



# Chapel Update

Chapel Team, Cray Inc.

SC17 Briefings

November 2017



---

COMPUTE

|  
STORE

|  
ANALYZE

# 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.



# 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.”



# 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***



# A Brief History of Chapel



## 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)



COMPUTE

STORE

ANALYZE

Copyright 2017 Cray Inc.

# Chapel Community R&D Partners



Lawrence Berkeley  
National Laboratory



Yale

(and several others...)

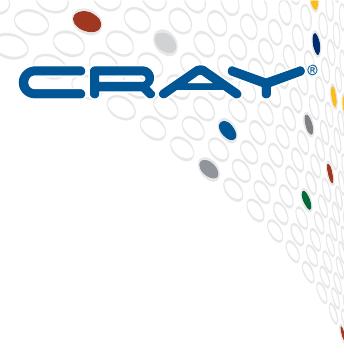
<https://chapel-lang.org/collaborations.html>



COMPUTE

STORE

ANALYZE



# Highlights of the Past Year or 4½



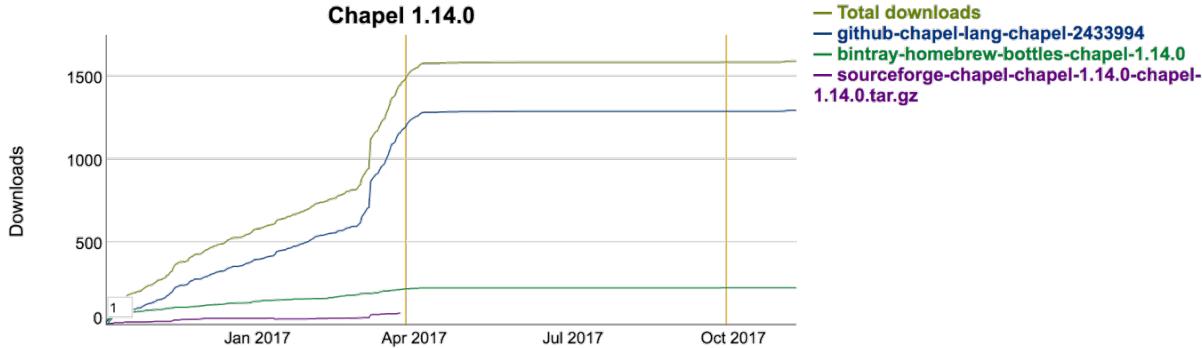
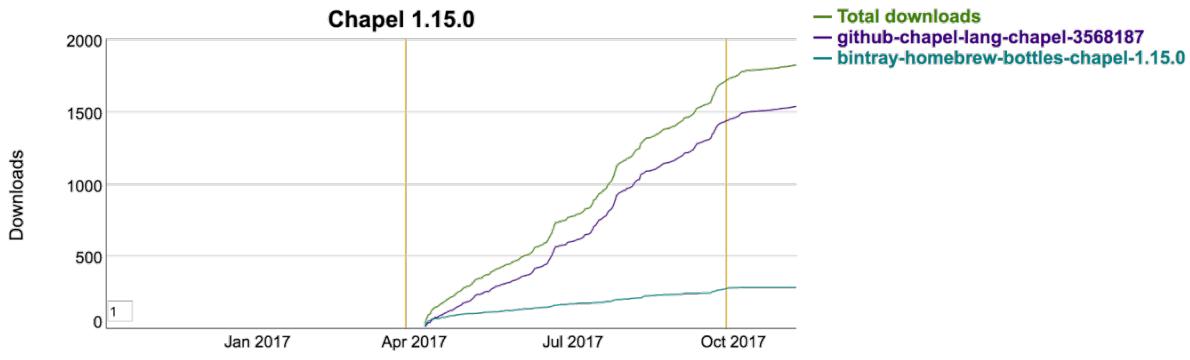
---

