



Chapel Comes of Age: Productive Parallelism at Scale

CUG 2018

Brad Chamberlain, Chapel Team, Cray Inc.





Or: What's Chapel been up to
since CUG 2013?

CUG 2018

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



CUG 2018

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]



CUG 2018

COMPUTE

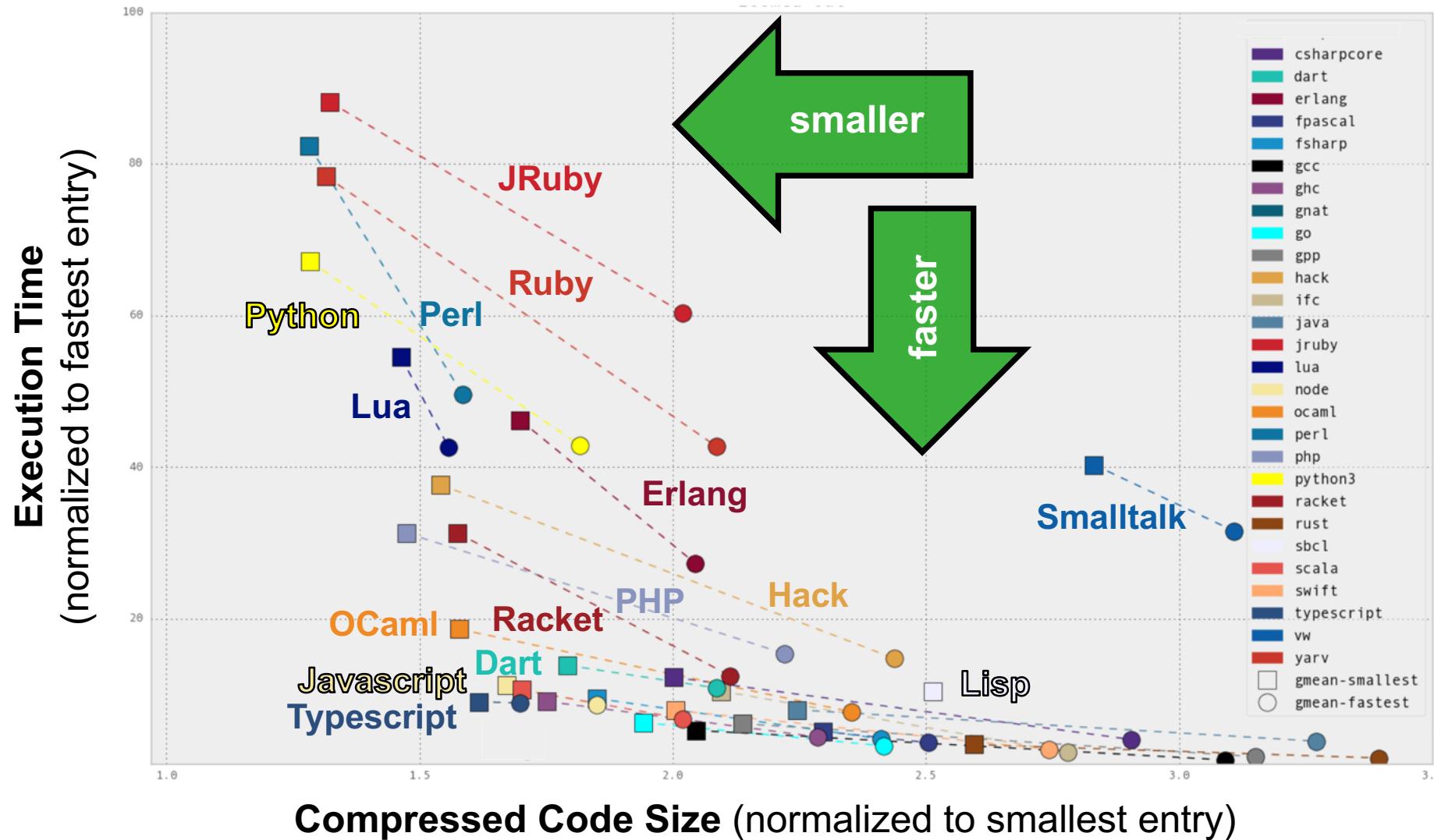
STORE

ANALYZE

Copyright 2018 Cray Inc.

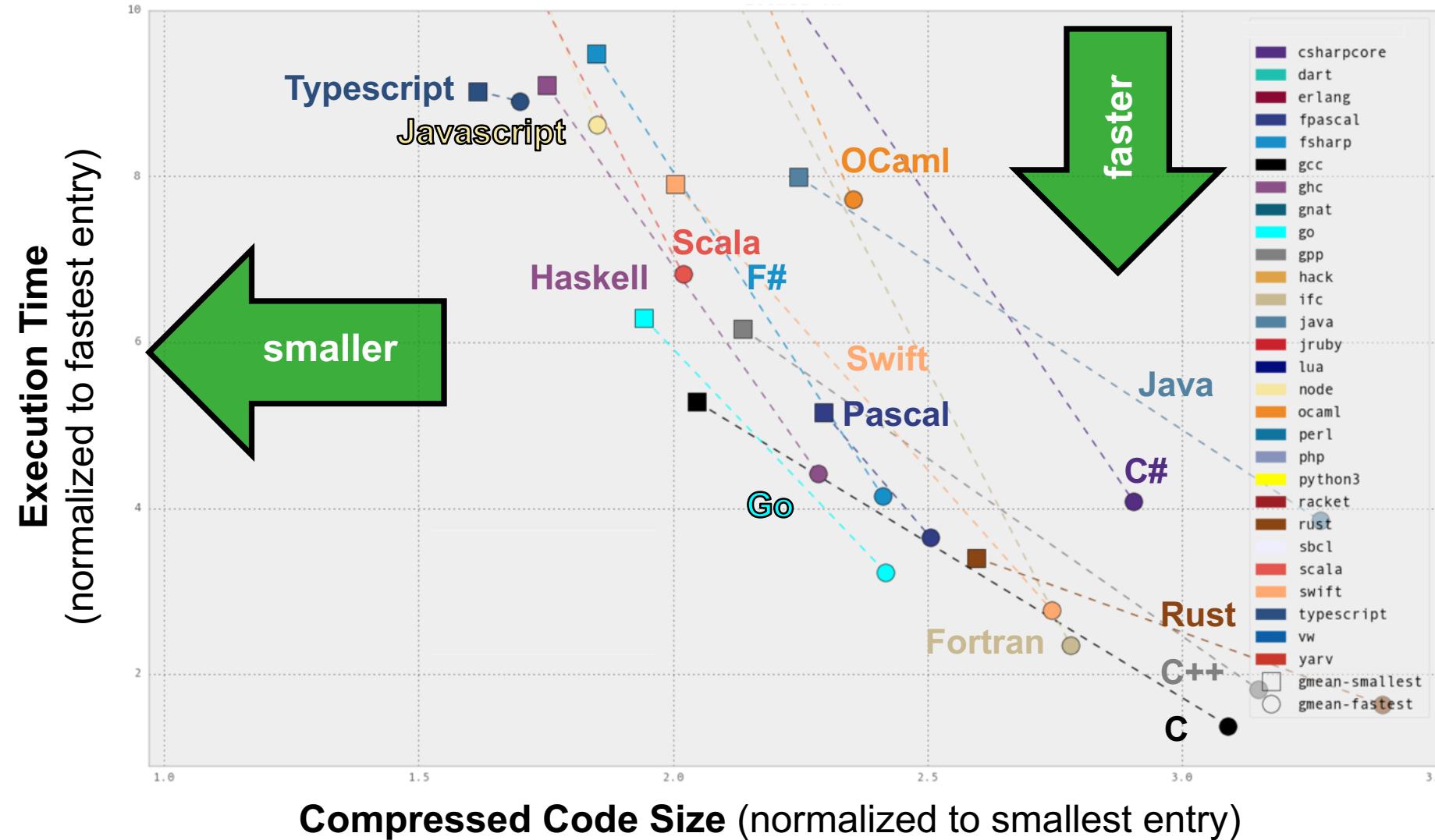
CLBG Cross-Language Summary

(Oct 2017 standings)



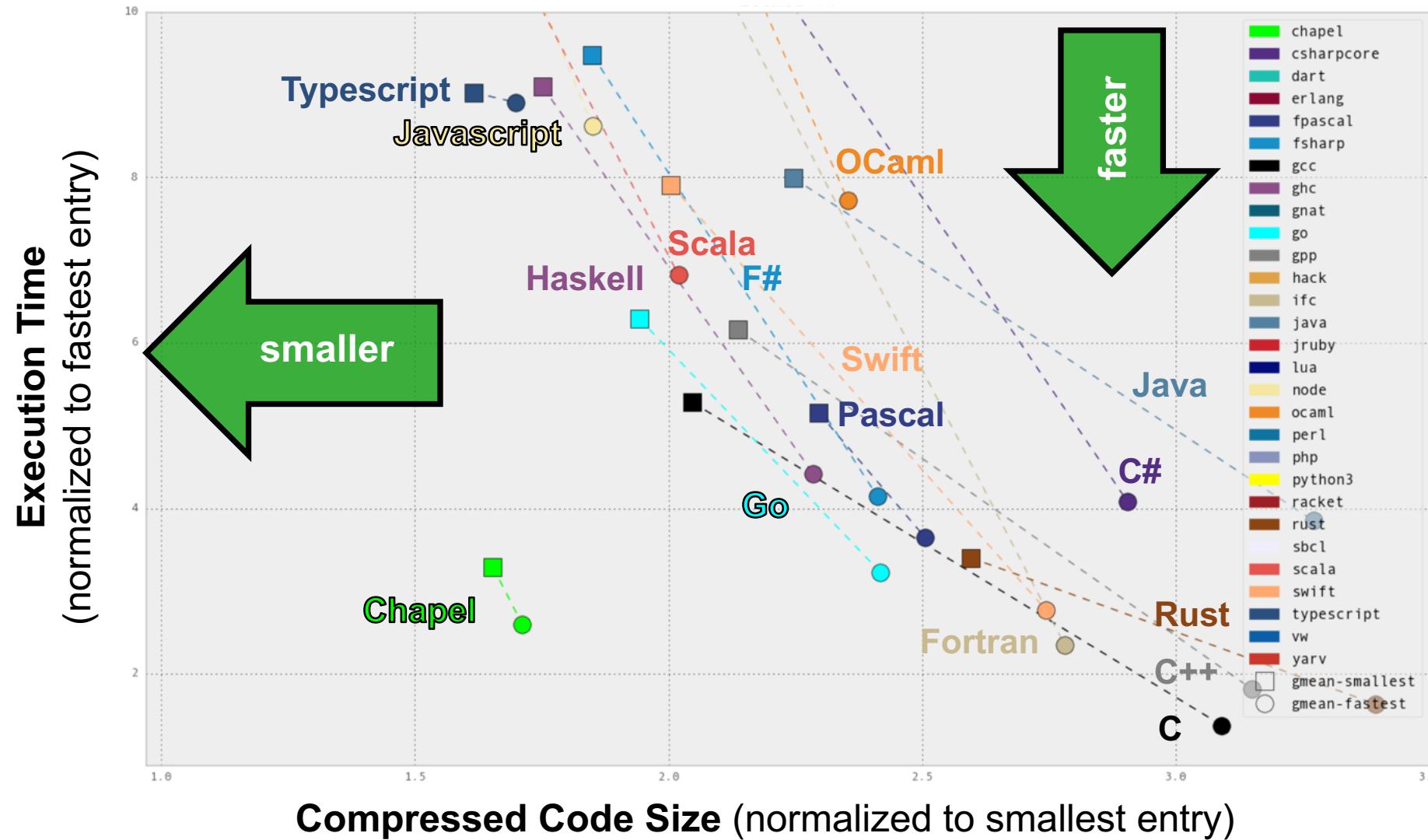
CLBG Cross-Language Summary

(Oct 2017 standings, zoomed in)



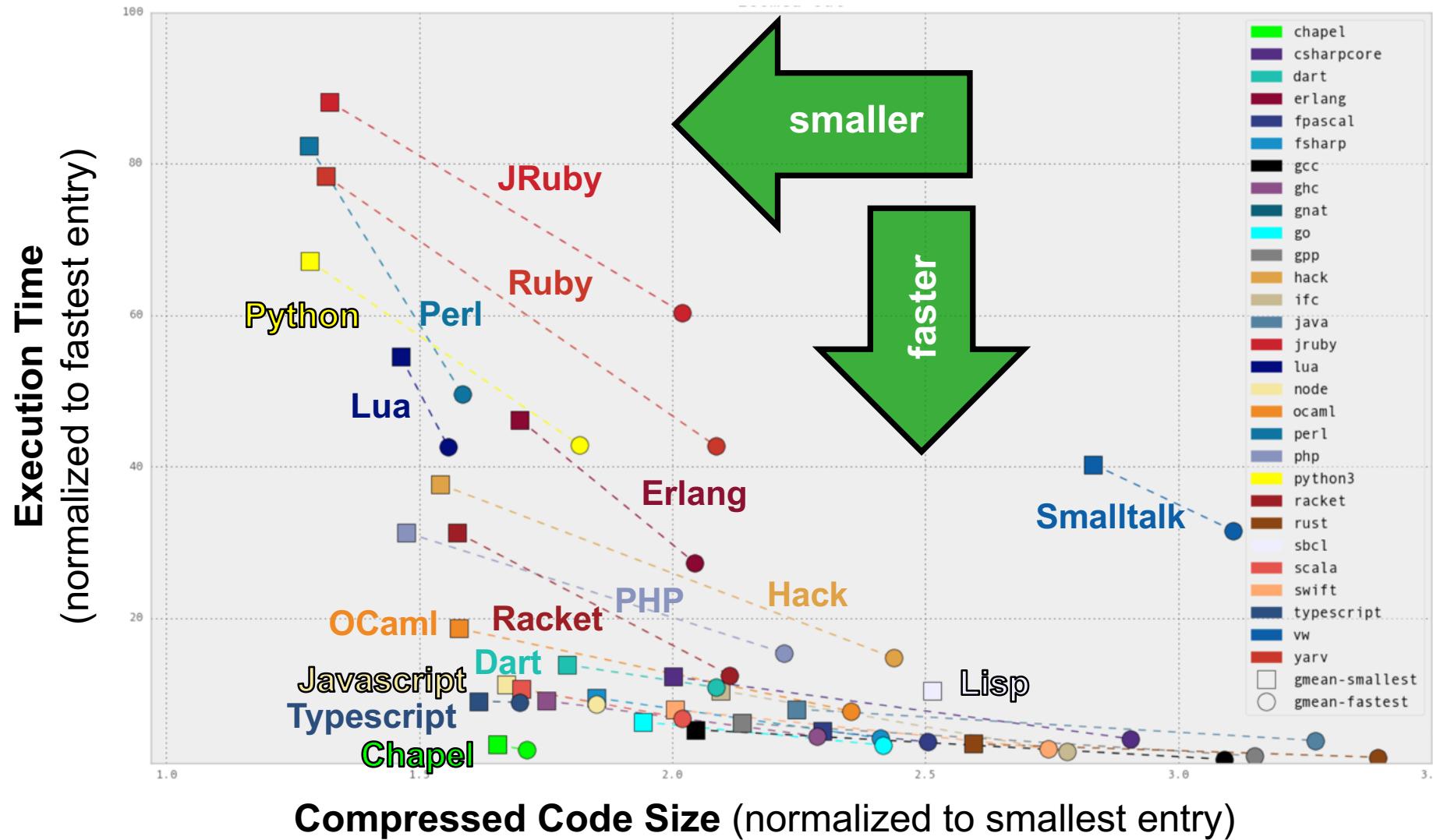
CLBG Cross-Language Summary

(Oct 2017 standings, zoomed in)



CLBG Cross-Language Summary

(Oct 2017 standings)



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



CUG 2018

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
// in population
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);
}
```

```
size_t
char const*
size_t
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";
n(core_id_str);
cores;
n(cpu_cores_str);

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

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

excerpt from 2863.gz C gcc entry



CUG 2018

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



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



The Chapel Team at Cray (May 2018)



CUG 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>



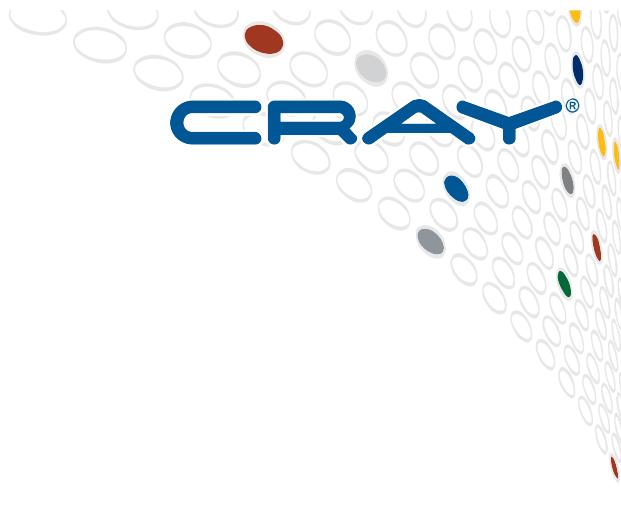
CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Outline

✓ What is Chapel?

➤ Chapel Overview

- Chapel: Then vs. Now
- Chapel User Profiles
- What's Next?



CUG 2018

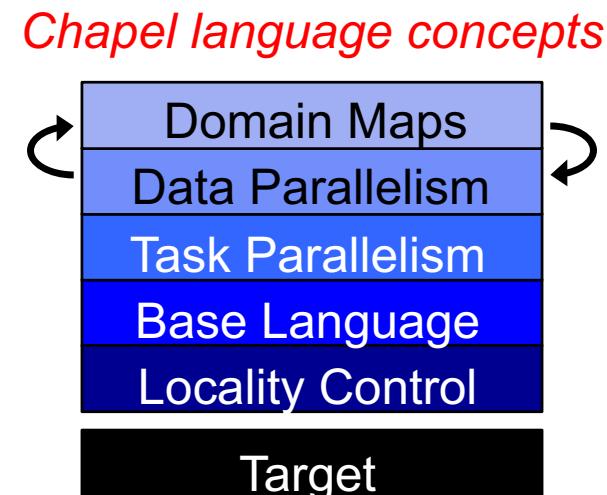
COMPUTE

STORE

ANALYZE

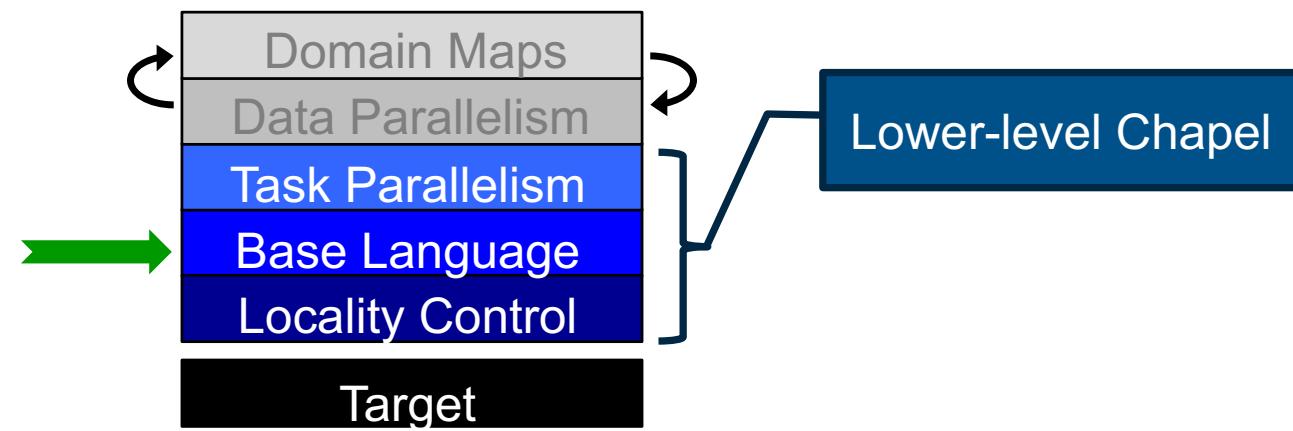
Copyright 2018 Cray Inc.

