Chapel 101

Brad Chamberlain

CHIUW 2019

June 22, 2019

bradc@cray.com

chapel-lang.org

@ChapelLanguage

What is Chapel?

Chapel: A modern 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

What does “Productivity” mean to you?

Recent Graduates:

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

Seasoned HPC Programmers:

“that sugary stuff which I don’t need because I require full control to get performance”

Computational Scientists:

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

Chapel Team:

“something that lets computational scientists express what they want,
without taking away the control that HPC programmers want,
implemented in a language as attractive as recent graduates want.”

Why Consider New Languages at all?

Syntax	• High level, elegant syntax • Improve programmer productivity
Semantics	• Static analysis can help with correctness • We need a compiler (front-end)
Performance	• If optimizations are needed to get performance • We need a compiler (back-end)
Algorithms	• Language defines what is easy and hard • Influences algorithmic thinking

[Source: Kathy Yelick, CHIUW 2018 keynote: *Why Languages Matter More Than Ever*]

Comparing Chapel to Other Languages

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]

Outline

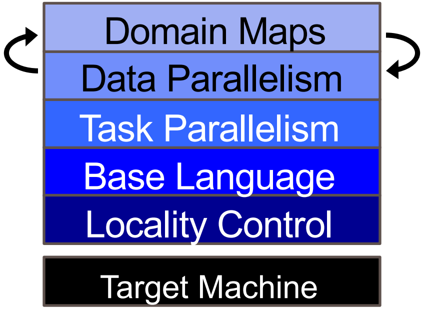
✓ Context and Motivation

➢ A Brief Tour of Chapel Features

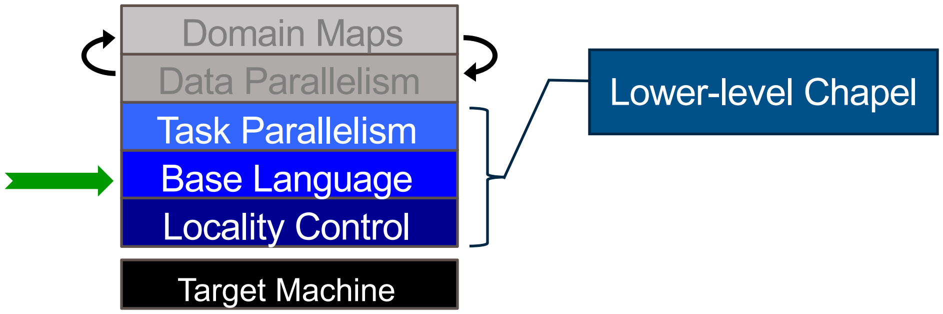
- Chapel Evaluations
- Summary and Resources

Chapel Feature Areas

Chapel language concepts



Base Language



Base Language Features, by example

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
…
```

Base Language Features, by example

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

Configuration declarations
(support command-line overrides)
`./fib --n=1000000`

```
config const n = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
…
```

Base Language Features, by example

Iterators

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
…
```

Base Language Features, by example

Static type inference for:

- arguments
- return types
- variables

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
…
```

Base Language Features, by example

Explicit types also supported

```
iter fib(n: int): int {
  var current: int = 0,
      next: int = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n: int = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
…
```

Base Language Features, by example

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
…
```

Base Language Features, by example

Zippered iteration

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for (i,f) in zip(0..#n, fib(n)) do
  writeln("fib #", i, " is ", f);
```

```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
…
```

Base Language Features, by example

Range types and operators

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for (i,f) in zip(0..#n, fib(n)) do
  writeln("fib #", i, " is ", f);
```

```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
…
```

Base Language Features, by example

Tuples

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for (i,f) in zip(0..#n, fib(n)) do
  writeln("fib #", i, " is ", f);
```

```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
…
```

Base Language Features, by example

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

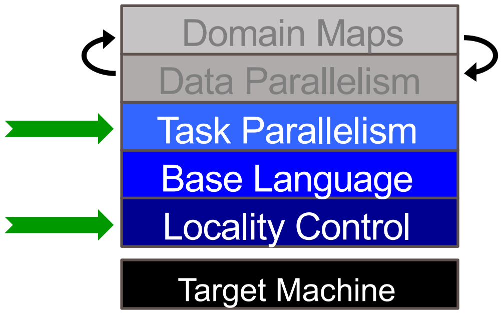
```
config const n = 10;

for (i,f) in zip(0..#n, fib(n)) do
  writeln("fib #", i, " is ", f);
```

```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
…
```

Other Base Language Features

- **Object-oriented programming** (value- and reference-based)
  - Managed objects and lifetime checking
  - Nilable vs. non-nilable class variables
- **Generic programming / polymorphism**
- **Error-handling**
- **Compile-time meta-programming**
- **Modules** (supporting namespaces)
- **Procedure overloading / filtering**
- **Arguments:** default values, intents, name-based matching, type queries
- and more…

Task Parallelism and Locality Control

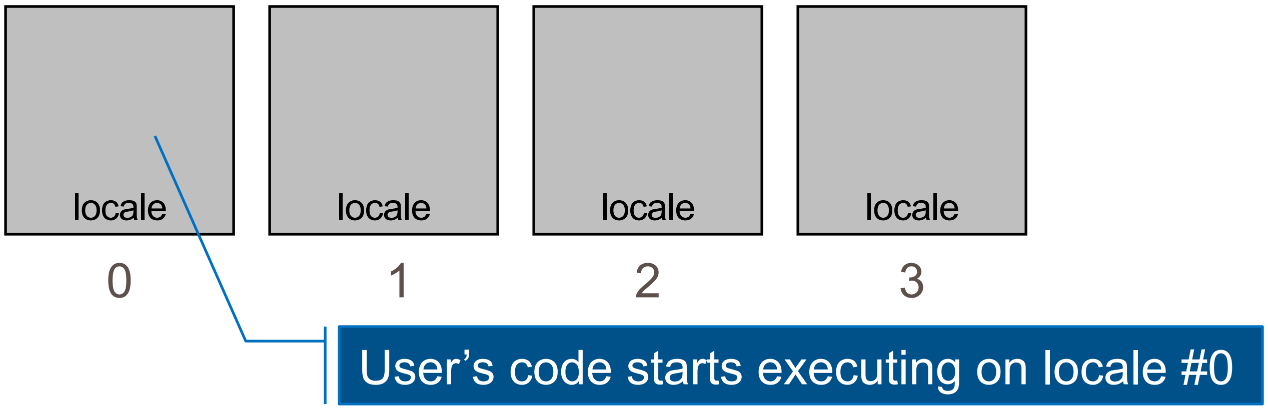


Locales, briefly

- Locales can run tasks and store variables
  - Think “compute node”
  - The number of locales is specified on the execution command-line

```
> ./myProgram --numLocales=4     # or `-nl 4`
```

Locales:



Task Parallelism and Locality, by example

taskParallel.chpl

```
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  writef("Hello from task %n of %n "+
         "running on %s\n",
         tid, numTasks, here.name);
```

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```

Task Parallelism and Locality, by example

Abstraction of System Resources

taskParallel.chpl

```
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  writef("Hello from task %n of %n "+
         "running on %s\n",
         tid, numTasks, here.name);
```

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```

Task Parallelism and Locality, by example

High-Level
Task Parallelism

taskParallel.chpl

```
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  writef("Hello from task %n of %n "+
         "running on %s\n",
         tid, numTasks, here.name);
```

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```

Task Parallelism and Locality, by example

So far, this is a shared memory program

Nothing refers to remote locales, explicitly or implicitly

taskParallel.chpl

