

Chapel: Programmability, Parallelism, and Performance

Summer 2018 overview

Brad Chamberlain, Chapel Team, Cray Inc.



What is Chapel?



Chapel: A productive parallel programming language

- portable & scalable
- open-source & collaborative

Goals:

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



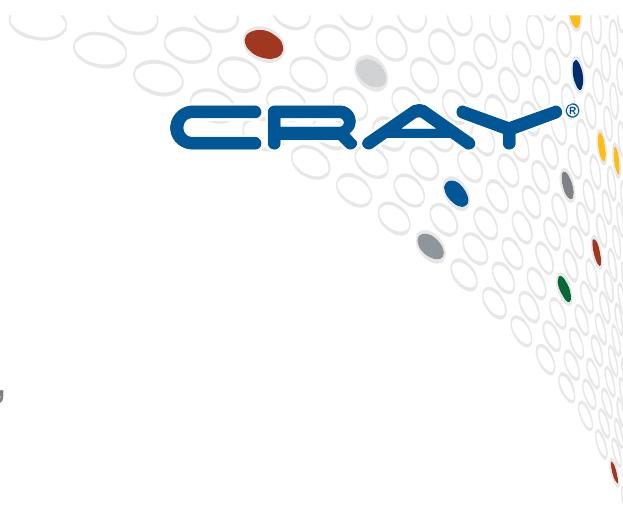
COMPUTE

| STORE |

ANALYZE

Copyright 2018 Cray Inc.

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 that I don’t need because I ~~was born to suffer~~
want full control to ensure 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.”



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel and Productivity



Chapel aims to be as...

- ...programmable as Python**
- ...fast as Fortran**
- ...scalable as MPI, SHMEM, or UPC**
- ...portable as C**
- ...flexible as C++**
- ...fun as [your favorite programming language]**



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



The Challenge

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

A: ~~Technical challenges?~~

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

A: Due to a lack 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

Copyright 2018 Cray Inc.

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 x ~27 languages x many implementations
 - exercise key computational idioms
 - specific approach prescribed



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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!

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

Which are fast? Trust, and verify

{ for researchers }

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

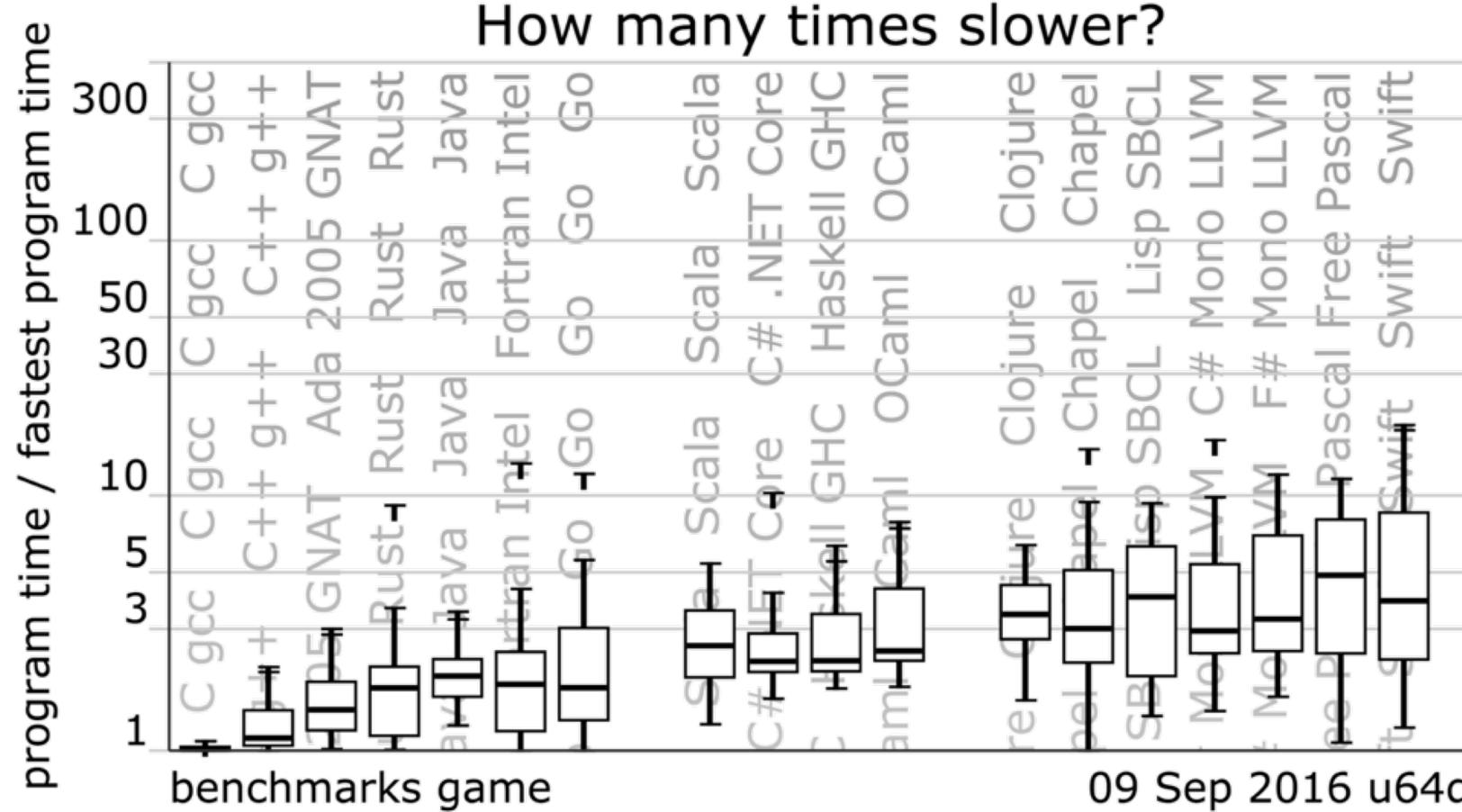
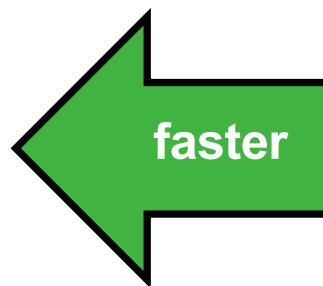
ANALYZE

Copyright 2018 Cray Inc.

CLBG: Fast-faster-fastest graph (Sep 2016)



Relative performance, sorted by geometric mean

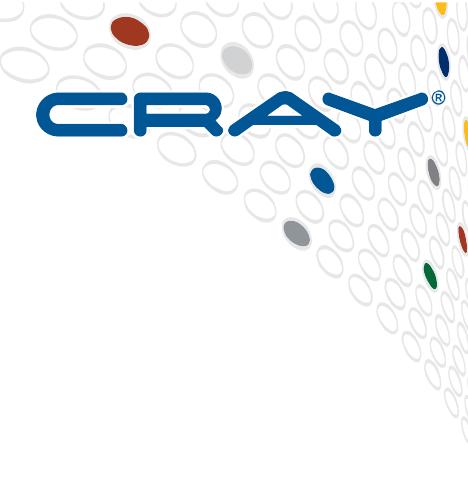


COMPUTE

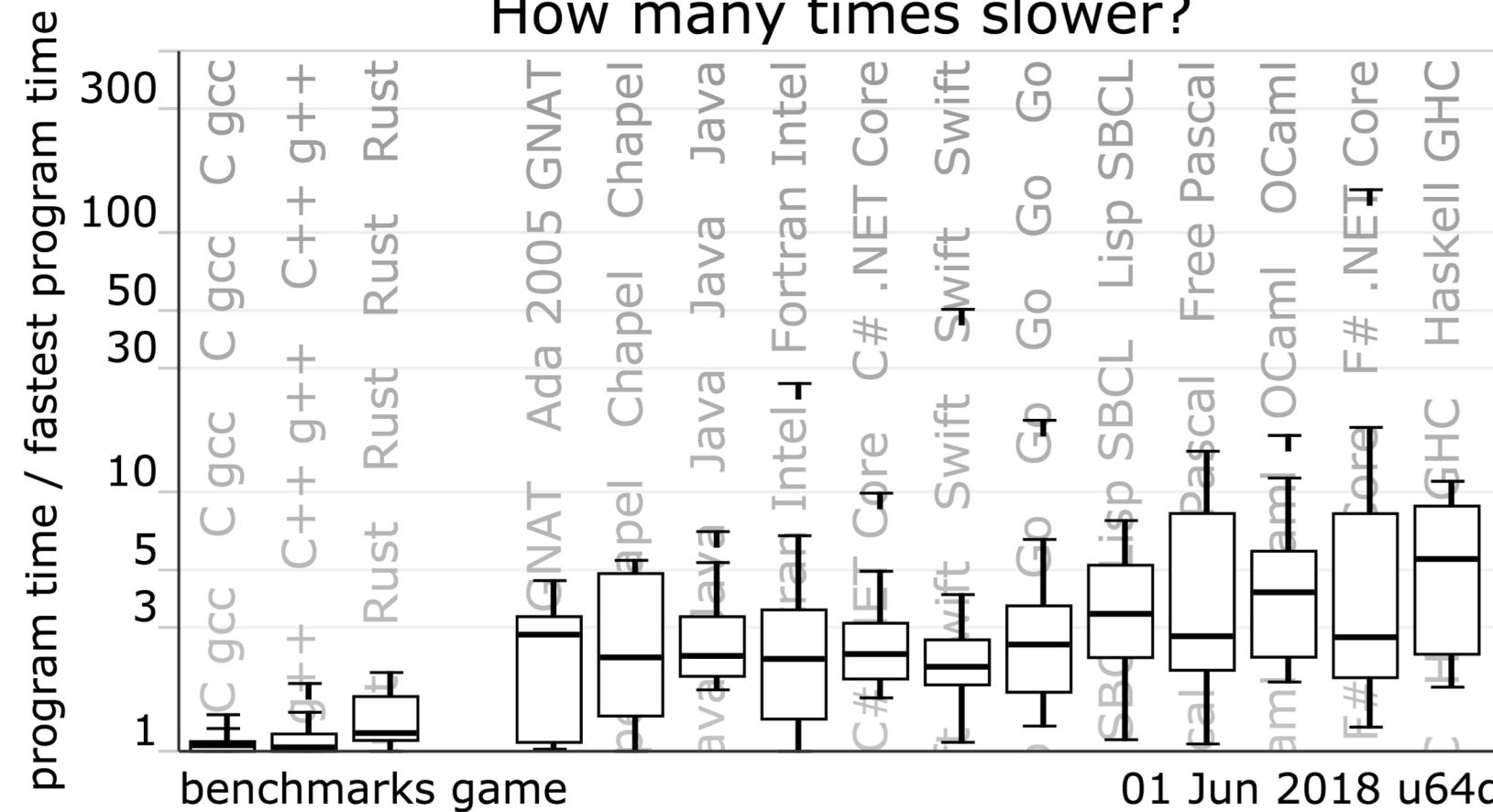
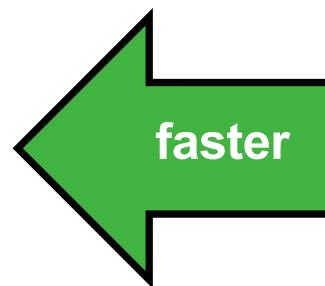
STORE

ANALYZE

CLBG: Fast-faster-fastest graph (June 2018)



Relative performance, sorted by geometric mean



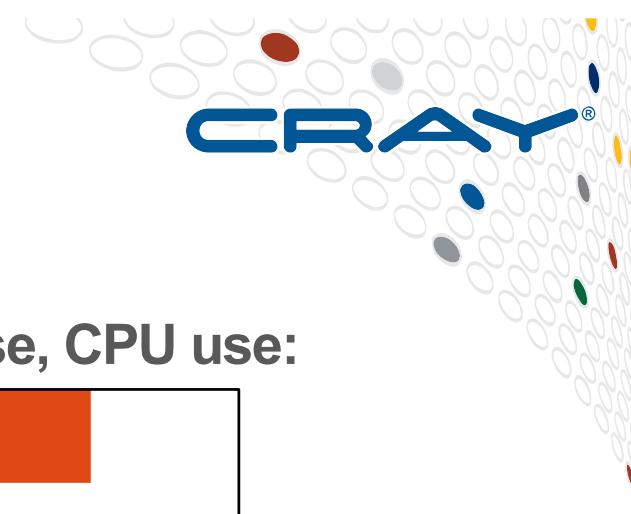
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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						
x	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						
x	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	1% 0% 1% 1%
1.7	C gcc	1.75	2,728	452	1.74	1% 0% 1% 1%
1.7	Racket	27.58	124,156	453	27.56	1% 0% 1% 1%
1.8	OCaml #5	6.72	19,836	458	6.71	1% 0% 1% 1%
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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Can also compare languages pair-wise:

- but only sorted by execution speed...

