

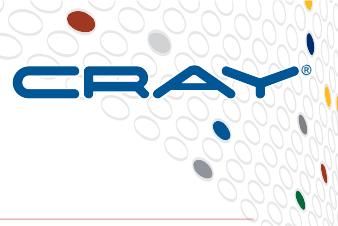
Chapel: Programmability, Parallelism, and Performance

PuPPy Scientific Computing SIG

Brad Chamberlain, Chapel Team, Cray Inc.



Cray: The Supercomputer Company



Jump-Start Your AI Deployment

Now organizations can get value from artificial intelligence faster and with less complexity than ever before. IT and data science teams can accelerate their infrastructure with our new Unika®-CS AI and Analytics software suite.

[READ THE NEWS](#)

Can LS-DYNA Scale Any Higher?
Cray, LSTC, NCSA and Rolls-Royce are continuing to explore the future of implicit finite element analyses of large-scale models with LS-DYNA.

[READ THE BLOG >](#)

How to Choose the Right Data Storage the First Time
An organization's storage plan must be driven by its needs and cost. But along came data... and then more data and bigger data and, well, it's not stopping.

[READ THE BLOG >](#)

Business Cards Change, but the Passion for Open-Source Lustre Doesn't
Proprietary file systems will always be part of the requirements of future leadership-class environments, so we're focused on Lustre.

[READ THE NEWS](#)

GENCI's New Supercomputer Powered by ClusterStor Storage
GENCI in Paris is getting a new supercomputer. The new system, "Joliot-Curie," will be empowered by Cray ClusterStor storage.

[READ THE NEWS](#)

Gray newsletter — get it once a month, delivered straight to your inbox.

Ask Your Biggest Questions

SUPERCOMPUTING
Scale your goals with high-performance compute solutions

DATA STORAGE
Get faster insights with simpler storage and data management

BIG DATA ANALYTICS
Think bigger about big data with agile analytics technology

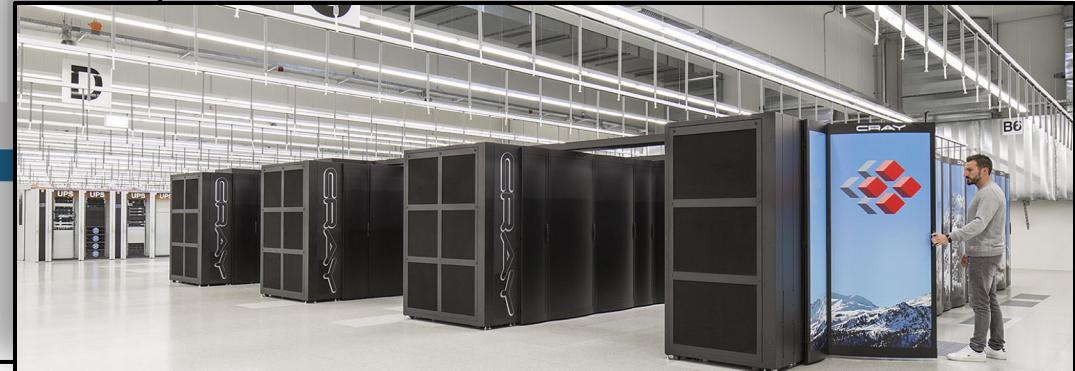
ARTIFICIAL INTELLIGENCE
Create tomorrow with compute tools for AI development

CLOUD
Extend your possibilities with cloud-based supercomputing and storage

Piz Daint specifications

Model Cray XC40/Cray XC50

Number of Hybrid Compute Nodes	5 320
Number of Multicore Compute Nodes	1 431
Peak Floating-point Performance per Hybrid Node	4.761 Teraflops Intel Xeon E5-2690 v3/Nvidia Tesla P100
Peak Floating-point Performance per Multicore Node	1.210 Teraflops Intel Xeon E5-2695 v4
Hybrid Peak Performance	25.326 Petaflops
Multicore Peak Performance	1.731 Petaflops
Hybrid Memory Capacity per Node	64 GB; 16 GB CoWoS HBM2
Multicore Memory Capacity per Node	64 GB, 128 GB
Total System Memory	437.9 TB; 83.1 TB
System Interconnect	Cray Aries routing and communications ASIC, and Dragonfly network topology
Sonexion 3000 Storage Capacity	6.2 PB
Sonexion 3000 Parallel File System Theoretical Peak Performance	112 GB/s
Sonexion 1600 Storage Capacity	2.5 PB
Sonexion 1600 Parallel File System Theoretical Peak Performance	138 GB/s



<https://www.cray.com>

<https://www.cscs.ch/computers/piz-daint/>



COMPUTE

STORE

ANALYZE

Copyright 2018 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



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.”



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]



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

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	34,024	423	1.64	99% 3% 1% 4%
1.0	Chapel	1.62	33,652	501	1.64	100% 0% 1% 1%
1.1	Pascal Free Pascal #3	1.73	2,284	482	1.72	1% 100% 1% 1%
1.1	C gcc	1.73	2,116	448	1.73	1% 99% 1% 0%
1.1	Ada 2005 GNAT #2	1.74	3,776	1065	1.73	1% 0% 100% 0%
1.1	Rust #2	1.74	7,876	1306	1.74	1% 100% 1% 1%
1.1	Rust	1.74	7,892	1420	1.74	100% 1% 2% 1%
1.1	Swift #2	1.75	8,532	601	1.75	100% 1% 1% 0%
1.1	Lisp SBCL #4	1.79	25,164	940	1.79	3% 2% 1% 100%
1.2	C++ g++ #4	1.89	3,868	508	1.89	100% 1% 2% 1%
1.2	Lua #5	1.94	3,248	479	1.93	1% 1% 1% 99%
1.2	Go #3	2.02	10,744	603	2.02	2% 0% 5% 96%
1.3	PHP #5	2.15	9,884	394	2.15	1% 0% 100% 1%
1.3	PHP #4	2.16	9,856	384	2.16	100% 0% 0% 2%
1.3	Racket #2	2.17	27,660	1122	2.17	100% 0% 1% 0%