Chapel language feature areas





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  
...
```



CUG 2018

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  
...
```



CUG 2018

COMPUTE

STORE

Copyright 2018 Cray Inc.

ANALYZE



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
...
...
```



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Base Language Features, by example

Explicit types also permitted

```
iter fib(n : int): int { config const n: int = 10;
    var current: int = 0,
        next: int = 1;

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

```
0  
1  
1  
2  
3  
5  
8  
...
```



CUG 2018

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  
...
```



CUG 2018

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



CUG 2018

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
...
```



CUG 2018

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
...
```



CUG 2018

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
...
```



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Other 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...



CUG 2018

COMPUTE

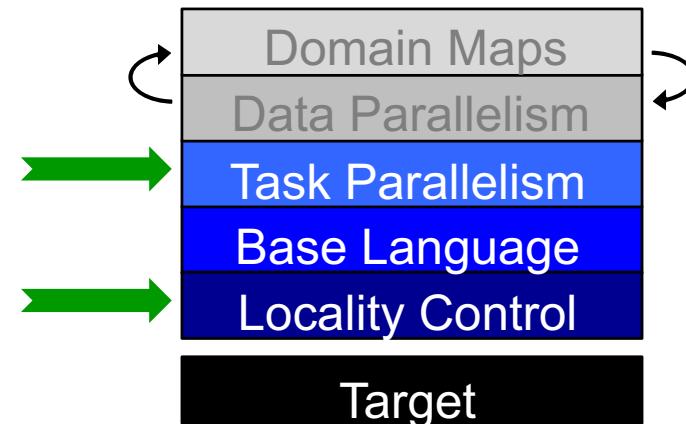
STORE

ANALYZE

Copyright 2018 Cray Inc.



Task Parallelism and Locality Control



CUG 2018

COMPUTE

STORE

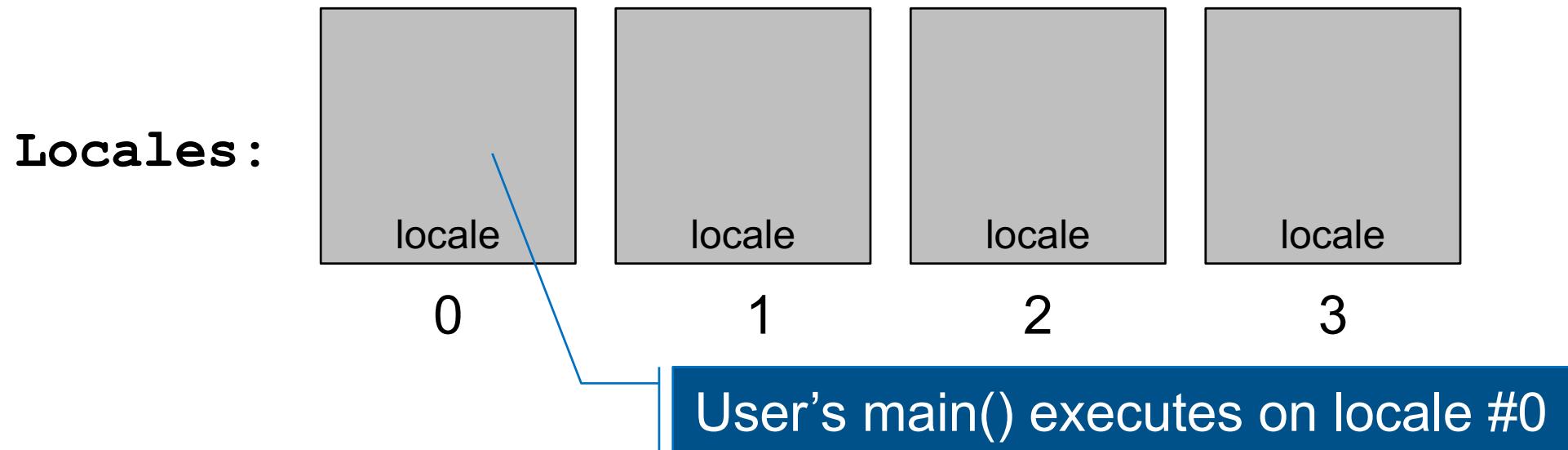
ANALYZE

Copyright 2018 Cray Inc.



Locales, briefly

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



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
```



CUG 2018

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
```



CUG 2018

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
```

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
```



CUG 2018

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
```



CUG 2018

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
```



CUG 2018

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
```



CUG 2018

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
```



CUG 2018

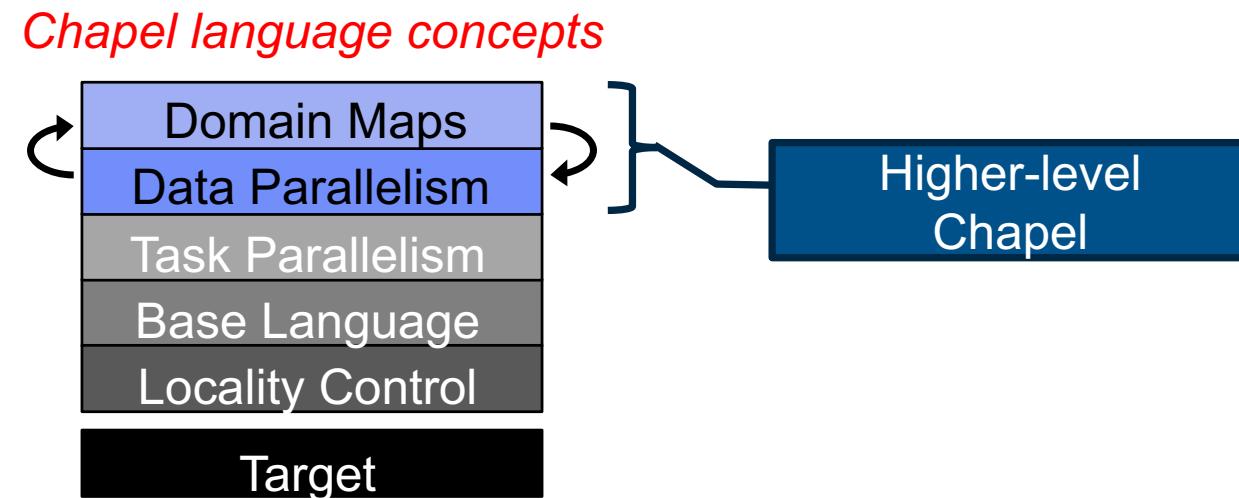
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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
```



CUG 2018

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
```



CUG 2018

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
```



CUG 2018

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
```



CUG 2018

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
```



CUG 2018

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
```



CUG 2018

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
```



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



A Brief History of Chapel



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

A Brief History of Chapel: Infancy



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

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

The State of the Chapel Union

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



CUG 2018

COMPUTE

|

STORE

|

ANALYZE

Copyright 2018 Cray Inc.



Crossing the Stream of Adoption

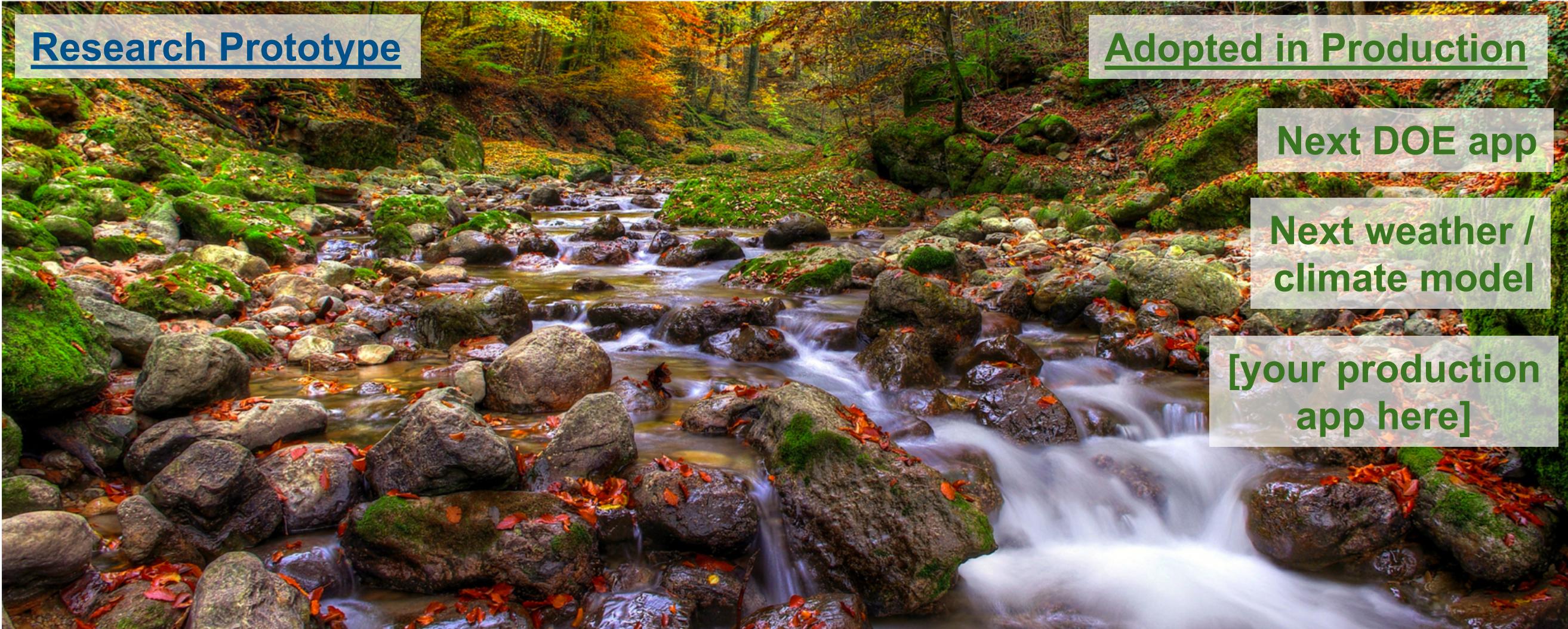
Research Prototype

Adopted in Production

Next DOE app

Next weather /
climate model

[your production
app here]



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

image credit: <http://feelgrafix.com/813578-free-stream-wallpaper.html>



Crossing the Stream of Adoption: Post-HPCS Barriers

Research Prototype

Performance & Scalability

Immature Language Features

Insufficient Libraries

Memory Leaks

Lack of Tools

Lack of Documentation

Fear of Being the Only User

Adopted in Production

Next DOE app

Next weather /
climate model

[your production
app here]



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

image credit: <http://feelgrafix.com/813578-free-stream-wallpaper.html>

A Brief History of Chapel: Adolescence



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

- Motivated by user enthusiasm for Chapel
- Development focus:
 - address weak points in HPCS prototype
 - support and grow the Chapel community
- ~13–14 FTEs
- This CUG 2018 talk & paper reports on progress during this time

Then Now



Chapel Performance: Then vs. Now vs. Reference



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Performance Focus Areas (during 5-year push)



Array Optimizations:

- shifted data optimization (eliminates arbitrary indexing overhead)
- loop-invariant code motion (eliminates meta-data overhead)
- eliminated multiply in indexing for 1D (and innermost dim of 2D+) arrays

Runtime Library Improvements:

- scalable parallel memory allocator
- tasks mapped to affinity aware user-level threads
- native/optimized comm with RDMA and limited software overhead

Optimized Communication:

- compiler locality analysis improvements
- bulk array assignments
- remote-value-forwarding, new distributions, fast-ons, ...



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Experimental Methodology



Methodology for the next several slides:

- Resurrected a copy of **Chapel 1.7**
 - updated it to build with current versions of gcc/g++
- Compared it to **Chapel 1.17**, released April 2018
- Used today's Cray systems
- Used today's benchmark codes
 - with modest edits for 1.7 in response to language changes



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

LCALS Serial Kernel



- Chapel source:

```
for i in 0..#len do  
    bvc[i] = cls * (compression[i] + 1.0);
```



CUG 2018

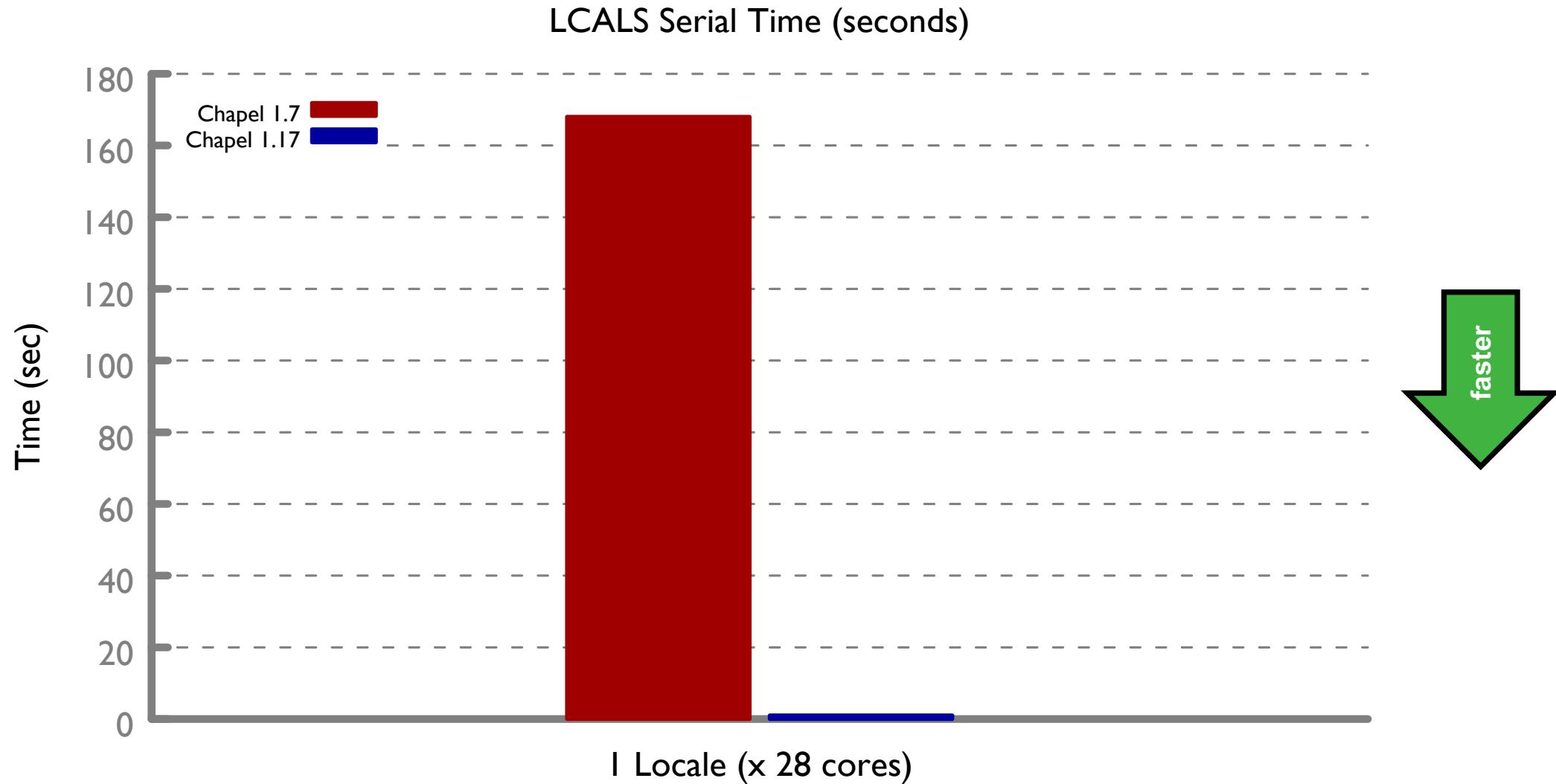
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

LCALS Serial Kernel: Chapel Then vs. Now



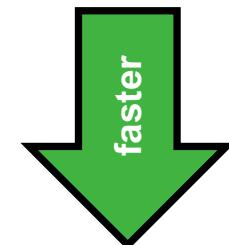
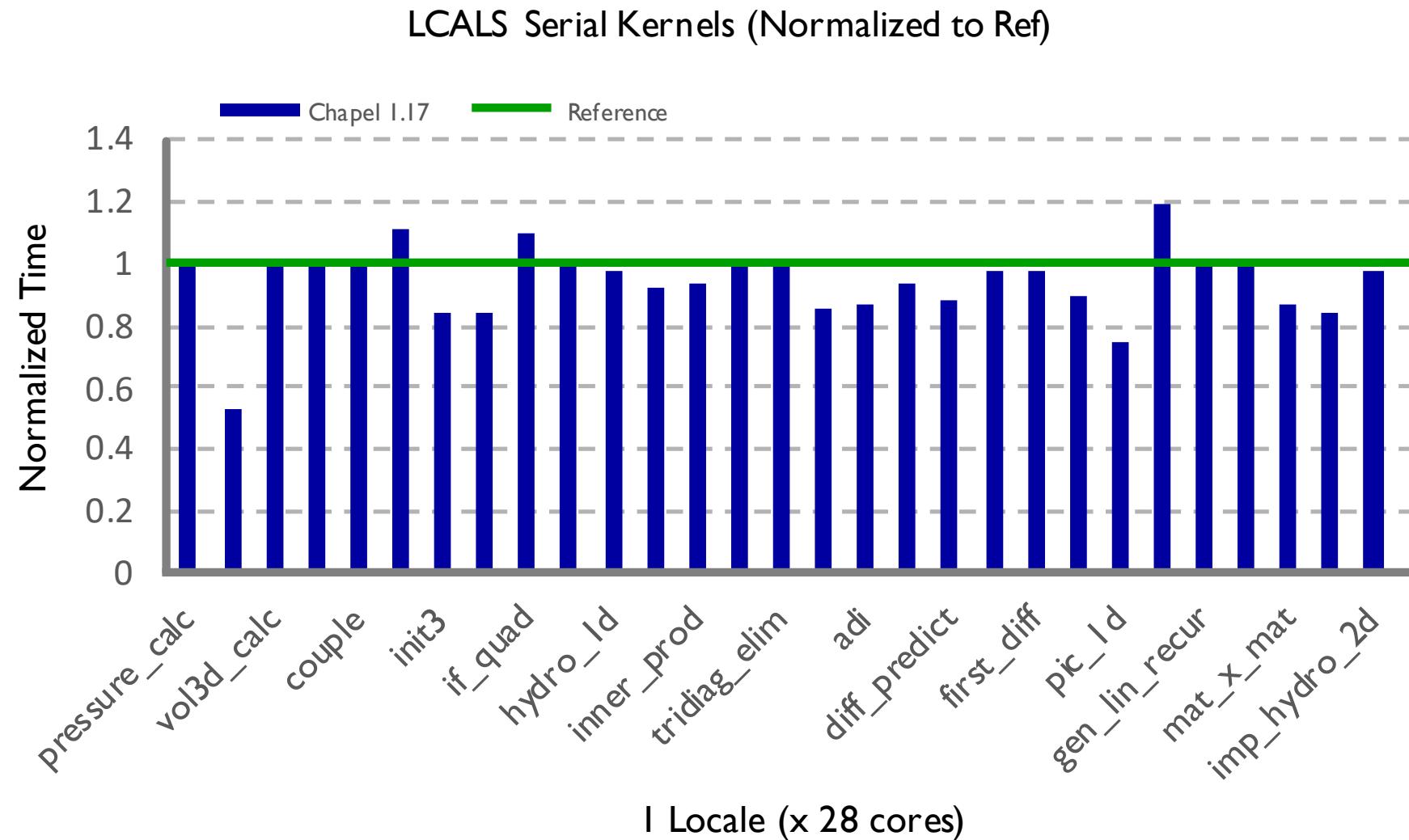
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

LCALS Serial Kernels: Chapel Now vs. Ref



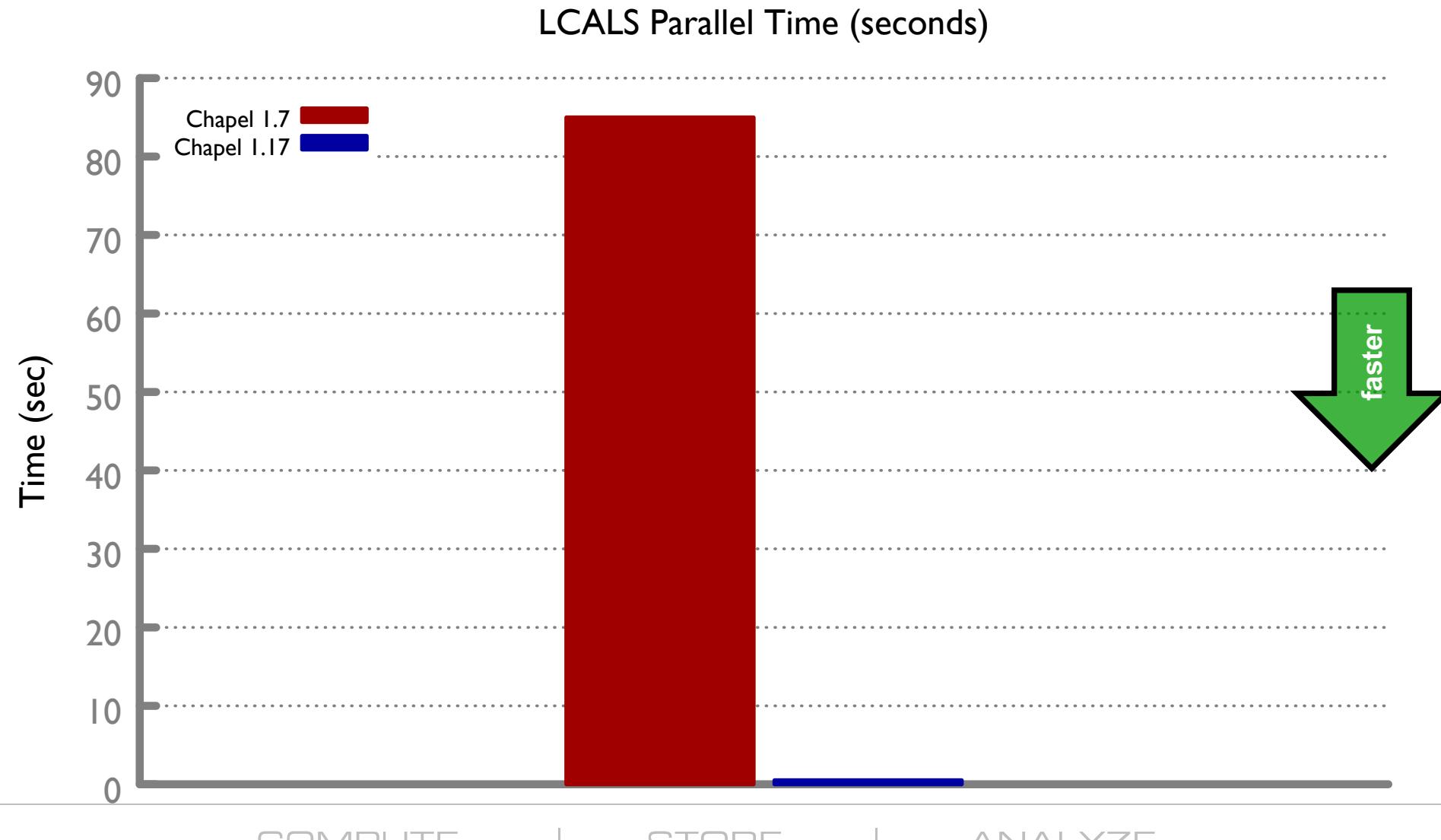
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

LCALS Parallel Kernel: Chapel Then vs. Now



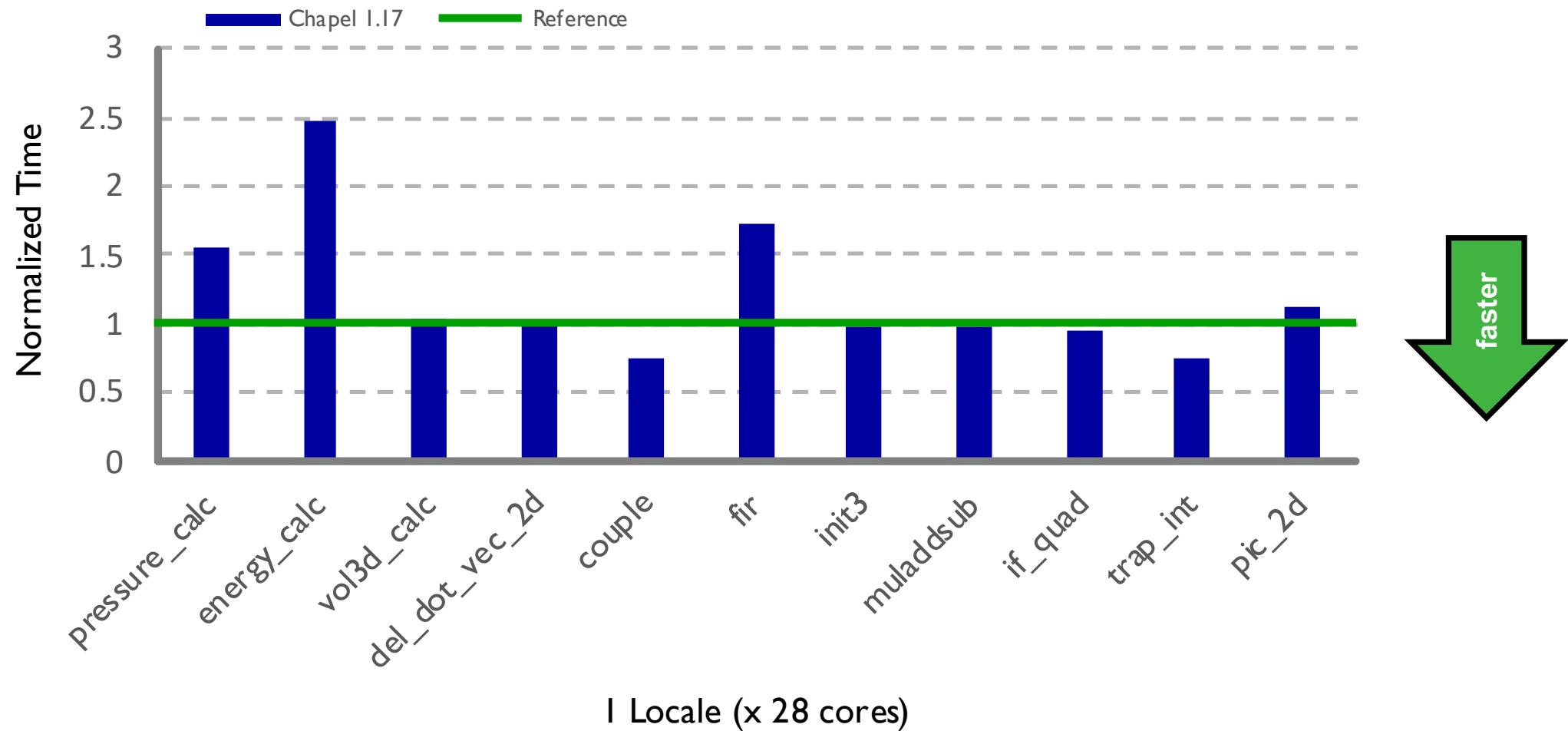
CUG 2018

Copyright 2018 Cray Inc.

LCALS Parallel Kernels: Chapel Now vs. Ref



LCALS Parallel Kernels (Normalized to Ref)



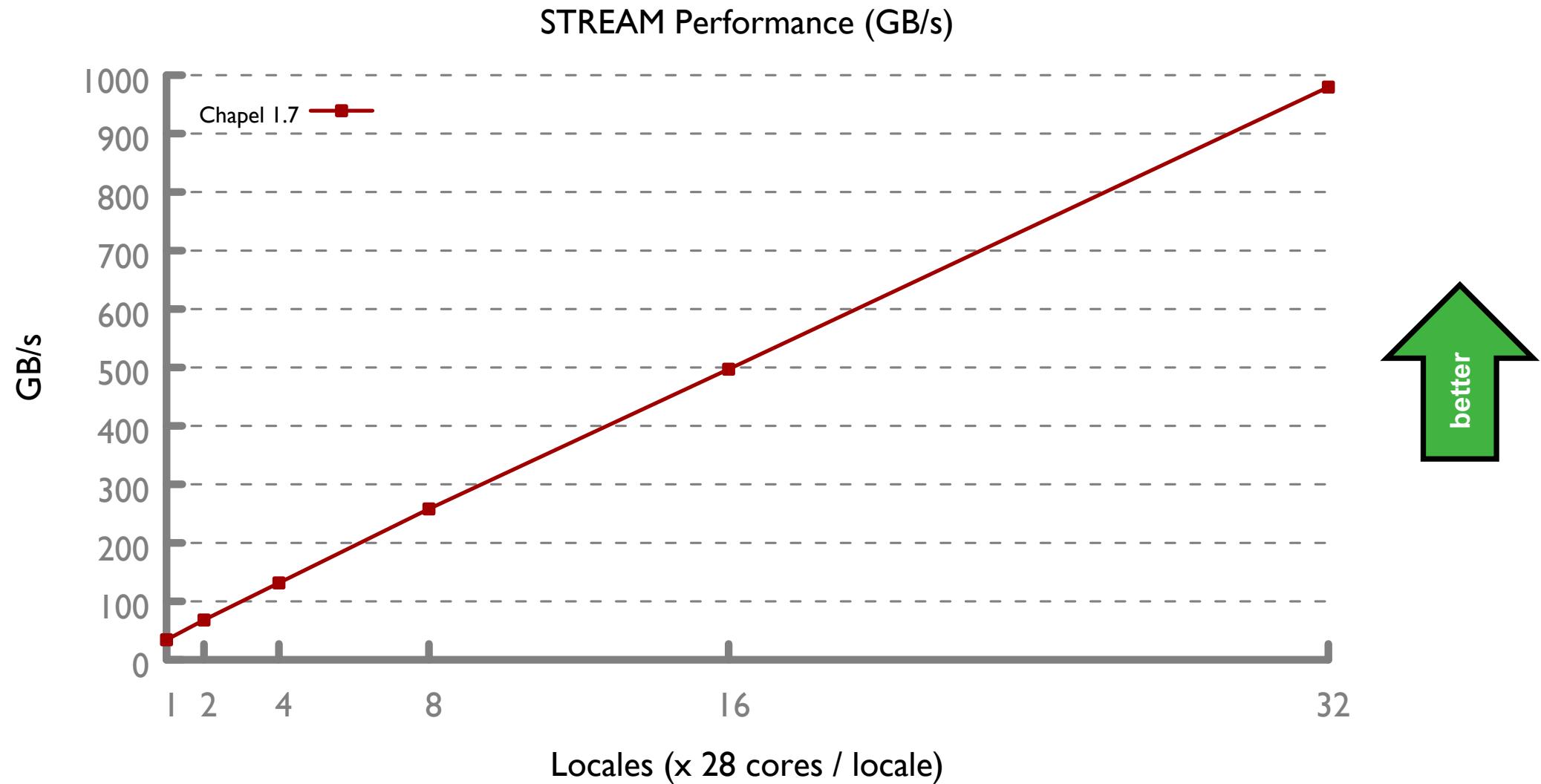
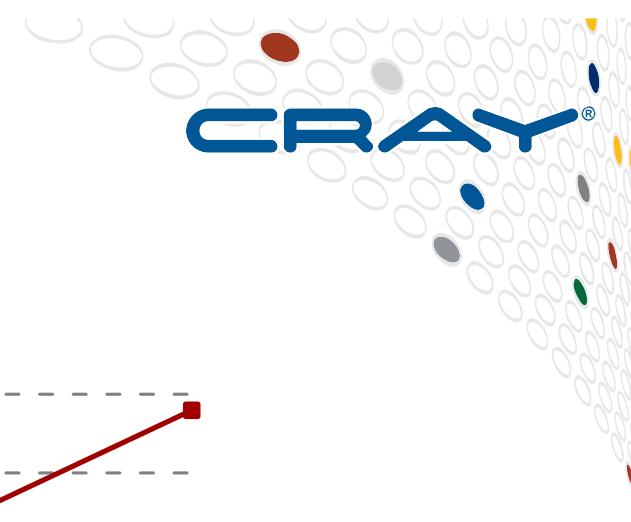
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

HPCC STREAM Triad: Chapel Then



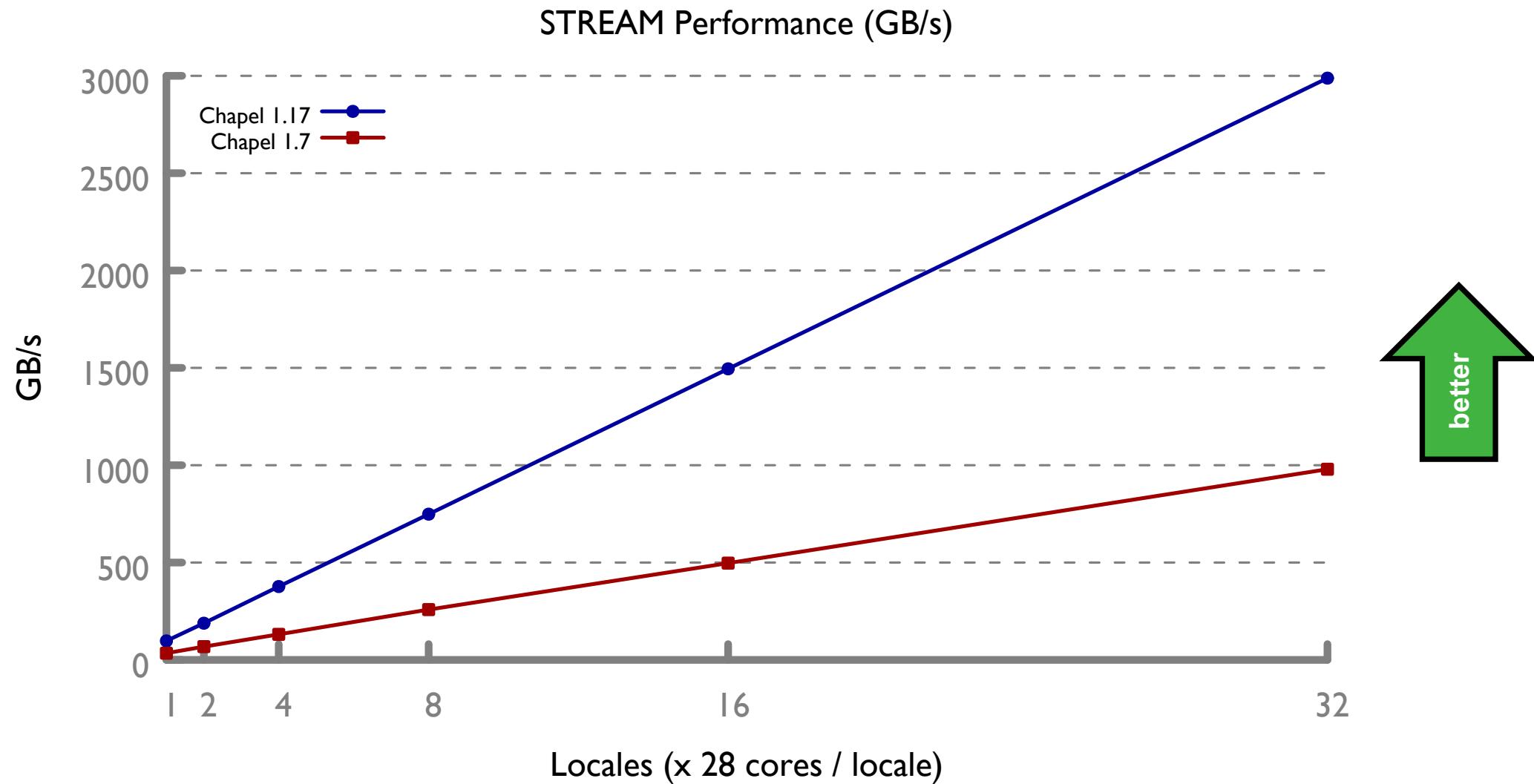
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