The Computer Language Benchmarks Game						
Chapel versus C++ g++ fastest programs						
	<u>vs C</u>	<u>vs C++</u>	<u>vs Go</u>	<u>vs Java</u>	<u>vs Python</u>	
by faster benchmark performance						
<u>reverse-complement</u>						
source	secs	mem	gz	cpu		cpu load
<u>Chapel</u>	2.20	1,497,876	707	5.10	96% 42% 58%	38%
<u>C++ g++</u>	2.95	980,472	2280	4.56	50% 41% 16%	50%
<u>pidigits</u>						
source	secs	mem	gz	cpu		cpu load
<u>Chapel</u>	1.62	6,488	501	1.63	99% 1% 1%	1%
<u>C++ g++</u>	1.89	4,284	513	1.88	5% 0% 1%	100%
<u>fannkuch-redux</u>						
source	secs	mem	gz	cpu		cpu load
<u>Chapel</u>	12.07	4,556	728	48.05	100% 100% 100%	100%
<u>C++ g++</u>	10.62	2,040	980	41.91	100% 95% 100%	100%

The Computer Language Benchmarks Game						
Chapel versus Python 3 fastest programs						
	<u>vs C</u>	<u>vs C++</u>	<u>vs Go</u>	<u>vs Java</u>	<u>vs Python</u>	
by faster benchmark performance						
<u>mandelbrot</u>						
source	secs	mem	gz	cpu		cpu load
<u>Chapel</u>	5.09	36,328	620	20.09	99% 99%	99% 99%
<u>Python 3</u>	279.68	49,344	688	1,117.29	100% 100%	100% 100%
<u>spectral-norm</u>						
source	secs	mem	gz	cpu		cpu load
<u>Chapel</u>	3.97	5,488	310	15.75	99% 99%	99% 99%
<u>Python 3</u>	193.86	50,556	443	757.23	98% 98%	99% 99%
<u>fannkuch-redux</u>						
source	secs	mem	gz	cpu		cpu load
<u>Chapel</u>	12.07	4,556	728	48.05	100% 100%	100% 100%
<u>Python 3</u>	547.23	48,052	950	2,162.70	99% 100%	97% 100%



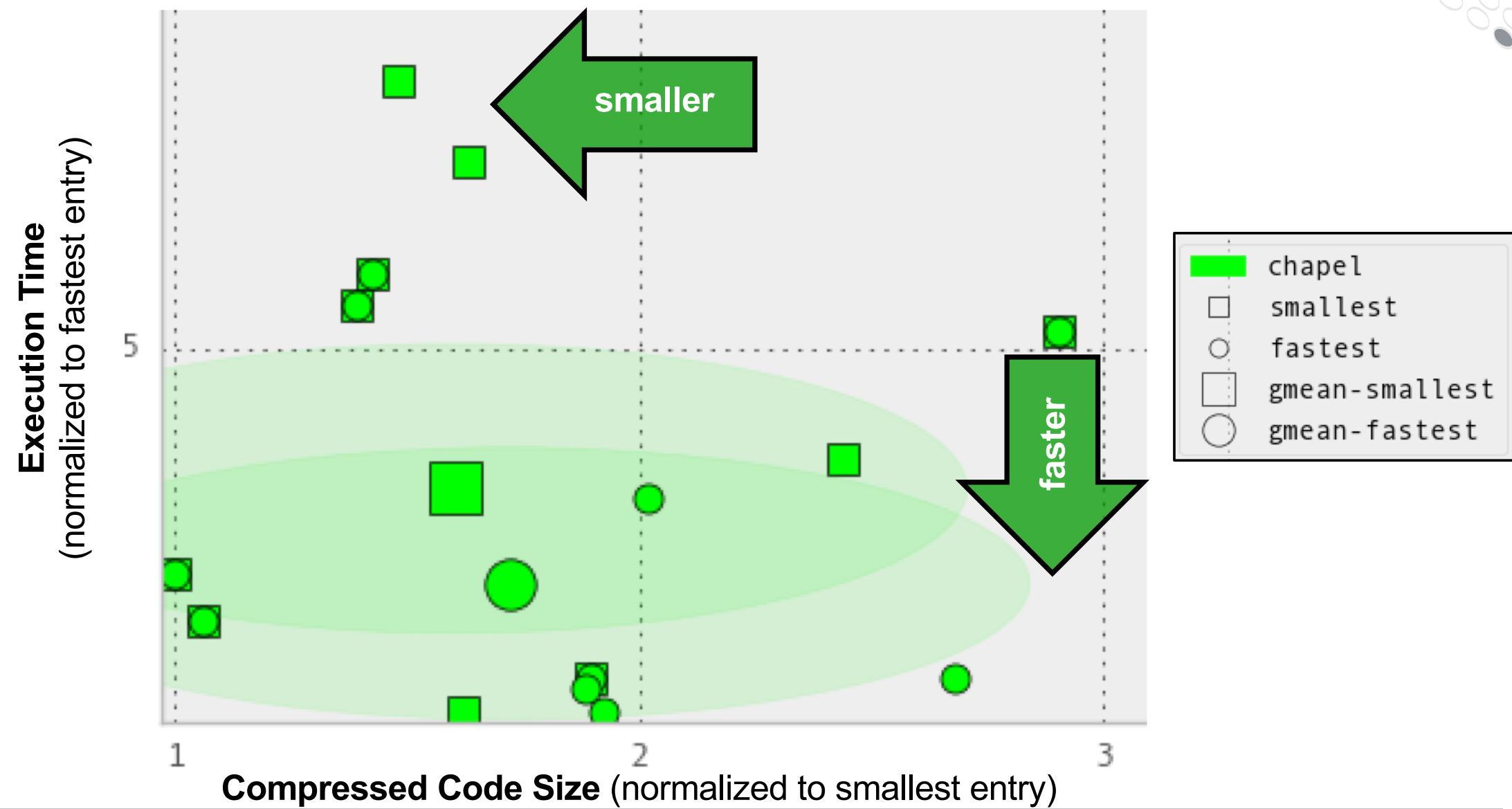
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

CLBG: Chapel Entries (September 21, 2018)

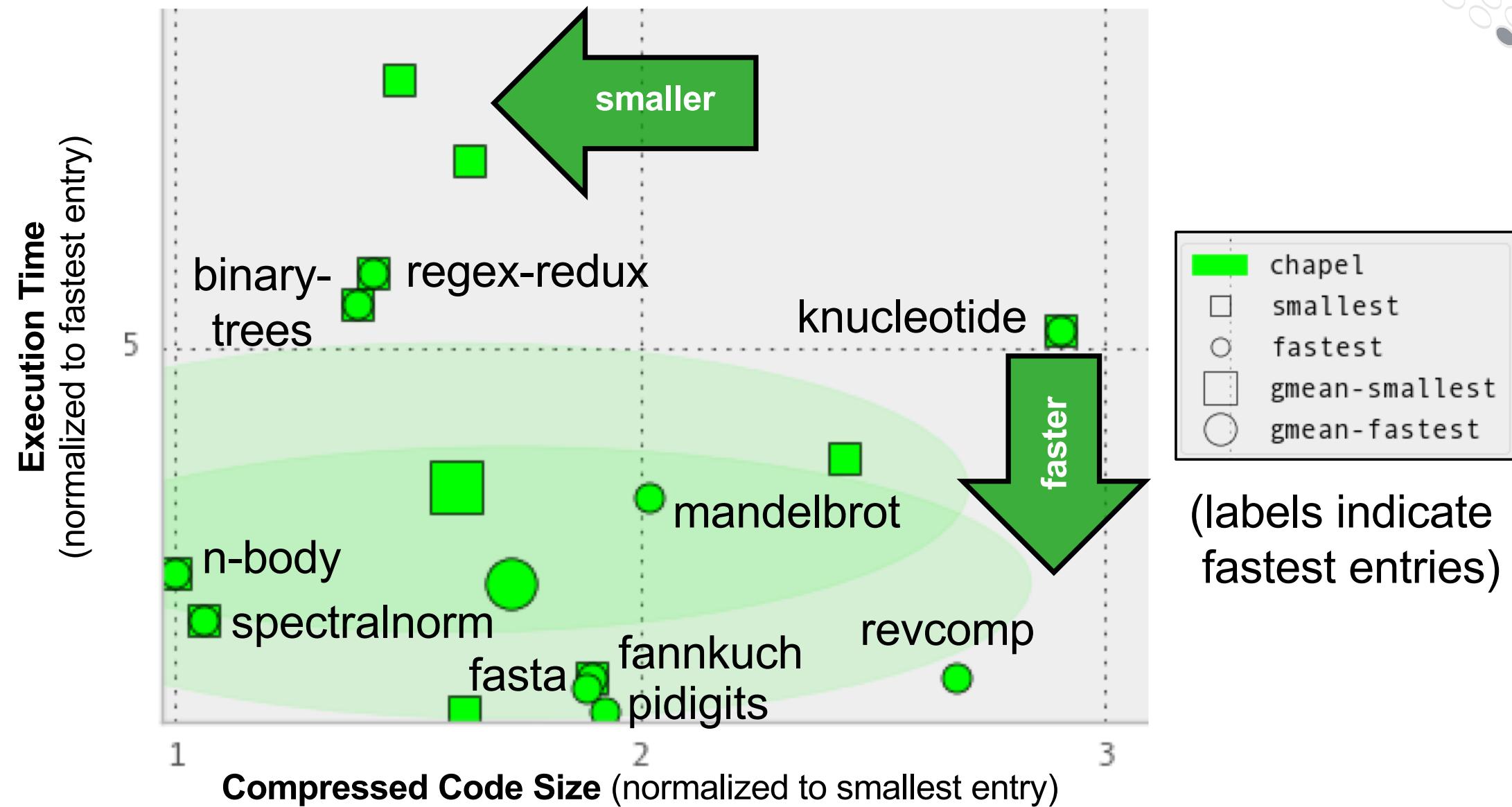


COMPUTE

STORE

ANALYZE

CLBG: Chapel Entries (September 21, 2018)



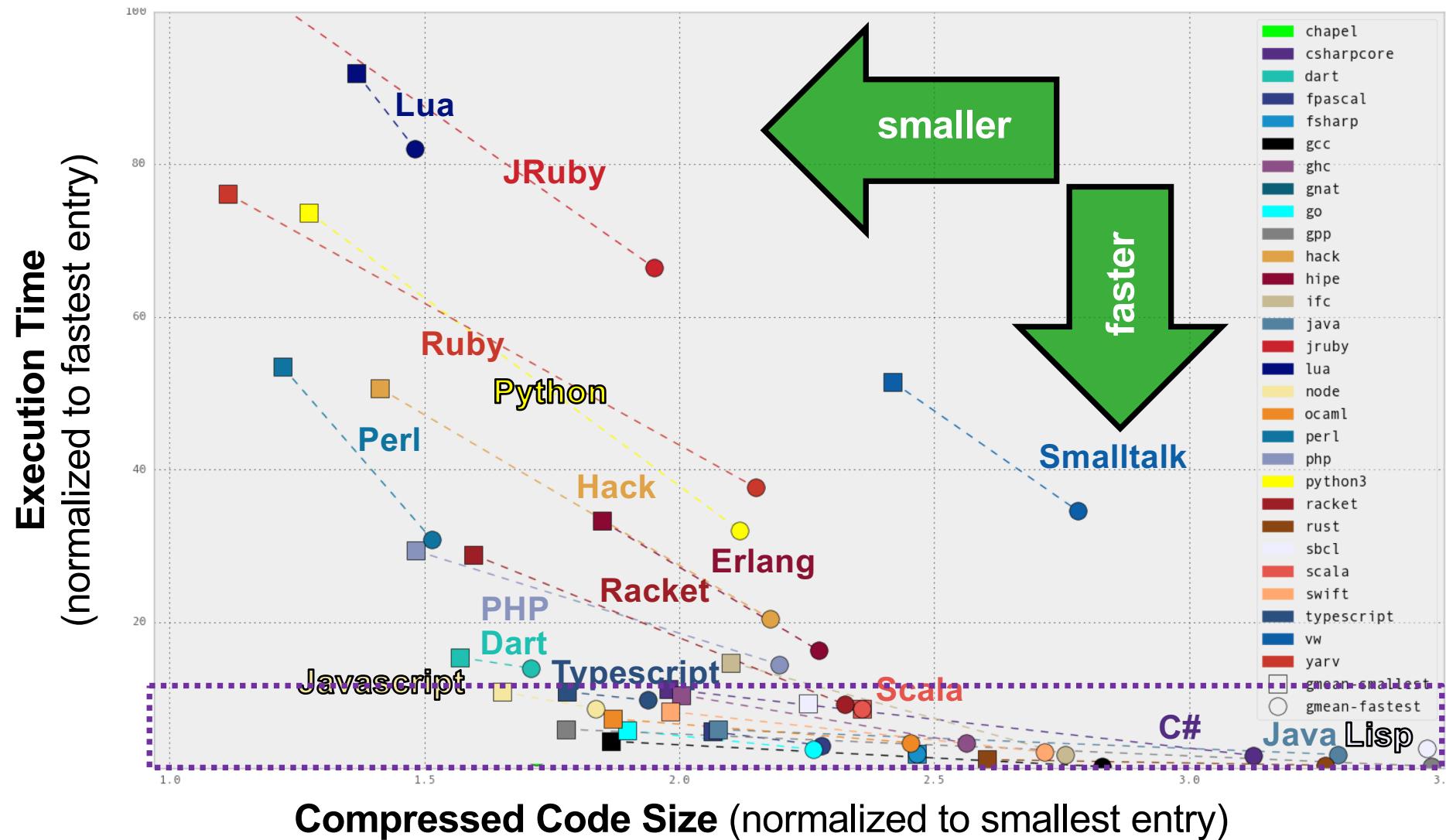
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(September 21, 2018 standings)