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	3.53	6,836	261	3.52	0% 0% 1% 100%
1.5	Python 3 #2	3.51	10,344	382	3.50	0% 2% 1% 100%
1.5	PHP #4	2.16	9,856	384	2.16	100% 0% 0% 2%
1.5	Perl #2	3.92	6,784	385	3.92	1% 0% 33% 68%
1.5	PHP #5	2.15	9,884	394	2.15	1% 0% 100% 1%
1.6	Chapel #2	1.62	34,024	423	1.64	99% 3% 1% 4%
1.7	C gcc	1.73	2,116	448	1.73	1% 99% 1% 0%
1.7	Perl	15.87	9,032	452	15.86	1% 100% 1% 1%
1.7	Racket	25.63	130,528	453	25.58	100% 0% 1% 1%
1.8	Lua #7	3.76	3,192	477	3.75	1% 100% 0% 2%
1.8	Ruby #5	3.14	477,097	478	3.12	0% 100% 2% 1%
1.8	Lua #5	3.14	477,097	478	3.12	0% 1% 1% 99%
1.8	Pascal Free Pas	3.14	477,097	478	3.12	0% 100% 1% 1%
1.9	Lisp SBCL #3	3.14	477,097	478	3.12	0% 1% 100% 1%
1.9	PHP #3	3.14	477,097	478	3.12	0% 0% 0% 1%

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



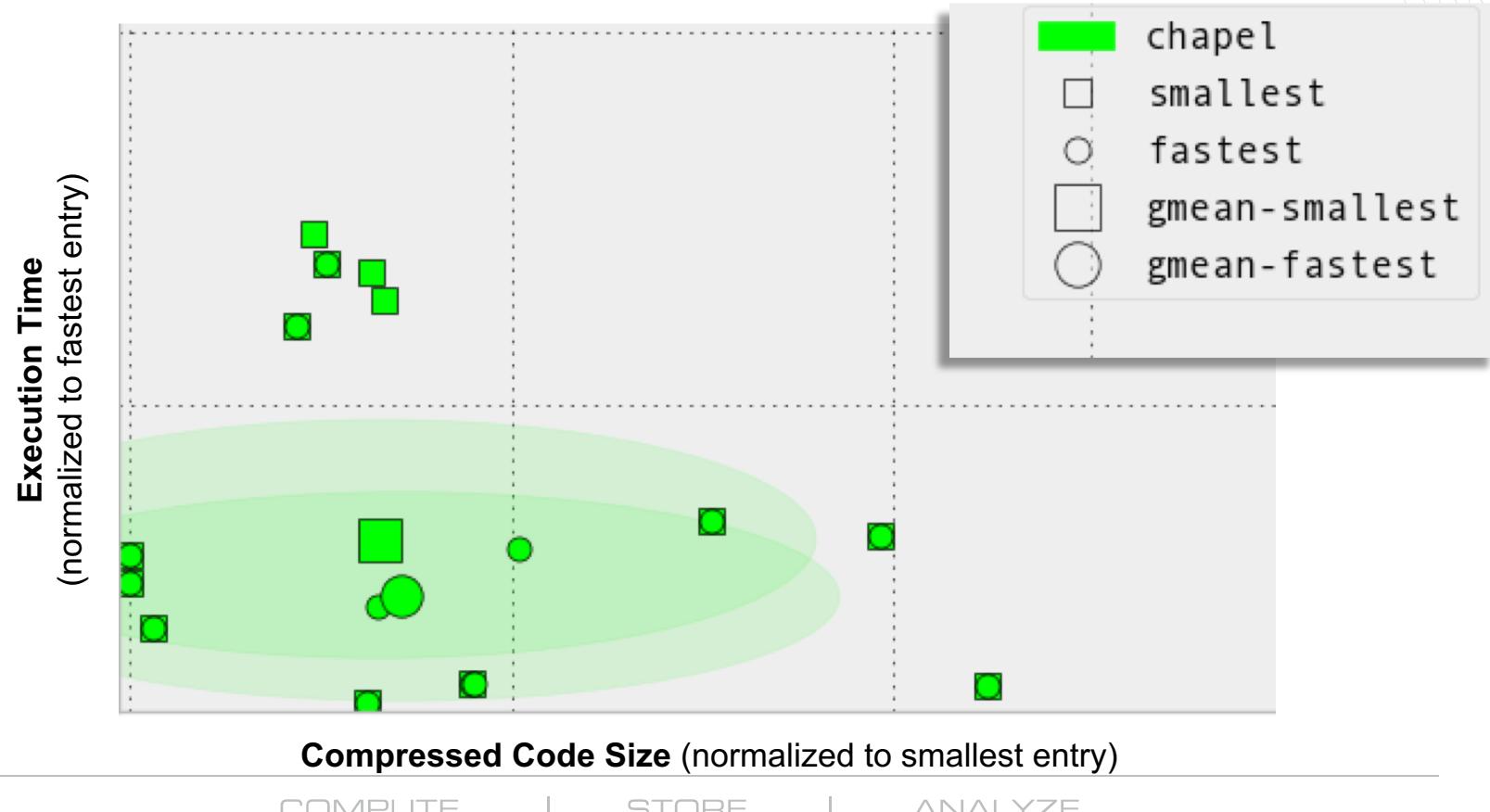
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Plotting normalized CLBG code size x speed