```
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  writef("Hello from task %n of %n "+
         "running on %s\n",
         tid, numTasks, here.name);
```

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```

Task Parallelism and Locality, by example

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```

Task Parallelism and Locality, by example

Abstraction of System Resources

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```

Task Parallelism and Locality, by example

High-Level
Task Parallelism

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```

Task Parallelism and Locality, by example

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

Control of Locality/Affinity

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```

Task Parallelism and Locality, by example

taskParallel.chpl

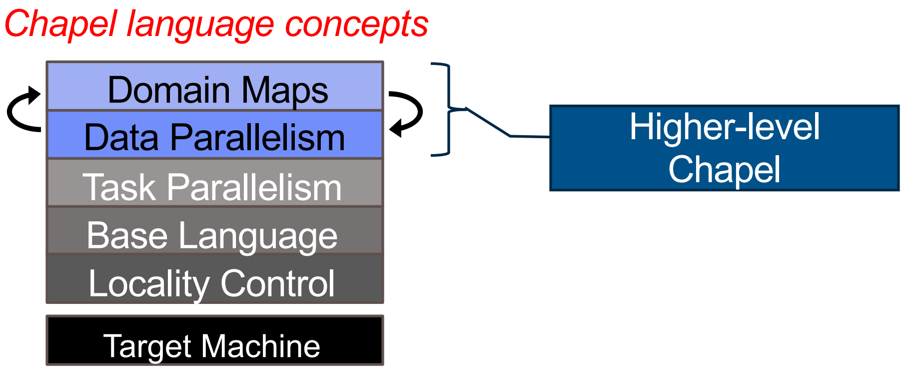
```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```

Other Task Parallel Features

- **atomic / synchronized variables:** for sharing data & coordination
- **begin / cobegin statements:** other ways of creating tasks
- **task intents:** for specifying how outer-scope variables are passed to tasks

Data Parallelism in Chapel



Data Parallelism, by example

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```

Data Parallelism, by example

Domains (Index Sets)

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```

Data Parallelism, by example

Arrays

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```

Data Parallelism, by example

Data-Parallel Forall Loops

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```

Data Parallelism, by example

So far, this is a shared memory program

Nothing refers to remote locales, explicitly or implicitly

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```

Distributed Data Parallelism, by example

dataParallel.chpl

```
use CyclicDist;
config const n = 1000;
var D = {1..n, 1..n}
          dmapped Cyclic(startIdx = (1,1));
var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

Domain Maps
(Map Data Parallelism to the System)

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5 --numLocales=4
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```

Distributed Data Parallelism, by example

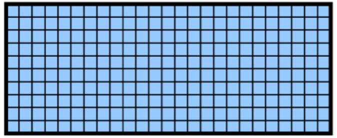
dataParallel.chpl

```
use CyclicDist;
config const n = 1000;
var D = {1..n, 1..n}
          dmapped Cyclic(startIdx = (1,1));
var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

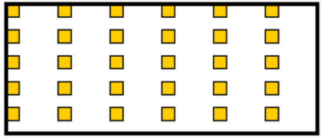
```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5 --numLocales=4
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```

Other Data Parallel Features

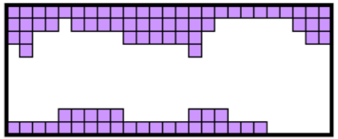
- **Parallel Iterators and Zippering**
- **Slicing:** refer to subarrays using ranges / domains
- **Promotion:** execute scalar functions in parallel using array arguments
- **Reductions:** collapse arrays to scalars or subarrays
- **Scans:** parallel prefix operations
- **Several Domain/Array Types:**



dense



strided



sparse

associative

Chapel Evaluations

Computer Language Benchmarks Game (CLBG)

The Computer Language Benchmarks Game

Which programs are faster?

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

Ada C Chapel C# C++ Dart

Erlang F# Fortran Go Hack

Haskell Java JavaScript Lisp Lua

OCaml Pascal Perl PHP Python

Racket Ruby Rust Smalltalk Swift

TypeScript

Which are fast? Trust, and verify

{ for researchers }

Website supporting cross-language comparisons

- 10 toy benchmark programs ×
  ~27 languages ×
  several implementations
  - specific approach prescribed

Chapel's approach to the CLBG:

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

CLBG: Website

Can sort results by various metrics: execution time, code size, memory use, CPU use:

The Computer Language Benchmarks Game

pidigits

description

program source code, command-line and measurements

×	source	secs	mem	gz	cpu	cpu load
1.0	**Chapel** #2	**1.62**	6,484	423	1.63	99% 1% 1% 2%
1.0	Chapel	1.62	6,488	501	1.63	99% 1% 1% 1%
1.1	Free **Pascal** #3	**1.73**	2,428	530	1.72	0% 2% 100% 1%
1.1	**Rust** #3	**1.74**	4,488	1366	1.74	1% 100% 1% 0%
1.1	Rust	1.74	4,616	1420	1.74	1% 100% 1% 0%
1.1	Rust #2	1.74	4,636	1306	1.74	1% 100% 0% 0%
1.1	**C** gcc	**1.75**	2,728	452	1.74	1% 2% 0% 100%
1.1	**Ada** 2012 GNAT #2	**1.75**	4,312	1068	1.75	1% 0% 100% 0%
1.1	**Swift** #2	**1.76**	8,492	601	1.76	1% 100% 1% 0%
1.1	**Lisp** SBCL #4	**1.79**	20,196	940	1.79	1% 2% 1% 100%
1.2	**C++** g++ #4	**1.89**	4,284	513	1.88	5% 0% 1% 100%
1.3	**Go** #3	**2.04**	8,976	603	2.04	1% 0% 100% 0%
1.3	**PHP** #5	**2.12**	10,664	399	2.11	100% 0% 1% 1%
1.3	PHP #4	2.12	10,512	389	2.12	100% 0% 0% 2%

The Computer Language Benchmarks Game

pidigits

description

program source code, command-line and measurements

×	source	secs	mem	gz	cpu	cpu load
1.0	**Perl** #4	3.50	7,348	**261**	3.50	100% 1% 1% 1%
1.5	**Python 3** #2	3.51	10,500	**386**	3.50	1% 1% 0% 100%
1.5	**PHP** #4	2.12	10,512	**389**	2.12	100% 0% 0% 2%
1.5	Perl #2	3.83	7,320	389	3.83	2% 1% 100% 1%
1.5	PHP #5	2.12	10,664	399	2.11	100% 0% 1% 1%
1.6	**Chapel** #2	1.62	6,484	**423**	1.63	[illegible]
1.7	**C** gcc	1.75	2,728	**452**	1.74	[illegible]
1.7	**Racket**	27.58	124,156	**453**	27.56	[illegible]
1.8	**OCaml** #5	6.72	19,836	**458**	6.71	[illegible]
1.8	Perl	15.45	10,876	463	15.44	0% 81% 19% 1%
1.9	**Ruby** #5	3.29	277,496	**485**	6.58	8% 63% 32% 100%
1.9	**Lisp** SBCL #3	11.99	325,776	**493**	11.96	0% 1% 100% 0%
1.9	Chapel	1.62	6,488	501	1.63	99% 1% 1% 1%
1.9	PHP #3	2.14	10,672	504	2.14	1% 0% 0% 100%

gz == code size metric
strip comments and extra whitespace, then gzip

CLBG: Website

Can also compare languages pair-wise:

- but only sorted by execution speed…

The Computer Language Benchmarks Game

Chapel versus C++ g++ fastest programs

vs C **vs C++** vs Go vs Java vs Python

by faster benchmark performance

reverse-complement

source	secs	mem	gz	cpu	cpu load
Chapel	**2.20**	1,497,876	707	5.10	96% 42% 58% 38%
C++ g++	2.95	980,472	2280	4.56	50% 41% 16% 50%

pidigits

source	secs	mem	gz	cpu	cpu load
Chapel	**1.62**	6,488	501	1.63	99% 1% 1% 1%
C++ g++	1.89	4,284	513	1.88	5% 0% 1% 100%

fannkuch-redux

source	secs	mem	gz	cpu	cpu load
Chapel	12.07	4,556	728	48.05	100% 100% 100% 100%
C++ g++	10.62	2,040	980	41.91	100% 95% 100% 100%

The Computer Language Benchmarks Game

Chapel versus Python 3 fastest programs

vs C vs C++ vs Go vs Java **vs Python**

by faster benchmark performance

mandelbrot

source	secs	mem	gz	cpu	cpu load
Chapel	**5.09**	36,328	620	20.09	99% 99% 99% 99%
Python 3	279.68	49,344	688	1,117.29	100% 100% 100% 100%

spectral-norm

source	secs	mem	gz	cpu	cpu load
Chapel	**3.97**	5,488	310	15.75	99% 99% 99% 99%
Python 3	193.86	50,556	443	757.23	98% 98% 99% 99%

fannkuch-redux

source	secs	mem	gz	cpu	cpu load
Chapel	**12.07**	4,556	728	48.05	100% 100% 100% 100%
Python 3	547.23	48,052	950	2,162.70	99% 100% 97% 100%

CLBG: Qualitative Code 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 Code Comparisons

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

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

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

  coforall c in population do              // creat
    c.haveMeetings(place, population);

  delete place;
}
```

