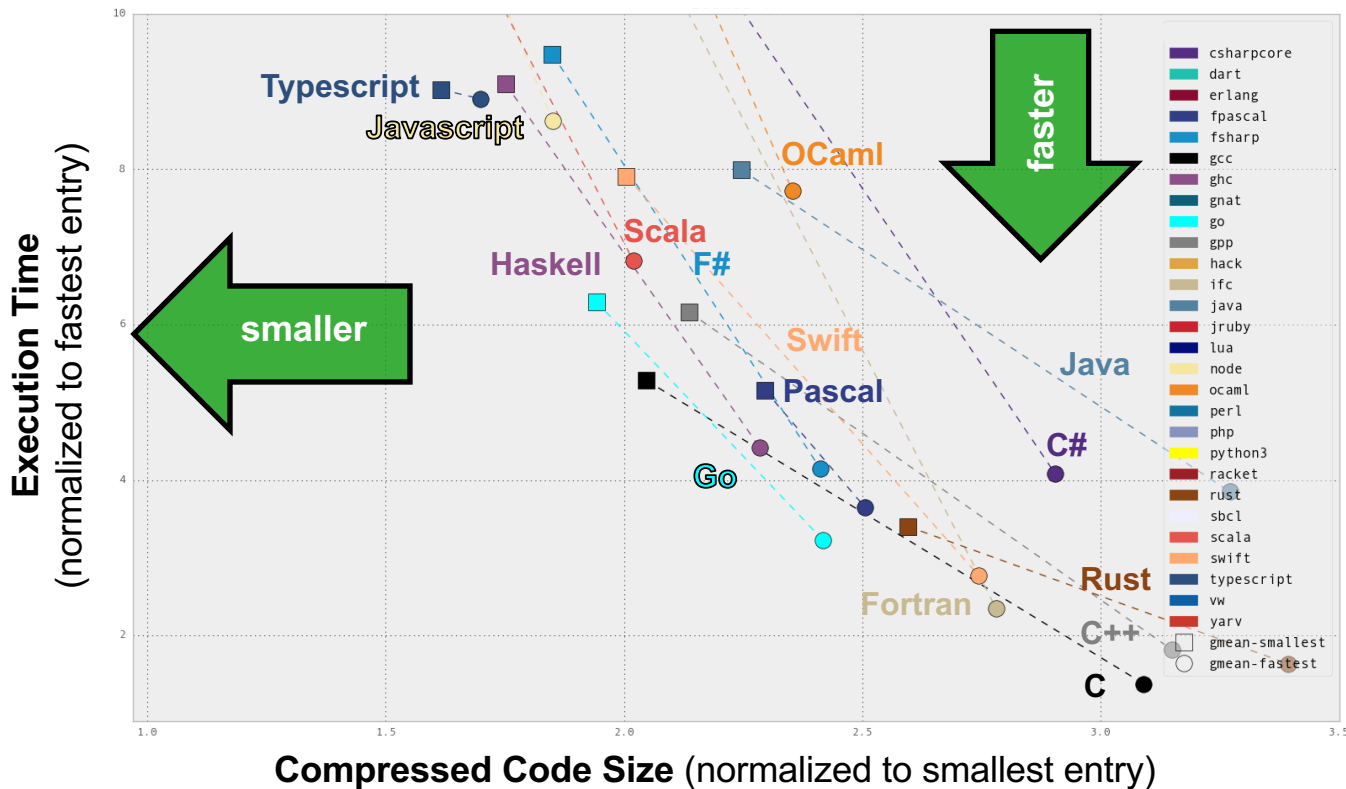
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(Oct 2017 standings)