COMPUTE

| STORE

| ANALYZE

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



COMPUTE

STORE

ANALYZE

# 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

# 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.”

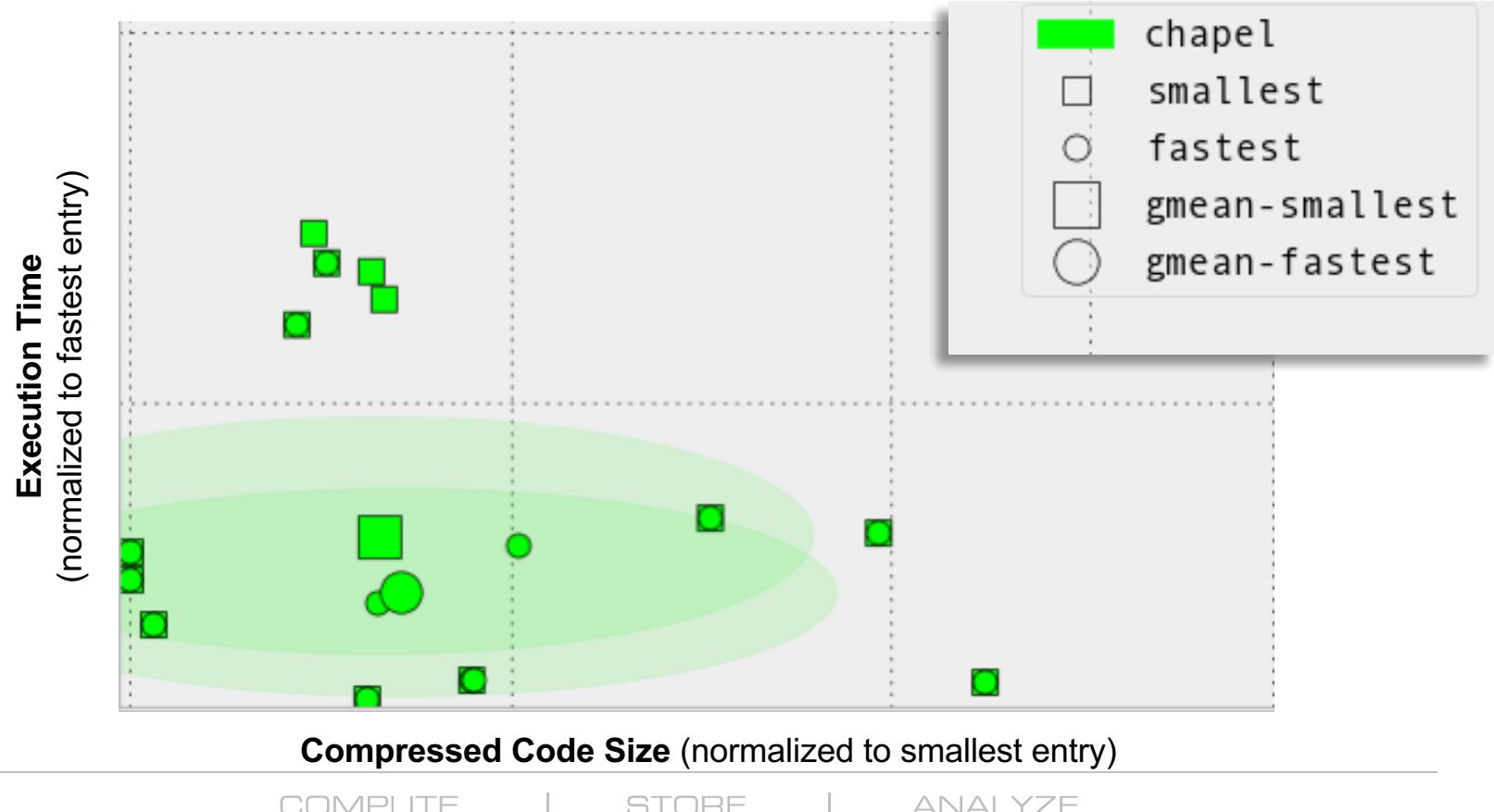


COMPUTE

STORE

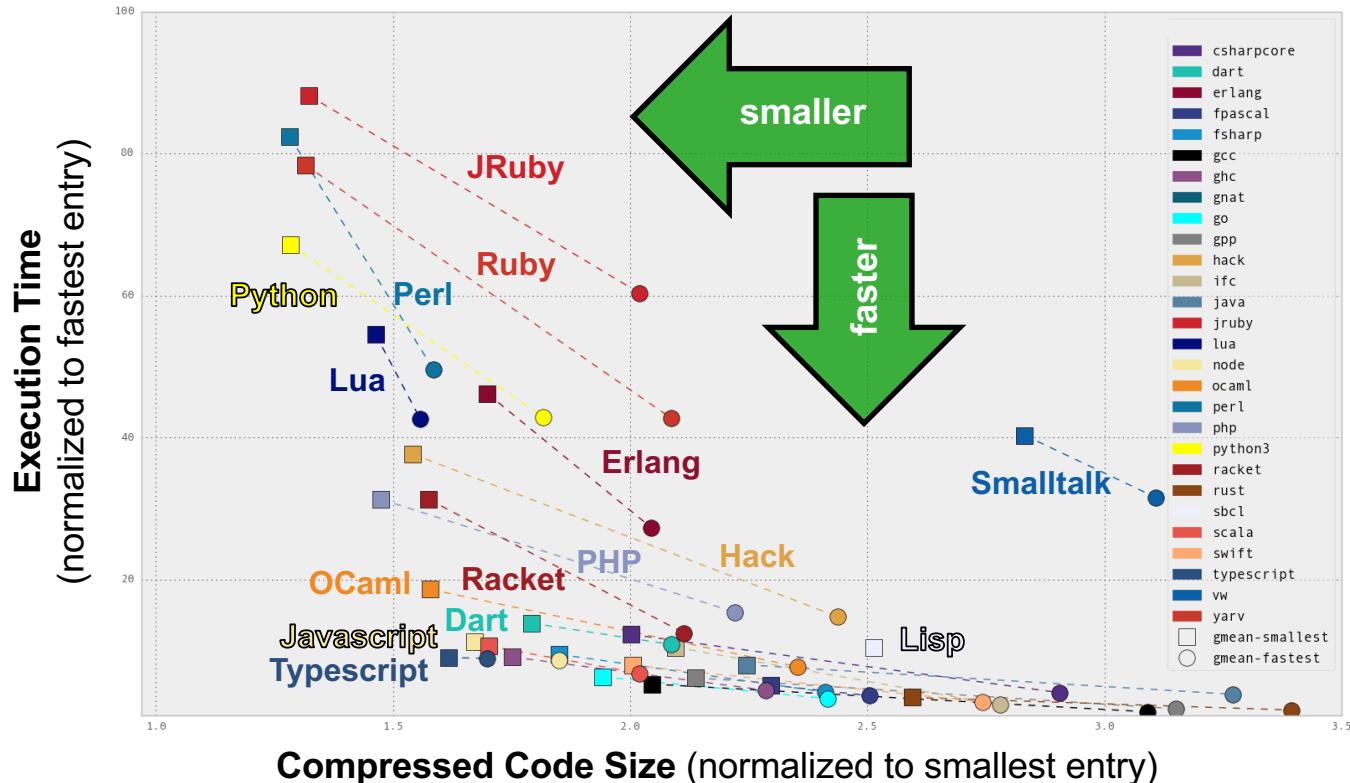
ANALYZE

# Scatter plots of CLBG code size x speed



# CLBG Cross-Language Summary

(Oct 2017 standings)