excerpt from 1210 gz Chapel entry

excerpt from 2863 gz C gcc entry

CLBG: Qualitative Code Comparisons

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

```
proc main() {
```

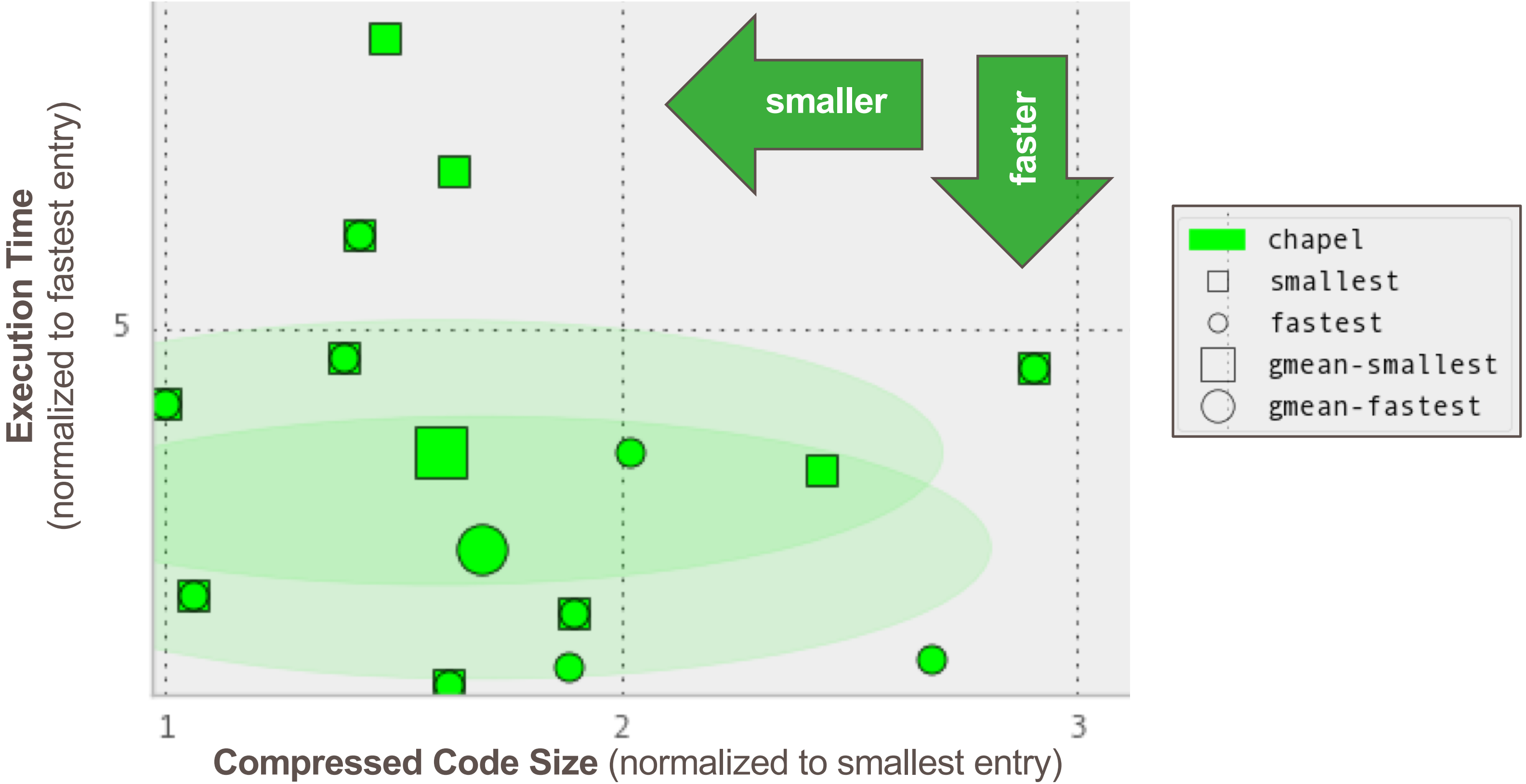
```
char const*     core_id_str          = "core id"
size_t          core_id_str_len      = strlen(co
char const*     cpu_cores_str        = "cpu core
size_t          cpu_cores_str_len    = strlen(cp

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

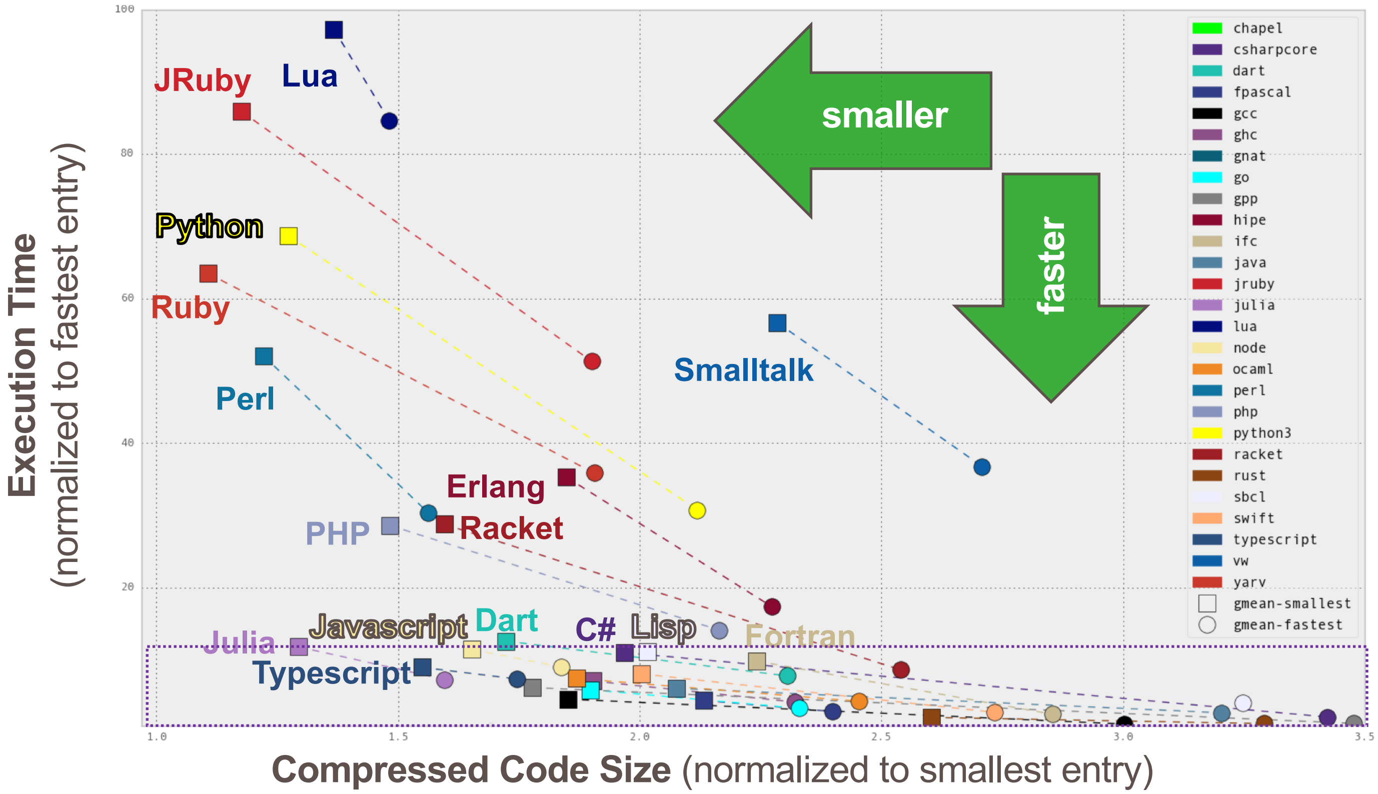
```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
    cpu_set_t                   active_cpus;
    FILE*                       f;
    char                        buf [2048];
    char const*                 pos;
    int                         cpu_idx;
    int                         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