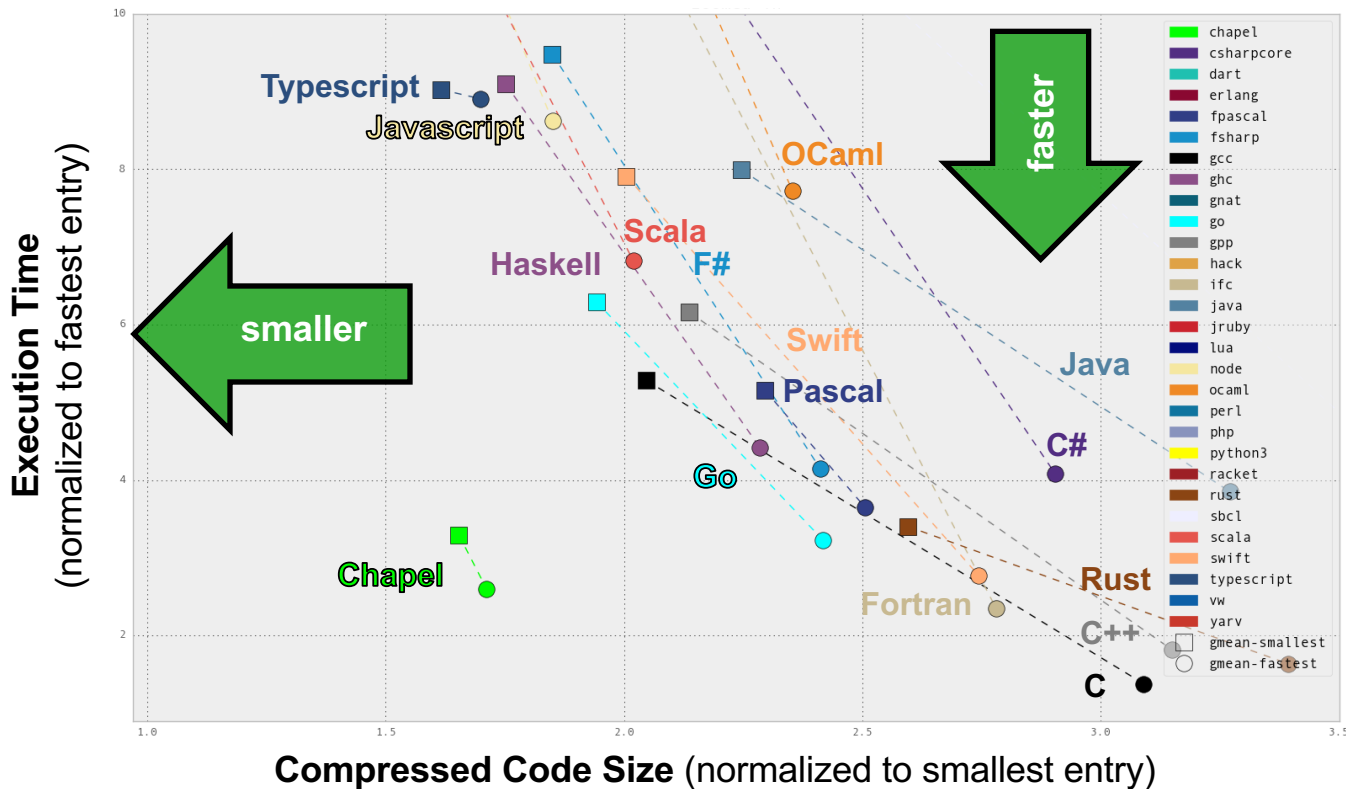
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(Oct 2017 standings)



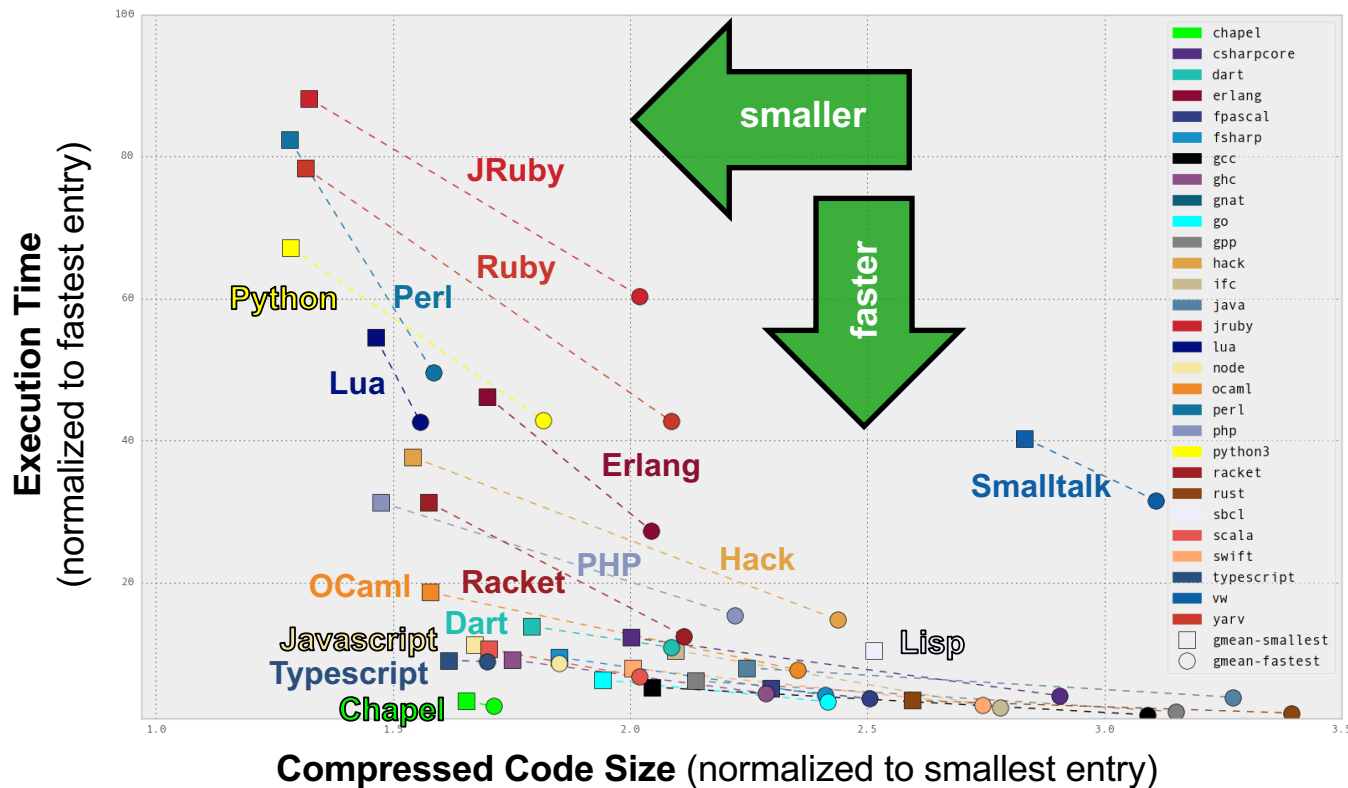
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(Oct 2017 standings)