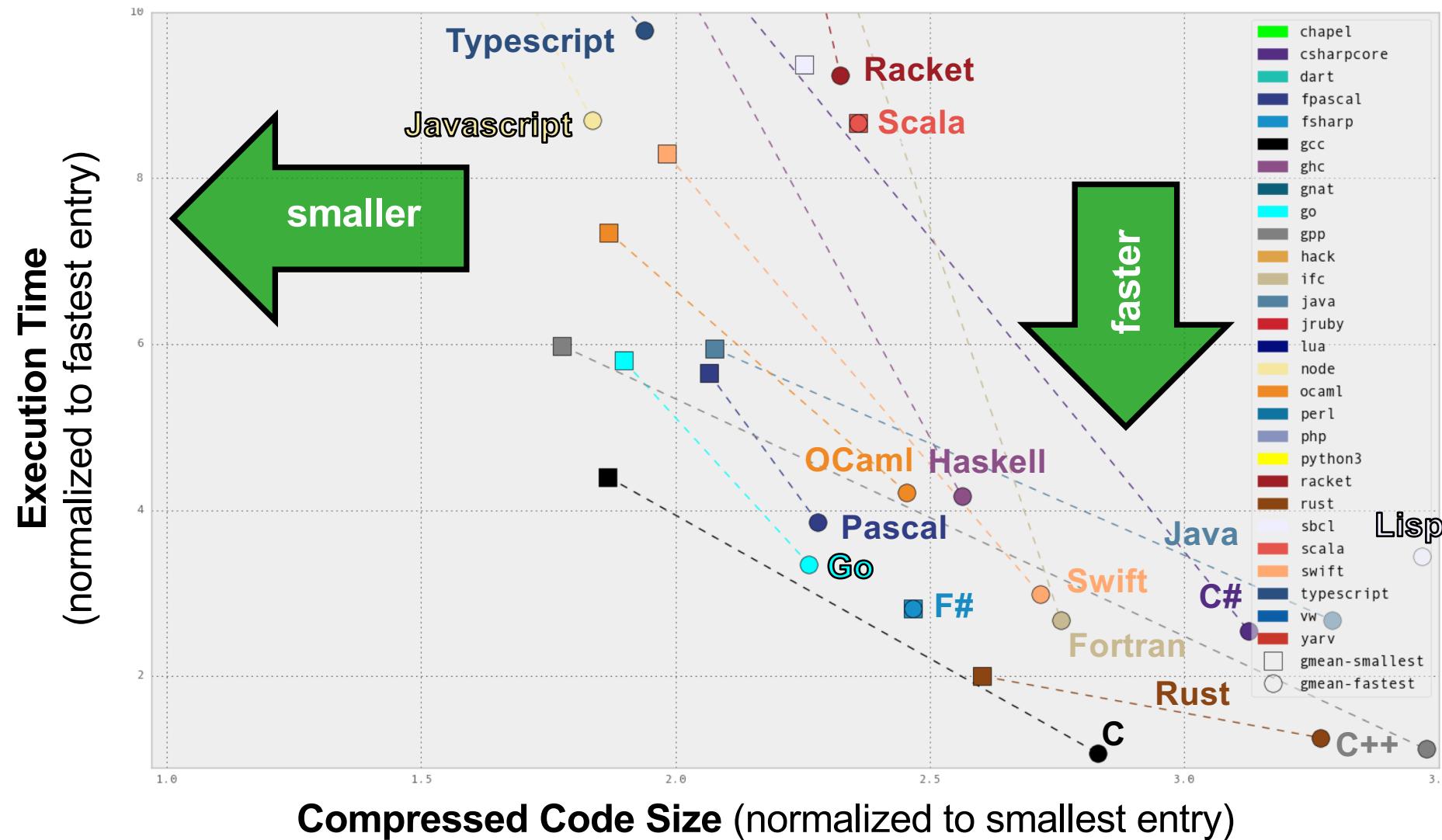
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(September 21, 2018 standings, zoomed in)



COMPUTE

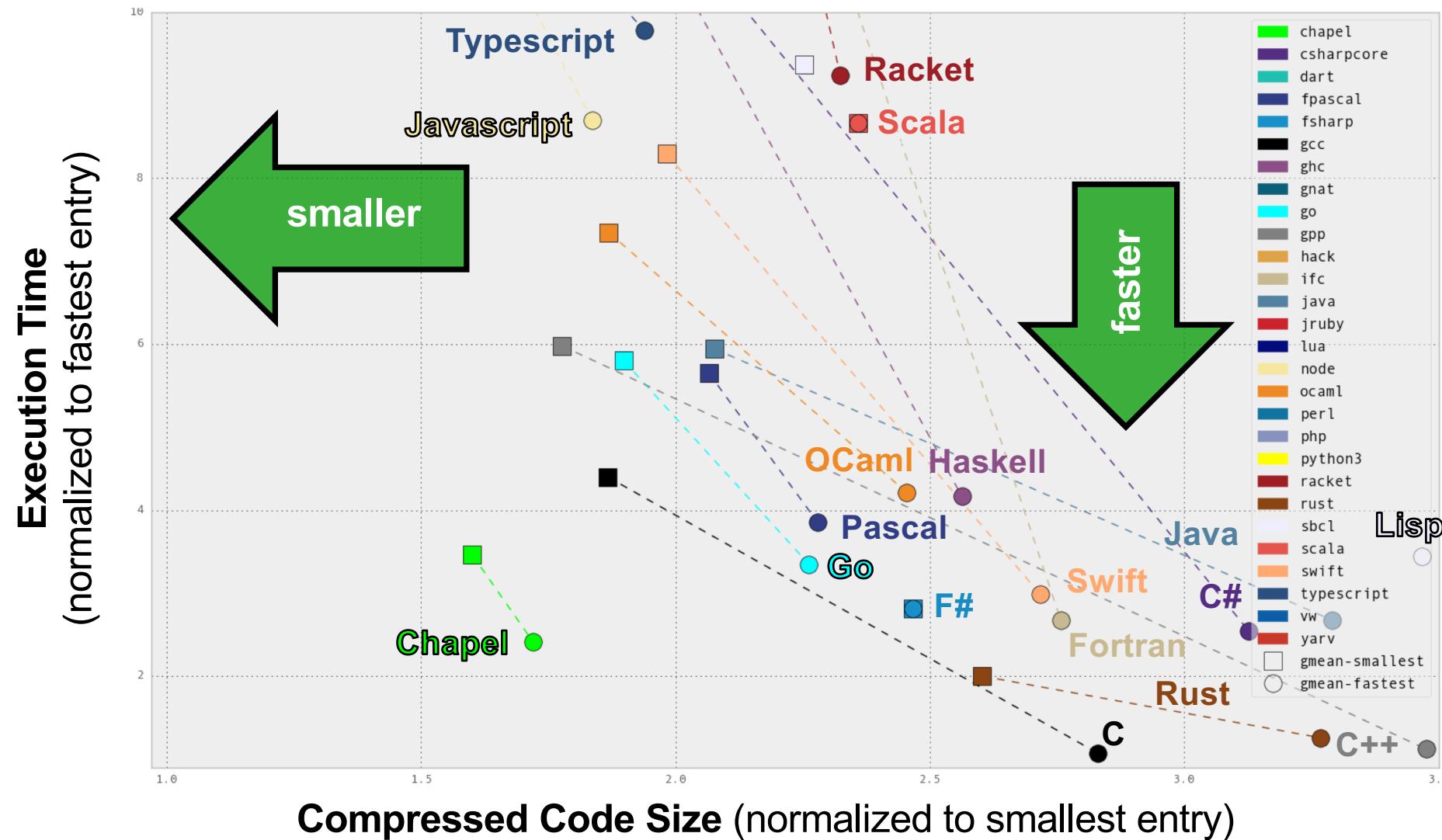
STORE

ANALYZE

Copyright 2018 Cray Inc.

CLBG Cross-Language Summary

(September 21, 2018 standings, zoomed in)



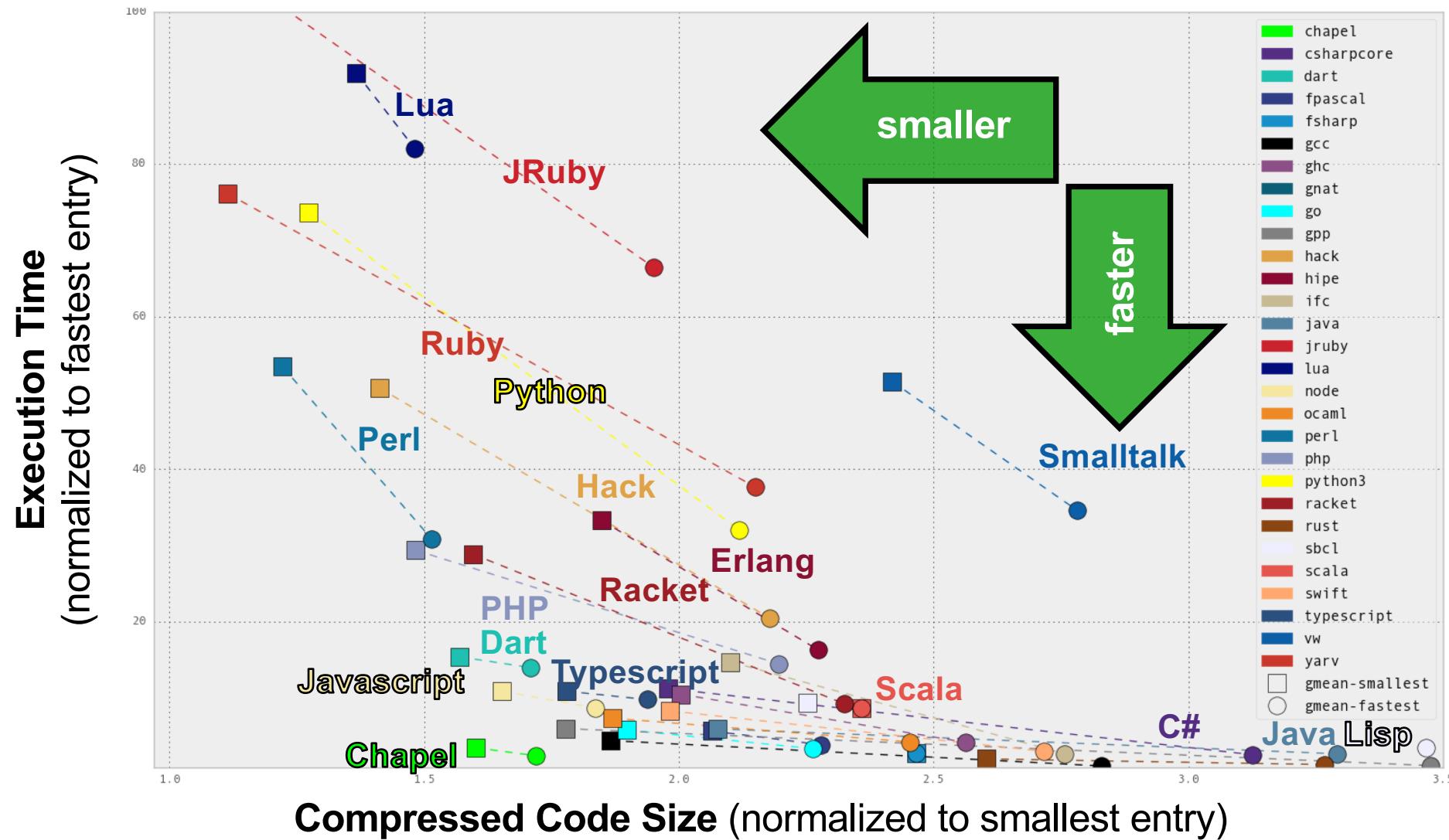
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(September 21, 2018 standings)



COMPUTE

STORE

ANALYZE

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

    char const* 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);
    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] = 0;
        return;
    }

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

excerpt from 2863.gz C gcc entry



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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, 0);
    const group2 = [i in 1..popSize2] new Chameneos(i, 0);

    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

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

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

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

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;
}

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

excerpt from 2863.gz C gcc entry



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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

excerpt from 2863.gz C gcc entry



COMPUTE

STORE

ANALYZE



The Chapel Team at Cray (May 2018)



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Community Partners



Lawrence Berkeley
National Laboratory



Yale

(and several others...)

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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Outline

✓ What is Chapel?

➤ Overview of Chapel Features

- Chapel Results and Resources
- Wrap-up



COMPUTE

STORE

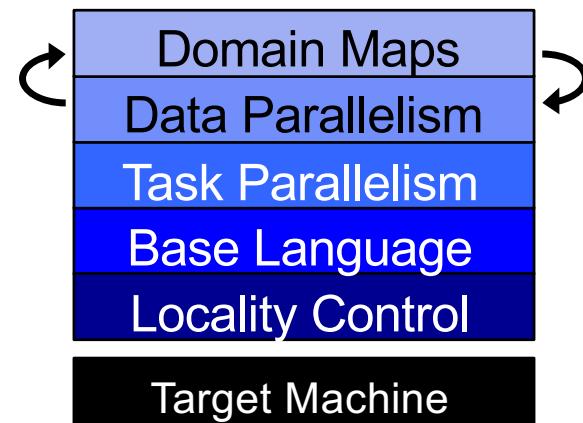
ANALYZE

Copyright 2018 Cray Inc.