HPCC STREAM Triad: Chapel Then vs. Now



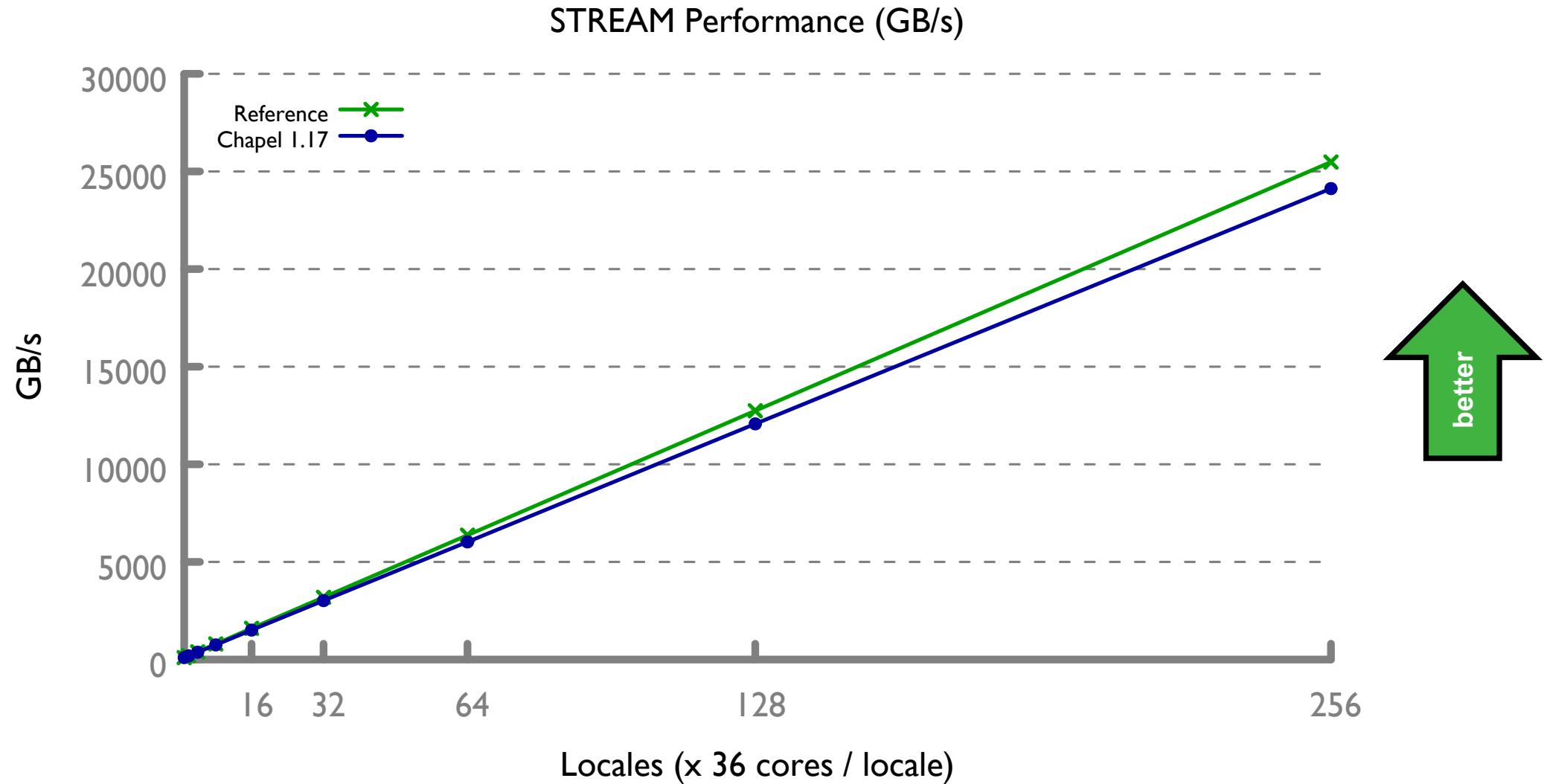
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

HPCC STREAM Triad: Chapel Now vs. Ref



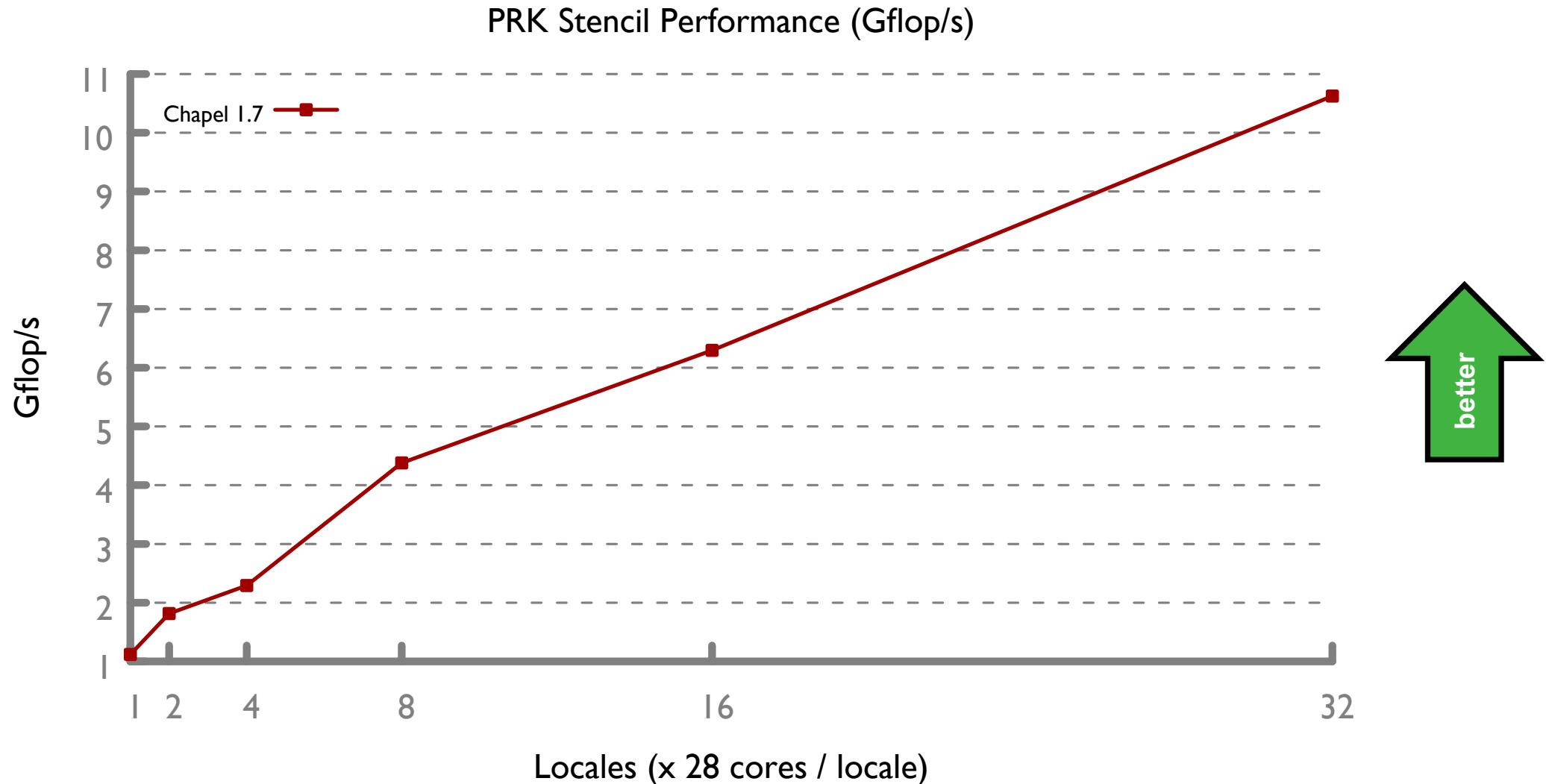
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

PRK Stencil: Chapel Then



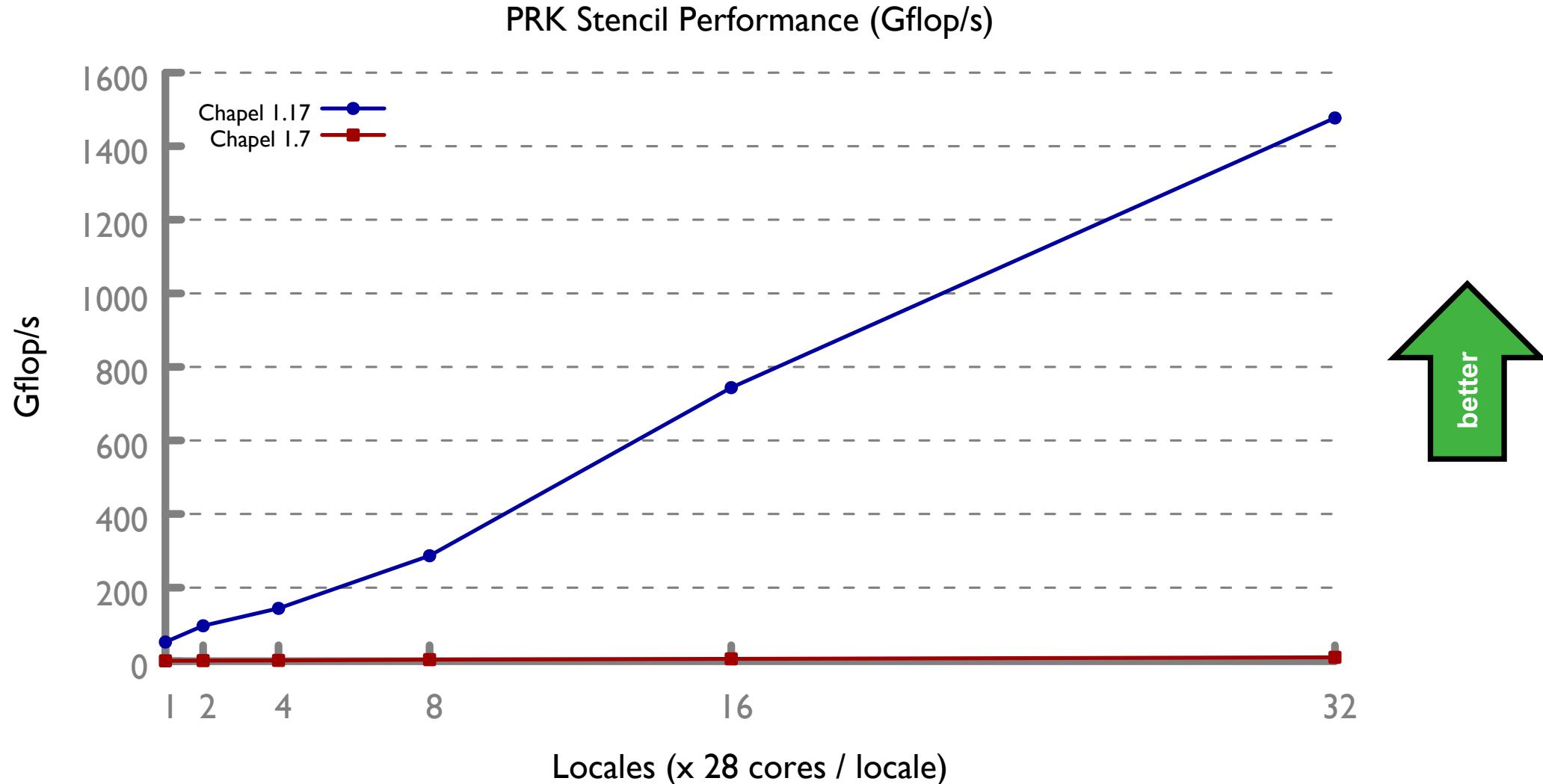
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

PRK Stencil: Chapel Then vs. Now

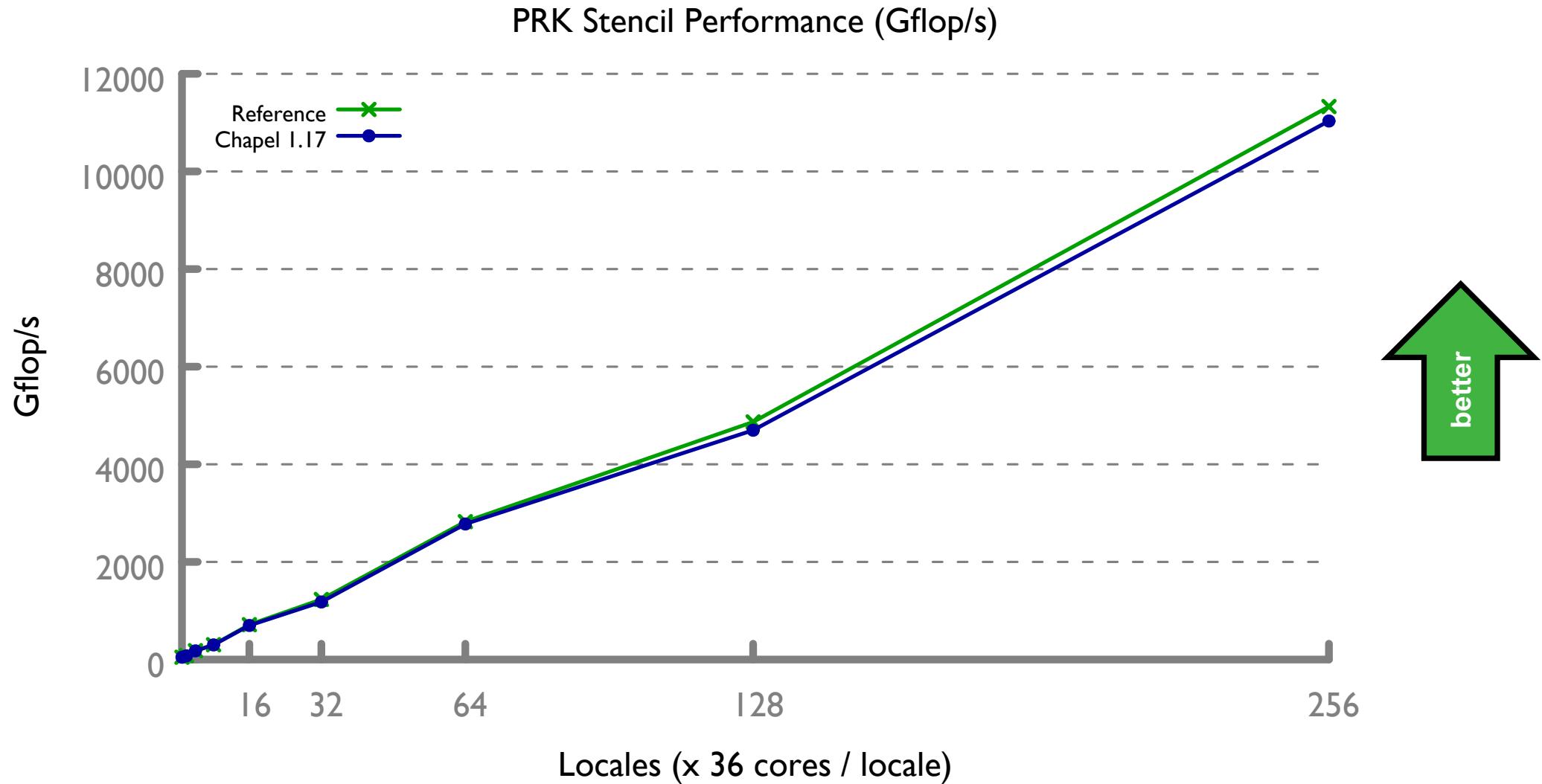


COMPUTE

STORE

ANALYZE

PRK Stencil: Chapel Now vs. Ref



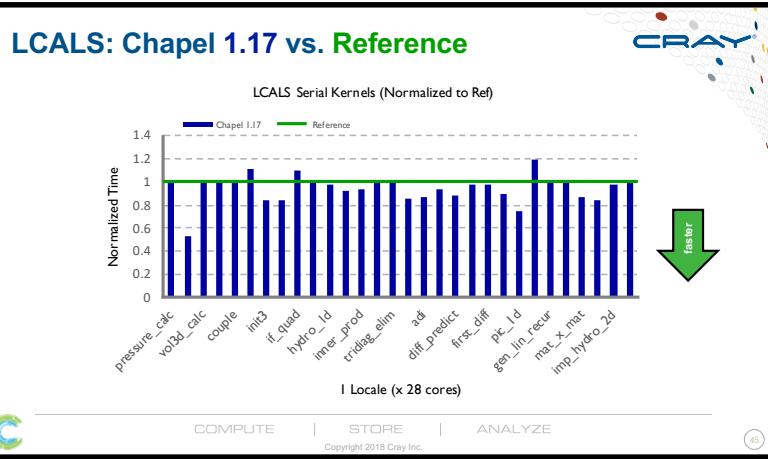
COMPUTE

STORE

ANALYZE

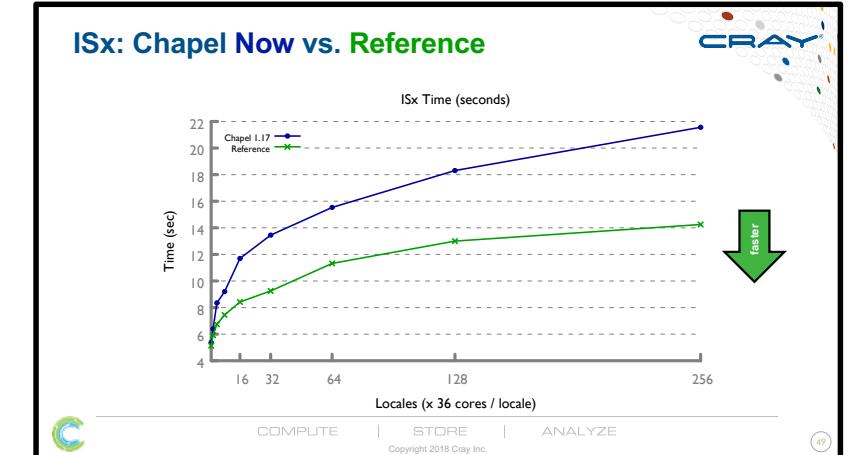
Copyright 2018 Cray Inc.

HPC Patterns: Chapel Now vs. reference



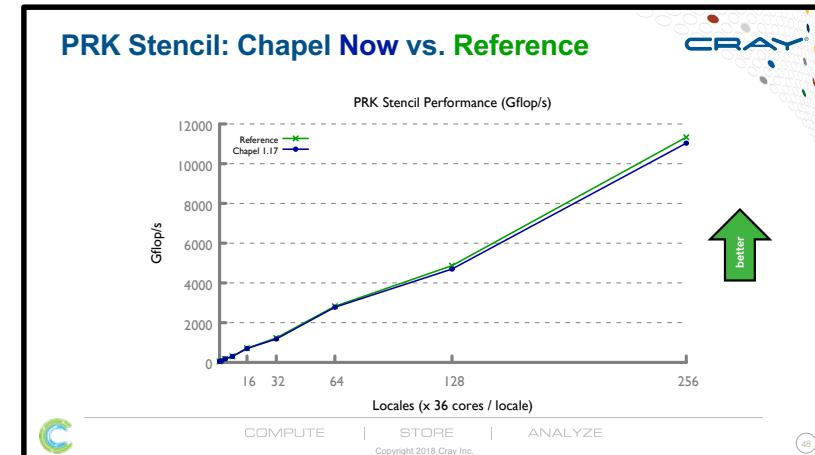
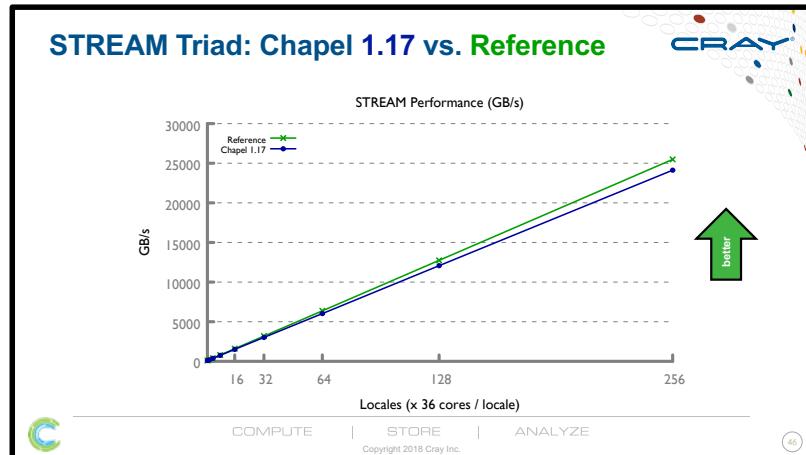
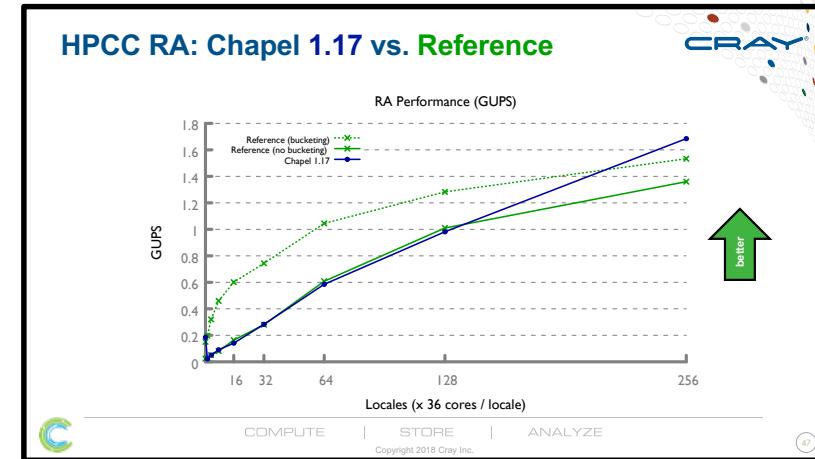
LCALS

STREAM
Triad



HPCC RA

PRK
Stencil



COMPUTE

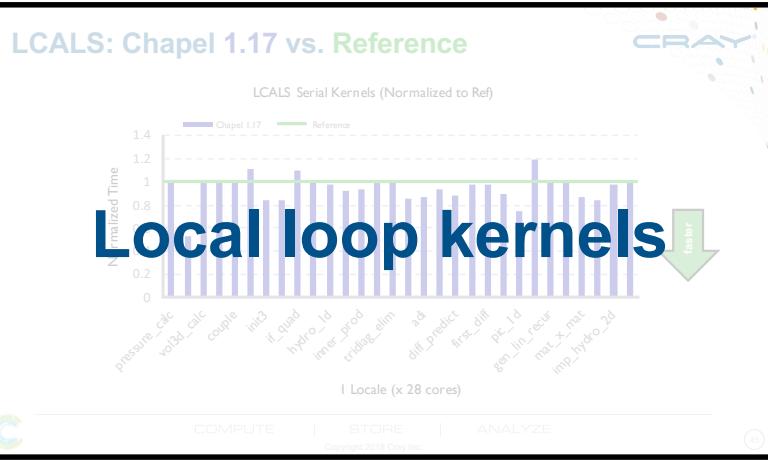
STORE