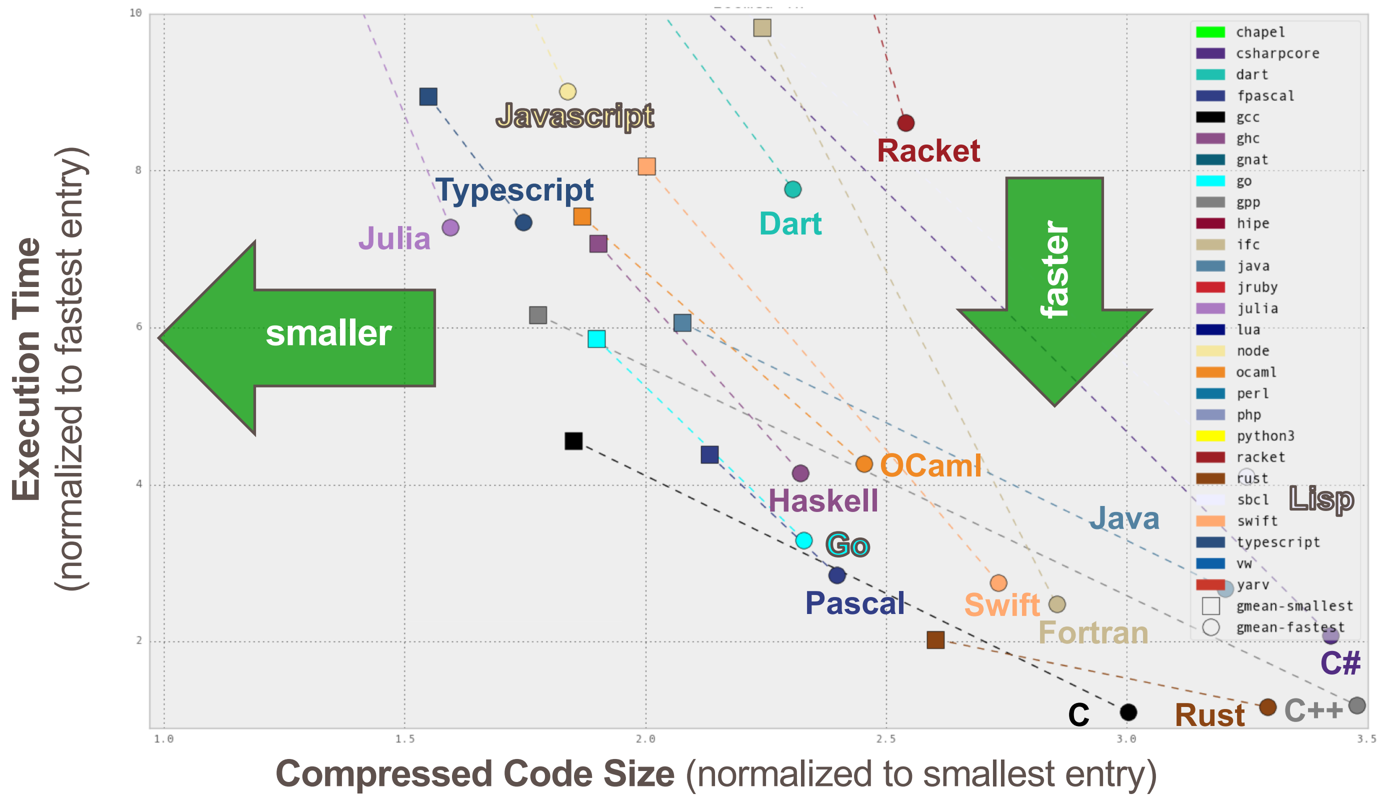
CLBG: Chapel Entries (May 14, 2019)



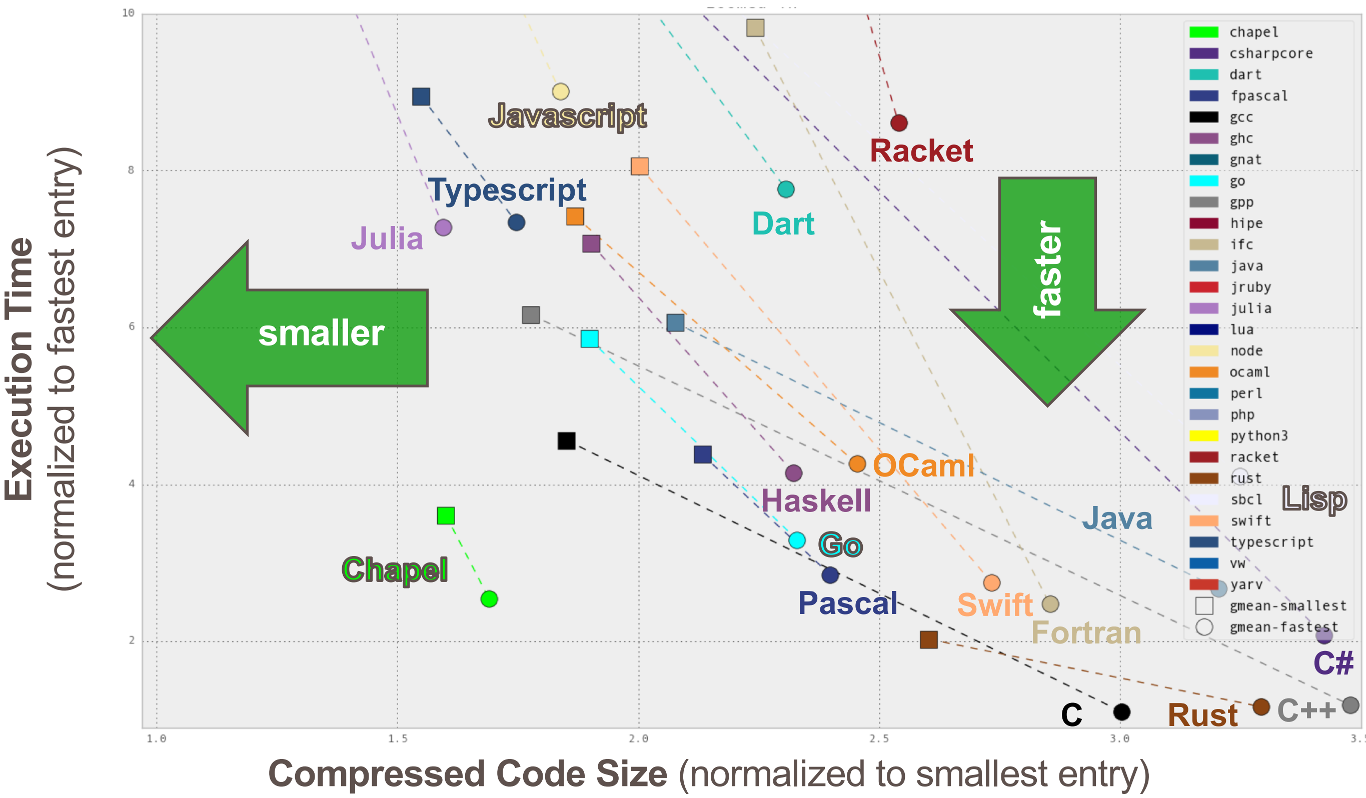
CLBG Cross-Language Summary (May 14, 2019)