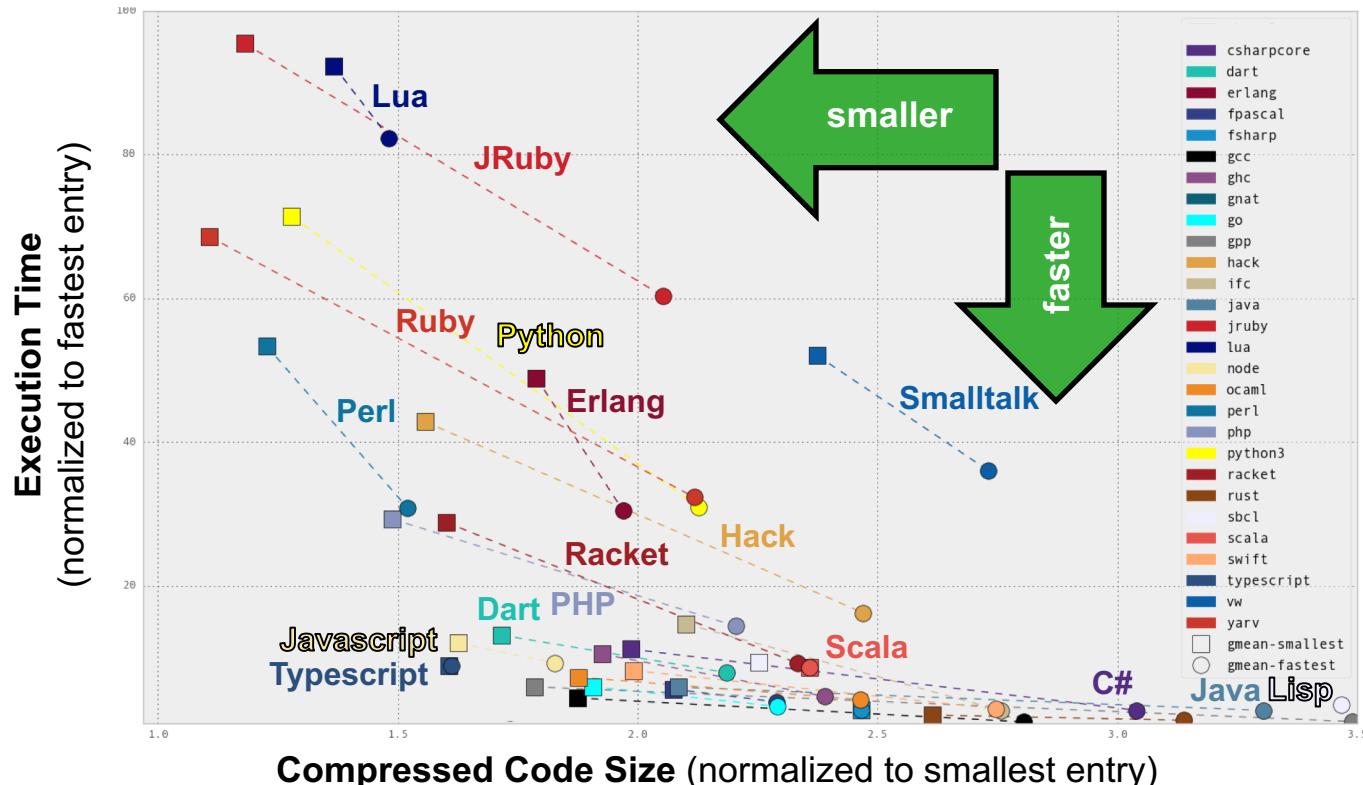
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(June 1, 2018 standings)



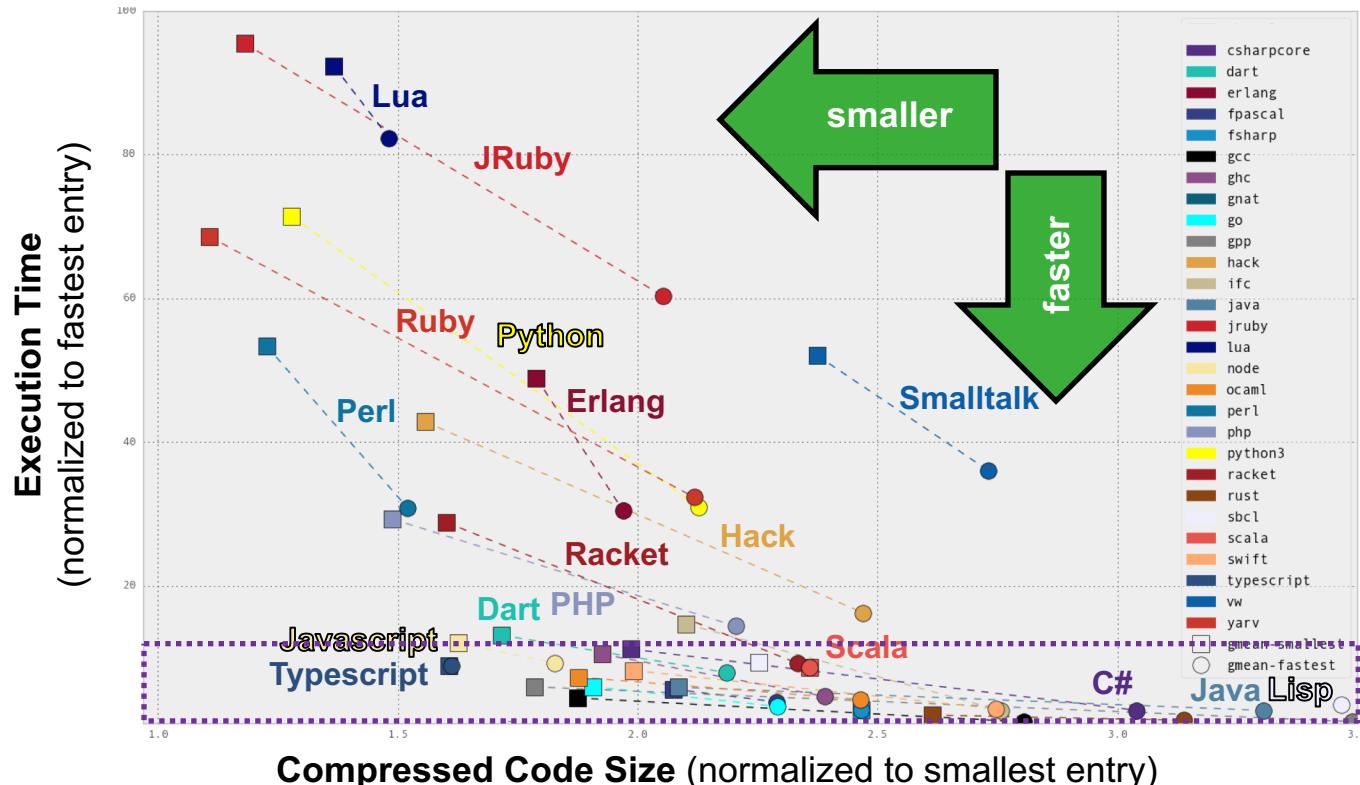
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(June 1, 2018 standings)