Chapel language feature areas



Chapel language concepts



COMPUTE

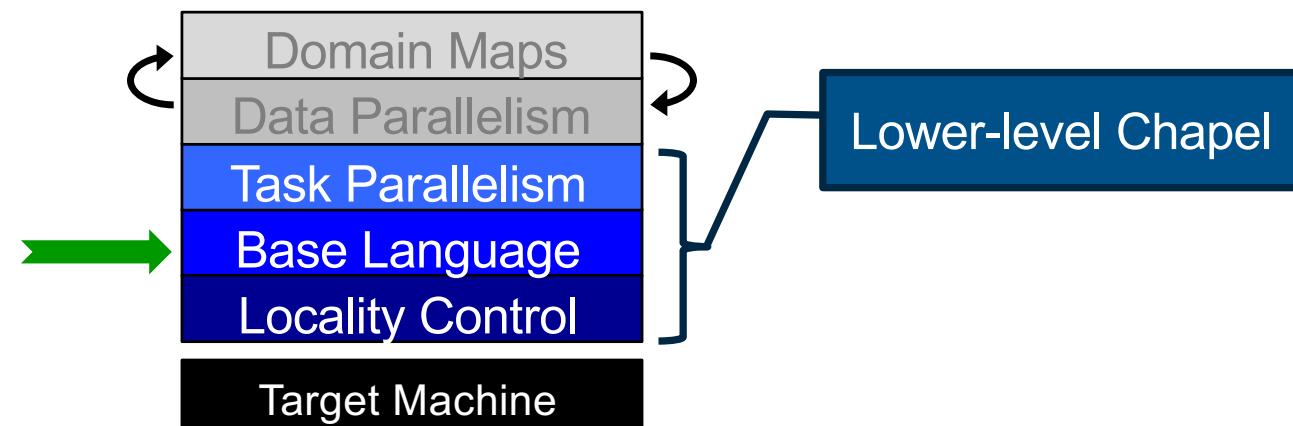
STORE

ANALYZE

Copyright 2018 Cray Inc.



Base Language



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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  
...
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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  
...
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



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  
...
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



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  
...
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



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  
...
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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  
...
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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  
...
```

Zippered iteration



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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  
...
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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);
```