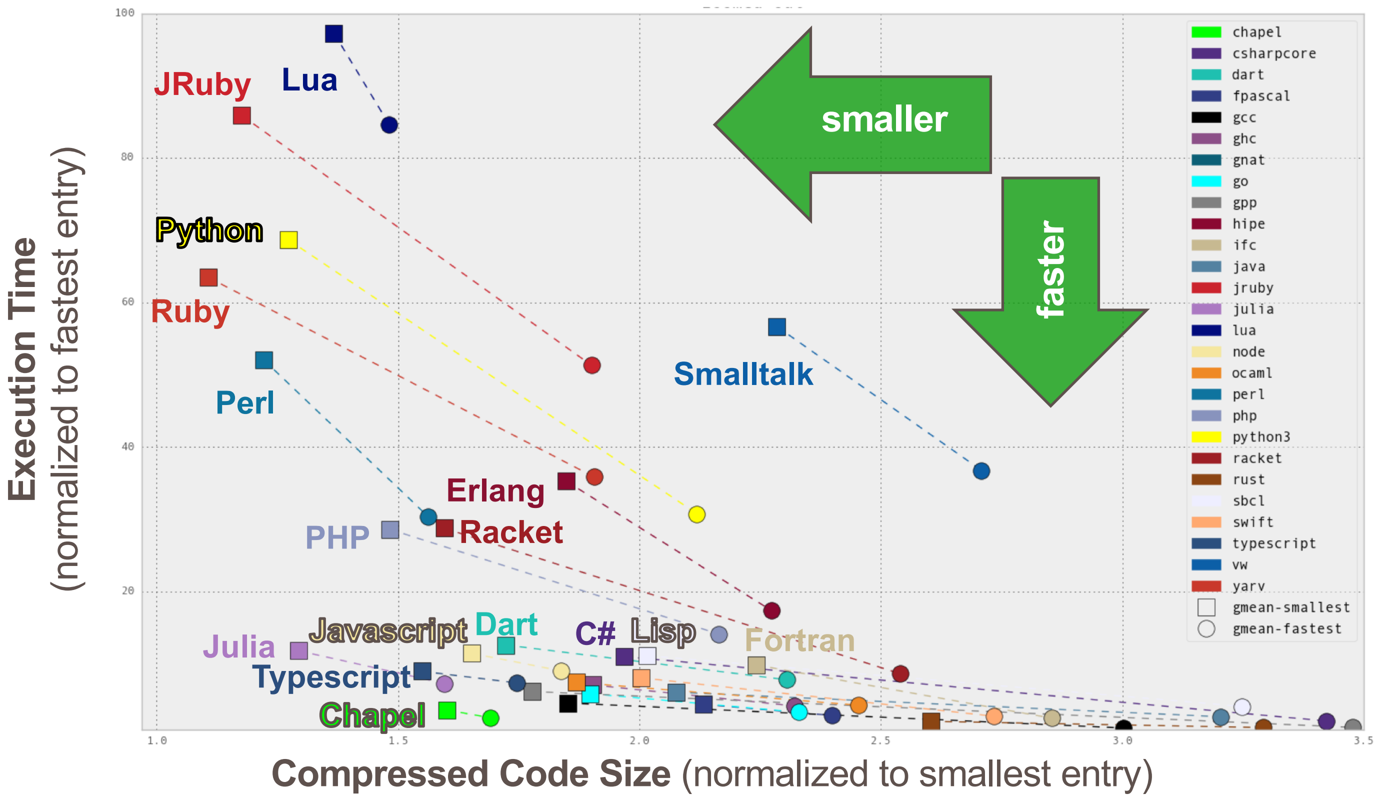
CLBG Cross-Language Summary (May 14, 2019, zoomed)



CLBG Cross-Language Summary (May 14, 2019, zoomed)



CLBG Cross-Language Summary (May 14, 2019)

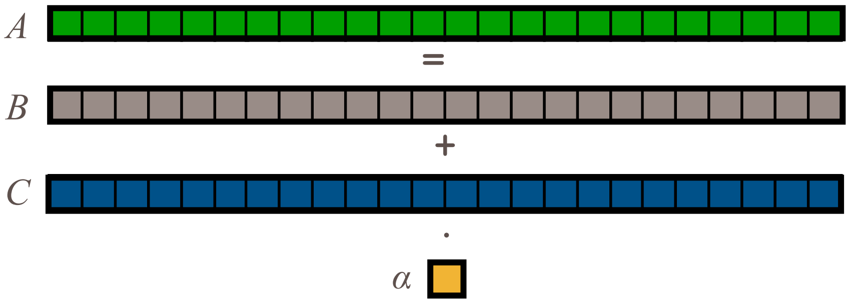


STREAM Triad: a trivial parallel computation

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m,\ A_i = B_i + \alpha \cdot C_i$

In pictures:

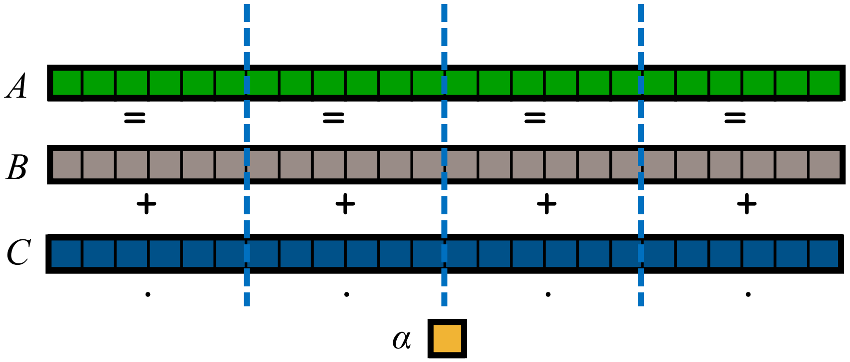


STREAM Triad: a trivial parallel computation

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m, A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel (shared memory / multicore):

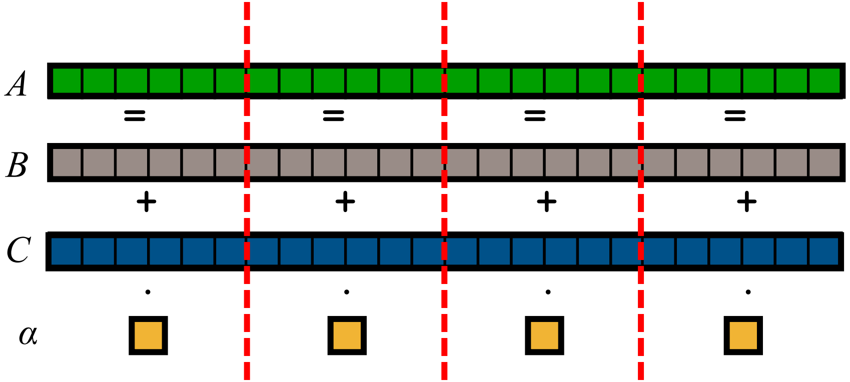


STREAM Triad: a trivial parallel computation

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m,\ A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel (distributed memory):

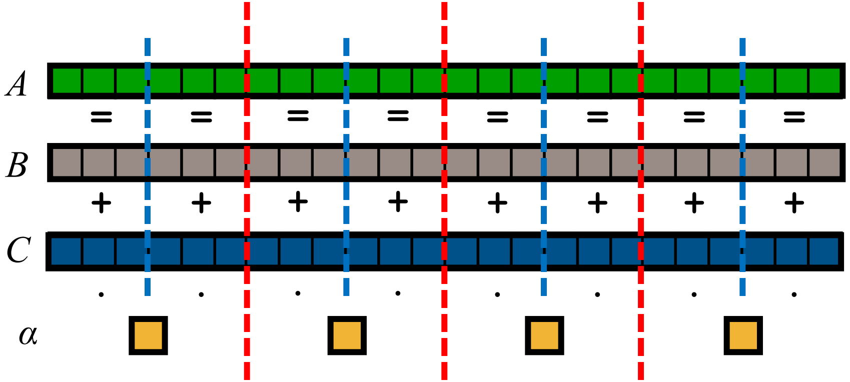


STREAM Triad: a trivial parallel computation

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m,\ A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel (distributed memory multicore):



STREAM Triad: C + MPI

```c
#include <hpcc.h>



static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );

  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
      fprintf( outFile, "Failed to allocate memory (%d).\n", VectorSize );
      fclose( outFile );
    }
    return 1;
  }



  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 1.0;
  }
  scalar = 3.0;



  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);

  return 0;
}
```

STREAM Triad: C + MPI + OpenMP

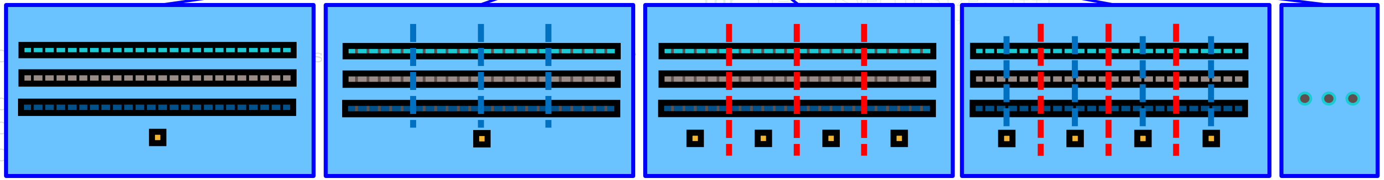
```c
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );
```

```c
  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
      fprintf( outFile, "Failed to allocate memory (%d).\n", VectorSize );
      fclose( outFile );
    }
    return 1;
  }

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 1.0;
  }
  scalar = 3.0;

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);

  return 0;
}
```