ANALYZE

Copyright 2018 Cray Inc.

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

HPC Patterns: Chapel Now vs. reference

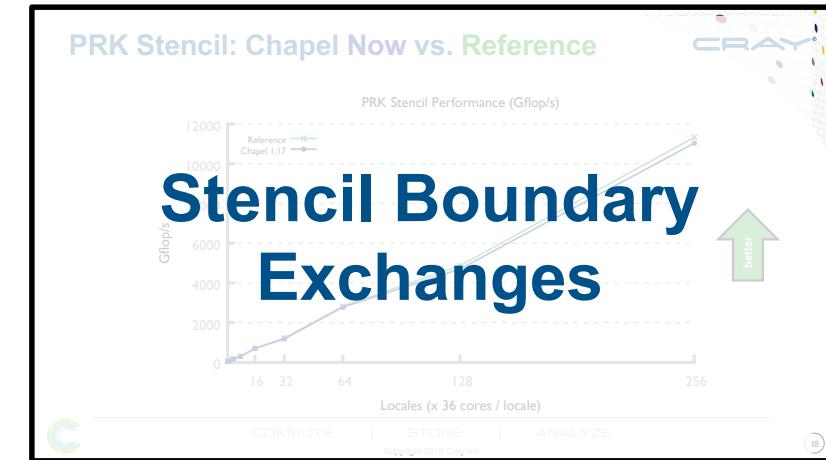
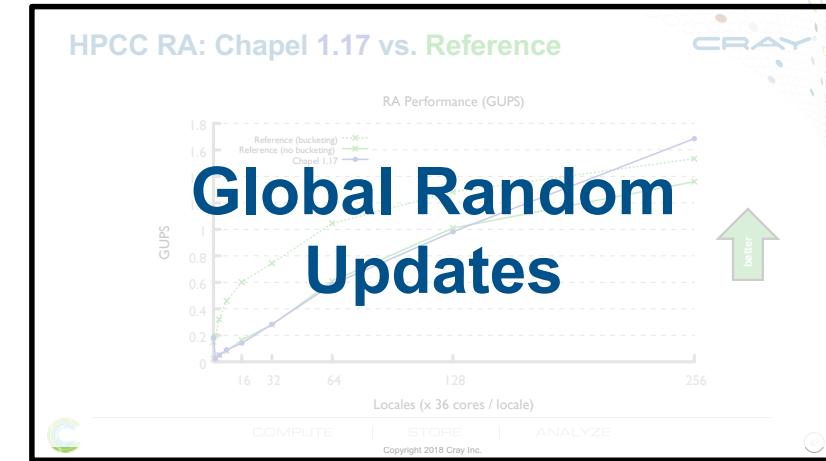
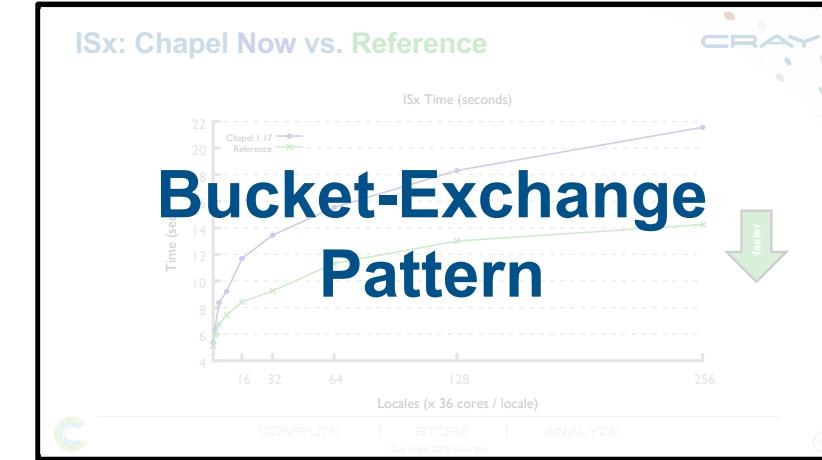
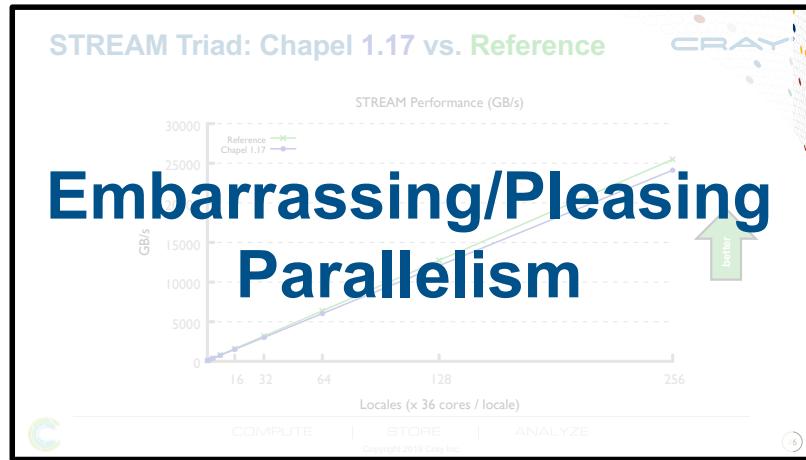


LCALS

STREAM
Triad

HPCC RA

PRK
Stencil



COMPUTE

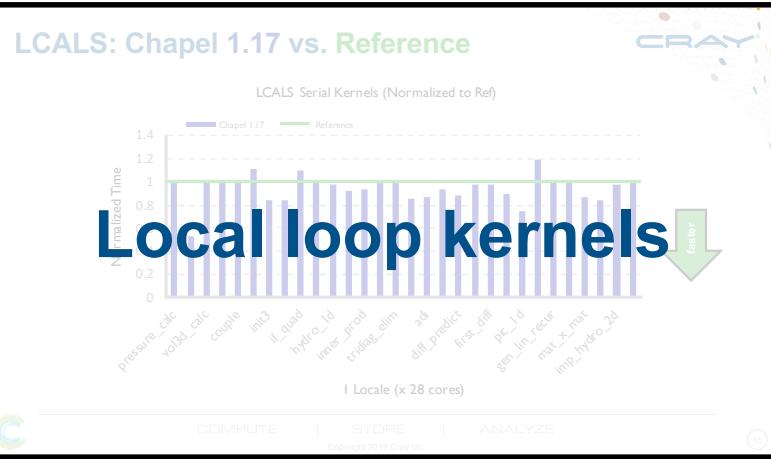
STORE

ANALYZE

Copyright 2018 Cray Inc.

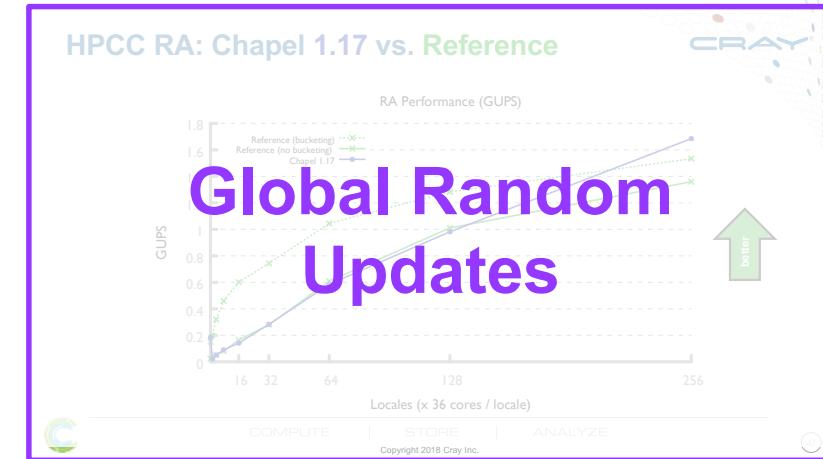
Nightly performance tickers online at:
<https://chapel-lang.org/perf-nightly.html>

HPC Patterns: Chapel Now vs. reference



LCALS

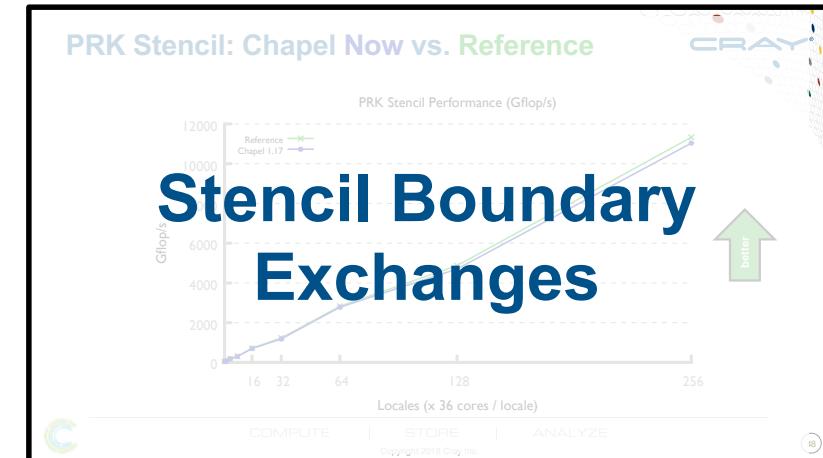
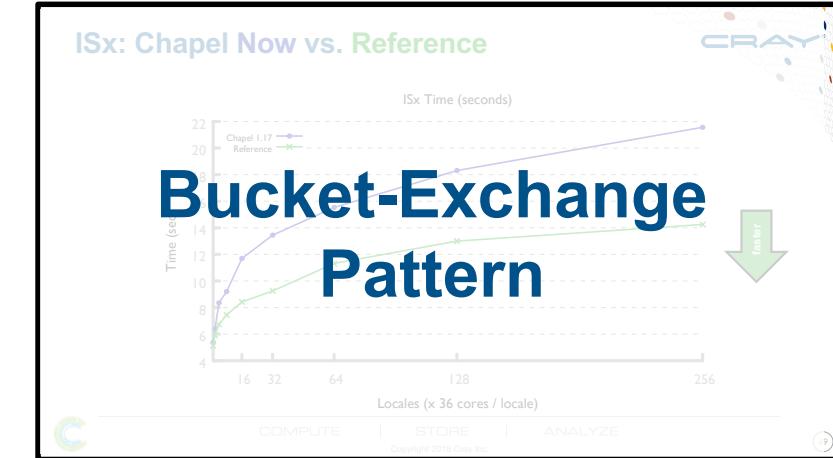
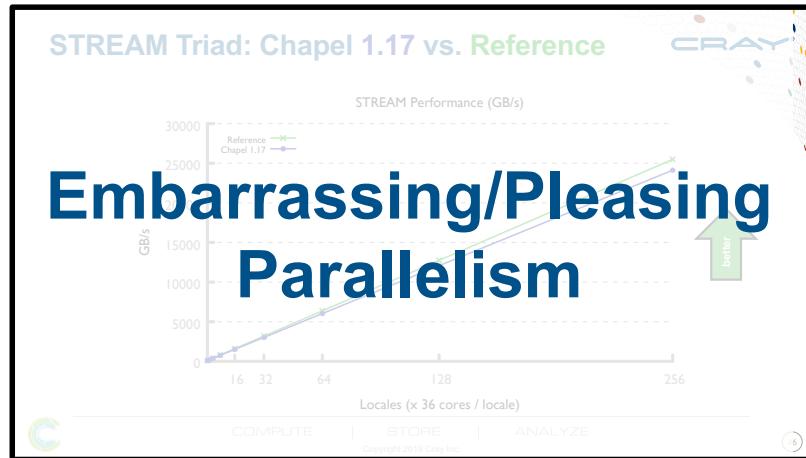
HPCC RA



STREAM
Triad

ISx

PRK
Stencil



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

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

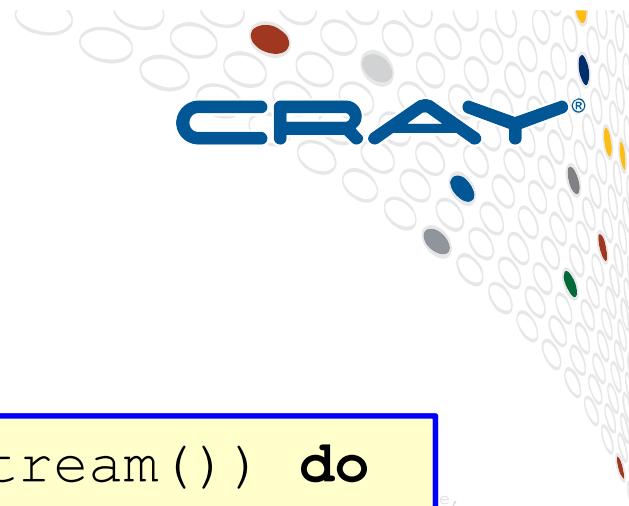
HPCC Random Access Kernel: MPI