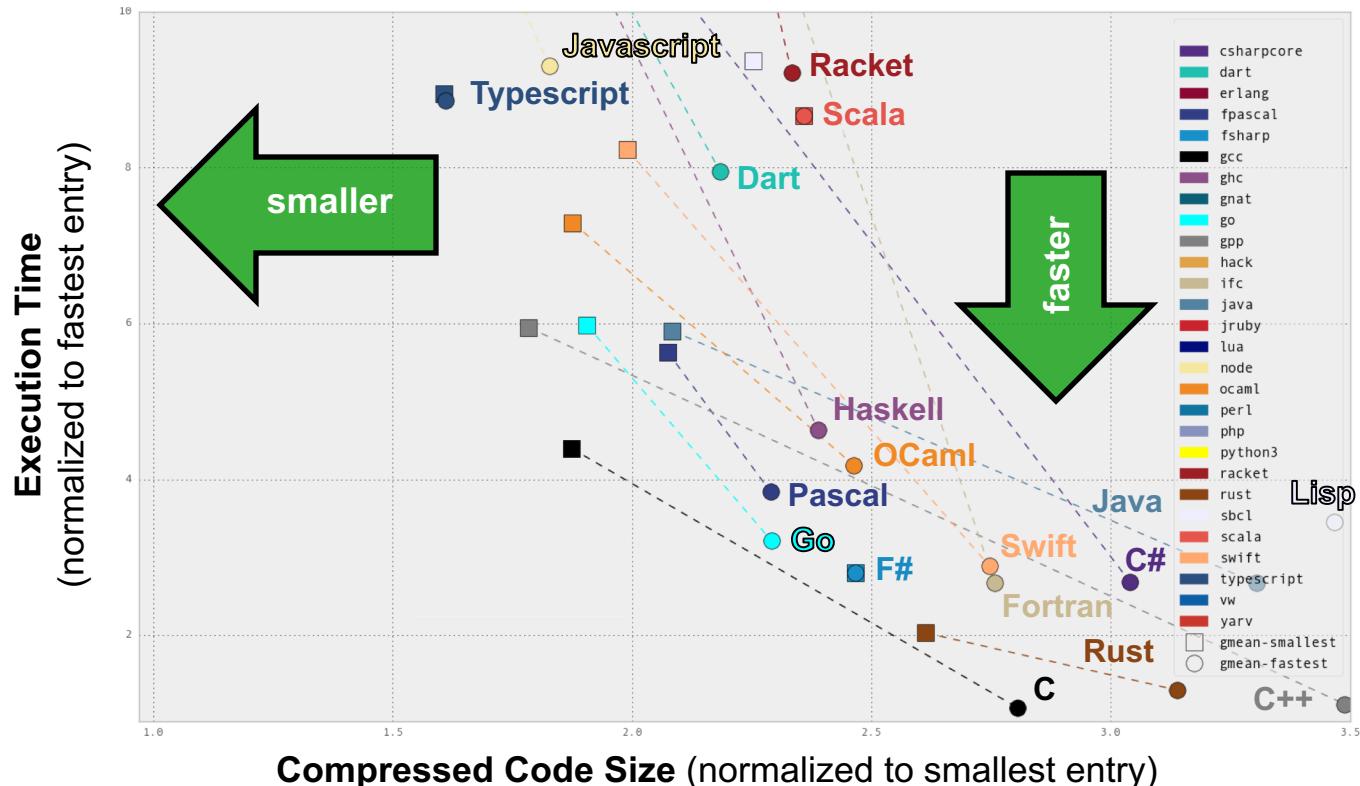
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(June 1, 2018 standings, zoomed in)