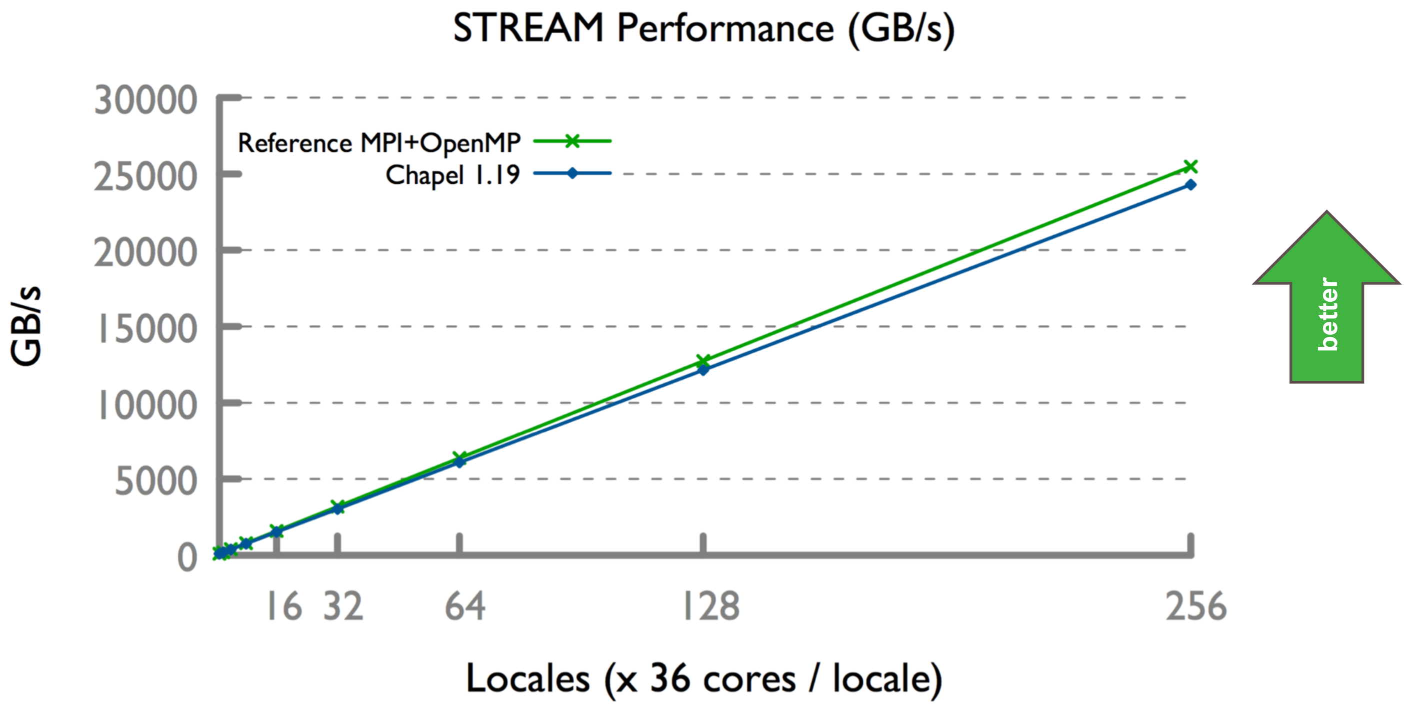
STREAM Triad: Chapel

```
use …;

config const m = 1000,
             alpha = 3.0;

const ProblemSpace = {1..m} dmapped …;

var A, B, C: [ProblemSpace] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```

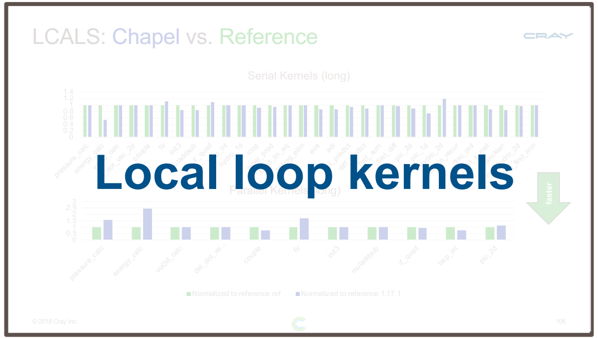
The special sauce:
How should this index set—and any arrays and computations over it—be mapped to the system?



HPCC STREAM Triad: Chapel vs. C+MPI+OpenMP

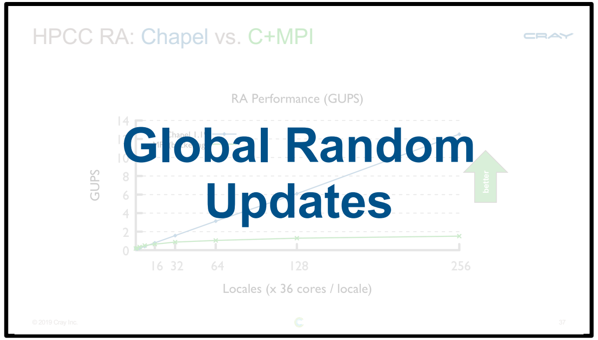


HPC Patterns: Chapel vs. Reference



LCALS

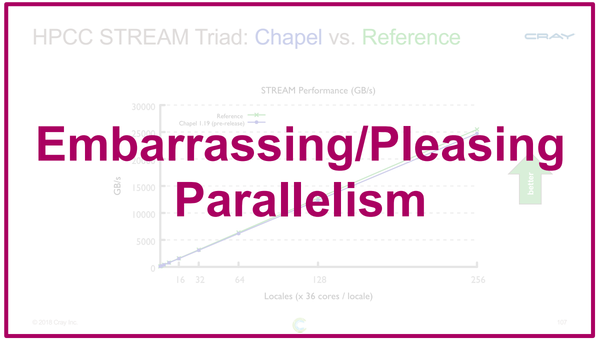
HPCC RA

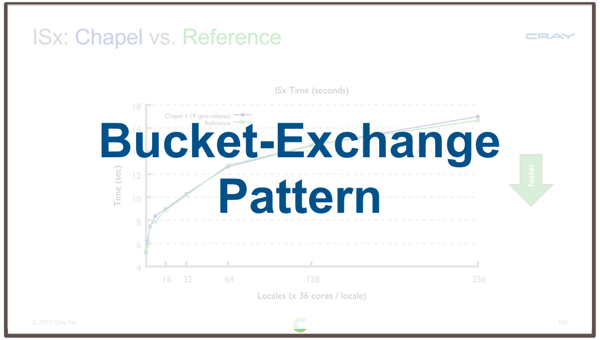


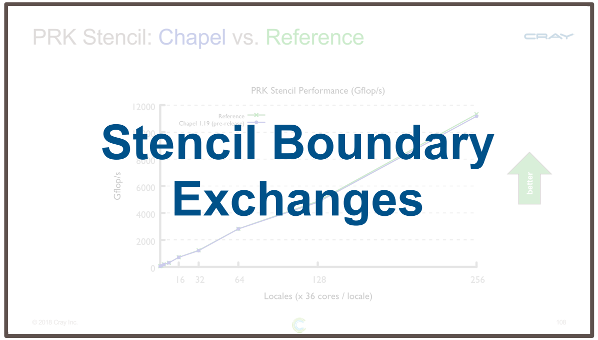
STREAM Triad

ISx

PRK Stencil

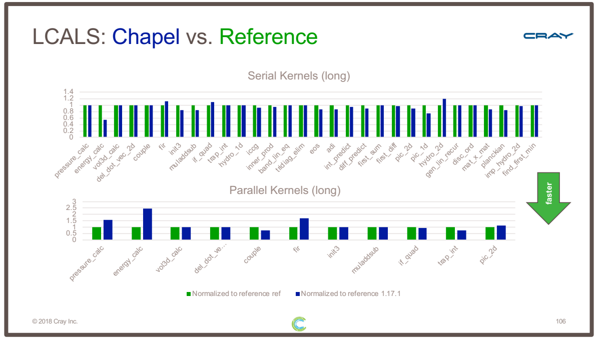






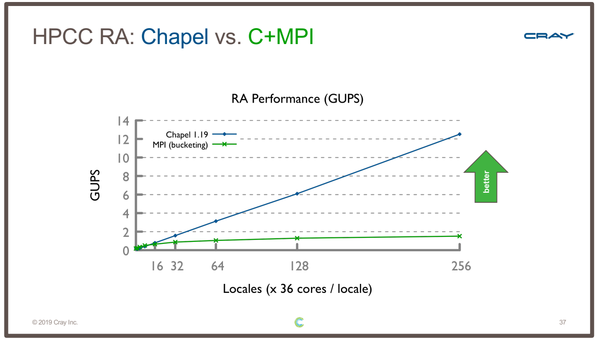
Nightly performance tickers online at:
https://chapel-lang.org/perf-nightly.html

HPC Patterns: Chapel vs. Reference



LCALS

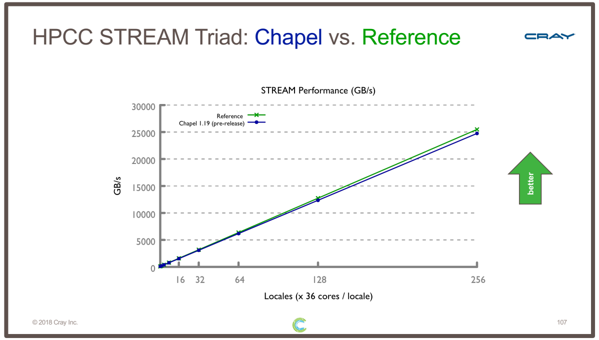
HPCC RA

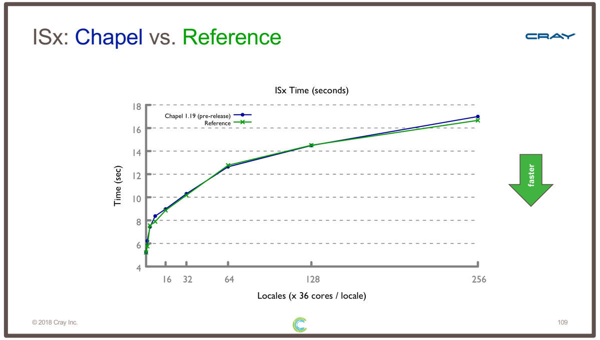


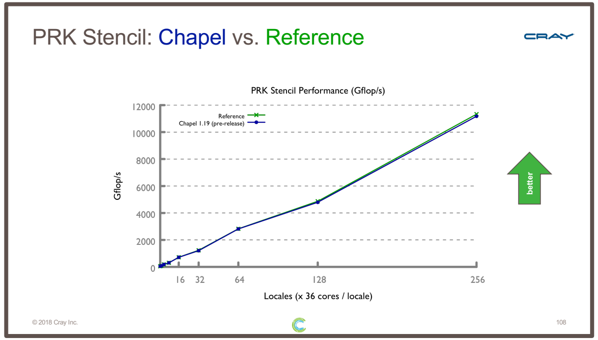
STREAM Triad

ISx

PRK Stencil







Nightly performance tickers online at:
https://chapel-lang.org/perf-nightly.html

Summary and Resources

Summarizing this Talk

Chapel cleanly and orthogonally supports…

…expression of parallelism and locality

…specifying how to map computations to the system

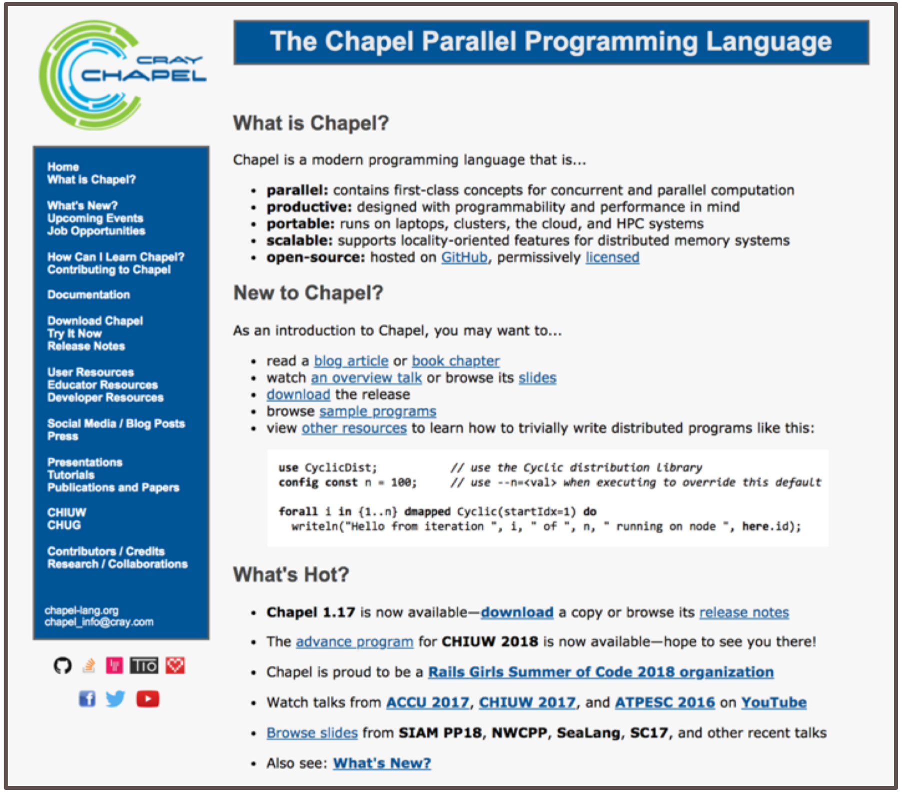
Chapel is powerful:

- supports succinct, straightforward code
- can result in performance that competes with (or beats) C+MPI+OpenMP

Chapel Central

https://chapel-lang.org

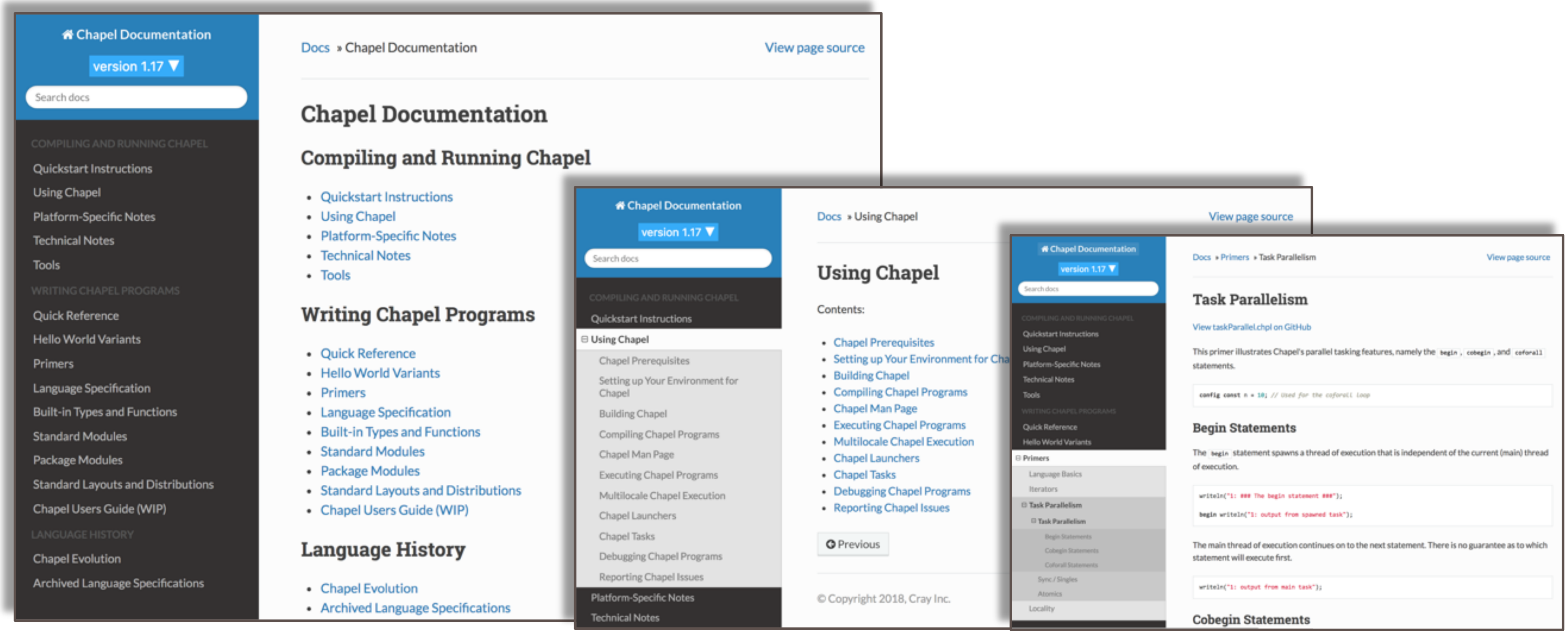
- download Chapel
- presentations
- papers
- resources
- documentation



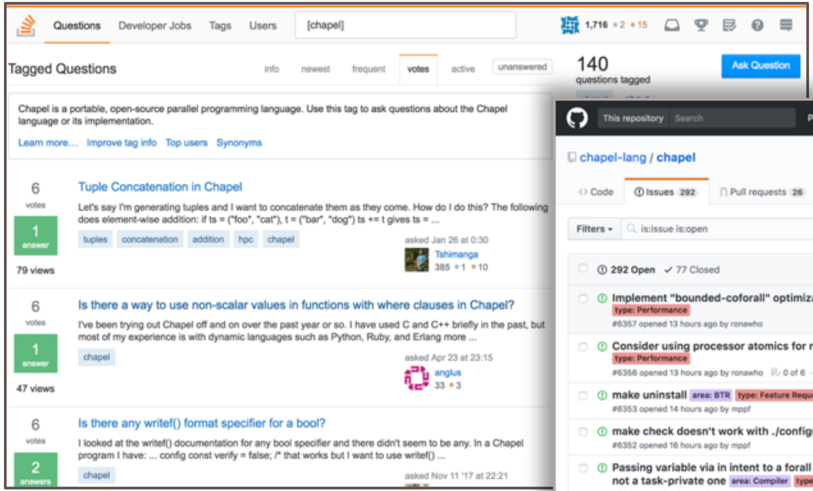
Chapel Online Documentation

CRAY

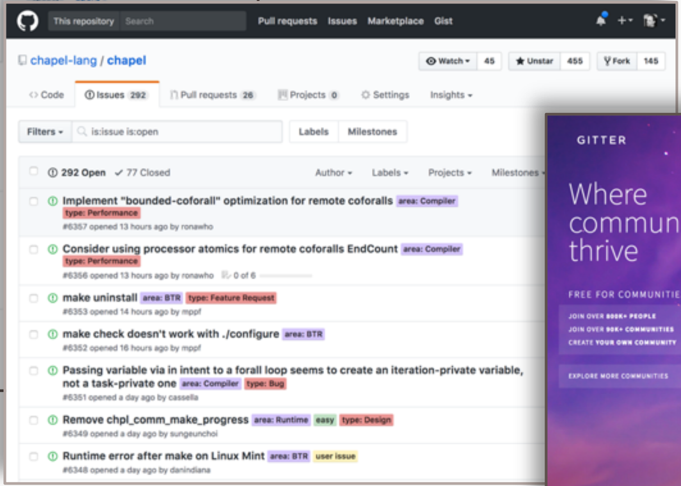
https://chapel-lang.org/docs: ~200 pages, including primer examples



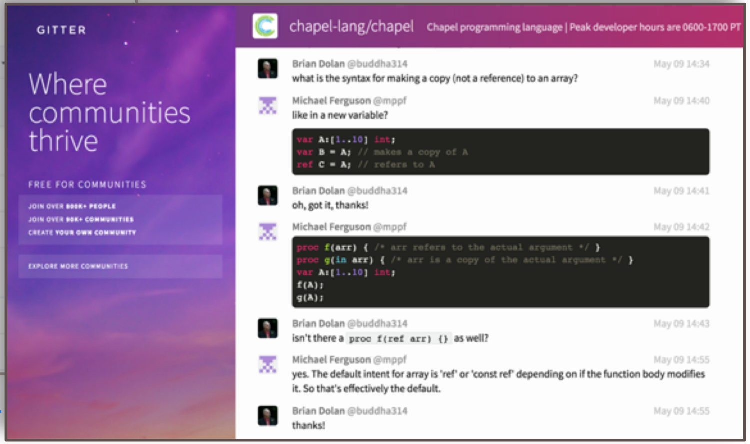
Chapel Community



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