Tuples

```
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  
...
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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  
...
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Other Key Base Language Features

- Object-oriented features
- Generic programming / polymorphism
- Procedure overloading / filtering
- Default args, arg intents, keyword-based arg passing
- Argument type queries / pattern-matching
- Compile-time meta-programming
- Modules (namespaces)
- Error-handling
- and more...



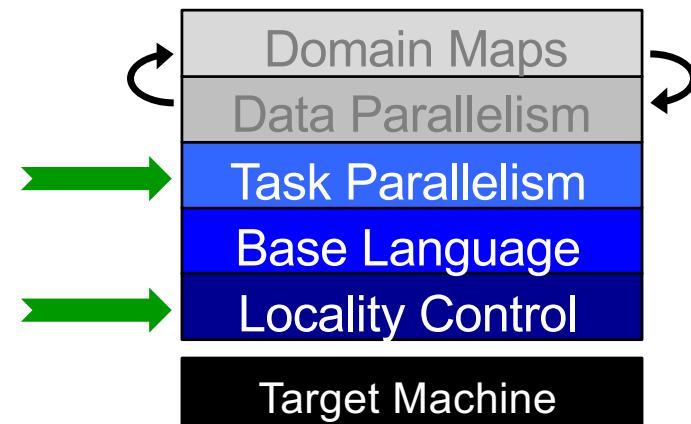
COMPUTE

STORE

ANALYZE



Task Parallelism and Locality Control



COMPUTE

STORE

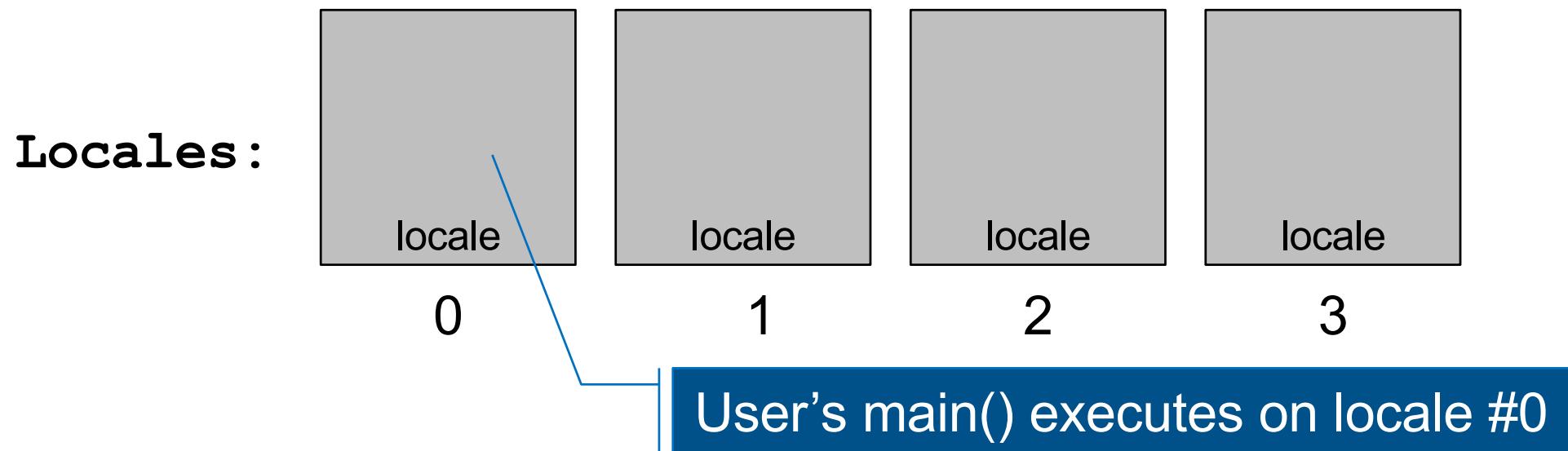
ANALYZE

Copyright 2018 Cray Inc.



Locales, briefly

- Locales can run tasks and store variables
 - Think “compute node”



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Task Parallelism and Locality, by example



This is a shared memory program
Nothing has referred 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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Task Parallelism and Locality, by example



Control of Locality/Affinity

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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Other Key Task Parallel Features



- **atomic / sync variables:** for sharing data & coordination
- **begin / cobegin statements:** other ways of creating tasks



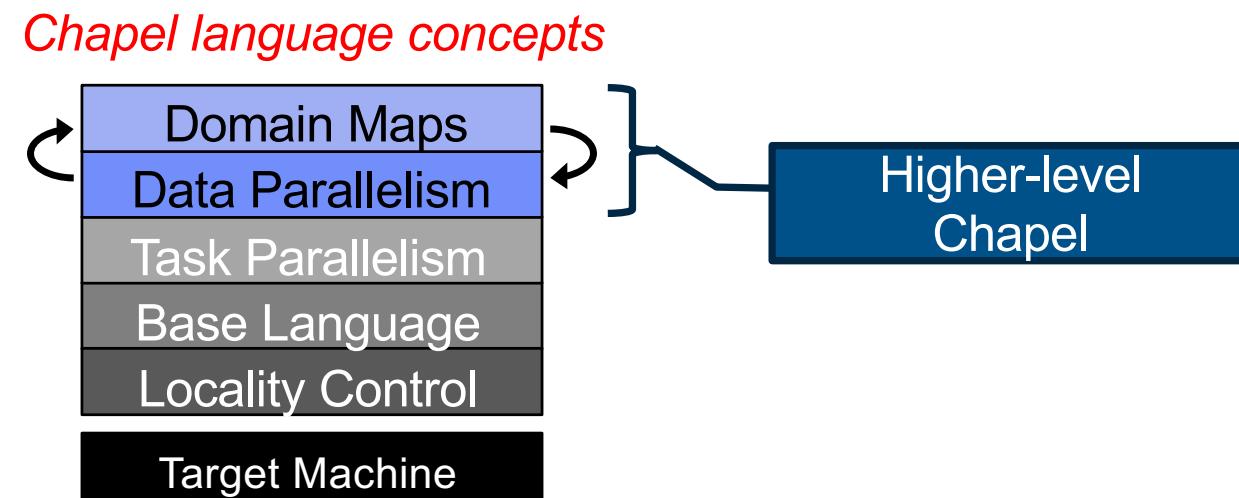
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Data Parallelism in Chapel



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Data Parallelism, by example



This is a shared memory program
Nothing has referred 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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Distributed Data Parallelism, by example



Domain Maps
(Map Data Parallelism to the System)

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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Other Key Data Parallel Features



- **Flavors of Domains/Arrays:** sparse, associative, ...
- **Slicing:** refer to subarrays using ranges / domains
- **Promotion:** call scalar arrays with array arguments
- **Reductions:** collapse arrays to scalars or subarrays
- **Scans:** compute parallel prefix operations



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Chapel Results and Resources



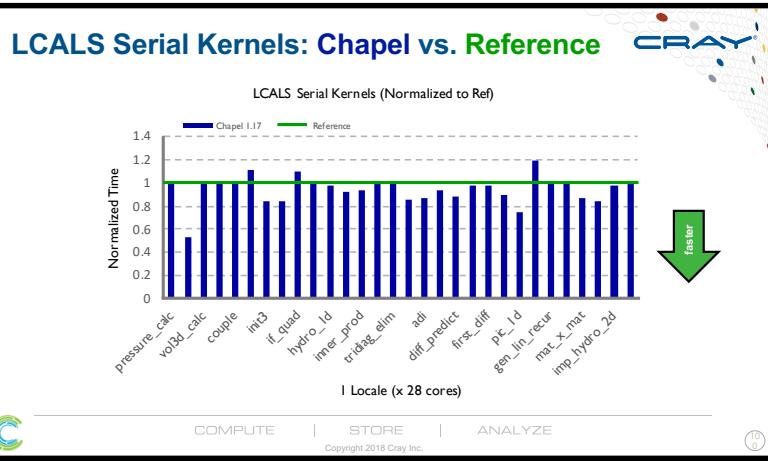
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

HPC Patterns: Chapel vs. Reference

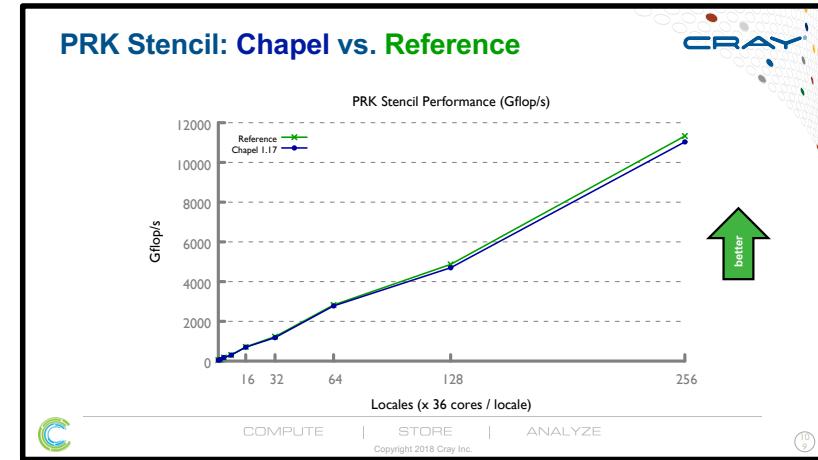
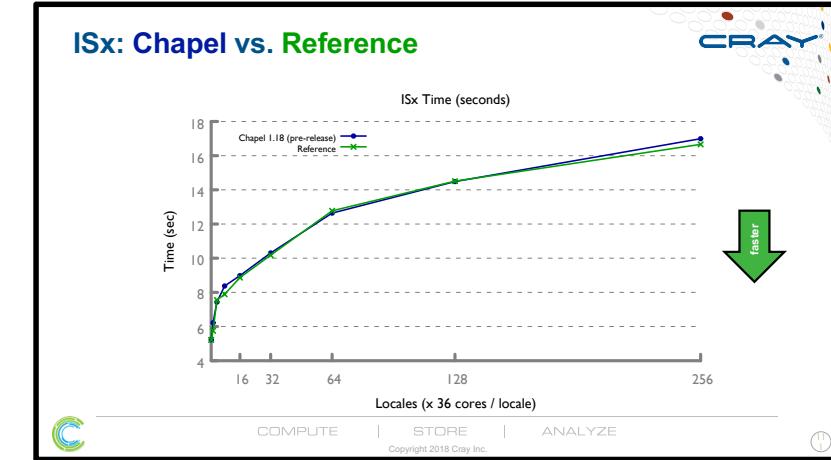
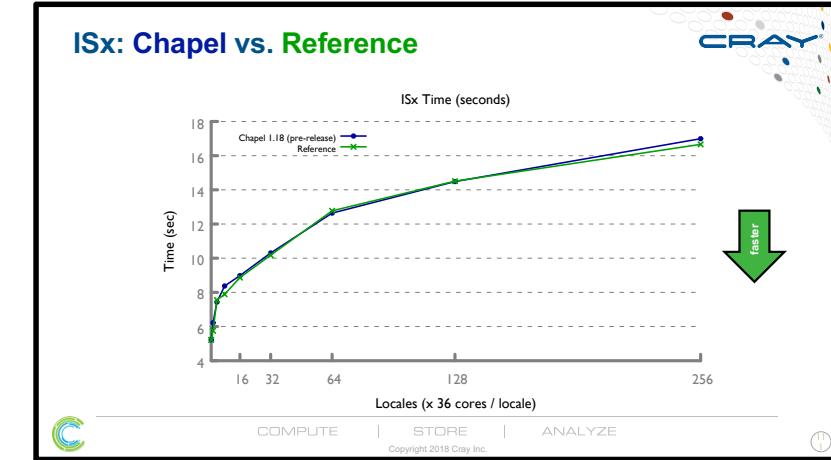
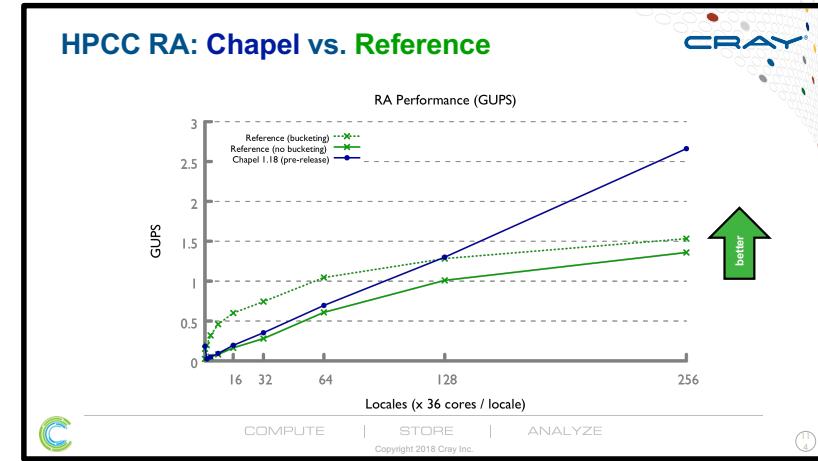


LCALS

HPCC RA

STREAM
Triad

PRK
Stencil



COMPUTE

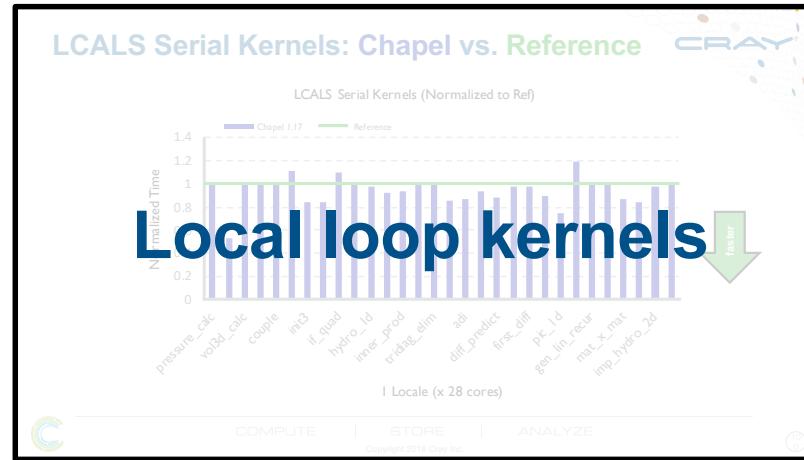
STORE

ANALYZE

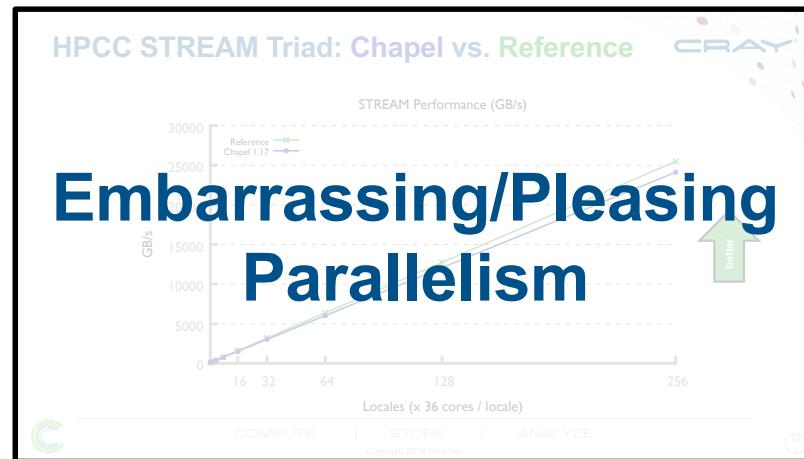
Copyright 2018 Cray Inc.

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

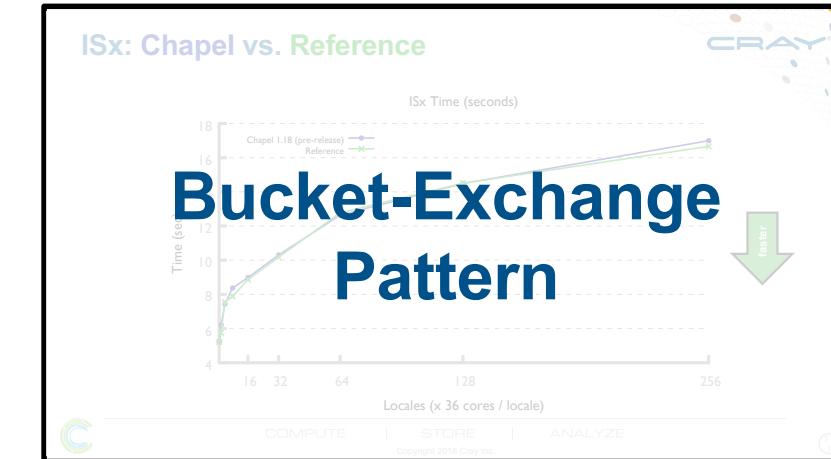
HPC Patterns: Chapel vs. Reference



LCALS

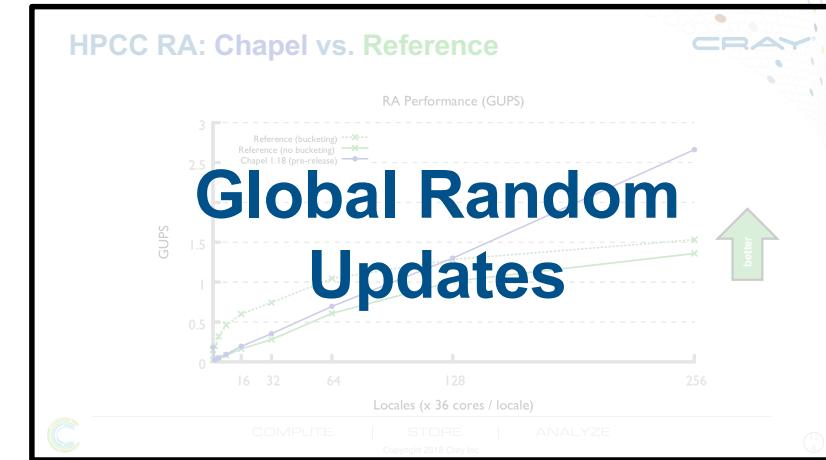


STREAM
Triad

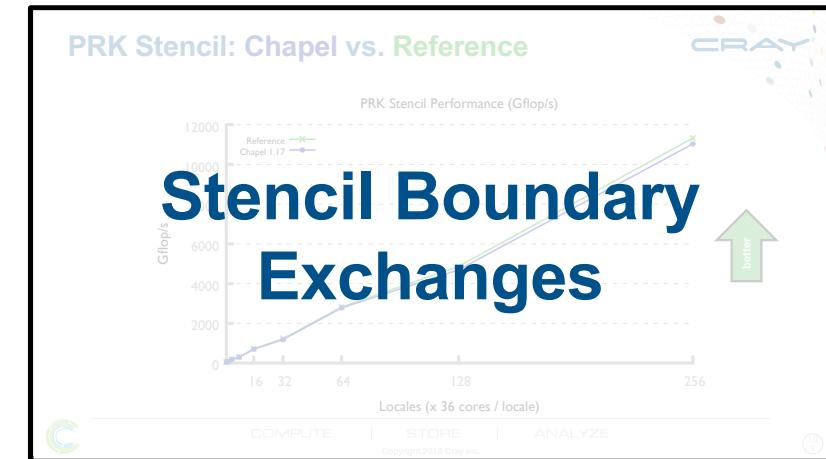


ISx

HPCC RA



PRK
Stencil



COMPUTE

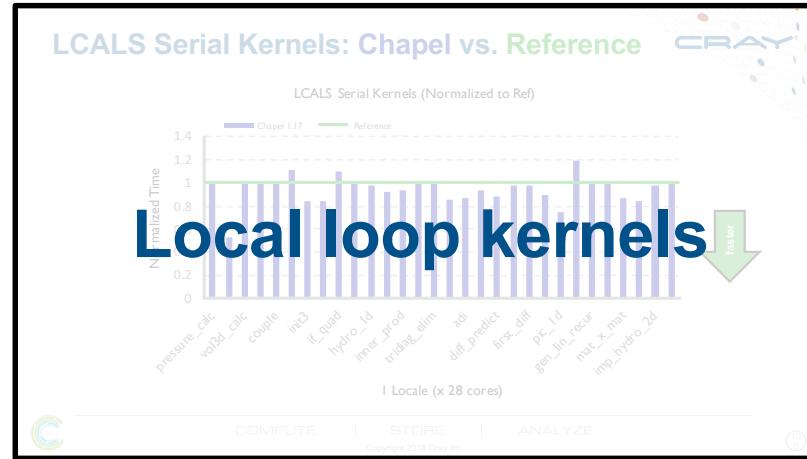
STORE

ANALYZE

Copyright 2018 Cray Inc.

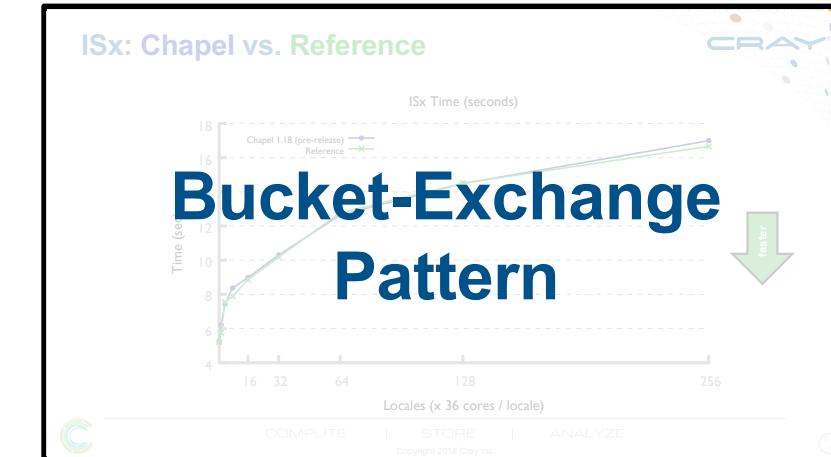
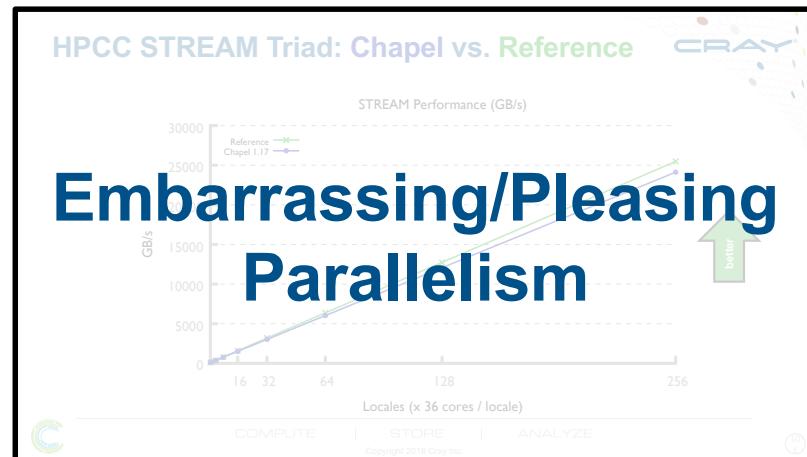
Nightly performance tickers online at:
<https://chapel-lang.org/perf-nightly.html>

HPC Patterns: Chapel vs. Reference



LCALS

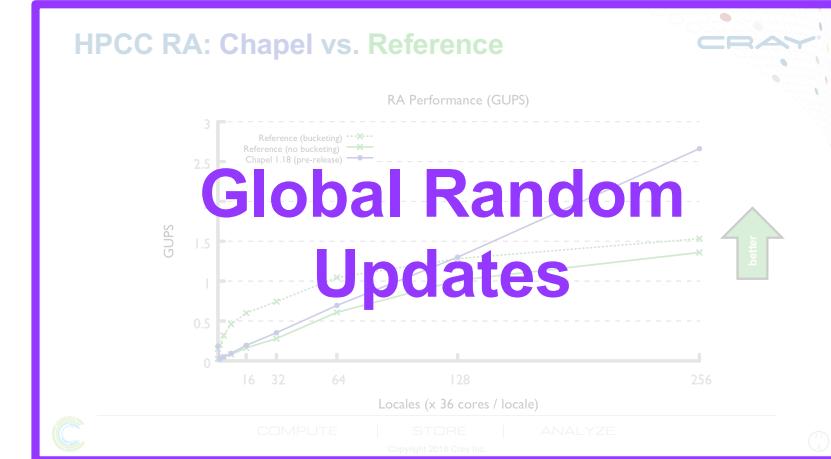
HPCC RA



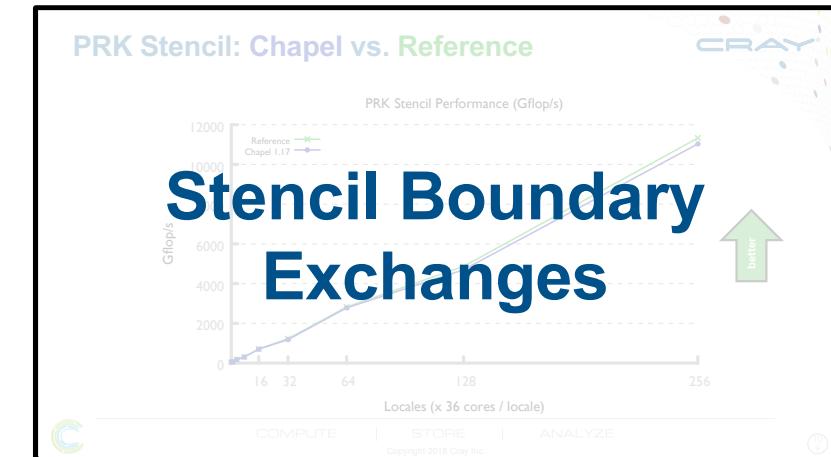
COMPUTE

STORE ANALYZE

Copyright 2018 Cray Inc.



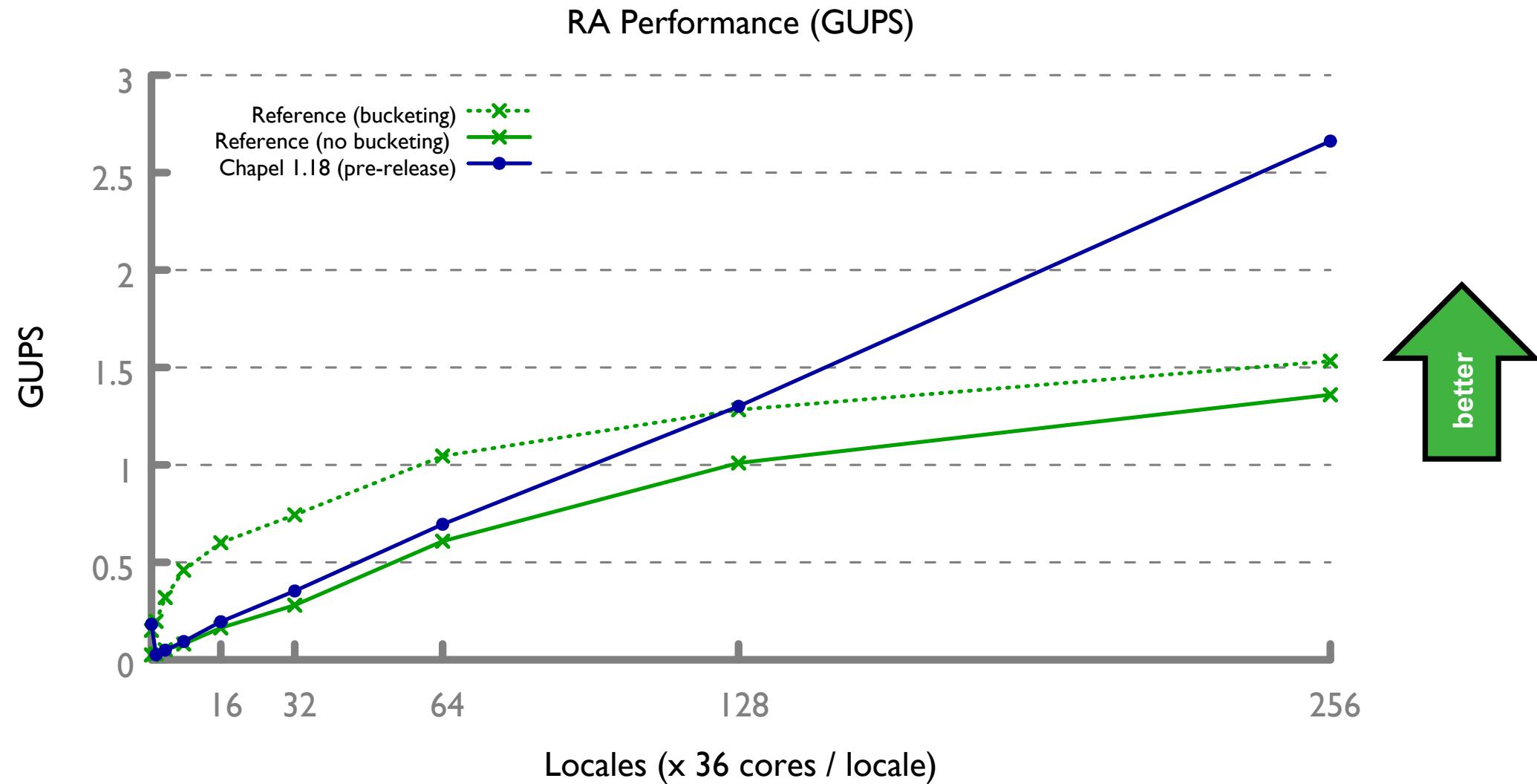
PRK
Stencil



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

HPCC RA: Chapel vs. Reference

CRAY®



COMPUTE

STORE | ANALYZE

Copyright 2018 Cray Inc.

HPCC Random Access Kernel: MPI