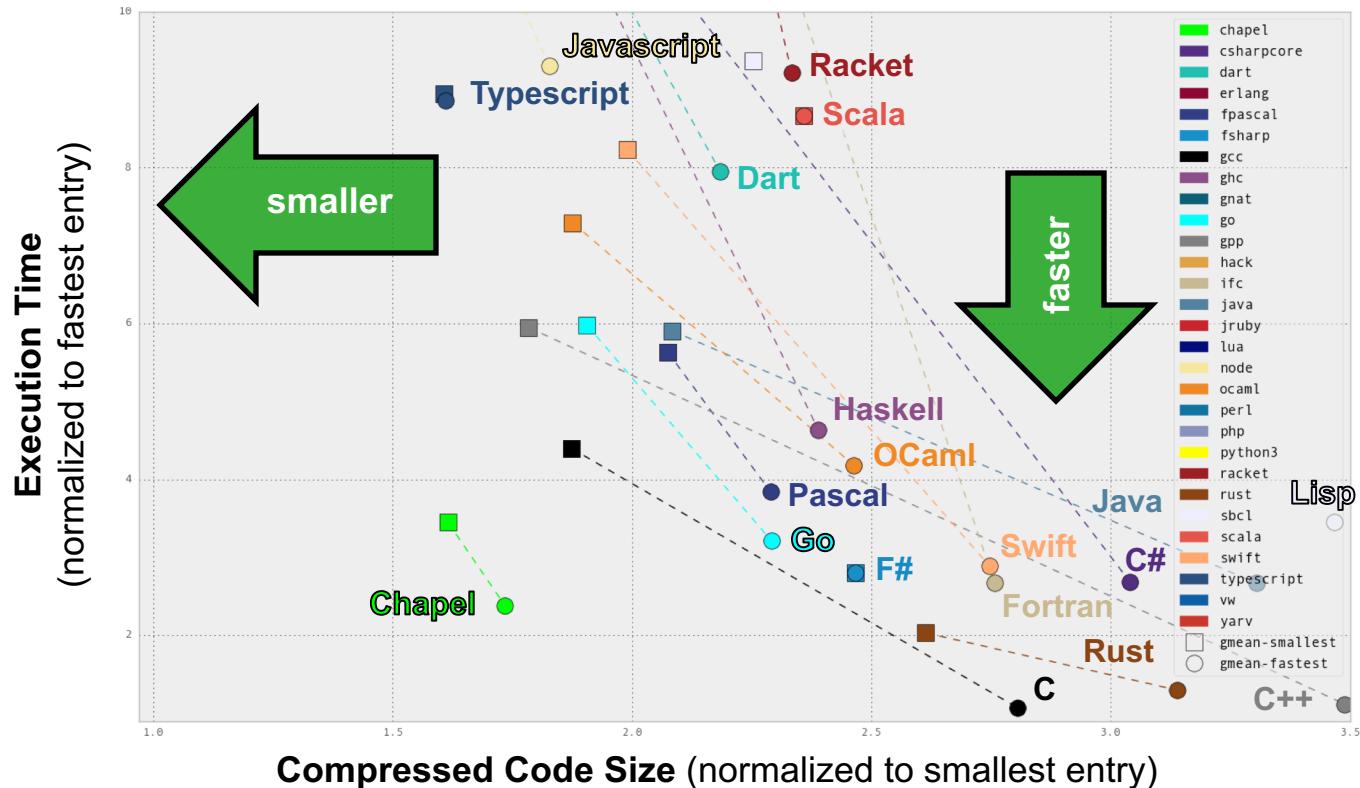
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(June 1, 2018 standings, zoomed in)



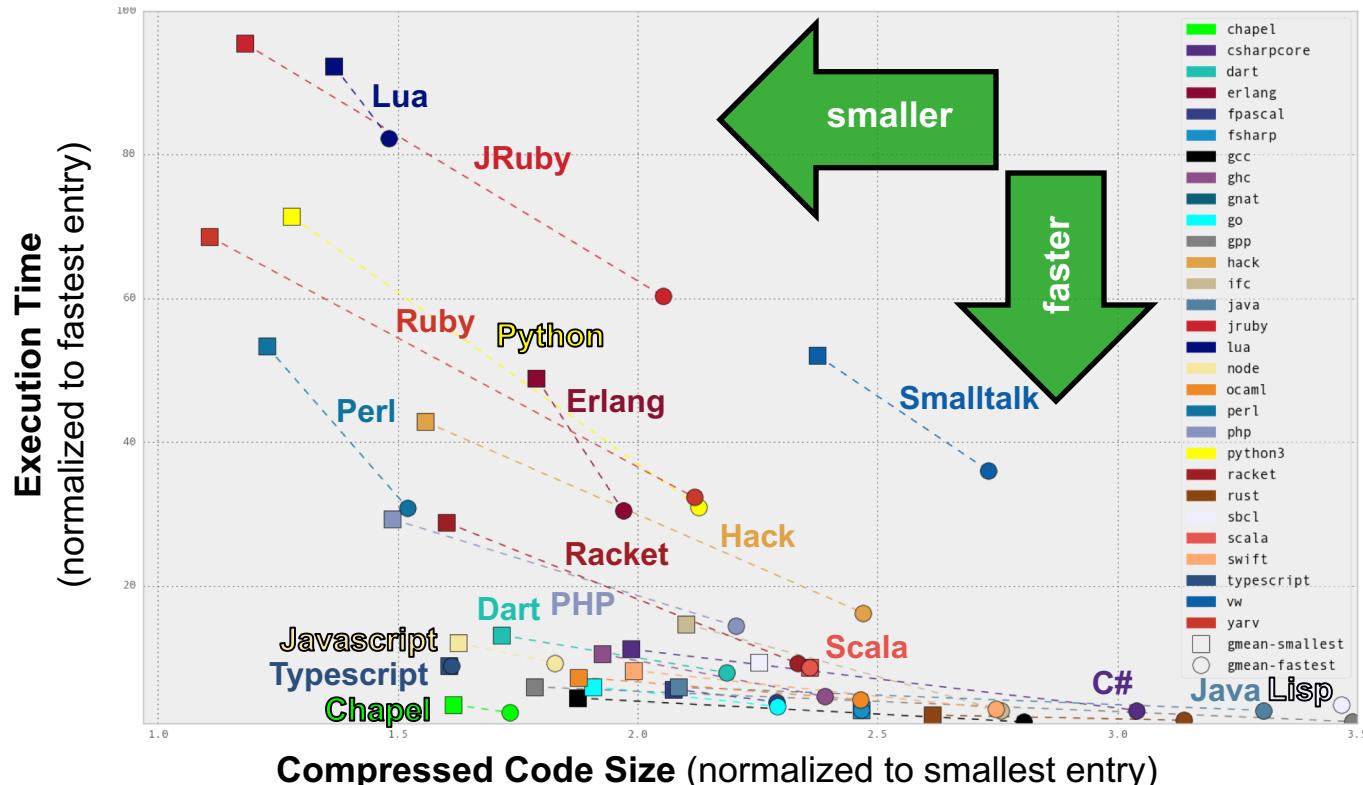
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(June 1, 2018 standings)