CLBG: Qualitative Comparisons



Can also browse program source code (*but this requires actual thought!*):

```
proc main() {
  printColorEquations();

  const group1 = {i in 1..popSize1} new Chameneos(i, ((i-1)%3):Color);
  const group2 = {i in 1..popSize2} new Chameneos(i, colors10[i]);

  cobegin {
    holdMeetings(group1, n);
    holdMeetings(group2, n);
  }

  print(group1);
  print(group2);

  for c in group1 do delete c;
  for c in group2 do delete c;
}

//
// Print the results of getNewColor() for all color pairs.
//
proc printColorEquations() {
  for c1 in Color do
    for c2 in Color do
      writeln(c1, " + ", c2, " -> ", getNewColor(c1, c2));
    writeln();
  }

  //
  // Hold meetings among the population by creating a shared meeting
  // place, and then creating per-chameneos tasks to have meetings.
  //
  proc holdMeetings(population, numMeetings) {
    const place = new MeetingPlace(numMeetings);

    coforall c in population do // create a task per chameneos
      c.haveMeetings(place, population);

    delete place;
  }
}
```

excerpt from 1210 gz Chapel entry

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
  cpu_set_t      active_cpus;
  FILE*          f;
  char           buf [2048];
  char const*    pos;
  int            cpu_idx;
  int            physical_id;
  int            core_id;
  int            cpu_cores;
  int            apic_id;
  size_t         cpu_count;
  size_t         i;

  char const*    processor_str   = "processor";
  size_t         processor_str_len = strlen(processor_str);
  char const*    physical_id_str = "physical id";
  size_t         physical_id_str_len = strlen(physical_id_str);
  char const*    core_id_str     = "core id";
  size_t         core_id_str_len = strlen(core_id_str);
  char const*    cpu_cores_str   = "cpu cores";
  size_t         cpu_cores_str_len = strlen(cpu_cores_str);

  CPU_ZERO(&active_cpus);
  sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
  cpu_count = 0;
  for (i = 0; i != CPU_SETSIZE; i += 1)
  {
    if (CPU_ISSET(i, &active_cpus))
    {
      cpu_count += 1;
    }
  }

  if (cpu_count == 1)
  {
    is_smp[0] = 0;
    return;
  }

  is_smp[0] = 1;
  CPU_ZERO(affinity1);
```

excerpt from 2863 gz C gcc entry



CLBG: Qualitative Comparisons



Can also browse program source code (*but this requires actual thought!*):

```
proc main() {
  printColorEquations();

  const group1 = [i in 1..popSize1] new Chameneos(i, c);
  const group2 = [i in 1..popSize2] new Chameneos(i, c);

  cobegin {
    holdMeetings(group1, n);
    holdMeetings(group2, n);
  }

  print(group1);
  print(group2);

  for c in group1 do delete c;
  for c in group2 do delete c;
}

//
// Print the results of getNewColor() for all colors
//
proc printColorEquations() {
  for c1 in Color do
    for c2 in Color do
      writeln(c1, " + ", c2, " = ", getNewColor(c1, c2));
    }
  writeln();
}