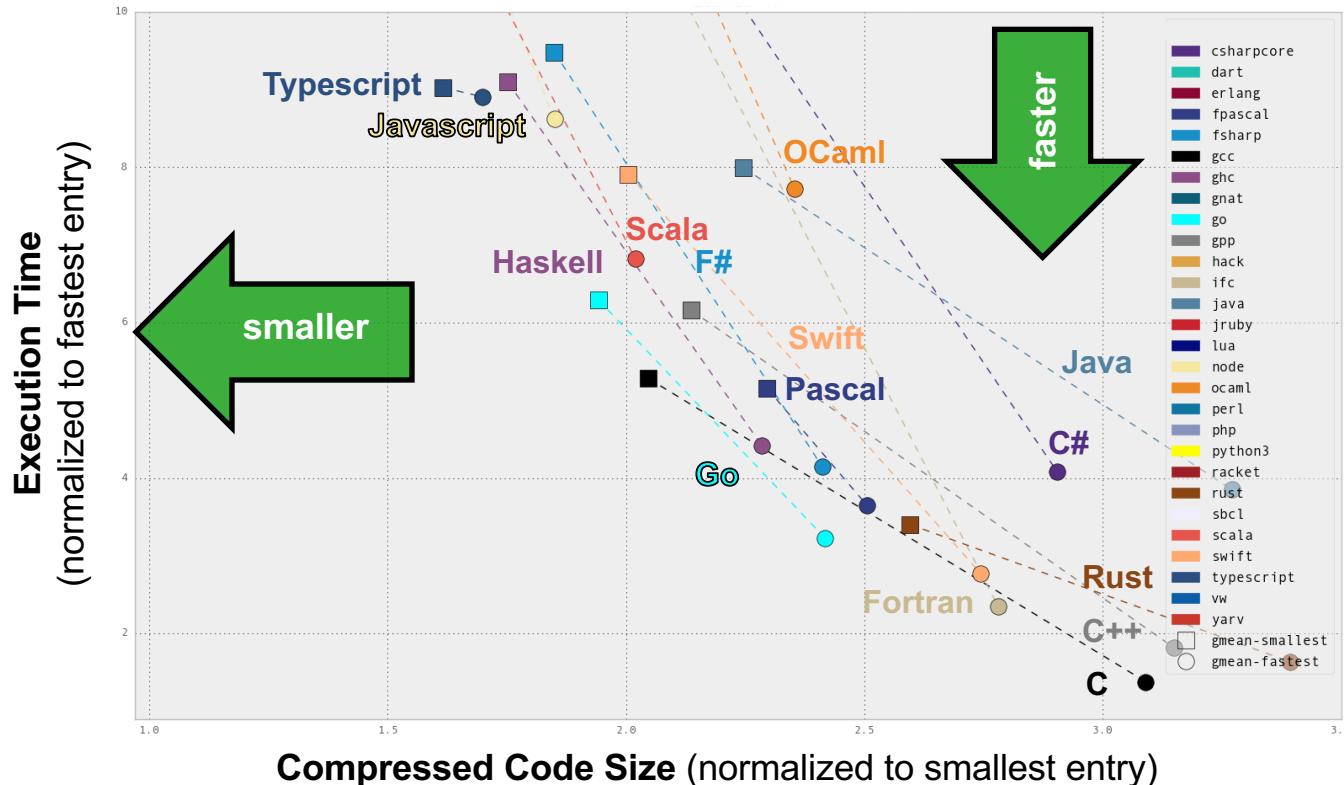
COMPUTE

STORE

ANALYZE

# CLBG Cross-Language Summary

(Oct 2017 standings)



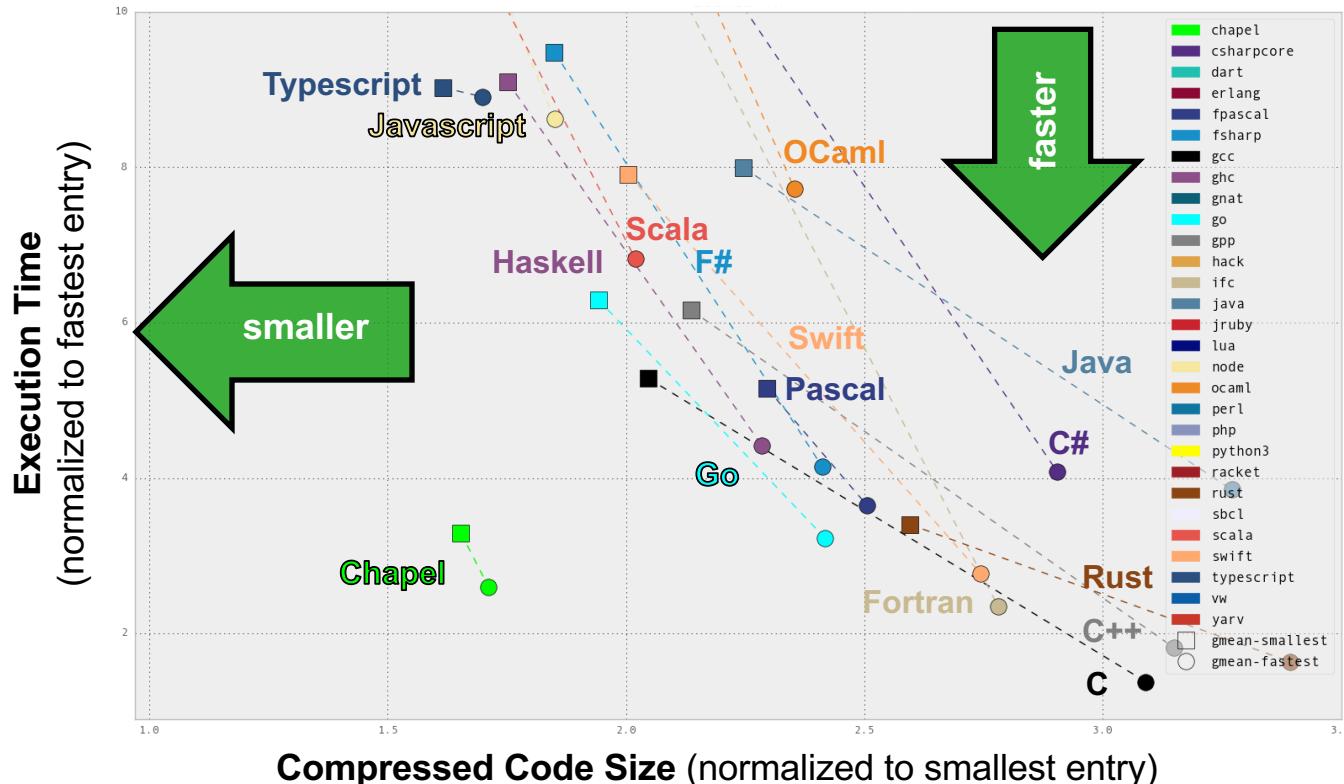
COMPUTE

STORE

ANALYZE

# CLBG Cross-Language Summary

(Oct 2017 standings)