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



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

read-only mailing list: chapel-announce@lists.sourceforge.net (~15 mails / year)

Chapel Social Media (no account required)

http://twitter.com/ChapelLanguage

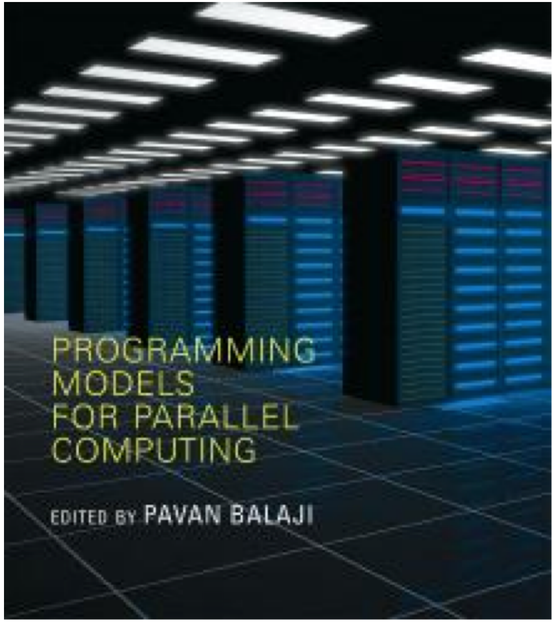
http://facebook.com/ChapelLanguage

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

Suggested Reading: Chapel history and overview

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



Suggested Reading: Recent Progress (CUG 2018)

CRAY

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
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 in terms of portability, and with C++ in terms of flexibility and extensibility. Chapel is designed to be general-purpose in the sense that when you have a parallel algorithm in mind and a parallel system on which you wish to run it, Chapel should be able to handle that scenario.

Chapel's design and implementation are led by Cray Inc. with feedback and code contributed by users and the open-source community. Though developed by Cray, Chapel's design and implementation are portable, permitting its programs to scale up from multicore laptops to commodity clusters to Cray systems. In addition, Chapel programs can be run on cloud-computing platforms and HPC systems from other vendors. Chapel is being developed in an open-source manner under the Apache 2.0 license and is hosted at GitHub.[1]

The development of the Chapel language was undertaken by Cray Inc. as part of its participation in the DARPA High Productivity Computing Systems program (HPCS). HPCS wrapped up in late 2012, at which point Chapel was a compelling prototype, having successfully demonstrated several key research challenges that the project had undertaken. Chief among these was supporting data- and task-parallelism in a unified manner within a single language. This was accomplished by supporting the creation of high-level data-parallel abstractions like parallel loops and arrays in terms of lower-level Chapel features such as classes, iterators, and tasks.