```
/* 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);
}
```



HPCC Random Access Kernel: MPI



```
/* Perform updates to main table. The scalar equivalent is:  
*  
*   for (i=0; i<NUPDATE; i++) {  
*     Ran = (Ran << 1) ^ (((s64lnt) Ran < 0) ? POLY : 0);  
*     Table[Ran & (TABSIZ-1)] ^= Ran;  
*   }  
*/  
  
MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,  
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,  
          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;  
  
/* Perform update to main table */  
*  
*   for (i=0; i<NUPDATE; i++) {  
*     Ran = (Ran << 1) ^ (((s64lnt) Ran < 0) ? POLY : 0);  
*     Table[Ran & (TABSIZ-1)] ^= Ran;  
*   }  
*/
```

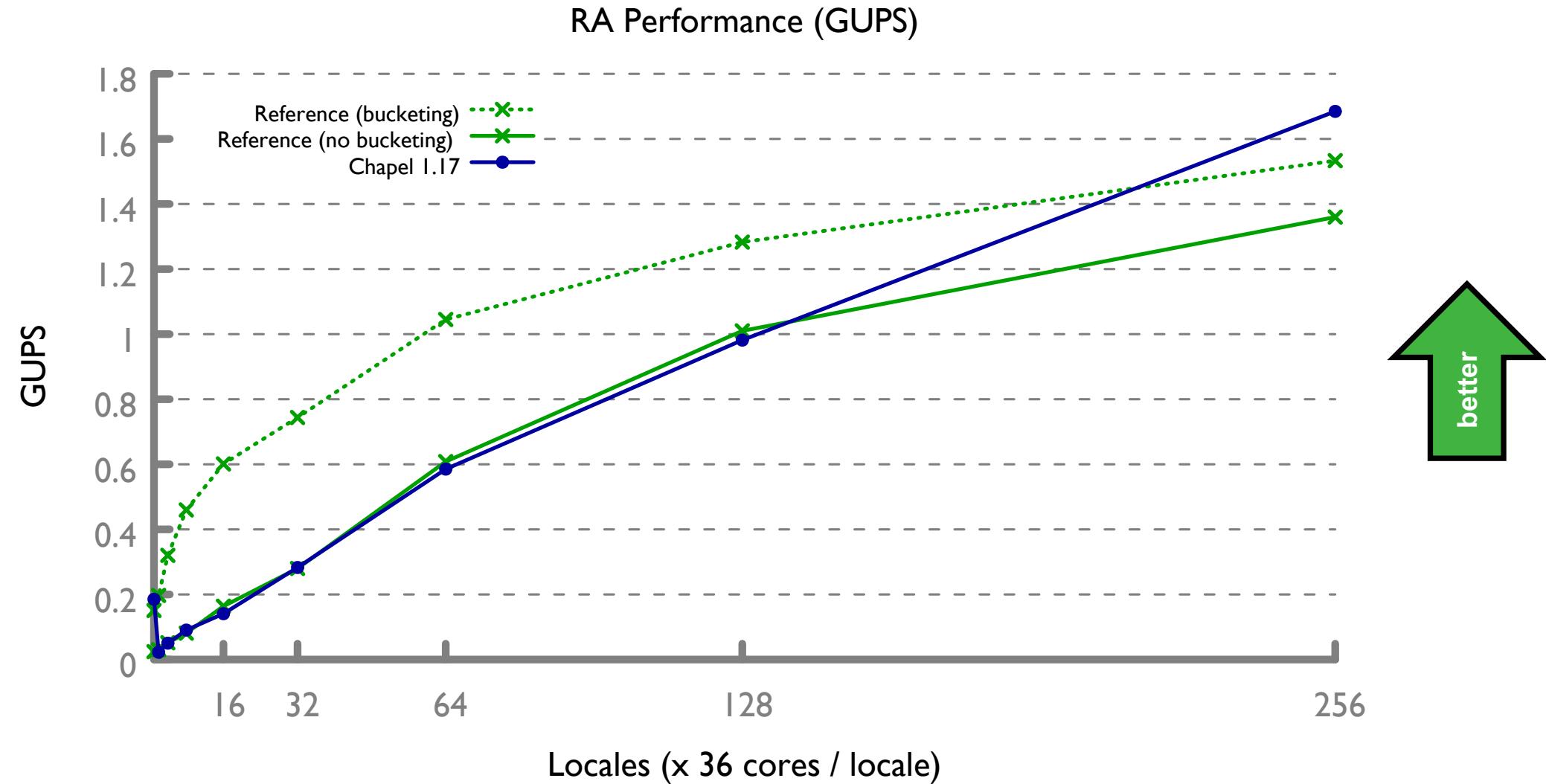
```
Chapel Kernel
forall (_, r) in zip(Updates, RASTream()) do
    T[r & indexMask] ^= r;
```

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 RA: Chapel Now vs. Ref



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Memory Leaks: Then vs. Now

(skipped at CUG due to time constraints)



CUG 2018

COMPUTE

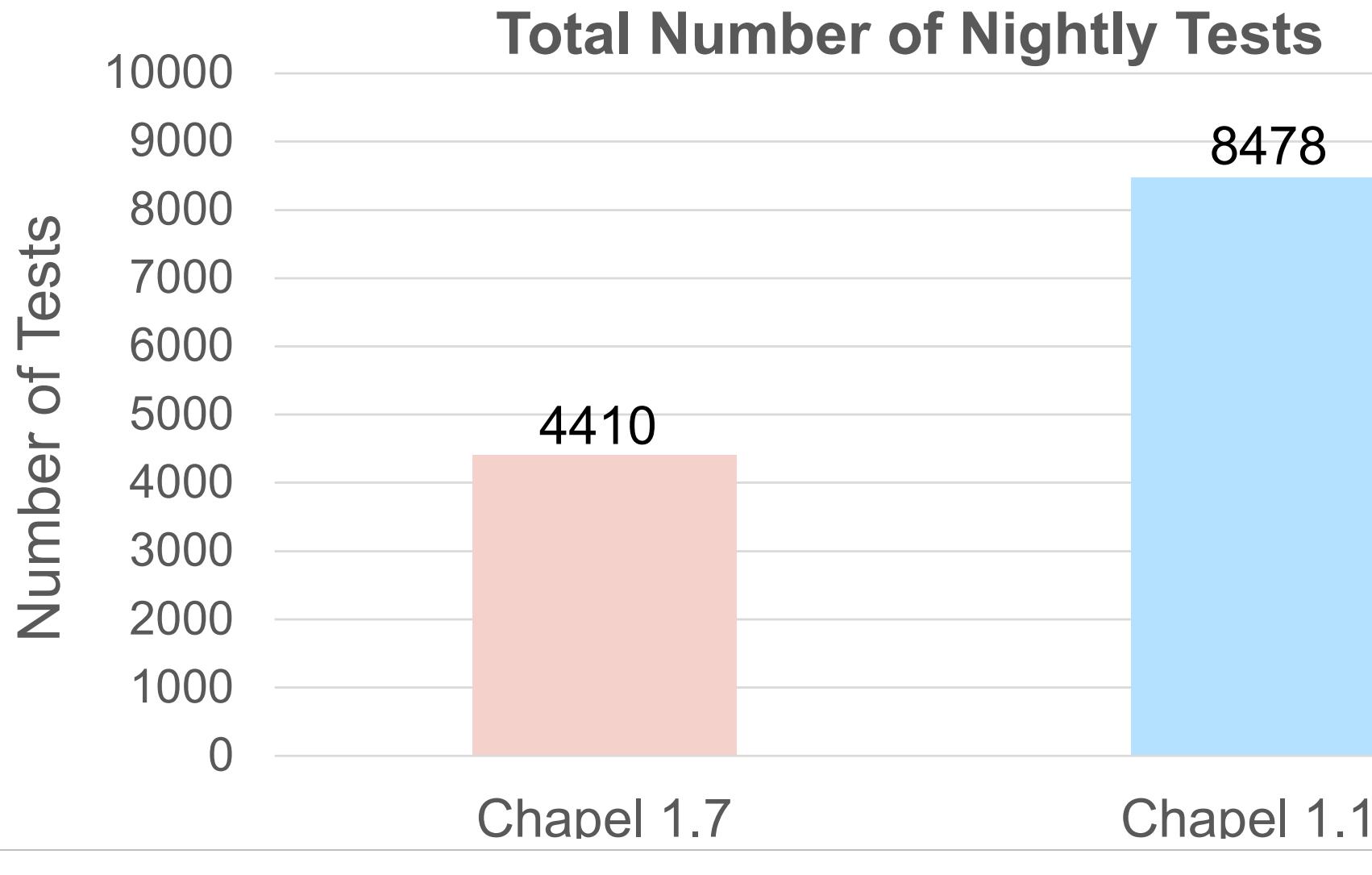
STORE

ANALYZE

Copyright 2018 Cray Inc.



Memory Leaks: Chapel Then vs. Now



CUG 2018

COMPUTE

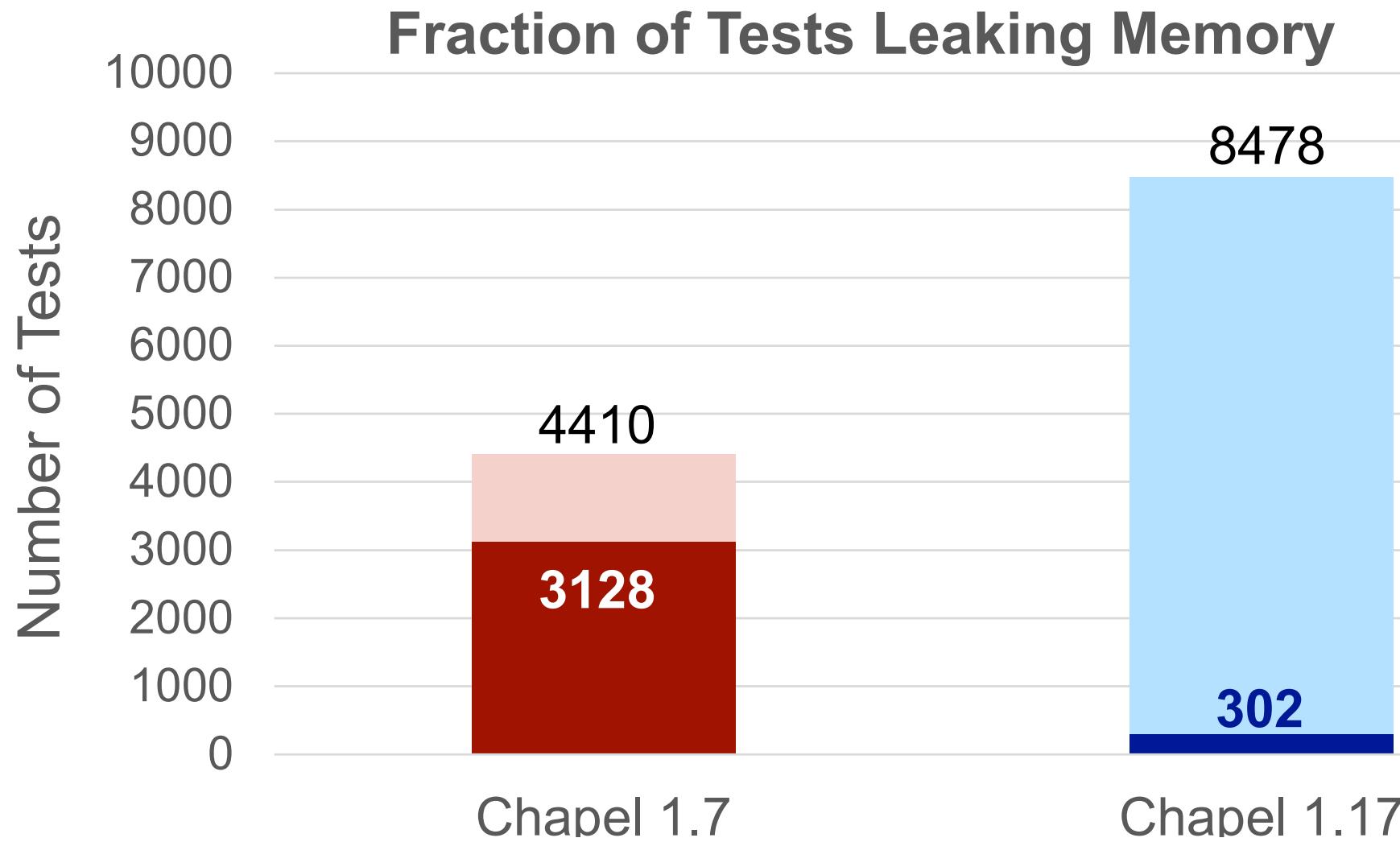
STORE

ANALYZE

Copyright 2018 Cray Inc.

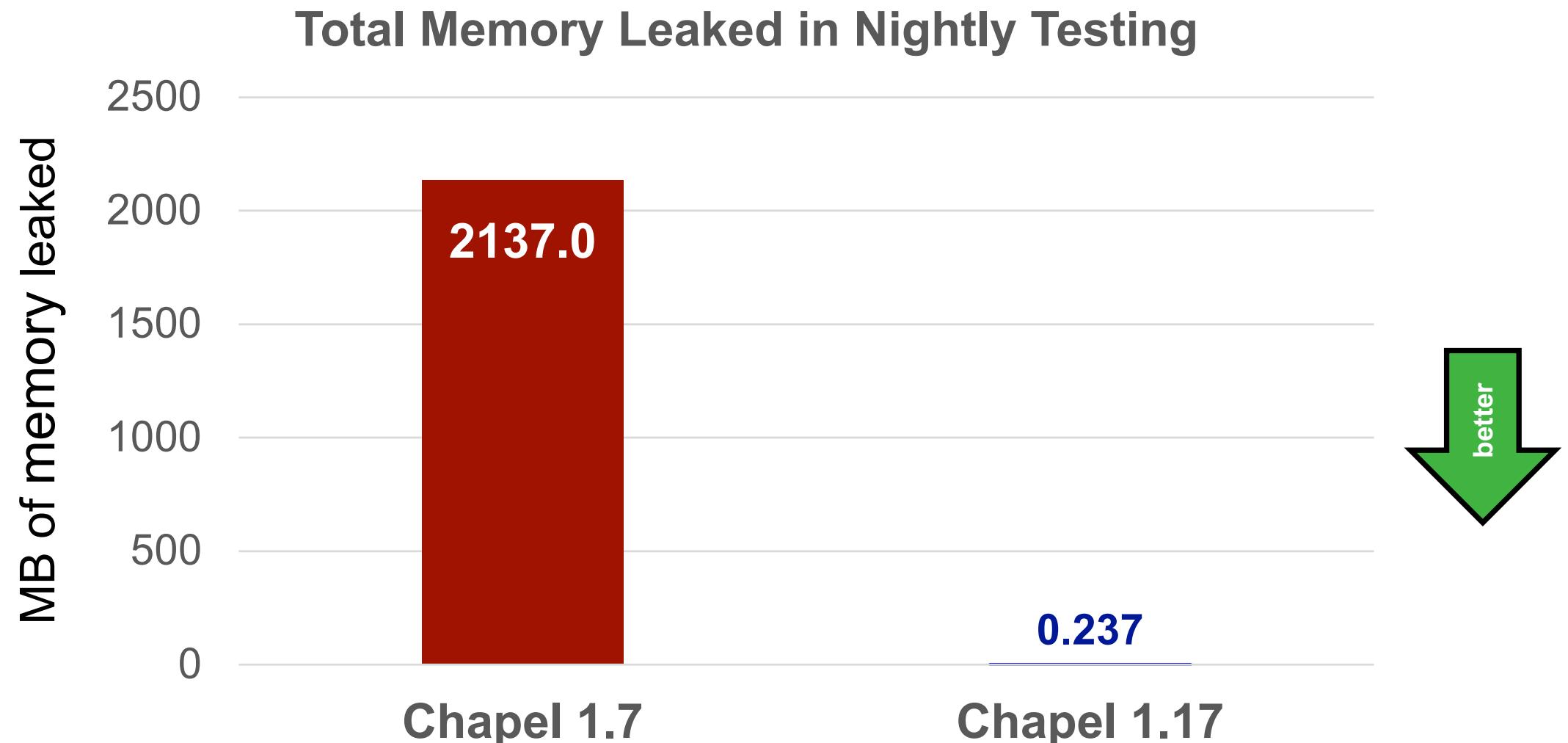


Memory Leaks: Chapel Then vs. Now

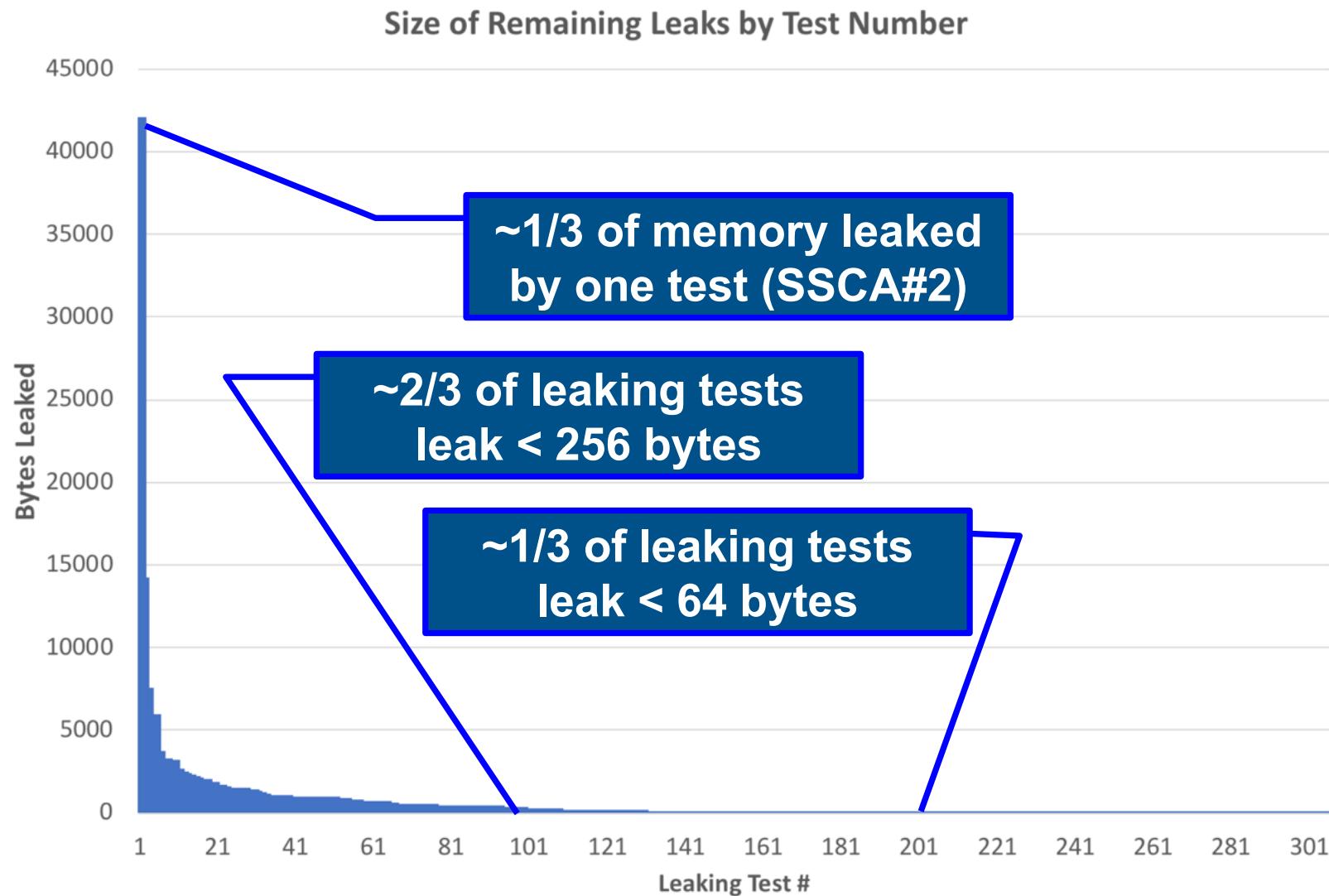




Memory Leaks: Chapel Then vs. Now



Memory Leaks: Remaining Leaks



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Chapel Language: Then vs. Now



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Language: Then

Parallelism and Locality: Generally in good shape

- not many changes here since HPCS

Base Language: Left much to be desired

- lots of focus here since HPCS



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Language: Now

Parallelism and Locality

- introduced *task intents* to reduce chances of race conditions
- and *user-defined locale models* to support new node architectures

Base Language

- fixed a number of problems with **object-oriented programming**
 - **records**: poor memory management discipline
 - **classes**: problems with generic classes, class hierarchies
 - made **strings** usable
 - added **error-handling** features
 - made **namespace** improvements
- (and **much more...**)



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Chapel Ecosystem: Then vs. Now



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Documentation: Then

After HPCS:

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

The terminal window displays the following content:

```
bradc -- ssh bradc@troll.cray.com -- bash
File Edit Options Buffers Tools Help
=====
Chapel doc README
=====

This directory contains the following documentation:

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

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

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

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

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

-----
For more Information

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

At the bottom of the terminal window:

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

The background shows the Chapel Language Specification Version 0.93 document and the Chapel Quick Reference sheet.

Chapel Language Specification Version 0.93

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

April 18, 2013

Chapel Quick Reference

Quick Start

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

1. Create the file `hello.chpl`
`writeln("Hello, world")`
2. Compile and run:
`chpl hello.chpl > ./a.out`

Expression Precedence and Associativity

Operators	Uses
<code>., ()[]</code>	member access, call and index
<code>.., ::[]</code>	member access, call and index
<code>**</code>	exponentiation
<code>reduce scan</code>	reduction, scan, apply domain
<code>map/reduce</code>	map
<code><=</code>	labeled and lexical operators

Statements

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

Selection Expressions

```
select expr {  
  when cond do stmt1();  
  when equiv2 [ stmt2(); ]  
  otherwise stmt3(); }
```

Iteration

```
while condition do ...  
do ... until ...  
for index in aggregate do ...  
for index in aggregate (-) ...  
label label for ...  
break; or break outers  
continue; or continue outers
```

Procedures

```
proc func (real, ii: imag): complex {  
  var ci: complex = x + iy;  
  return cz;  
}  
proc foo(ii) returns (x*x + i*i);
```

Formal Argument Intents

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

Named Formal Arguments

```
proc function (int, arg1: real) { ... }  
func(arg0=3.14, arg1=2);
```

Default Values for Formal Arguments

```
func function (int, arg1: real = 3.14);  
func(2);
```

Page 1



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Documentation: Now



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

The screenshot displays a hierarchical documentation structure for Chapel. At the top left is the main navigation bar for "Chapel Documentation 1.16". Below it, the "COMPILING AND RUNNING CHAPEL" section includes links to "Quickstart Instructions", "Using Chapel", "Platform-Specific Notes", "Technical Notes", and "Tools". The "WRITING CHAPEL PROGRAMS" section includes links to "Quick Reference", "Hello World Variants", "Primers", "Language Specification", "Built-in Types and Functions", "Standard Modules", "Package Modules", "Standard Layouts and Distributions", and "Chapel Users Guide (WIP)". The "LANGUAGE HISTORY" section includes links to "Chapel Evolution" and "Archived Language Specifications". The central page is titled "Chapel Documentation" and "Compiling and Running Chapel". It lists "Quickstart Instructions", "Using Chapel", "Platform-Specific Notes", "Technical Notes", and "Tools". Below this is a detailed list of "Chapel Prerequisites", "Setting up Your Environment for Chapel", "Building Chapel", "Compiling Chapel Programs", "Chapel Man Page", "Executing Chapel Programs", "Multilocale Chapel Execution", "Chapel Launchers", "Chapel Tasks", "Debugging Chapel Programs", and "Reporting Chapel Issues". A "View page source" link is located at the top right of this page. To the right, there are two more pages: "Using Chapel" and "Task Parallelism". The "Using Chapel" page has a "Contents:" sidebar with links to "Chapel Prerequisites", "Setting up Your Environment for Chapel", "Building Chapel", "Compiling Chapel Programs", "Chapel Man Page", "Executing Chapel Programs", "Multilocale Chapel Execution", "Chapel Launchers", "Chapel Tasks", "Debugging Chapel Programs", and "Reporting Chapel Issues". The "Task Parallelism" page includes sections for "Begin Statements" and "Cobegin Statements", each with code examples and descriptions. A "View page source" link is also present on the "Task Parallelism" page.



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Libraries: Then

After HPCS: ~25 library modules

- documented via source comments, if at all:

```
bradc -- ssh bradc@troll.cray.com -- bash
File Edit Options Buffers Tools chpl Help
// Copyright (c) 2004-2013, Cray Inc. (See LICENSE file for more details)

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

```
bradc -- ssh bradc@troll.cray.com -- bash
File Edit Options Buffers Tools chpl Help
// Copyright (c) 2004-2013, Cray Inc. (See LICENSE file for more details)

extern type qio_regexp_t;

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

extern proc qio_regexp_null():qio_regexp_t;
extern proc qio_regexp_init_default_options(ref options:qio_regexp_options_t);
extern proc qio_regexp_create_compile(str:string, strlen:int(64), ref options:qio_regexp_options_t, ref compiled:qio_regexp_t);
extern proc qio_regexp_create_compile_flags(str:string, strlen:int(64), flags:$\string, flagslen:int(64), isutf8:bool, ref compiled:qio_regexp_t);
extern proc qio_regexp_create_compile_flags_2(str:c_ptr, strlen:int(64), flags:c_ptr, flagslen:int(64), isutf8:bool, ref compiled:qio_regexp_t);
extern proc qio_regexp_retain(ref compiled:qio_regexp_t);
extern proc qio_regexp_release(ref compiled:qio_regexp_t);

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

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

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

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





Libraries: Now

Now: ~60 library modules

- web-documented, many user-contributed

The screenshot shows two pages from the Chapel Documentation 1.16 website side-by-side.

Standard Modules Page:

- Header: Docs > Standard Modules
- Section: Standard Modules
- Description: Standard modules are those which describe features that are considered part of the Standard Library.
- Text: All Chapel programs automatically use the modules `Assert`, `IO`, `Math`, and `UtilReplicatedVar`.
- List: `Assert`, `Barrier`, `Barriers`, `BigInteger`, `BitOps`, `Buffers`, `CommDiagnostics`, `DateTime`, `DynamicIterators`, `FileSystem`, `GMP`, `Help`, `IO`, `List`, `Math`, `Memory`, `Path`, `Random`, `Reflection`, `Regexp`, `Spawn`, `Sys`, `SysBasic`, `SysCTypes`, `SysError`, `Time`, `Types`, `UtilReplicatedVar`.

Package Modules Page:

- Header: Docs > Package Modules
- Section: Package Modules
- Description: Package modules are libraries that currently live outside of the Chapel Standard Library, either because they are not considered to be fundamental enough or because they are not yet mature enough for inclusion there.
- List: `BLAS`, `Collection`, `Crypto`, `Curl`, `DistributedBag`, `DistributedDeque`, `DistributedIterators`, `FFTW`, `FFTW_MT`, `Futures`, `HDFS`, `HDFSIterator`, `LAPACK`, `LinearAlgebra`, `MPI`, `Norm`, `OwnedObject`, `RangeChunk`, `RecordParser`, `Search`, `SharedObject`, `Sort`, `VisualDebug`, `ZMQ`.



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Libraries: Now

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, ...