```

/* Perform updates to main table. The scalar equivalent is:
 *
 * for (i=0; i<NUPDATE; i++) {
 *   Ran = (Ran << 1) ^ ((s64Int) Ran < 0) ? POLY : 0;
 *   Table[Ran & (TABSIZE-1)] ^= Ran;
 * }
 */
MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
while (i < SendCnt) {
    /* receive messages */
    do {
        MPI_Test(&inreq, &have_done, &status);
        if (have_done) {
            if (status.MPI_TAG == UPDATE_TAG) {
                MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
                bufferBase = 0;
                for (j=0; j < recvUpdates; j++) {
                    inmsg = LocalRecvBuffer[bufferBase+j];
                    LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                                 tparams.GlobalStartMyProc;
                    HPCC_Table[LocalOffset] ^= inmsg;
                }
            } else if (status.MPI_TAG == FINISHED_TAG) {
                NumberReceiving--;
            } else
                MPI_Abort( MPI_COMM_WORLD, -1 );
            MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                      MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
        }
    } while (have_done && NumberReceiving > 0);
    if (pendingUpdates < maxPendingUpdates) {
        Ran = (Ran << 1) ^ ((s64Int) Ran < ZERO64B ? POLY : ZERO64B);
        GlobalOffset = Ran & (tparams.TableSize-1);
        if ( GlobalOffset < tparams.Top)
            WhichPe = ( GlobalOffset / (tparams.MinLocalTableSize + 1) );
        else
            WhichPe = ( (GlobalOffset - tparams.Remainder) /
                         tparams.MinLocalTableSize );
        if (WhichPe == tparams.MyProc) {
            LocalOffset = (Ran & (tparams.TableSize - 1)) -
                         tparams.GlobalStartMyProc;
            HPCC_Table[LocalOffset] ^= Ran;
        }
    } else {
        HPCC_InsertUpdate(Ran, WhichPe, Buckets);
        pendingUpdates++;
    }
    i++;
}
else {
    MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
    if (have_done) {
        outreq = MPI_REQUEST_NULL;
        pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                             &peUpdates);
        MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
                  UPDATE_TAG, MPI_COMM_WORLD, &outreq);
        pendingUpdates -= peUpdates;
    }
}
/* send remaining updates in buckets */
while (pendingUpdates > 0) {
    /* receive messages */
    do {
        MPI_Test(&inreq, &have_done, &status);
        if (have_done) {
            if (status.MPI_TAG == UPDATE_TAG) {
                MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
                bufferBase = 0;
                for (j=0; j < recvUpdates; j++) {
                    inmsg = LocalRecvBuffer[bufferBase+j];
                    LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                                 tparams.GlobalStartMyProc;
                    HPCC_Table[LocalOffset] ^= inmsg;
                }
            } else if (status.MPI_TAG == FINISHED_TAG) {
                /* we got a done message. Thanks for playing... */
                NumberReceiving--;
            } else
                MPI_Abort( MPI_COMM_WORLD, -1 );
            MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                      MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
        }
    } while (have_done && NumberReceiving > 0);
    MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
    if (have_done) {
        outreq = MPI_REQUEST_NULL;
        pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                             &peUpdates);
        MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
                  UPDATE_TAG, MPI_COMM_WORLD, &outreq);
        pendingUpdates -= peUpdates;
    }
}
/* send our done messages */
for (proc_count = 0 ; proc_count < tparams.NumProcs ; ++proc_count) {
    if (proc_count == tparams.MyProc) { tparams.finish_req[tparams.MyProc] =
                                         MPI_REQUEST_NULL; continue; }
    /* send garbage - who cares, no one will look at it */
    MPI_Isend(&Ran, 0, tparams.dtype64, proc_count, FINISHED_TAG,
              MPI_COMM_WORLD, tparams.finish_req + proc_count);
}
/* Finish everyone else up... */
while (NumberReceiving > 0) {
    MPI_Wait(&inreq, &status);
    if (status.MPI_TAG == UPDATE_TAG) {
        MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
        bufferBase = 0;
        for (j=0; j < recvUpdates; j++) {
            inmsg = LocalRecvBuffer[bufferBase+j];
            LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                         tparams.GlobalStartMyProc;
            HPCC_Table[LocalOffset] ^= inmsg;
        }
    } else if (status.MPI_TAG == FINISHED_TAG) {
        /* we got a done message. Thanks for playing... */
        NumberReceiving--;
    } else
        MPI_Abort( MPI_COMM_WORLD, -1 );
    MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
              MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
}
MPI_Waitall( tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);

```



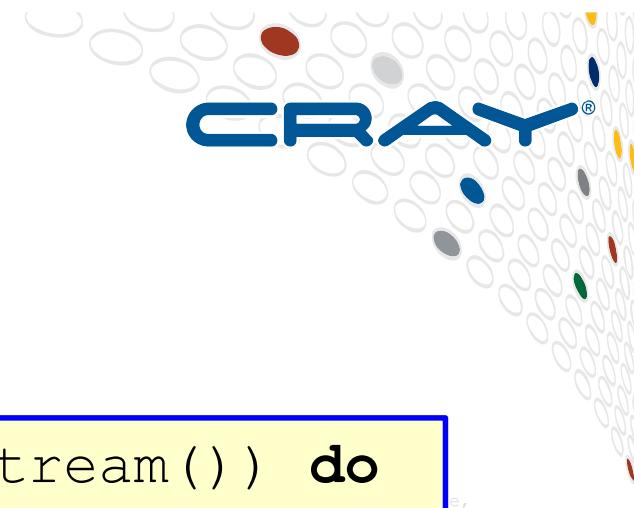
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

HPCC Random Access Kernel: MPI



```

/* Perform updates to main table. The scalar equivalent is:
 */
*   for (i=0; i<NUPDATE; i++) {
*     Ran = (Ran << 1) ^ ((s64lInt) Ran < 0) ? POLY : 0);
*     Table[Ran & (TABSIZ-1)] ^= Ran;
*   }
*/
```

```

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
          MPI_STATUS_IGNORE);

while (i < SendCnt) {
    /* receive messages */
    do {
        MPI_Test(&inreq, &have_done, &status);
        if (have_done) {
            if (status.MPI_TAG == UPDATE_TAG) {
                MPI_Get_count(&status, tparams.dtype64, &recvUp-
                bufferBase = 0;
```

```
Chapel Kernel
forall (_, r) in zip(Updates, RASTream()) do
    T[r & indexMask] ^= r;
```

MPI Comment