//
// Hold meetings among the population by creating a shared place,
// and then creating per-chameneos tasks to have meetings
//
proc holdMeetings(population, numMeetings) {
  const place = new MeetingPlace(numMeetings);

  coforall c in population do // create a task for each
    c.haveMeetings(place, population);

  delete place;
}
```

```
void getAffinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2) {
  active_cpus;
  f;
  buf [2048];
  pos;
  cpu_idx;
  physical_id;
  core_id;
  cpu_cores;
  apic_id;
  cpu_count;
  i;

  processor_str = "processor";
  processor_str_len = strlen(processor_str);
  physical_id_str = "physical id";
  physical_id_str_len = strlen(physical_id_str);
  core_id_str = "core id";
  core_id_str_len = strlen(core_id_str);

  is_smp[0] = 1;
  CPU_ZERO(affinity1);
```

excerpt from 1210 gz Chapel entry

excerpt from 2863 gz C gcc entry



COMPUTE

STORE

ANALYZE

CLBG: Qualitative Comparisons



Can also browse program source code (*but this requires actual thought!*):

```
proc main() {  
    char const*      core_id_str      = "core id"  
    size_t           core_id_str_len  = strlen(co  
    char const*      cpu_cores_str    = "cpu core  
    size_t           cpu_cores_str_len = strlen(cpu  
  
    CPU_ZERO(&active_cpus);  
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);  
    cpu_count = 0;  
    for (i = 0; i != CPU_SETSIZE; i += 1)  
    {  
        if (CPU_ISSET(i, &active_cpus))  
        {  
            cpu_count += 1;  
        }  
    }  
  
    if (cpu_count == 1)  
    {  
        is_smp[0] = 0;  
        return;  
    }  
}
```

excerpt from 1210 gz Chapel entry

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)  
{  
    cpu_set_t      active_cpus;  
    FILE*          f;  
    char           buf [2048];  
    char const*    pos;  
    int            cpu_idx;  
    int            physical_id;  
    int            core_id;  
    int            cpu_cores;  
    int            apic_id;  
    size_t         cpu_count;  
    size_t         i;  
  
    char const*    processor_str      = "processor";  
    size_t         processor_str_len  = strlen(processor_str);  
    char const*    physical_id_str    = "physical id";  
    size_t         physical_id_str_len = strlen(physical_id_str);  
    char const*    core_id_str        = "core id";  
    size_t         core_id_str_len    = strlen(core_id_str);  
    char const*    cpu_cores_str      = "cpu cores";  
    size_t         cpu_cores_str_len  = strlen(cpu_cores_str);  
  
    CPU_ZERO(&active_cpus);  
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);  
    cpu_count = 0;  
    for (i = 0; i != CPU_SETSIZE; i += 1)  
    {  
        if (CPU_ISSET(i, &active_cpus))  
        {  
            cpu_count += 1;  
        }  
    }  
  
    if (cpu_count == 1)  
    {  
        is_smp[0] = 0;  
        return;  
    }  
  
    is_smp[0] = 1;  
    CPU_ZERO(affinity1);  
}
```

excerpt from 2863 gz C gcc entry

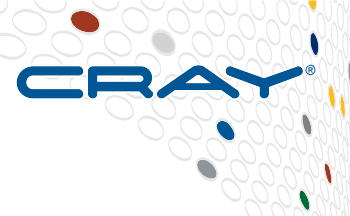


COMPUTE

STORE

ANALYZE

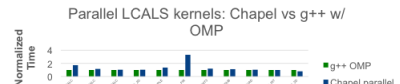
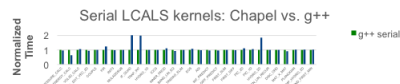
Chapel Performance: HPC Benchmarks



LCALS: Chapel vs. C + OpenMP



Shared memory performance competitive with hand-coded



LCALS

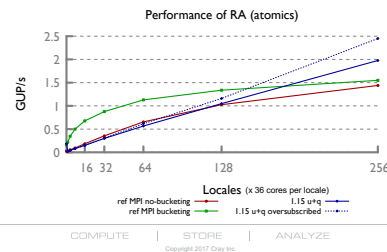
HPCC RA

STREAM
Triad

ISx

PRK
Stencil

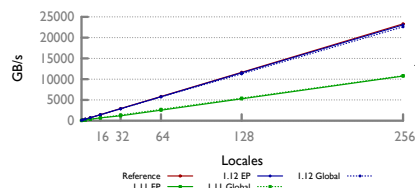
HPCC RA Performance: Chapel vs. MPI



HPCC Stream Triad: Chapel vs. MPI+OpenMP



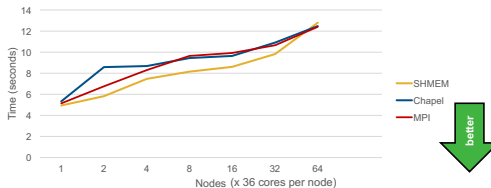
Performance of STREAM
(GASNet/multi+qthreads)