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Tools: Then



After HPCS:

- **highlighting modes** for emacs and vim
- **chpldoc**: documentation tool (early draft)



CUG 2018

COMPUTE

STORE

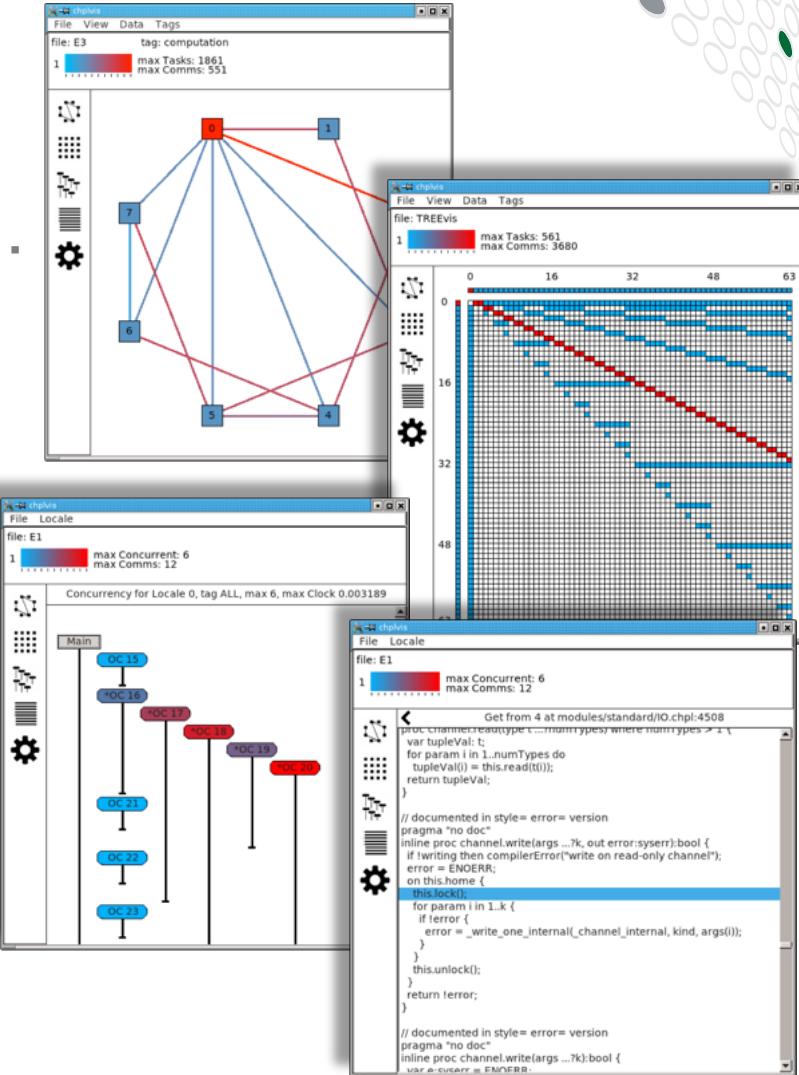
ANALYZE

Copyright 2018 Cray Inc.

Tools: Now

Now:

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



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Then vs. Now: And so much more...



Interoperability:

- passing arrays & functions to C, working with C pointers, ...

Development process:

- GitHub, Jenkins, Travis, interactive nightly performance graphs...

Social media:

Twitter, Facebook, YouTube

User support:

GitHub issues, StackOverflow, Gitter, email

Web presence:

CLBG, Try It Online, CyberDojo, ...

Memory Leaks:

significantly reduced

CHIUW:

annual community workshop



Chapel User Profiles



CUG 2018

COMPUTE

STORE

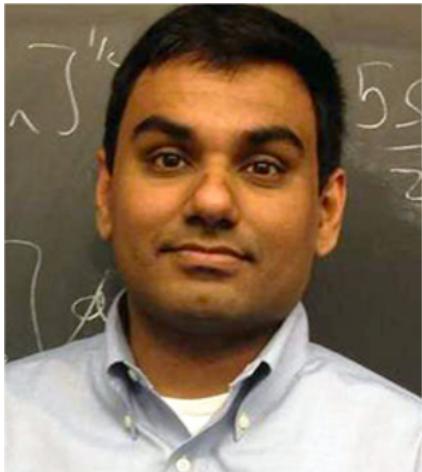
ANALYZE

Copyright 2018 Cray Inc.

Chapel User Profiles



Current Users:



Time-to-science
Cosmologist



Commercial AI
Scientist

Potential Users:



Genomic
Researcher

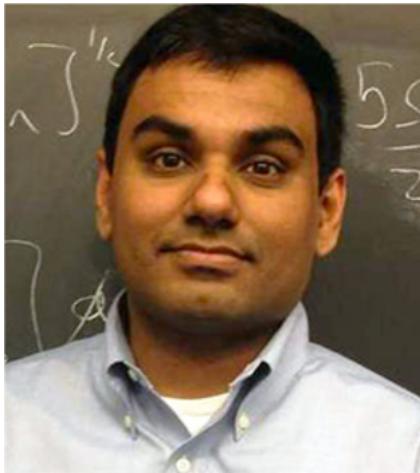


DOE Scientist

Chapel User Profiles



Current Users:

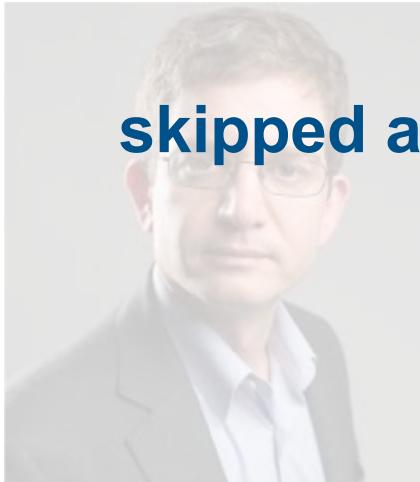


Time-to-science
Cosmologist



Commercial AI
Scientist

Potential Users:

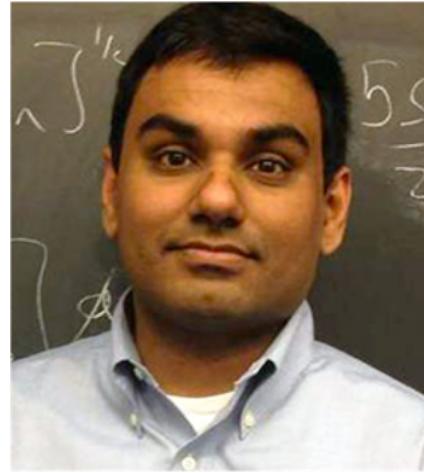


Genomic
Researcher



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.”



CUG 2018

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.”



CUG 2018

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.**”



CUG 2018

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.**”



CUG 2018

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]**



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



What's Next?



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Crossing the Stream of Adoption



Research Prototype

Adopted in Production

Performance & Scalability

Immature Language Features

Insufficient Libraries

Memory Leaks

Lack of Tools

Lack of Documentation

Fear of Being the Only User

Next DOE app

Next weather /
climate model

[your production
app here]



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

image credit: <http://feelgrafix.com/813578-free-stream-wallpaper.html>

Crossing the Stream of Adoption



Research Prototype

Adopted in Production

Performance & Scalability

Immature Language Features

Insufficient Libraries

Memory Leaks

Lack of Tools

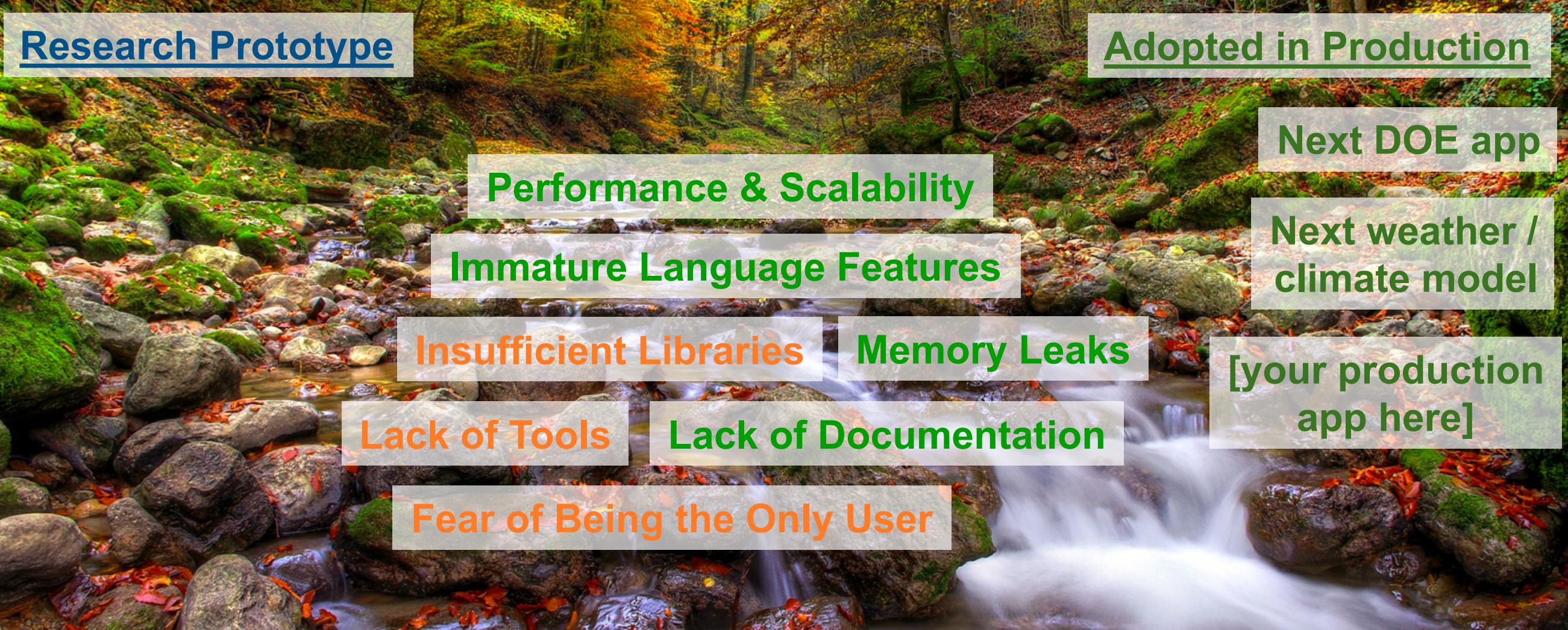
Lack of Documentation

Fear of Being the Only User

Next DOE app

Next weather /
climate model

[your production
app here]



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

image credit: <http://feelgrafix.com/813578-free-stream-wallpaper.html>

Crossing the Stream of Adoption



Research Prototype

CLBG

MiniMD

ISx

CoMD

PRK Stencil

RA

LULESH

Stream

LCALS

Time-to-science
academic codes

Adopted in Production

Next DOE app

Next weather /
climate model

[your production
app here]

What are the next
stepping stones?

Where can Chapel help your
workflow's productivity?



CUG 2018

COMPUTE

STORE

ANALYZE

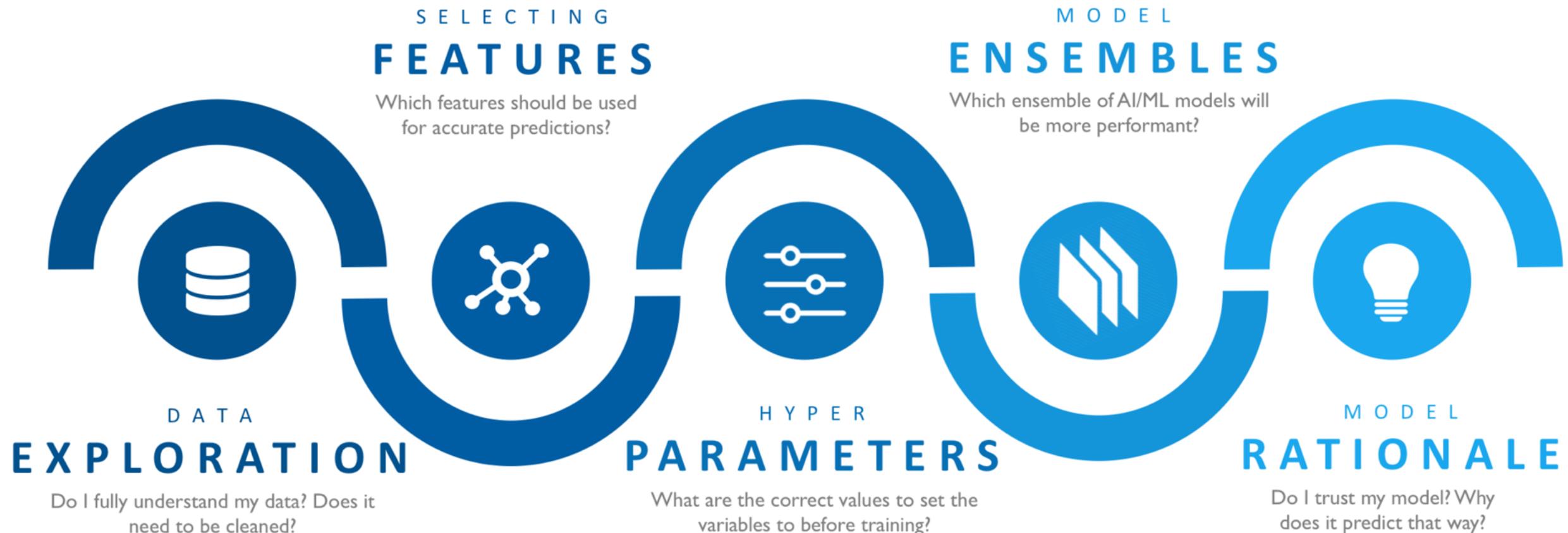
Copyright 2018 Cray Inc.

image credit: <http://feelgrafix.com/813578-free-stream-wallpaper.html>



Discovery Roadblocks

Data Science Pain Points



CUG 2018

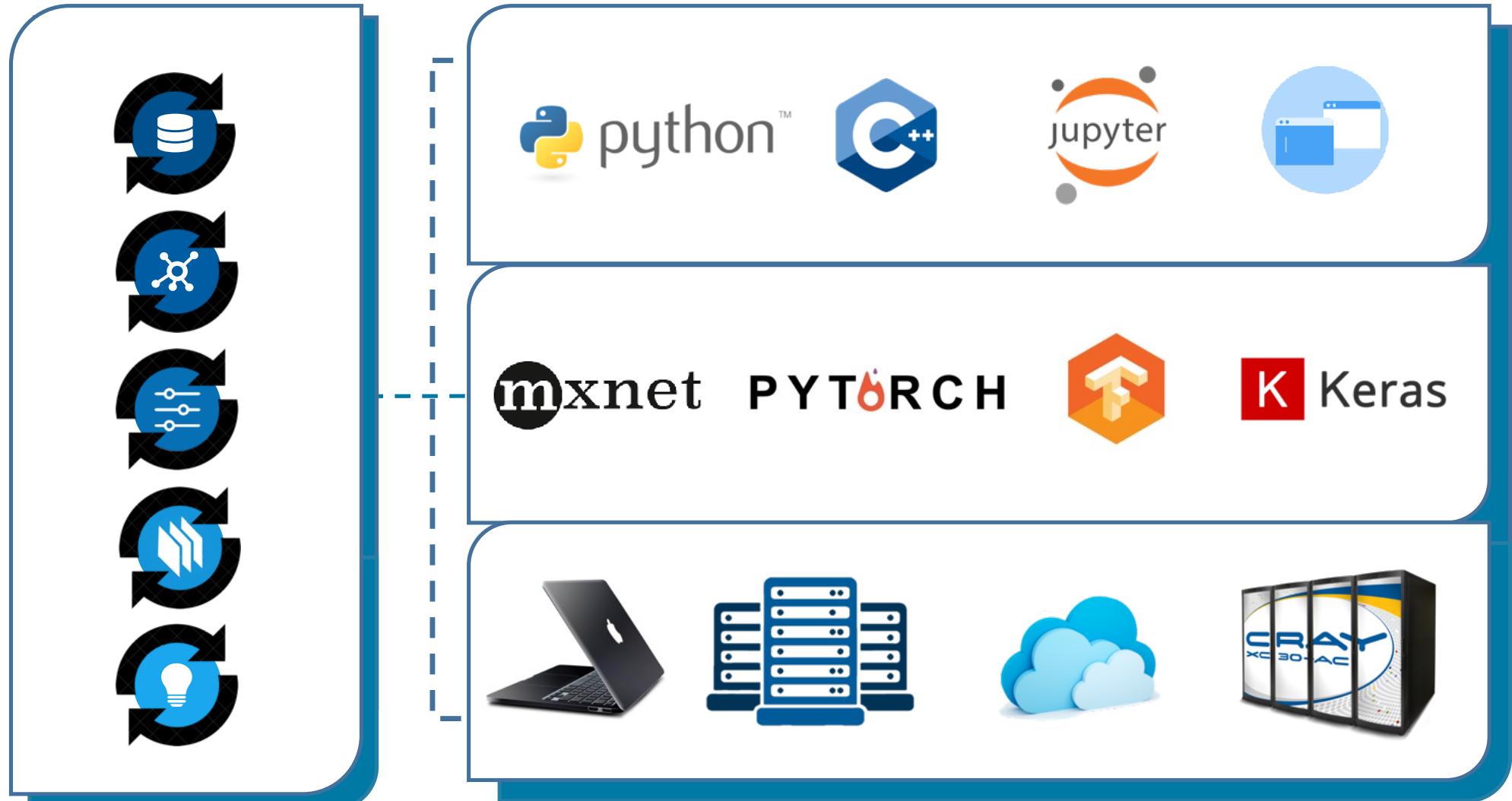
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel AI Ecosystem



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Sample Chapel AI Workflow



- User works from within a Jupyter notebook
- Uses Chapel to ingest large HDF5 data files
 - read in parallel
 - transformed / analyzed during ingestion
 - stored in a distributed Dataframe
- Starts working on model locally on laptop
- As confidence in model grows, tunes it at scale
 - feature selection
 - hyperparameter optimization



CUG 2018

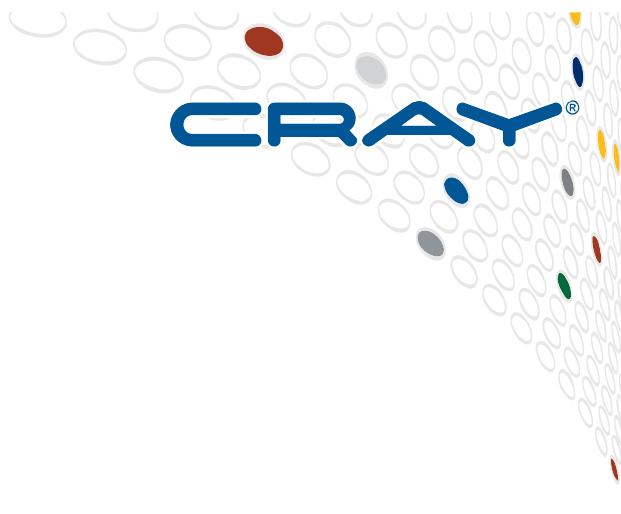
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

What's Next?



Chapel's college years: plans for 2018-2021

- Language Core
- Interoperability / Usability
- Portability
- Data Ingestion
- Chapel AI



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

What's Next?



Chapel's college years: plans for 2018-2021

- **Language Core**
 - Language stabilization: avoid backward-breaking changes
 - Sparse array improvements, partial reductions, delete-free features, ...
 - Additional performance and scalability improvements
- **Interoperability / Usability**
- **Portability**
- **Data Ingestion**
- **Chapel AI**



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

What's Next?



Chapel's college years: plans for 2018-2021

- Language Core
- Interoperability / Usability
 - Python / C++ interoperability
 - Support for Jupyter notebooks / REPL
- Portability
- Data Ingestion
- Chapel AI



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

What's Next?