COMPUTE

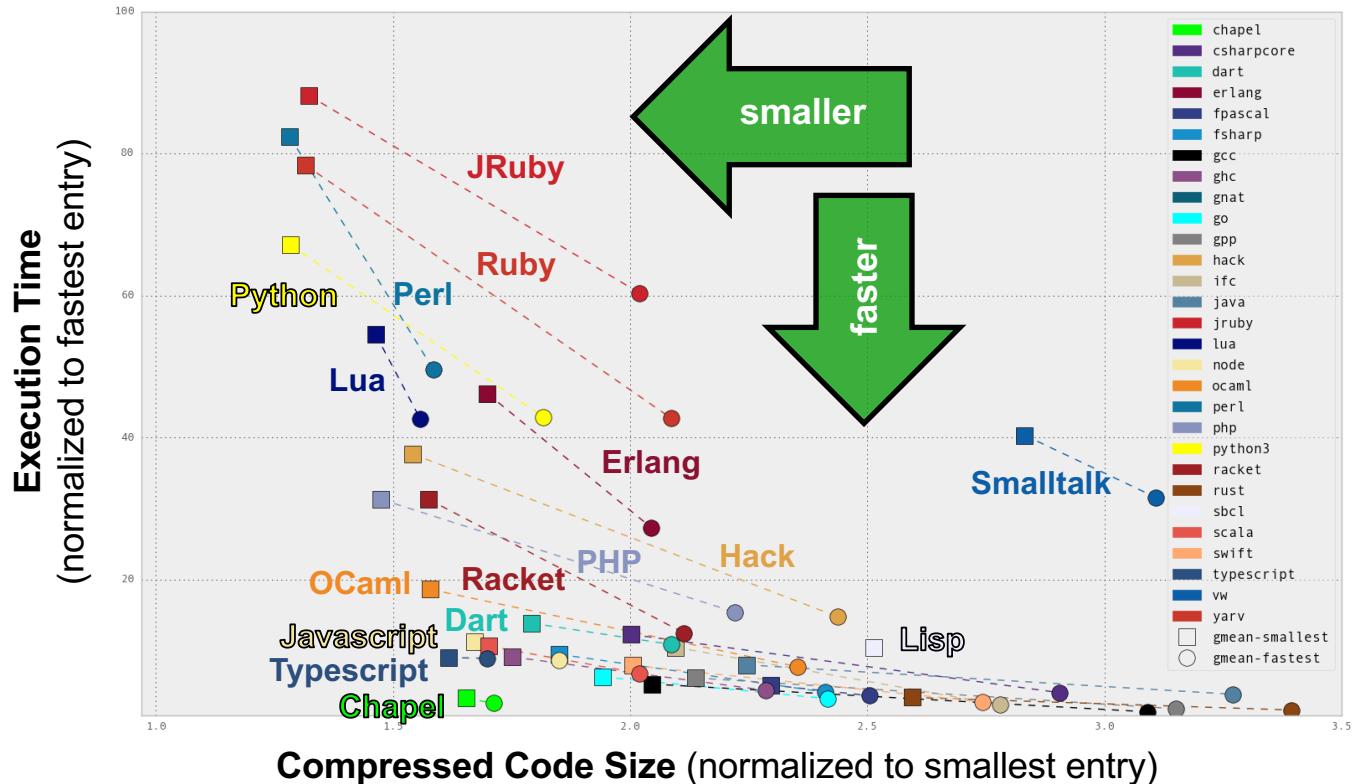
STORE

ANALYZE



# CLBG Cross-Language Summary

(Oct 2017 standings)

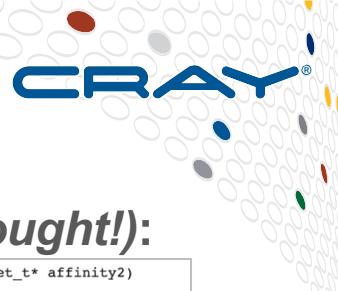


COMPUTE

STORE

ANALYZE

# 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];
    pos;
    cpu_idx;
    physical_id;
    core_id;
    cpu_cores;
    apic_id;
    cpu_count;
    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

# 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
// them meet
proc holdMeetings(population, numMeetings) {
    const place = new MeetingPlace(numMeetings);

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

    delete place;
}
```

excerpt from 1210.gz Chapel entry

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

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
```

```
size_t
char const*
size_t
char const*
```

```
proc holdMeetings(population, numMeetings) {
    const place = new MeetingPlace(numMeetings);

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

    delete place;
}
```

excerpt from 2863.gz C gcc entry

```
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";
cores              = n(core_id_str);
cores              = n(cpu_cores_str);
```

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



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(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;
    }
}
```

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;
    buf [2048];
    pos;
    cpu_idx;
    physical_id;
    core_id;
    cpu_cores;
    apic_id;
    cpu_count;
    i;

    char const* processor_str = "processor";
    size_t 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);
    cpu_cores_str = "cpu cores";
    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] = 1;
        CPU_ZERO(affinity1);
    }
}
```