ISx Performance: Chapel vs. MPI, SHMEM



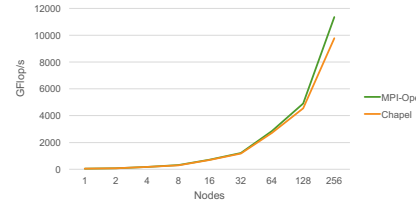
ISx weakISO Total Time



Stencil PRK Scalability



Stencil PRK (weak scaling)



COMPUTE

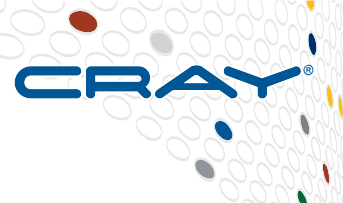
STORE

ANALYZE

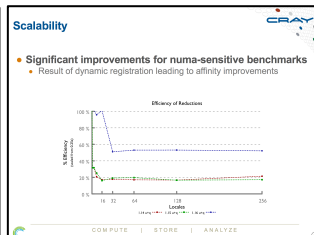
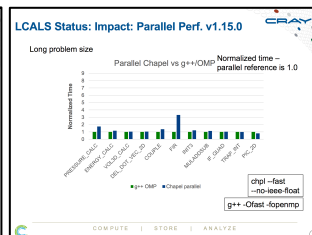
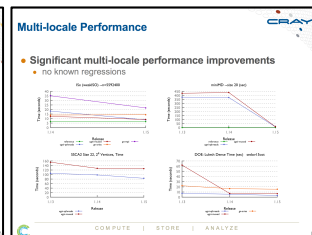
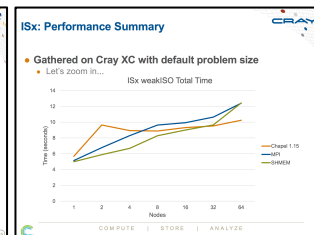
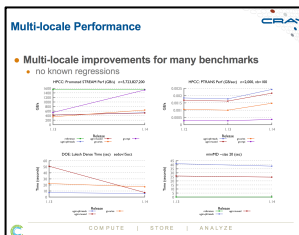
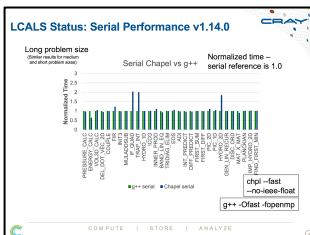
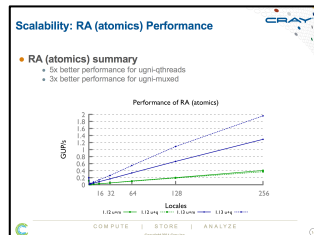
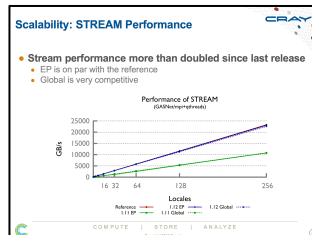
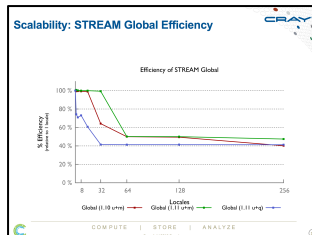
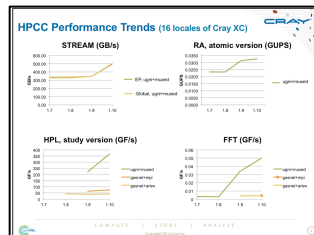
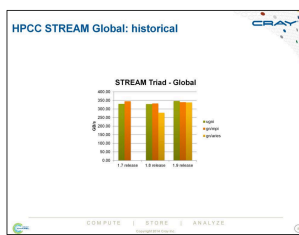
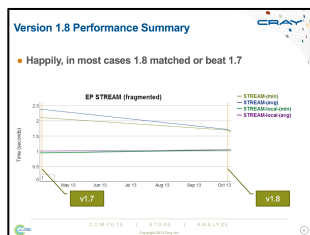
Copyright 2017 Cray Inc.

Nightly performance graphs online
at: <https://chapel-lang.org/perf>

Performance: Progress Since HPCS



Significant improvements throughout the past 4½ years



COMPUTE

STORE

ANALYZE

Copyright 2017 Cray Inc.