```
/* Perform updates to main table. The scalar equivalent is:  
*  
*      for (i=0; i<NUPDATE; i++) {  
*          Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);  
*          Table[Ran & (TABSIZ-1)] ^= Ran;  
*      }  
*/
```

HPCC Table[LocalOffset] ^= Ran;

COMPLITE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Libraries



~60 library modules

- web-documented, many user-contributed

The image displays two side-by-side screenshots of the Chapel Documentation website, version 1.17. Both screenshots show a dark-themed sidebar with navigation links and a light-colored main content area.

Standard Modules Screenshot:

- Header:** Shows "Chapel Documentation" and "version 1.17".
- Left Sidebar:** Includes links for "COMPILING AND RUNNING CHAPEL" (Quickstart Instructions, Using Chapel, Platform-Specific Notes, Technical Notes, Tools) and "WRITING CHAPEL PROGRAMS" (Quick Reference, Hello World Variants, Primers, Language Specification, Built-in Types and Functions).
- Content Area:** A heading "Standard Modules" followed by a paragraph about standard modules being part of the Chapel Standard Library. Below this is a list of modules: Assert, Barriers, BigInteger, BitOps, CommDiagnostics, DateTime, DynamicIterators, FileSystem, GMP, Help, IO, List, Math, Memory, Path, Random, Reflection, Regexp, Spawn, Sys, SysBasic, SysCtypes, SysError, Time, Types, and UtilReplicatedVar.
- Right Sidebar:** Includes links for "View page source" and "Docs > Standard Modules".

Package Modules Screenshot:

- Header:** Shows "Chapel Documentation" and "version 1.17".
- Left Sidebar:** Same as the Standard Modules screenshot.
- Content Area:** A heading "Package Modules" followed by a paragraph about package modules living outside the standard library. Below this is a list of modules: AllLocalesBarriers, BLAS, Buffers, Collection, Crypto, Curl, DistributedBag, DistributedDeque, DistributedIterators, FFTW, FFTW_MT, Futures, HDFS, HDFSIterator, LAPACK, LinearAlgebra, MPI, Norm, OwnedObject, RangeChunk, RecordParser, ReplicatedVar, Search, SharedObject, Sort, VisualDebug, and ZMQ.
- Right Sidebar:** Includes links for "View page source" and "Docs > Package Modules".



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Libraries



Math: FFTW, BLAS, LAPACK, LinearAlgebra, Math

Inter-Process Communication: MPI, ZMQ (ZeroMQ)

Parallelism: Futures, Barrier, DynamicIterators

Distributed Computing: DistributedIterators, DistributedBag,
DistributedDeque, Block, Cyclic, Block-Cyclic, ...

File Systems: FileSystem, Path, HDFS

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



COMPUTE

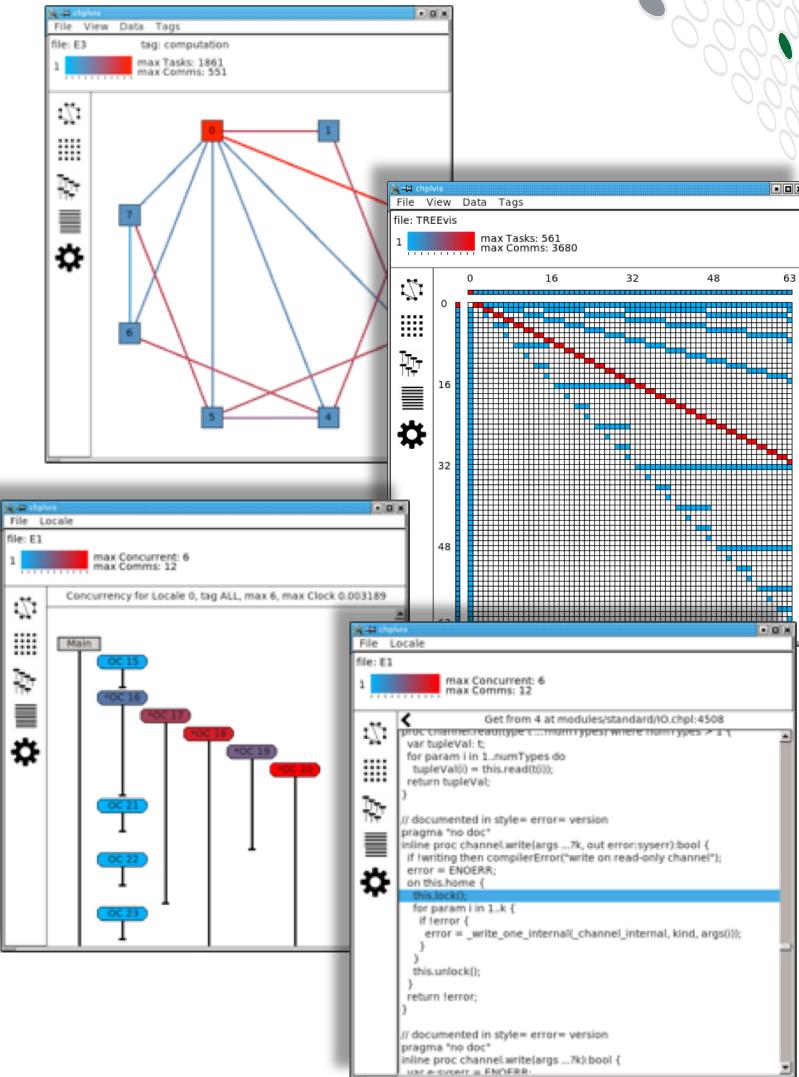
STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Tools

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



COMPUTE

STORE

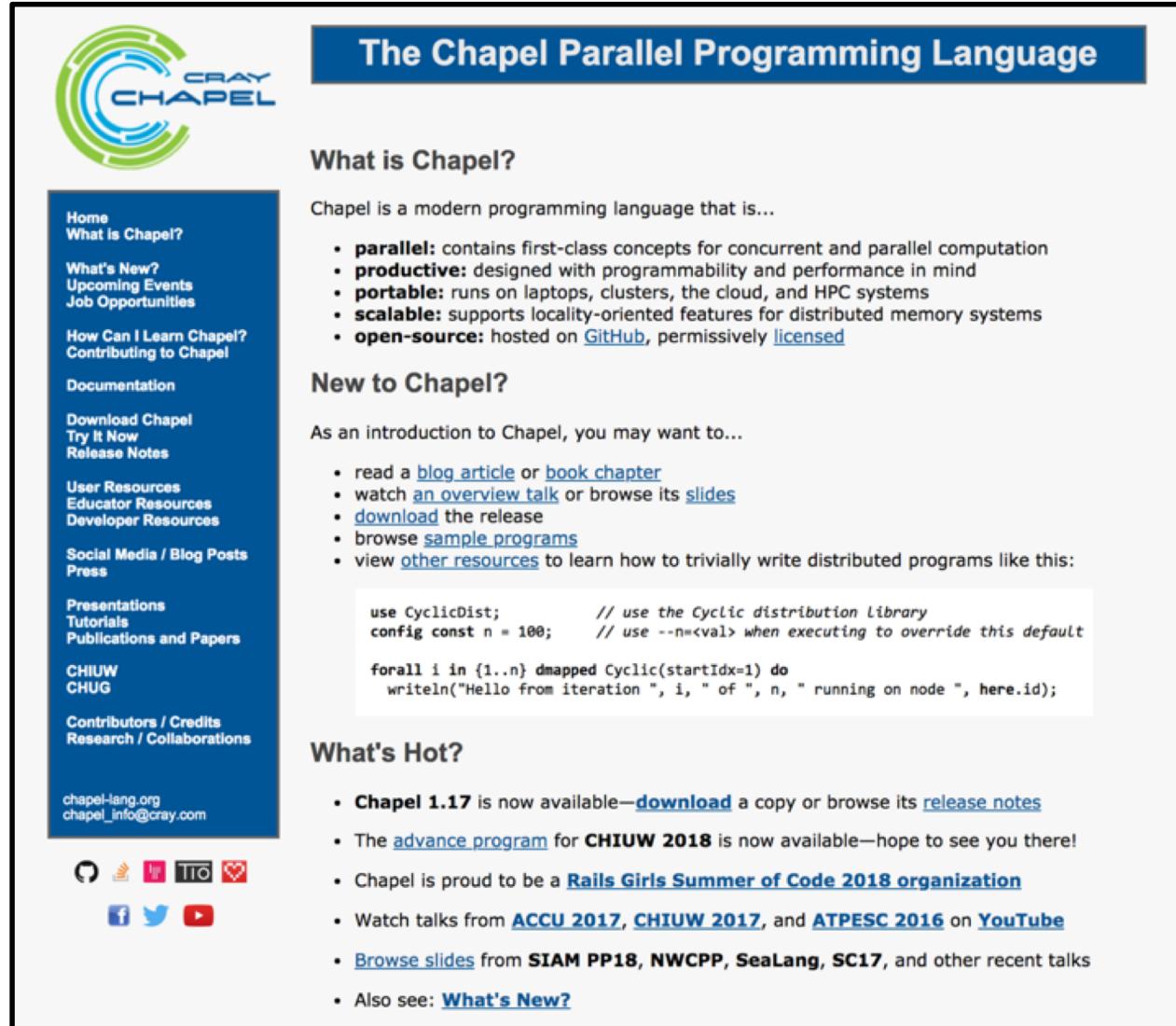
ANALYZE

Copyright 2018 Cray Inc.

Chapel Central

<https://chapel-lang.org>

- downloads
- presentations
- papers
- resources
- documentation



The screenshot shows the homepage of the Chapel Parallel Programming Language website. The header features the CRAY logo with a stylized 'C' composed of colored circles. The main title is "The Chapel Parallel Programming Language". Below the title, there's a section titled "What is Chapel?" which defines Chapel as a modern programming language with specific characteristics: parallel, productive, portable, scalable, and open-source. It includes a GitHub link and a license notice. The "New to Chapel?" section provides links to introductory resources like blog articles, book chapters, overview talks, and sample programs. A code snippet in the "New to Chapel?" section shows a CyclicDist configuration example. The "What's Hot?" section lists recent news items, including the release of Chapel 1.17, the availability of an advance program for CHI UW 2018, and contributions to the Rails Girls Summer of Code 2018 organization. The footer contains the website address, email, and social media links.

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 --n=<val> when executing to override this default
forall i in {1..n} dmapped Cyclic(startIdx=1) do
    writeln("Hello from iteration ", i, " of ", n, " running on node ", here.id);
```

What's Hot?

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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Online Documentation



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

The screenshot displays three pages from the Chapel Documentation website:

- Compiling and Running Chapel**: This page covers the basics of setting up the environment, building Chapel programs, and executing them. It includes sections on Quickstart Instructions, Platform-Specific Notes, Technical Notes, and Tools.
- Using Chapel**: This page provides an overview of the Chapel language, including Prerequisites, Environment Setup, and various execution models like Multilocale Chapel Execution and Chapel Launchers.
- Task Parallelism**: This page focuses on parallel tasks, specifically the `begin`, `cobegin`, and `coforall` statements. It includes code snippets and explanations for each statement.

The sidebar on the left contains links to other documentation sections such as Writing Chapel Programs, Language History, and Archived Language Specifications.



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Social Media (no account required)



Tweets 576 Following 48 Followers 278 Likes 200 Lists 1

Chapel Language @ChapelLanguage

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

chapel-lang.org Joined March 2016 256 Photos and videos

Interview with Brad Chamberlain about Chapel, a productive parallel programming language

As you might know, I am a big fan of concurrent, data-driven migration of tasks, but I know almost nothing about high performance computing. So I decided to get out from my comfort area. This is Chamberlain about Chapel, a productive parallel programming language.

<http://twitter.com/ChapelLanguage>

Chapel highlights

- Syntactic constructs for creating task parallelism:
coforall (concurrent forall): create a task per iteration
- Control over locality/affinity:
on-clauses: data-driven migration of tasks
- Static type inference (optionally):
Supports programmability with performance
- Modules for namespace management:
CyclicCdr: standard modules providing cyclic distributions
- Configuration variables and constants:
Never write an argument parser again (unless you want to)
- Domains and Arrays:
Index sets and arrays that can optionally be distributed
- Data parallel forall loops and operations:
Use available parallelism for data-driven computations

taskParallel.chpl

```
on loc {  
    const numTasks = here.maxTaskPar;  
    coforall tid in 1..numTasks do  
        write("Hello from task %d of %n running on %s",  
            tid, numTasks, here.name);  
}
```

dataParallel.chpl

```
use CyclicList;  
  
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);
```

<http://facebook.com/ChapelLanguage>

Chapel Parallel Programming Language 72 subscribers

HOME VIDEOS PLAYLISTS CHANNELS ABOUT

Chapel videos PLAY ALL

A playlist of featured Chapel presentations

CHI17 2017 keynote: Chapel's Home in the New Landscape of Scientific Frameworks, Jonathan Dursi Chapel Parallel Programming Language • 348 views • 10 months ago

This is Jonathan Dursi's keynote talk from CHI17: the 4th Annual Chapel Implementers and Users Workshop. The slides are available at: <https://jdursi.github.io/CHI17/> [Due to technical difficulties, the video is not available at the moment.]

The Audacity of Chapel: Scalable Parallel Programming Done Right - Brad Chamberlain [ACCU 2017] ACCU Conference • 1.1K views • 1 year ago

Programming language designers have to date largely failed the large-scale parallel computing community, and arguably even parallel programmers targeting desktops or modest-scale clusters.

PYCON UK 2017: On Big Computation and Python PyCon UK • 590 views • 6 months ago

Russel Winder | Thursday 17:00 | Assembly Room Python is a programming language slow of execution but fast of program development – except for some sorts of bug that a statically compiled

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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Community



Questions Developer Jobs Tags Users [chapel] 1,716 2 15

Tagged Questions info newest frequent votes active unanswered 140 questions tagged Ask Question

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

Learn more... Improve tag info Top users Synonyms

Tuple Concatenation in Chapel
Let's say I'm generating tuples and I want to concatenate them as they come. How do I do this? The following does element-wise addition: if ts = ("foo", "cat"), t = ("bar", "dog") ts + t gives ts = ...
tuples concatenation addition hpc chapel asked Jan 26 at 0:30 Tahimanga 385 1 10 6 votes 1 answer 79 views

Is there a way to use non-scalar values in functions with where clauses in Chapel?
I've been trying out Chapel off and on over the past year or so. I have used C and C++ briefly in the past, but most of my experience is with dynamic languages such as Python, Ruby, and Erlang more ...
chapel asked Apr 23 at 23:15 angus 33 3 6 votes 1 answer 47 views

Is there any writef() format specifier for a bool?
I looked at the writef() documentation for any bool specifier and there didn't seem to be any. In a Chapel program I have: ... config const verify = false; /* that works but I want to use writef() ...
chapel asked Nov 11 '17 at 22:21 6 votes 2 answers

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

This repository Search Pull requests Issues Marketplace Gist Watch 45 Star Unstar 455 Fork 145

chapel-lang / chapel

Code Issues 292 Pull requests 26 Projects 0 Settings Insights

Filters is:issue is:open Labels Milestones

292 Open 77 Closed Author Labels Projects Milestones

Implement "bounded-coforall" optimization for remote coforalls area: Compiler type: Performance #6357 opened 13 hours ago by ronawho 1 of 6 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 mpff 1 of 1 make check doesn't work with ./configure area: BTR type: Bug #6352 opened 16 hours ago by mpff 1 of 1 Passing variable via 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 1 of 1 Remove chpl_comm_make_progress area: Runtime easy type: Design #6349 opened a day ago by sungeunchoi 1 of 1 Runtime error after make on Linux Mint area: BTR user issue #6348 opened a day ago by danindiana 1 of 1

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

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

Brian Dolan @buddha314 what is the syntax for making a copy (not a reference) to an array? May 09 14:34 Michael Ferguson @mpff like in a new variable? May 09 14:40 var A:[1..10] int; var B = A; // makes a copy of A ref C = A; // refers to A May 09 14:41 Brian Dolan @buddha314 oh, got it, thanks! May 09 14:42 Michael Ferguson @mpff May 09 14:42 proc f(arr) { /* arr refers to the actual argument */ } proc g(in arr) { /* arr is a copy of the actual argument */ } var A:[1..10] int; f(A); g(A); May 09 14:43 Brian Dolan @buddha314 isn't there a proc f(ref arr) {} as well? May 09 14:45 Michael Ferguson @mpff yes. The default intent for array is 'ref' or 'const ref' depending on if the function body modifies it. So that's effectively the default. May 09 14:55 Brian Dolan @buddha314 thanks! May 09 14:55

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

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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Suggested Reading (short attention spans)



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

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

Chapel: Productive Parallel Programming, [Cray Blog](#), May 2013.

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

Six Ways to Say “Hello” in Chapel (parts [1](#), [2](#), [3](#)), [Cray Blog](#), Sep-Oct 2015.

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

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

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

[Ten] Myths About Scalable Programming Languages, [IEEE TCSC Blog](#)

([index available on chapel-lang.org “blog posts” page](#)), Apr-Nov 2012.