excerpt from 2863.gz C gcc entry

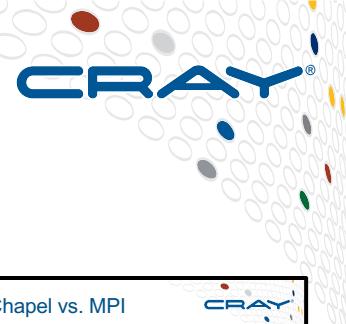


COMPUTE

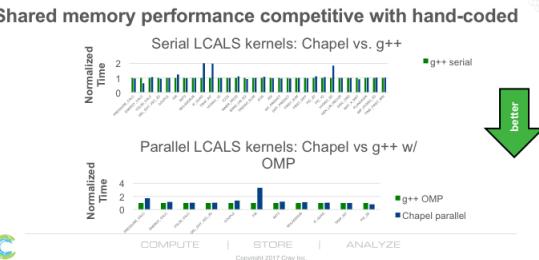
STORE

ANALYZE

# Chapel Performance: HPC Benchmarks



LCALS: Chapel vs. C + OpenMP



LCALS

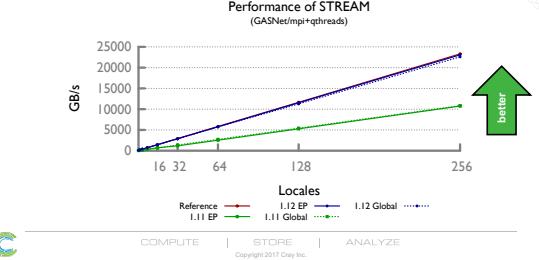
HPCC RA

STREAM  
Triad

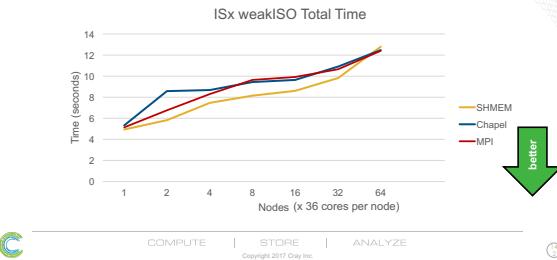
ISx

PRK  
Stencil

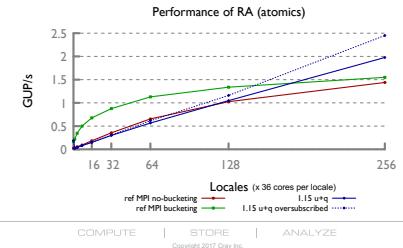
HPCC Stream Triad: Chapel vs. MPI+OpenMP



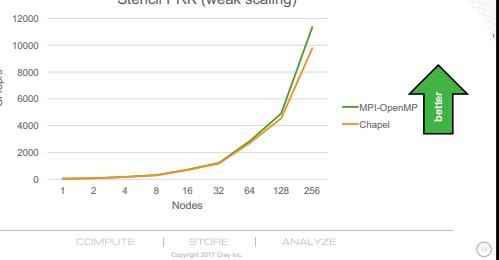
ISx Performance: Chapel vs. MPI, SHMEM



HPCC RA Performance: Chapel vs. MPI



Stencil PRK Scalability



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
- DistributedIter
- 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
- DistributedIter
- 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)

```
// 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
```

```
extern type qio_regex_t;
extern record qio_regex_options_t {
    var utf8:bool;
    var posixbc:bool;
    var casefold: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_regex_null():qio_regex_t;
extern proc qio_regex_init_default_options(ref options:qio_regex_options_t);
extern proc qio_regex_create_compile(strstring, strlen:int(64), ref options:qio_regex_options_t, ref compiled:qio_regex_t);
extern proc qio_regex_create_flags_1(strstring, strlen:int(64), flags:s\string, flagslen:int(64), isUtf8:bool, ref compiled:qio_regex_t);
extern proc qio_regex_create_flags_2(str:c_ptr, strlen:int(64), flags:c_ptr, flagslen:int(64), isUtf8:bool, ref compiled:qio_regex_t);
extern proc qio_regex_retain(ref compiled:qio_regex_t);
extern proc qio_regex_release(ref compiled:qio_regex_t);

extern proc qio_regex_get_options(ref regexp:qio_regex_t, ref options: qio_re\gexp_options_t);
extern proc qio_regex_get_pattern(ref regexp:qio_regex_t, ref pattern: string\);
extern proc qio_regex_get_ncaptures(ref regexp:qio_regex_t):int(64);
extern proc qio_regex_ok(ref regexp:qio_regex_t):bool;
extern proc qio_regex_error(ref regexp:qio_regex_t):string;

extern const QIO_REGEX_ANCHOR_UNANCHORED:c_int;
extern const QIO_REGEX_ANCHOR_START:c_int;
extern const QIO_REGEX_ANCHOR_BOTH:c_int;

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

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



COMPUTE

STORE

ANALYZE

# Libraries: Progress Since HPCS



Now: ~58 documented modules, many user-contributed

The screenshot displays two pages from the Chapel Documentation 1.16 website:

- Standard Modules**: A list of 34 modules including Assert, Barrier, Barriers, BigInteger, BitOps, Buffers, CommDiagnostics, DateTime, DynamicIterators, FileSystem, GMP, Help, IO, List, Math, Memory, Path, Random, Reflection, Regexp, Spawn, Sys, SysBasic, SysTypes, SysError, Time, Types, and UtilReplicatedVar.
- Package Modules**: A list of 34 modules including BLAS, Collection, Crypto, Curl, DistributedBag, DistributedDeque, DistributedIterators, FFTW, FFTW\_MT, Futures, HDF5, HDF5Iterator, LAPACK, LinearAlgebra, MPI, Norm, OwnedObject, RangeChunk, RecordParser, Search, SharedObject, Sort, VisualDebug, and ZMQ.



COMPUTE

STORE

ANALYZE

Copyright 2017 Cray Inc.

# Documentation: Progress Since HPCS

Then:

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

**Chapel Language Specification Version 0.93**

Brady E. Gammill  
901 Fifth Avenue, Suite 1000  
Seattle, WA 98164

April 18, 2013

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.contributors : names of the original Chapel contributors
README.multilocale : how to execute Chapel on multiple locales
README.threads   : explains how Chapel tasks are implemented using threads
README.xt-c      : notes for XT-C/C/C++ users
README.xc         : notes for C/C++ 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 HPC benchmarks. It 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. Chapman, David B. Kahan, and Hans P. Zima, published in the International Journal of High Performance Computing Applications, August 2007, 21(3), 291-312.

**Chapel Quick Reference**

**Quick Start**

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

- Create the file hello.chpl
- Run it with \$ ./hello.chpl
- Output: Hello, world!

**2. Compile and run it:**