Library Highlights: Past Year

New libraries:

- Crypto
- Collections: DistributedBag, DistributedDeque
- DateTime
- DistributedIters
- Futures
- LinearAlgebra (ongoing effort)
- OwnedObject / SharedObject
- TOML (ongoing effort)

Library improvements:

- BLAS
- FFTW
- MPI
- ZMQ
- various: added 'throw'ing versions of several routines





Library Highlights: Past Year

New libraries:

- Crypto
- Collections: DistributedBag, DistributedDeque
- DateTime
- DistributedIters
- Futures
- LinearAlgebra (ongoing effort)
- OwnedObject / SharedObject
- TOML (ongoing effort)

Library improvements:

- BLAS
- FFTW
- MPI
- ZMQ
- various: added 'throw'ing versions of several routines

(developed by GSoC student)
(developed by Cray intern)
(externally developed)



Libraries: Progress Since HPCS



Then: ~25 modules, documented via 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 'rand' 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
// the module.
//
//---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:qio_regexp_options_t, ref compiled:qio_regexp_t);
extern proc qio_regexp_create_compile_flags(str:string, strlen:int(64), flags:string, 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_regexp_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):bool;

//---F1 Regexp.chpl Top L1 (Chapel/L Abbrev)-----
```



Libraries: Progress Since HPCS



Now: ~58 documented modules, many user-contributed

The image displays two overlapping screenshots of the Chapel Documentation 1.16 website. The left screenshot shows the 'Standard Modules' page, which lists modules that are considered part of the standard library. The right screenshot shows the 'Package Modules' page, which lists modules that are currently live outside of the standard library.

Standard Modules

Standard modules are those which describe features that are considered part of the standard library.

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

- `Assert`
- `Barrier`
- `Barriers`
- `BigIntegers`
- `BitOps`
- `Buffers`
- `CommDiagnostics`
- `DateTime`
- `Dynamics`
- `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`
- `DistributedIterators`
- `FFTW`
- `FFTW_MT`
- `Futures`
- `HDFS`
- `HDF5`
- `HDF5Iterator`
- `LAPACK`
- `LinearAlgebra`
- `MPI`
- `Norm`
- `OwnedObject`
- `RangeChunk`
- `RecordParser`
- `Search`
- `SharedObject`
- `Sort`
- `VisualDebug`
- `ZMQ`



COMPUTE

STORE

ANALYZE

Copyright 2017 Cray Inc.

Documentation: Progress Since HPCS



Now: > 200 modern, hyperlinked, web-based doc pages

The image displays three overlapping screenshots of the Cray Chapel documentation website, illustrating its modern, hyperlinked, and web-based nature.

Top Screenshot (Main Index): Shows the "Chapel Documentation 1.16" header with a search bar. The left sidebar lists categories: "COMPILING AND RUNNING CHAPEL" (Quickstart Instructions, Using Chapel, Platform-Specific Notes, Technical Notes, Tools), "WRITING CHAPEL PROGRAMS" (Quick Reference, Hello World Variants, Primers), and "LANGUAGE HISTORY" (Language Specification, Built-in Types and Functions, Standard Modules, Package Modules, Standard Layouts and Distributions, Chapel Users Guide (WIP)). The main content area is titled "Chapel Documentation" and lists links to "Compiling and Running Chapel", "Writing Chapel Programs", and "Language History".

Middle Screenshot ("Using Chapel"): Shows the "Using Chapel" page. The left sidebar lists "Chapel Prerequisites", "Setting up Your Environment for Chapel", "Building Chapel", "Chapel Man Page", "Executing Chapel Programs", "Multilocale Chapel Execution", "Chapel Launchers", "Chapel Tasks", "Debugging Chapel Programs", and "Reporting Chapel Issues". The main content area is titled "Using Chapel" and lists links to "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", and "Reporting Chapel Issues".

Bottom Screenshot ("Task Parallelism"): Shows the "Task Parallelism" page. The left sidebar lists "Language Basics", "Iterators", "Task Parallelism", "Begin Statements", "Cobegin Statements", "Sync / Singles", "Atoms", "Locality", and "Data Parallelism". The main content area is titled "Task Parallelism" and includes a "View taskParallel.chpl on GitHub" link. It contains a code snippet:

```
config const n = 30; // Used for the coforall loop
```

 and a "Begin Statements" section with a code snippet:

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

 and a "Cobegin Statements" section.



COMPUTE

STORE

ANALYZE

Copyright 2017 Cray Inc.

Tool Highlights: Past Year



- Initial version of Chapel package manager, 'mason'
 - modeled after Cargo, enables community to develop and share decentralized libraries

```
> mason build
```

```
Updating mason-registry
```

```
Downloading dependency: Bob-1.1.0
```

```
Downloading dependency: Alice-0.3.0
```

- First release of 'c2chapel' tool
 - converts C header files to Chapel 'extern' declarations

C99

```
struct allInts {  
  int a;  
  unsigned int b;  
  long long c;  
};
```

```
void msg(const char* fmt);
```

Chapel

```
extern record allInts {  
  var a : c_int;  
  var b : c_uint;  
  var c : c_longlong;  
}
```