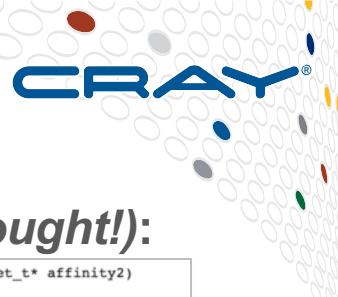


COMPUTE

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

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

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

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

excerpt from 2863.gz C gcc entry

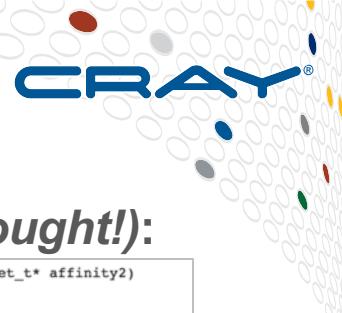


COMPUTE

STORE

ANALYZE

CLBG: Qualitative 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

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

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

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

processor_str      = "processor";
processor_str_len = strlen(processor_str);
physical_id_str   = "physical id";
physical_id_str_len = strlen(physical_id_str);
core_id_str        = "core id";
core_id_str_len   = strlen(core_id_str);
cores              = "cores";
cores_len          = strlen(cores);
cpu_cores_str      = "cpu cores";
cpu_cores_str_len = strlen(cpu_cores_str);
```

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

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

    delete place;
}
```

excerpt from 2863.gz C gcc entry

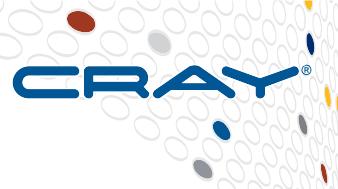


COMPUTE

STORE

ANALYZE

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] = 1;
        CPU_ZERO(affinity1);
    }
}
```

excerpt from 2863.gz C gcc entry



COMPUTE

STORE

ANALYZE

The Chapel Team at Cray (May 2018)



COMPUTE

STORE

ANALYZE

Chapel Community Partners



Lawrence Berkeley
National Laboratory



Yale

(and several others...)

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



COMPUTE

STORE

ANALYZE

Outline



✓ What is Chapel?

➤ Overview of Chapel Features

- Chapel Results and Resources
- Wrap-up



COMPUTE

|

STORE

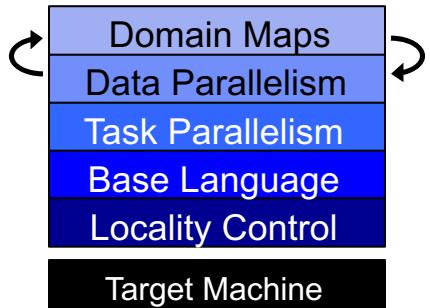
|

ANALYZE

Chapel language feature areas



Chapel language concepts

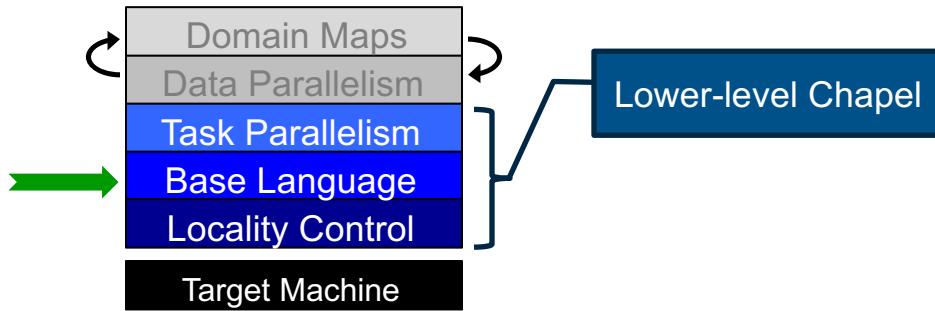


COMPUTE

STORE

ANALYZE

Base Language



COMPUTE

|

STORE

|

ANALYZE

Base Language Features, by example



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

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

```
config const n = 10;

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

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



Base Language Features, by example



```
iter fib(n) {  
    var current = 0,  
        next = 1;  
  
    for i in 1..n {  
        yield current;  
        current += next;  
        current <=> next;  
    }  
}
```

```
config const n = 10;  
  
for f in fib(n) do  
    writeln(f);
```

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

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



Base Language Features, by example



Iterators

```
iter fib(n) {  
    var current = 0,  
        next = 1;  
  
    for i in 1..n {  
        yield current;  
        current += next;  
        current <=gt; next;  
    }  
}
```

```
config const n = 10;  
  
for f in fib(n) do  
    writeln(f);
```