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



COMPUTE

STORE

ANALYZE

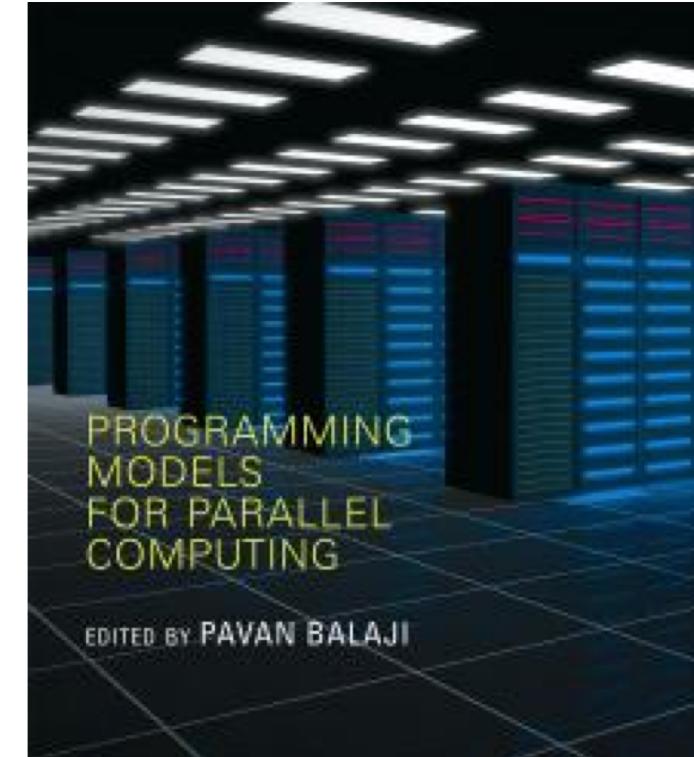
Copyright 2018 Cray Inc.

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 also available [online](#)



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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Report on Recent Progress: CUG 2018 Paper



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 C

Chapel Team

Cray Inc.

Seattle, WA, USA

chapel_info@cray.com

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

Keywords-Parallel programming; Computer languages

I. INTRODUCTION

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

The development of the Chapel language was driven by Cray Inc. as part of its participation in the DARPA High Productivity Computing Systems program. The project wrapped up in late 2012, at which point we had a working prototype, having successfully overcome several key research challenges that the team had identified. Chief among these was supporting distributed parallelism in a unified manner within a single language, which was accomplished by supporting the creation of parallel abstractions like parallel loops and parallel reductions in terms of lower-level Chapel features such as parallel for loops and parallel tasks.

Under HPCS, Chapel also succeeded in supporting the expression of parallelism using distinct constructs from those used to control locality and memory access. Programmers specify *which* computation is parallel distinctly from specifying *when* it should be run. This permits Chapel to support multicore, multi-node, and heterogeneous systems using a single unified language.

Chapel's implementation under HPCS demonstrated that the language could be implemented in a way that was optimized for HPC-specific features and support available in Cray® Gemini supercomputers. This allows Chapel to take advantage of the unique features of Cray systems.

[paper](#) and [slides](#) available at chapel-lang.org

CRAY

**Chapel Comes of Age:
Productive Parallelism at Scale
CUG 2018**

Brad Chamberlain, Chapel Team, Cray Inc.

CRAY CHAPEL



COMPUTE

STORE

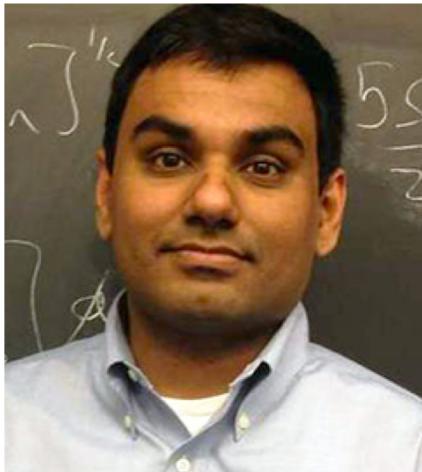
ANALYZE

Copyright 2018 Cray Inc.

Chapel User Profiles



Current Users:



Time-to-science
Cosmologist



Commercial AI
Scientist

Potential Users:

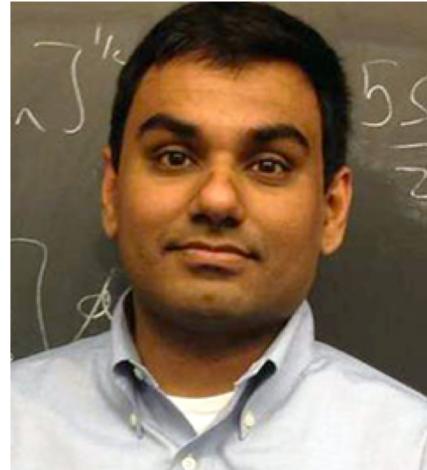


Genomic
Researcher



DOE Scientist

User Profile: Time-to-Science Cosmologist



Name: Nikhil Padmanabhan

Title: Associate Professor of Physics and Astronomy,
Yale University

Computations: Surveys of galaxies to constrain
cosmological models, n-body simulations of gravity

Why Chapel? “My interests in Chapel developed from a desire to have a lower barrier to writing parallel codes. In particular, I often find myself writing prototype codes (often serial), but then need to scale these codes to run on large numbers of simulations/datasets. **Chapel allows me to smoothly transition from serial to parallel codes with a minimal number of changes.**

“Another important issue for me is "my time to solution" (some measure of productivity vs performance). Raw performance is rarely the only consideration.”



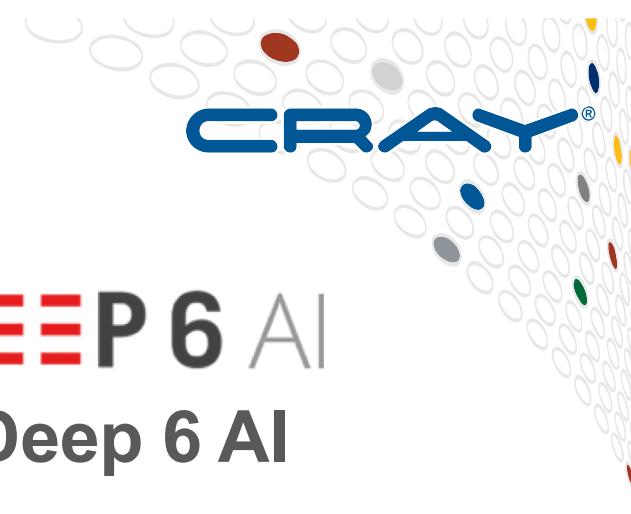
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

User Profile: Commercial AI Scientist



Name: Brian Dolan

DEEP 6 AI

Title: Co-Founder and Chief Scientist of Deep 6 AI

Computations: Natural language processing, AI and ML applications, network analysis, community detection, reinforcement learning in the form of Deep Q-Networks

Why Chapel? “I have used Fortran, R, Java and Python extensively. If I had to give up Chapel, I would probably move to C++. **I prefer Chapel due to the extreme legibility and performance.** We have abandoned Python on large problems for performance reasons.

“We’ve now developed thousands of lines of Chapel code and a half dozen open source libraries for things like database connectivity, numerical libraries, graph processing, and even a REST framework. We’ve done this because AI is about to face an HPC crisis, and the folks at Chapel understand the intersection of usability and scalability.”



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Potential User Profile: Genomic Researcher



Name: Jonathan Dursi

Title: Senior Research Associate, The Hospital for Sick Children, Toronto

Computations: Human genomics, bioinformatics, and medical informatics

Why Chapel? “My interest in Chapel lies in its potential for bioinformatics tools that are currently either written in elaborately crafted, threaded but single node, C++ code, or in Python. Either has advantages and disadvantages (performance vs rapid development cycles), but neither has a clear path to cross-node computation, for performance as well as larger memory and memory bandwidth. **Chapel has the potential to have some of the best of both worlds in terms of C++ and Python, as well as having a path to distributed memory.**”



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Potential User Profile: DOE Scientist



Name: Anshu Dubey

Title: Computer Scientist, Argonne National Laboratory

Computations: Design and development of Multiphysics software that can serve multiple science domains; solvers for PDEs and ODEs

Why Chapel? “In Multiphysics applications separation of concerns and use of high level abstractions is critical for sustainable software. Chapel combines language features that would enable this for clean implementation.

“HPC Scientific software is made more complex than it needs to be because the only language designed for scientific work, Fortran, is losing ground for various reasons. Its object oriented features are clunky and make it nearly as unsuitable as other languages for scientific work. **Chapel appears to be parallel and modern Fortran done better, therefore has the potential to become a more suitable language.**”



COMPUTE

STORE

ANALYZE

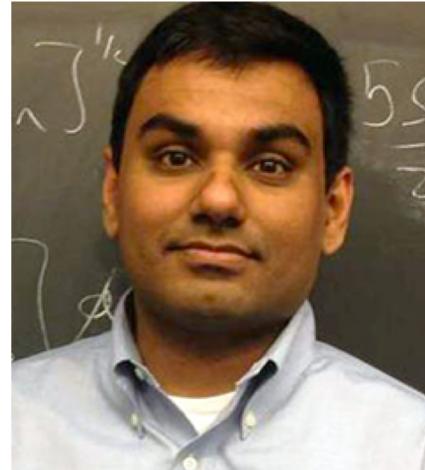
Copyright 2018 Cray Inc.

Chapel and Productivity



Chapel aims to be as...

- ...programmable as Python**
- ...fast as Fortran**
- ...scalable as MPI, SHMEM, or UPC**
- ...portable as C**
- ...flexible as C++**
- ...fun as [your favorite language]**



COMPUTE

STORE

ANALYZE

Copyright 2018 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

[Gitter \(chapel-lang/chapel\)](#): community chat with archives

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

Discuss Chapel development

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

[GitHub issues for chapel-lang/chapel](#): for feature requests, design discussions

Discuss Chapel's use in education

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

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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Wrap-up



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Summary



The Chapel language offers a unique combination of productivity, performance, and parallelism

We're interested in finding and working with the next generation of Chapel users

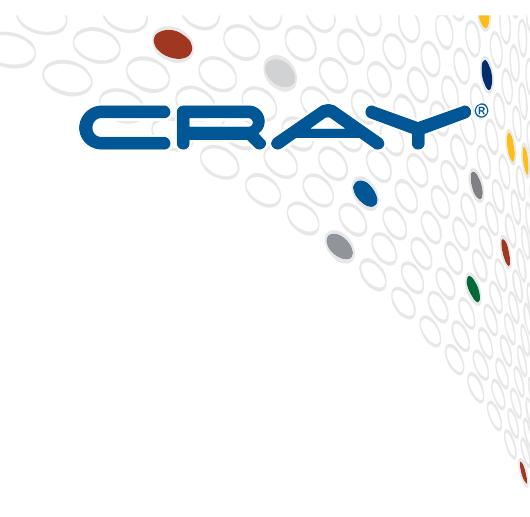


COMPUTE

STORE

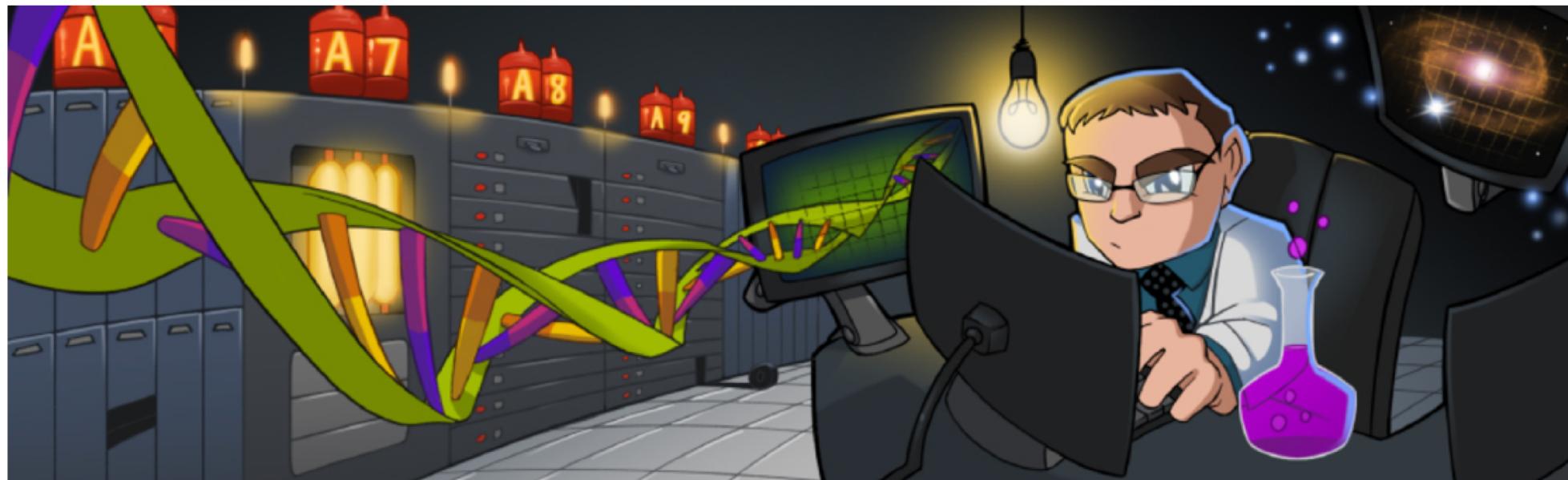
ANALYZE

Copyright 2018 Cray Inc.



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

Copyright 2018 Cray Inc.

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...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

Copyright 2018 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 publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

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

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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

CRAY