```
extern proc msg(fmt : c_string) : void;
```



What's Next?





What's Next? (Big Ticket Items)

- **Work towards Chapel 2.0 release**
 - goal: no changes that break backwards compatibility
- **LLVM back-end by default**
- **GPU support**
- **Support for delete-free computation**
- **Application studies / application partnerships**



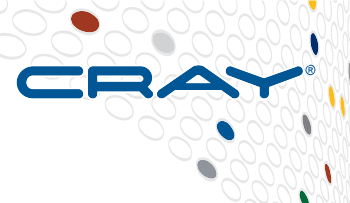
COMPUTE

| STORE

| ANALYZE

Copyright 2017 Cray Inc.

Crossing the Stream of Adoption



Research Prototype

Adopted in Production

Next MET Office model

Next DOE app

[your production app here]

What are the next stepping stones?

Who's interested in meeting us partway?

MiniMD

CoMD

ISx

CLBG

PRK Stencil

Codes from startups

RA

LULESH

Stream

LCALS

Time-to-science academic codes



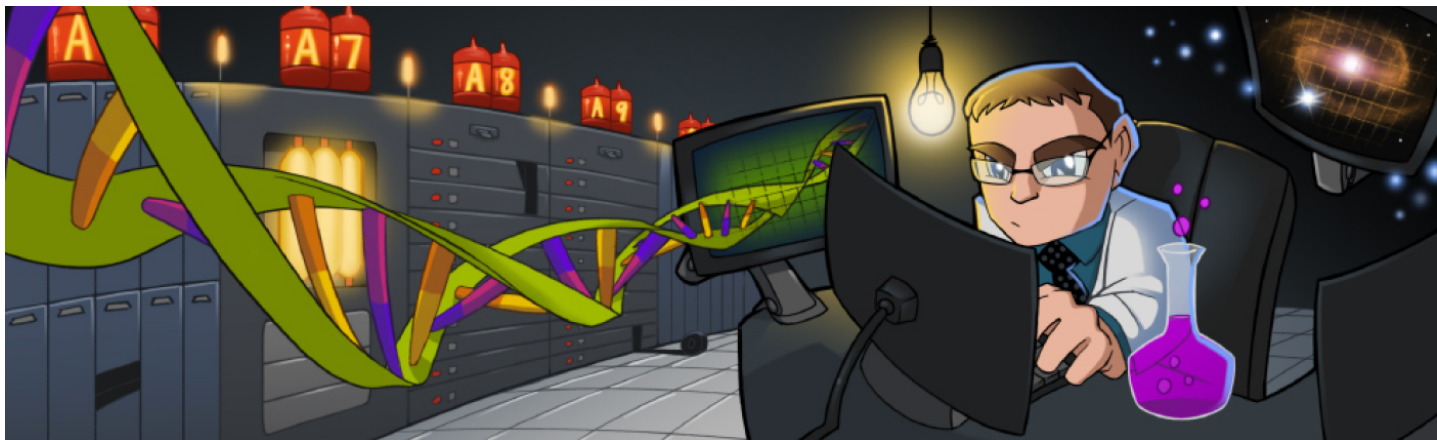
COMPUTE

STORE

ANALYZE

Chapel's Home in the Landscape of New Scientific Computing Languages (and what it can learn from the neighbours)

Jonathan Dursi, *The Hospital for Sick Children, Toronto*



COMPUTE

STORE

ANALYZE

Quote from CHI UW 2017 keynote



“My opinion as an outsider...is that Chapel is important, Chapel is mature, and Chapel is just getting started.

“If the scientific community is going to have frameworks for solving scientific problems that are actually designed for our problems, they’re going to come from a project like Chapel.

“And the thing about Chapel is that the set of all things that are ‘projects like Chapel’ is ‘Chapel.’”

—Jonathan Dursi

Chapel’s Home in the New Landscape of Scientific Frameworks

(and what it can learn from the neighbours)

CHI UW 2017 keynote

<https://ljdursi.github.io/CHI UW2017> / <https://www.youtube.com/watch?v=xj0rwdLOR4U>



COMPUTE


| STORE

| ANALYZE

Copyright 2017 Cray Inc.

Chapel Resources





Home
Chapel Overview

What's New?
Upcoming Events
Job Opportunities

How Can I Learn 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

CHIUV
CHUG
Lightning Talks

Contributors / Credits
Research Groups
License

chapel-lang.org
chapel_info@cray.com

The Chapel Parallel Programming Language

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 ./a.out --n=<val> 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 1.16** is now available—[download](#) a copy today!
- The **CHIUV 2018** [call for participation](#) is now available!
- A recent [Cray blog post](#) reports on highlights from CHIUV 2017.
- Chapel is now one of the supported languages on [Try It Online!](#)
- Watch talks from [ACCU 2017](#), [CHIUV 2017](#), and [ATPESC 2016](#) on [YouTube](#).
- [Browse slides](#) from **PADAL**, **EAGE**, **EMBRACE**, **ACCU**, and other recent talks.
- See also: [What's New?](#)



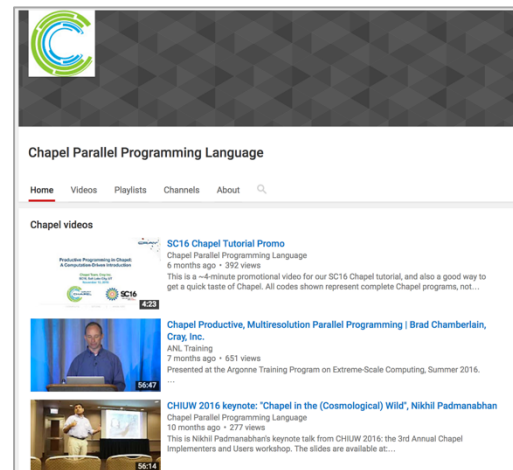
How to Stalk Chapel



<http://facebook.com/ChapelLanguage>

<http://twitter.com/ChapelLanguage>

[https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/
chapel-announce@lists.sourceforge.net](https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/chapel-announce@lists.sourceforge.net)



COMPUTE

STORE

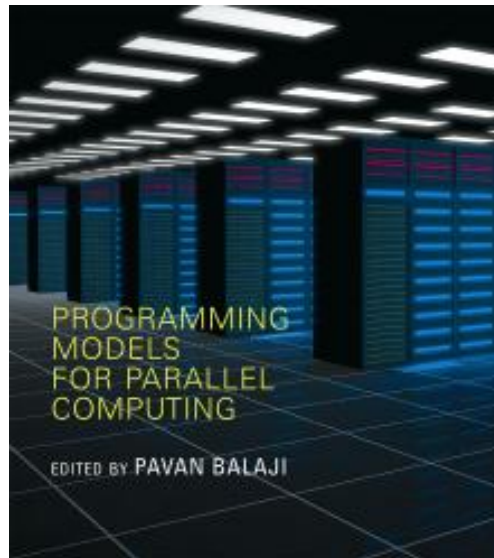
ANALYZE

Copyright 2017 Cray Inc.