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



Base Language Features, by example



Static type inference for:

- arguments
- return types
- variables

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

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

```
config const n = 10;

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

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



Base Language Features, by example



Explicit types also supported

```
iter fib(n: int): int {  
    var current: int = 0,  
        next: int = 1;  
  
    for i in 1..n {  
        yield current;  
        current += next;  
        current <=> next;  
    }  
}
```

```
config const n: int = 10;  
  
for f in fib(n) do  
    writeln(f);
```

0
1
1
2
3
5
8
...



Base Language Features, by example



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

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

```
config const n = 10;

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

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



Base Language Features, by example



```
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);
```

Zippered iteration

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



Base Language Features, by example



Range types and operators

```
iter fib(n) {  
    var current = 0,  
        next = 1;  
  
    for i in 1..n {  
        yield current;  
        current += next;  
        current <=> next;  
    }  
}
```

```
config const n = 10;  
  
for (i,f) in zip(0..#n, fib(n)) do  
writeln("fib #", i, " is ", f);
```

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



COMPUTE

STORE

ANALYZE

Base Language Features, by example



Tuples

```
iter fib(n) {  
    var current = 0,  
        next = 1;  
  
    for i in 1..n {  
        yield current;  
        current += next;  
        current <=> next;  
    }  
}
```

```
config const n = 10;  
  
for (i,f) in zip(0..#n, fib(n)) do  
writeln("fib #", i, " is ", f);
```

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



Base Language Features, by example



```
iter fib(n) {  
    var current = 0,  
        next = 1;  
  
    for i in 1..n {  
        yield current;  
        current += next;  
        current <=> next;  
    }  
}
```

```
config const n = 10;  
  
for (i,f) in zip(0..#n, fib(n)) do  
writeln("fib #", i, " is ", f);
```

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



Other 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

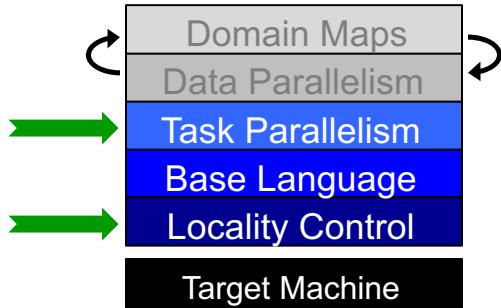
|

STORE

|

ANALYZE

Task Parallelism and Locality Control



COMPUTE

|

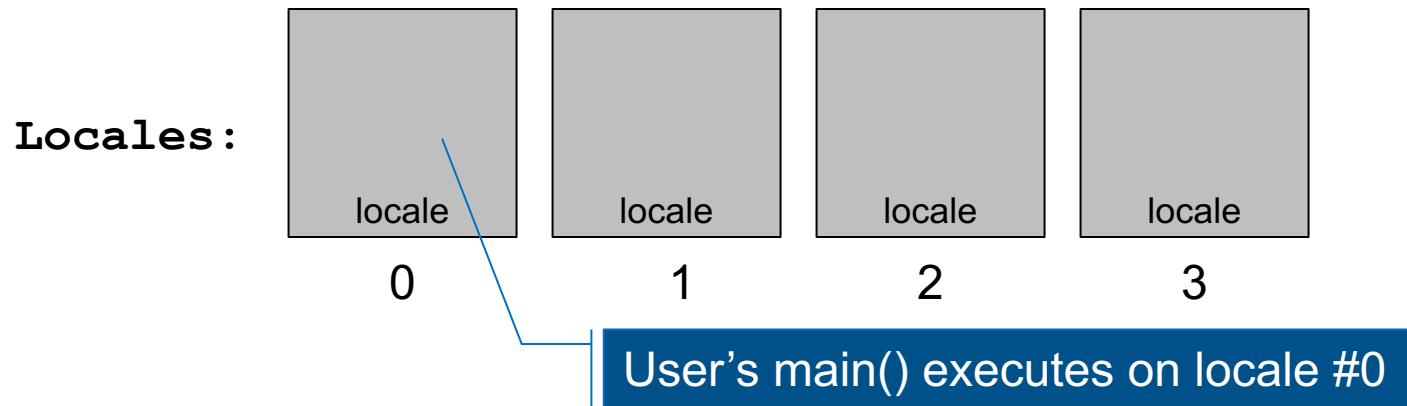
STORE

|

ANALYZE

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
```



COMPUTE

STORE

ANALYZE

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

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

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

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
```

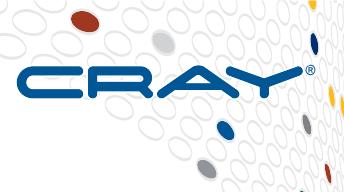


COMPUTE

STORE

ANALYZE

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

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

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

Other Key Task Parallel Features



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



COMPUTE

|

STORE

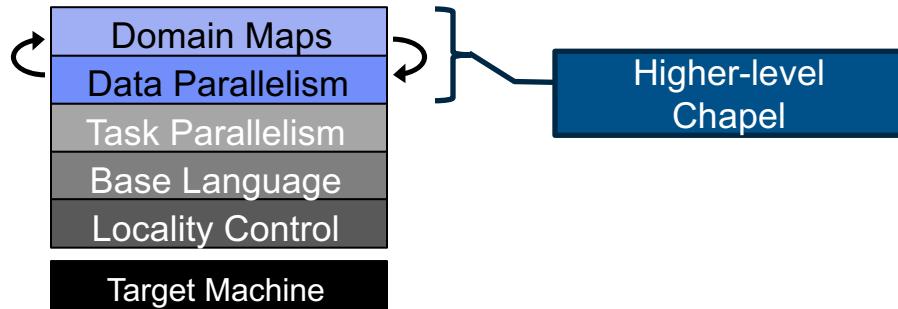
|

ANALYZE

Data Parallelism in Chapel



Chapel language concepts



COMPUTE

|

STORE

|

ANALYZE

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

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

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

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

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
```

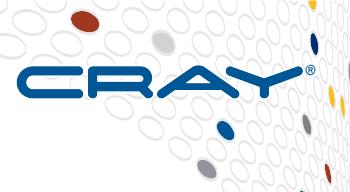


COMPUTE

STORE

ANALYZE

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
```