Chapel's college years: plans for 2018-2021

- Language Core
- Interoperability / Usability
- Portability
 - LLVM back-end
 - Target Libfabric/OFI
 - Target GPUs
 - Cloud computing support
- Data Ingestion
- Chapel AI



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

What's Next?



Chapel's college years: plans for 2018-2021

- Language Core
- Interoperability / Usability
- Portability
- Data Ingestion
 - Support HDF5, NetCDF, CSV, ...
 - Transform-on-ingest
 - Distributed DataFrames support
- Chapel AI



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

What's Next?



Chapel's college years: plans for 2018-2021

- Language Core
- Interoperability / Usability
- Portability
- Data Ingestion
- Chapel AI
 - Hyperparameter optimization
 - Deep Learning
 - ...



CUG 2018

COMPUTE

STORE

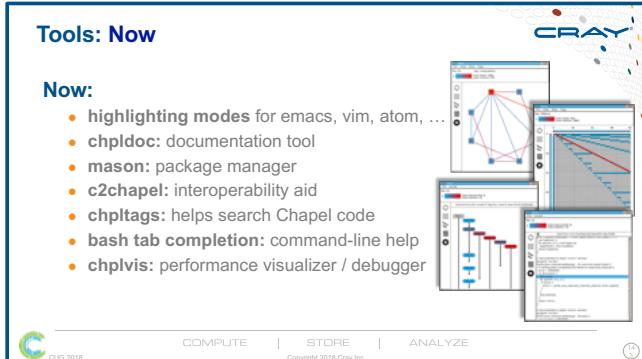
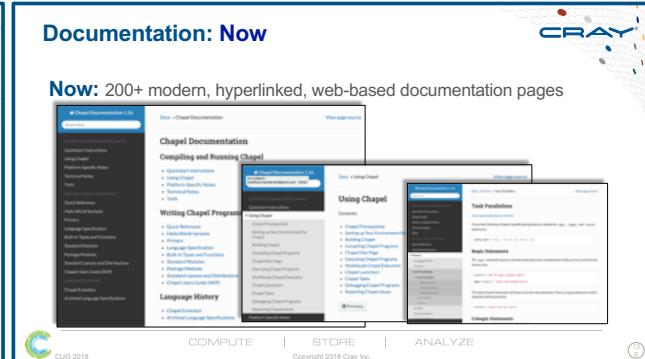
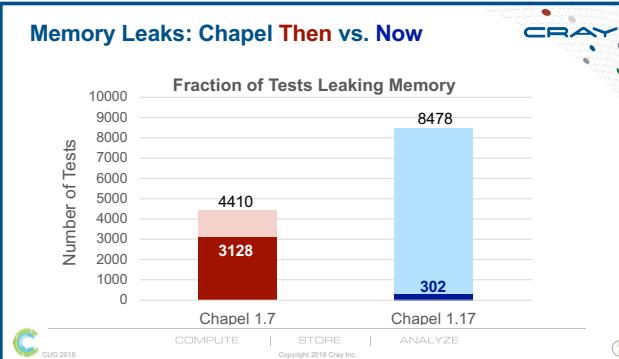
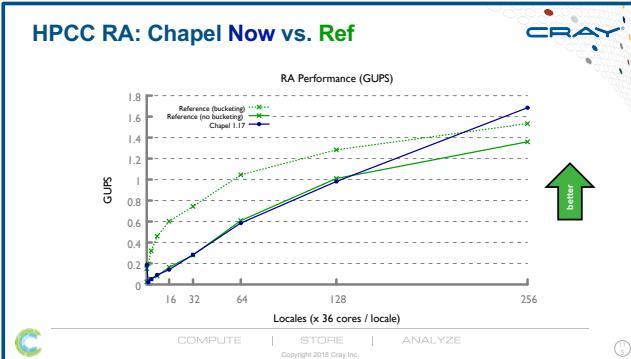
Copyright 2018 Cray Inc.

ANALYZE

Summary

Chapel has made huge strides over the past five years

We've addressed many historical barriers to using Chapel



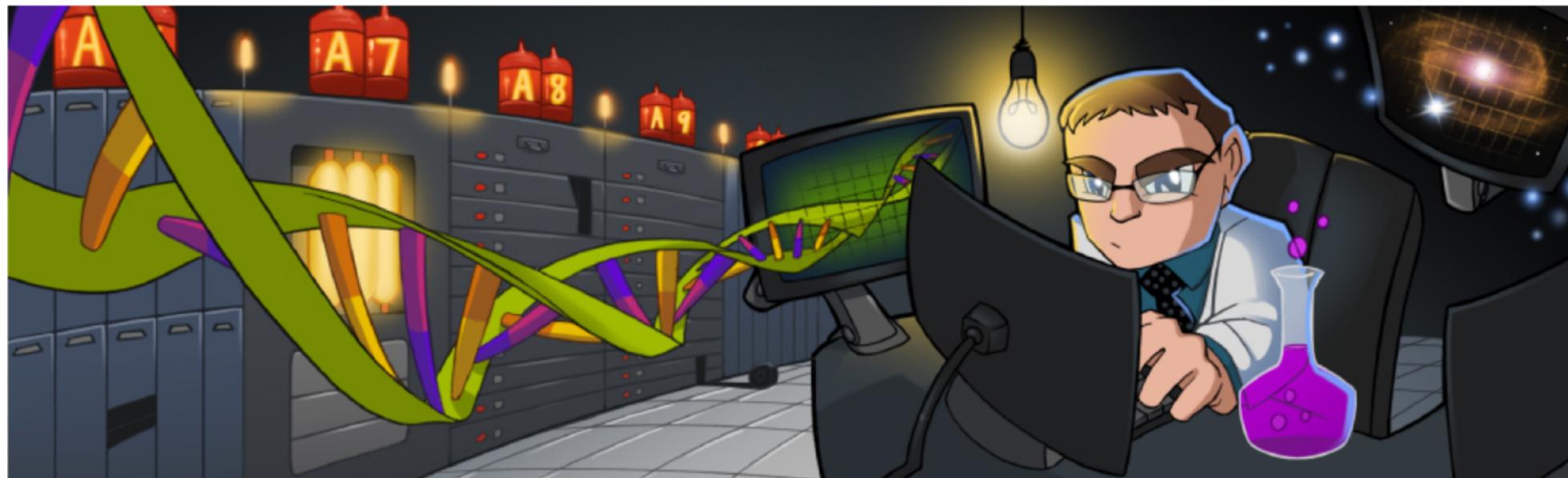
We're continuing our work to support and improve Chapel

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



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*



CUG 2018

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>



CUG 2018

COMPUTE

STORE

ANALYZE

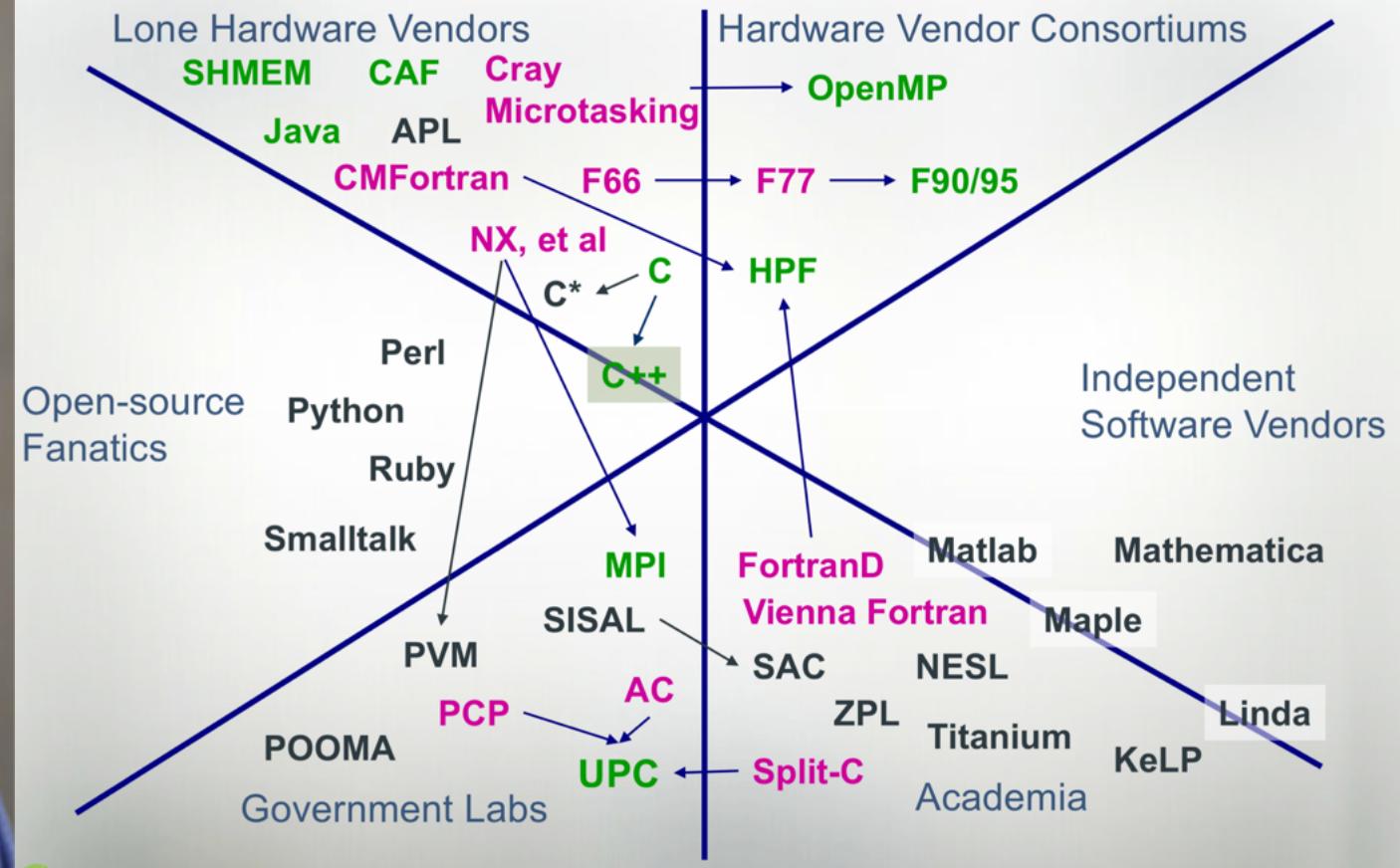
Copyright 2018 Cray Inc.

Dedicated to the Memory of Burton Smith



Where do Languages Come From?

CRAY
THE SUPERCOMPUTER COMPANY





Chapel Resources



CUG 2018

COMPUTE

STORE

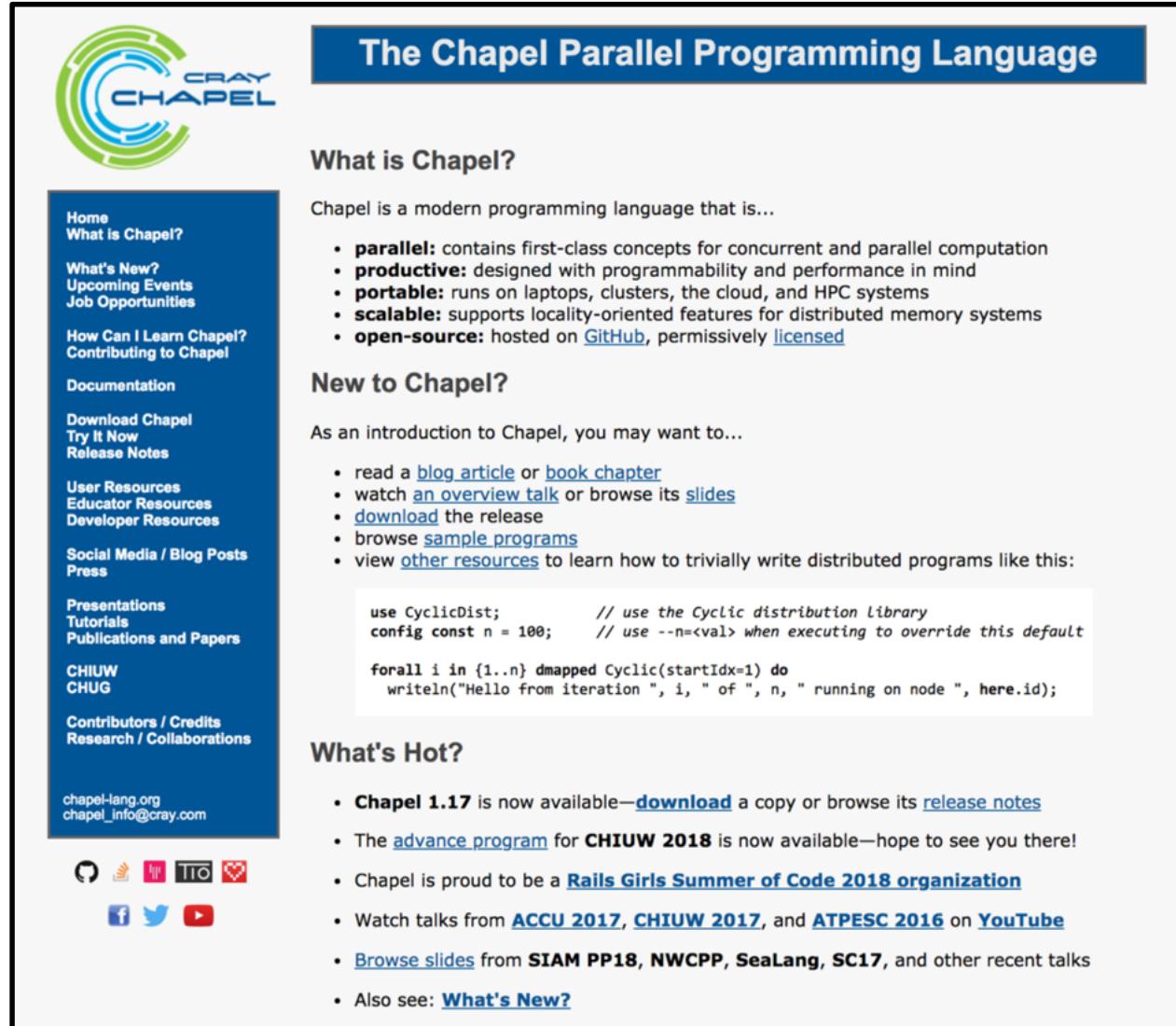
ANALYZE

Copyright 2018 Cray Inc.

Chapel Central

<https://chapel-lang.org>

- downloads
- documentation
- resources
- presentations
- papers



The screenshot shows the homepage of the Chapel Parallel Programming Language website. The header features the CRAY logo with a stylized 'C' composed of circles. The main title is "The Chapel Parallel Programming Language". Below the title, there's a section titled "What is Chapel?" which describes Chapel as a modern programming language with characteristics like parallel, productive, portable, scalable, and open-source. It includes links to "Home", "What is Chapel?", "What's New?", "Upcoming Events", "Job Opportunities", "How Can I Learn Chapel?", "Contributing to Chapel", "Documentation", "Download Chapel", "Try It Now", "Release Notes", "User Resources", "Educator Resources", "Developer Resources", "Social Media / Blog Posts", "Press", "Presentations", "Tutorials", "Publications and Papers", "CHIUW", "CHUG", "Contributors / Credits", and "Research / Collaborations". The footer contains the email address "chapel-lang.org" and "chapel_info@cray.com", along with social media icons for GitHub, YouTube, LinkedIn, and others.

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 **CHIUW 2018** is now available—hope to see you there!
- Chapel is proud to be a [Rails Girls Summer of Code 2018 organization](#)
- Watch talks from [ACCU 2017](#), [CHIUW 2017](#), and [ATPESC 2016](#) on [YouTube](#)
- [Browse slides](#) from **SIAM PP18**, **NWCPP**, **SeaLang**, **SC17**, and other recent talks
- Also see: [What's New?](#)



CUG 2018

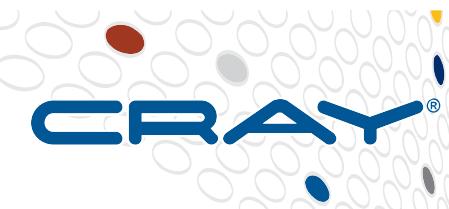
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Social Media (no account required)



<http://twitter.com/ChapelLanguage>

<http://facebook.com/ChapelLanguage>

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

The image displays four separate screenshots of social media interfaces for the Chapel programming language:

- Twitter:** Shows the Chapel Language account (@ChapelLanguage) with 576 tweets, 48 following, 278 followers, and 200 likes. A pinned tweet from Brad Chamberlain is visible.
- Facebook:** Shows the Chapel Programming Language page with 72 subscribers. It features a chart comparing program times across various languages, with Chapel being the fastest.
- YouTube:** Shows the "Chapel Parallel Programming Language" channel with 72 subscribers. It contains a playlist of videos, including keynotes from CHI UW 2017 and ACCU 2017, and a presentation from PYCON UK 2017.
- Custom Channel:** Shows a custom channel page for Chapel with a large "C" logo, a brief description, and links to the other social media platforms.



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Community



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

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

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

chapel-announce@lists.sourceforge.net

The collage consists of four screenshots:

- Stack Overflow Tagged Questions:** Shows the 'chapel' tag page with three questions: "Tuple Concatenation in Chapel", "Is there a way to use non-scalar values in functions with where clauses in", and "Is there any writef() format specifier for a bool?".
- Github Issues:** Shows the 'chapel-lang / chapel' repository issues page with 292 open issues, including topics like "Implement 'bounded-coforall' optimization for remote coforalls" and "make uninstall".
- Gitter Chat:** Shows a Gitter channel for 'chapel-lang/chapel' with a purple background. It includes promotional text for Gitter and a transcript of a conversation between Brian Dolan (@buddha314) and Michael Ferguson (@mppf) about array syntax.
- SourceForge Mailing List:** Shows the 'chapel-announce' mailing list with a purple background. It includes promotional text for Gitter and a transcript of a conversation between Brian Dolan (@buddha314) and Michael Ferguson (@mppf) about array syntax.



CUG 2018

COMPUTE

STORE

ANALYZE

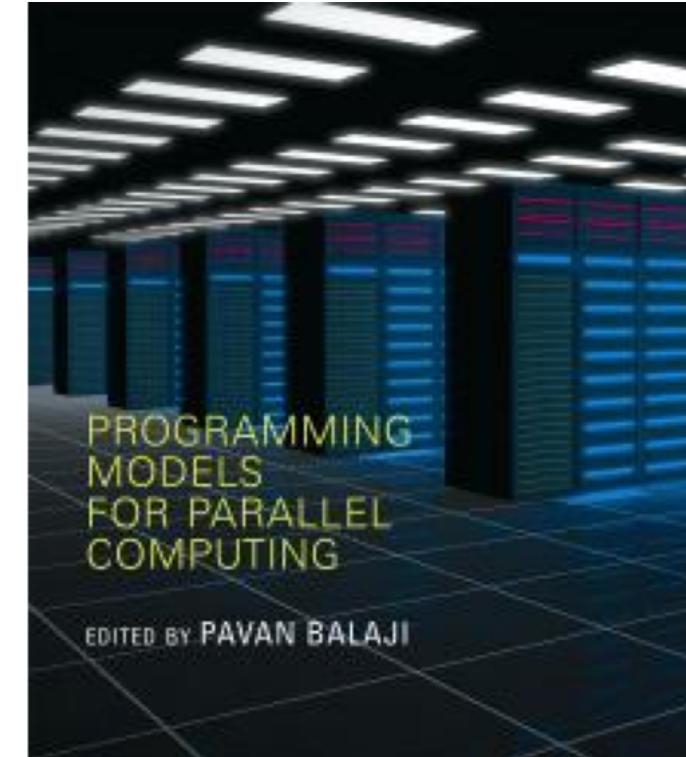
Copyright 2018 Cray Inc.

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>



CUG 2018

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*



CUG 2018

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



CUG 2018

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.



CUG 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Q&A

Brad Chamberlain
bradc@cray.com