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 now also available [online](#)



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



COMPUTE

| STORE

| ANALYZE

Suggested Reading (short attention spans)



[CHIUW 2017: Surveying the Chapel Landscape](#), [Cray Blog](#), July 2017.

- *a run-down of recent events*

[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*



Chapel StackOverflow and GitHub Issues



stackoverflow Questions Jobs Documentation TAGS Users [chapel] Log In Sign Up

Tagged Questions info newest frequent votes active

Chapel, the Cascade High Productivity Language, is a parallel programming language developed by Cray.
learn more... top users synonyms

2 votes
2 answers
22 views
Can one generate a grid of the Locales where a Distribution is mapped?
If I run the following code: use BlockDist; config const dimension: int = 5; const space = {0..# dimension}; const matrixBlock: domain(2) dmapped Block(boundingBox=space) = space
asked 13 hours ago by barrymoo 52 • 2

3 votes
1 answer
24 views
Is "[<var> in <distributed variable>]" equivalent to "forall"?
I noticed something in a snippet of code I was given: var D: domain(2) dmapped Block(boundingBox=Space; var A: [D] int; [a in A] a = a.locale.id; Is [a in A] equivalent to forall a in A a = ...
asked 15 hours ago by barrymoo 52 • 2

2 votes
1 answer
45 views
Get Non-primitive Variables from within a Cobegin - Chapel
I want to compute some information in parallel and use the result outside the cobegin. To be my requirement is to retrieve a domain (and other non primitive types) like this var a,b: ...
asked Apr 18 at 14:14 by xSo0Dx 151 • 1

3 votes
1 answer
Is there a default String conversion method in Chapel?
Is there a default method that gets called when I try to cast an object into a string? (E.g. toStr in Python.) I want to be able to do the following with an array of Objects, ...

This repository chapel-lang / chapel Pull requests Issues Marketplace Gist Watch 45 Unstar 455 Fork 145

<> Code Issues 292 Pull requests 26 Projects 0 Settings Insights

Filters is:issue is:open Labels Milestones New issue

292 Open 77 Closed Author Labels Projects Milestones Assignee Sort

- 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



COMPUTE

STORE

ANALYZE

Copyright 2017 Cray Inc.

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

#chapel-users (irc.freenode.net): user-oriented IRC channel

chapel-users@lists.sourceforge.net: user discussions

Discuss Chapel development

chapel-developers@lists.sourceforge.net: developer discussions

#chapel-developers (irc.freenode.net): developer-oriented IRC channel

Discuss Chapel's use in education

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

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



COMPUTE

STORE

ANALYZE

Copyright 2017 Cray Inc.

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 publically 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, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. 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 in this document are the property of their respective owners.





CRAY
THE SUPERCOMPUTER COMPANY