Under HPCS, Chapel also successfully supported the expression of parallelism using distinct language features from those used to control locality and affinity—that is, Chapel programmers specify *which* computations should run in parallel distinctly from specifying *where* those computations should be run. This permits Chapel programs to support multicore, multi-node, and heterogeneous computing within a single unified language.

Chapel's implementation under HPCS demonstrated that the language could be implemented portably while still being optimized for HPC-specific features such as the RDMA support available in Cray® Gemini™ and Aries™ networks. This allows Chapel to take advantage of native hardware support for remote puts, gets, and atomic memory operations.

Despite these successes, at the close of HPCS, Chapel was not at all ready to support production codes in the field. This was not surprising given the language's aggressive design and modest-sized research team. However, reactions from potential users were sufficiently positive that, in early 2013, Cray embarked on a follow-up effort to improve Chapel and move it towards being a production-ready language. Colloquially, we refer to this effort as "the five-year push."

This paper's contribution is to describe the results of this five-year effort, providing readers with an understanding of Chapel's progress and achievements since the end of the HPCS program. In doing so, we directly compare the status of Chapel version 1.17, released last month, with Chapel version 1.7, which was released five years ago in April 2013.

[1] https://github.com/chapel-lang/chapel

paper and slides available at chapel-lang.org

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.

THANK YOU

QUESTIONS?

bradc@cray.com
@ChapelLanguage
chapel-lang.org

cray.com
@cray_inc
linkedin.com/company/cray-inc-/