```

$ ./chpl hello.chpl
$ ./hello.chpl
Hello, world

```

**Comments**

```

// single-line comment
/* multi-line
comment */

```

**Primitive Types**

Type	Default size	Other sizes	Default int
int	64	8, 16, 32	0
uint	64	8, 16, 32	0
real	64	0.0	0.0
imag	64	0.0	0.0
complex	128	0.0+0.0i, 64	0.0+0.0i, 64
string	variable	variable	variable

**Variables, Constants and Configuration**

```

VAR al_rast = 0.15;
var rast: real;
const rast: real = 0.01;
var const: real = 0.01;
param const: real = 0.01;
config const: int = 4;
config const: real = 4;
config const: string = "4";

```

**Casts and coercions**

```

var i = 1.234;
var i: real = i;
var i: real = 2;
var i: real = 2.0;
const i: real = 2;
const i: real = 2.0;

```

**Control and Loop Expressions**

```

var half: if i < 1.0 then 1/2.0 else 1/3.0;
for i = 1 to 10 do
  print(i);
  if forall i
    conditional expression, parallel
    source expression, serial return
    expression
  else
    common sequential expression

```

**Procedures**

```

proc series( real, li: list ) complex {
  var i = 0;
  while i < li.length do
    print li[i];
    i = i + 1;
}
proc foos() returns i*2 + 1 + 1

```

**Function Argument Inference**

```

function f(x: int) returns x;
function g(x: int, y: int) returns x+y;

```

**Local Variables**

```

local var x;
local var y;
local var z;

```

**Costs and execution**

```

var i = 1.234;
var i: real = i;
var i: real = 2;
var i: real = 2.0;
const i: real = 2;
const i: real = 2.0;

```

**Conditional and Loop Expressions**

```

var half: if i < 1.0 then 1/2.0 else 1/3.0;
for i = 1 to 10 do
  print(i);
  if forall i
    conditional expression, parallel
    source expression, serial return
    expression
  else
    common sequential expression

```

**Named Formal Parameters**

```

proc foo(x: int, y: real) returns i - j;

```

**File Edit Options Buffers Tools Help Help**

```

// Task Parallel Primer
// This primer illustrates Chapel's parallel tasking features,
// namely the begin, cobegin, and forall 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 statement 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
// with the next task, 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");
  begin writeln("3: output from spawned task 3");
}

// The output from within the cobegin statement will always precede the
// following output from the main thread of execution.
writeln("3: output from main task");

-uU---FI taskParallel.chpl Top LI (Chapel/L Abbrev)
Loading /users/bradc/chape/highlight/emacs/22/chpl-mode.el (source)...done

```

COMPUTE

STORE

Copyright 2017 Cray Inc.

28

# Documentation: Progress Since HPCS



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

The image shows four screenshots of the Chapel Documentation 1.16 website:

- Compiling and Running Chapel**: A page with a sidebar containing links to Quickstart Instructions, Using Chapel, Platform-Specific Notes, Technical Notes, and Tools. The main content includes sections on Quickstart Instructions, Using Chapel, Writing Chapel Programs, Language History, and Standard Layouts and Distributions.
- Using Chapel**: A page with a sidebar containing links to Quickstart Instructions, Using Chapel, Platform-Specific Notes, Technical Notes, and Tools. The main content includes sections on 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.
- Task Parallelism**: A page with a sidebar containing links to Quickstart Instructions, Using Chapel, Platform-Specific Notes, Technical Notes, and Tools. The main content includes sections on Primers, Task Parallelism, and Cobegin Statements. It features code snippets for `config const n = 10;` and `begin writeln("I: output from main task");`, and a note about the `begin` statement.



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?



---

COMPUTE

| STORE

| ANALYZE

# 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

# Crossing the Stream of Adoption



Research Prototype

MiniMD

ISx

CoMD

CLBG

PRK Stencil

RA

LULESH

Stream

LCALS

Time-to-science  
academic codes

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?



COMPUTE

STORE

ANALYZE

Copyright 2017 Cray Inc.

image source: <http://feelgraffix.com/813578-free-stream-wallpaper.html>

## 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 CHIUW 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)*

*CHIUW 2017 keynote*

<https://ljdursi.github.io/CHIUW2017> / <https://www.youtube.com/watch?v=xj0rwdLOR4U>



COMPUTE

STORE

ANALYZE

# Chapel Resources



---

COMPUTE

| STORE

| ANALYZE

# Chapel Central: <https://chapel-lang.org/>





## 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} mapped 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 **CHI UW 2018** [call for participation](#) is now available!
- A recent [Cray blog post](#) reports on highlights from CHI UW 2017.
- Chapel is now one of the supported languages on [Try It Online!](#)
- Watch talks from **ACCU 2017**, **CHI UW 2017**, and **ATPESC 2016** on [YouTube](#).
- [Browse slides](#) from **PADAL**, **EAGE**, **EMBRACE**, **ACCU**, and other recent talks.
- See also: [What's New?](#)



COMPUTE

STORE

ANALYZE



# How to Stalk Chapel

<http://facebook.com/ChapelLanguage>

<http://twitter.com/ChapelLanguage>

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

A screenshot of a Facebook page for "Chapel Programming Language". The page has 300 likes. A post from April 21 at 8:47pm discusses Chapel's ranking in the Computer Language Benchmarks Game. It includes a chart comparing Chapel's performance against other languages like C, Fortran, and Java.

A screenshot of a Twitter profile for "Chapel Language" (@ChapelLanguage). The profile has 222 tweets, 12 following, 129 followers, and 32 likes. A tweet from April 2017 encourages users to submit interesting applications to the PAW 2017 workshop. It includes a link to [sourceryinstitute.github.io/PAW/](https://sourceryinstitute.github.io/PAW/). The profile also shows 115 photos and videos.

A screenshot of a YouTube channel for "Chapel Parallel Programming Language". The channel has 222 subscribers. It features a video titled "SC16 Chapel Tutorial Promo" which is a promotional video for the SC16 Chapel tutorial. Other videos include "Chapel Productive, Multiresolution Parallel Programming | Brad Chamberlain, Cray, Inc." and "CHIW 2016 keynote: 'Chapel in the (Cosmological) Wild', Nikhil Padmanabhan".



COMPUTE

STORE

ANALYZE

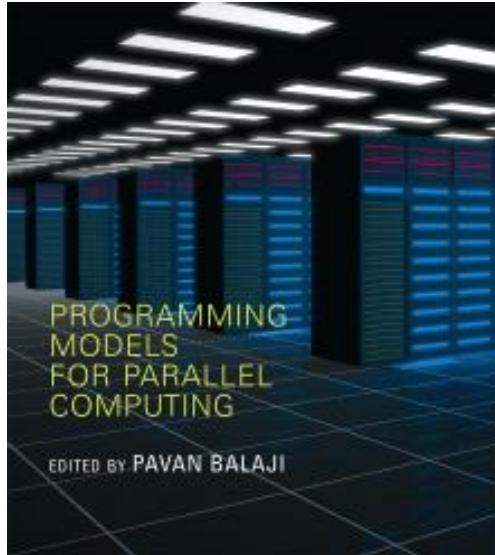
Copyright 2017 Cray Inc.

# 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](http://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



A screenshot showing two side-by-side web interfaces. On the left is the StackOverflow 'chapel' tag page, displaying several questions about the Chapel language. On the right is the GitHub repository 'chapel-lang/chapel' issues page, listing 292 open pull requests. Both pages show user interactions like voting, commenting, and filtering.



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](mailto: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](mailto:chapel-users@lists.sourceforge.net): user discussions

## Discuss Chapel development

[chapel-developers@lists.sourceforge.net](mailto: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](mailto:chapel-education@lists.sourceforge.net): educator discussions

## Directly contact Chapel team at Cray: [chapel\\_info@cray.com](mailto: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 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