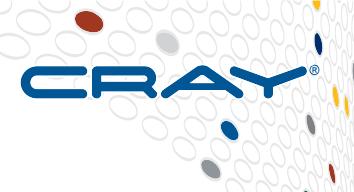


COMPUTE

STORE

ANALYZE

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
```

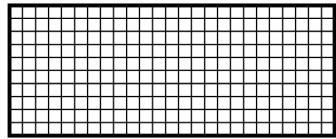


COMPUTE

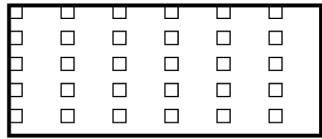
STORE

ANALYZE

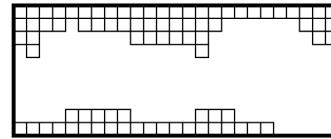
Chapel Has Several Domain / Array Types



dense



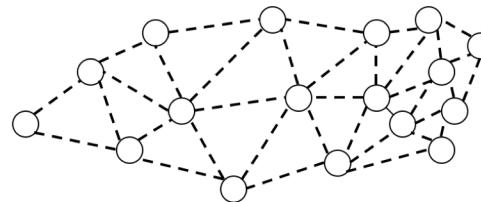
strided



sparse



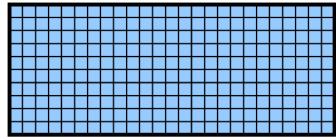
associative



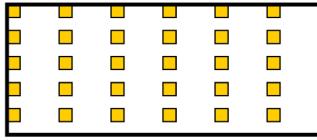
unstructured



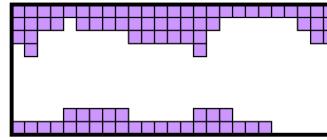
Chapel Has Several Domain / Array Types



dense



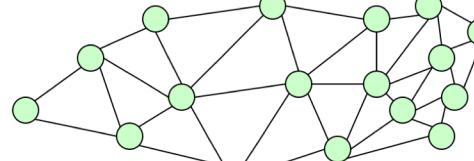
strided



sparse



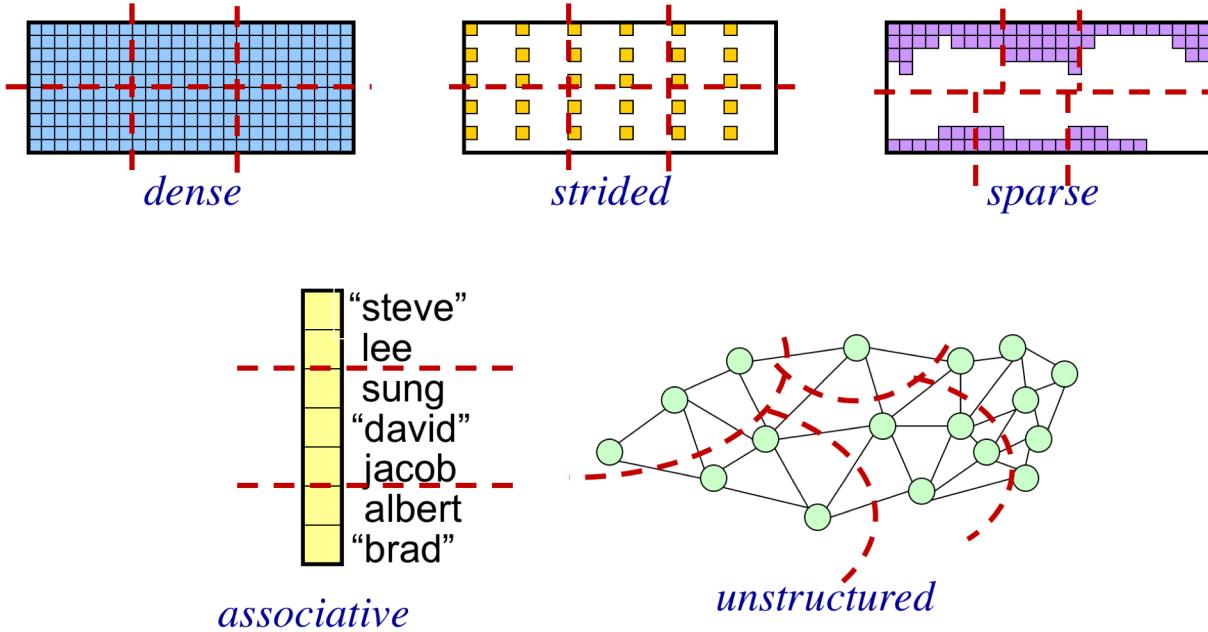
associative



unstructured



Chapel Has Several Domain / Array Types



COMPUTE

STORE

ANALYZE

Other Key Data Parallel Features



- **Promotion:** call scalar arrays with array arguments

```
var B = sin(A); // results in parallel evaluation
```

- **Slicing:** refer to subarrays using range / domain arguments

...A[lo..hi, ...]

...OceanTemp[Coastal] ...

- **Reductions:** collapse arrays to scalars or subarrays

```
const hottest = max reduce OceanTemp;
```



Chapel Results and Resources



COMPUTE

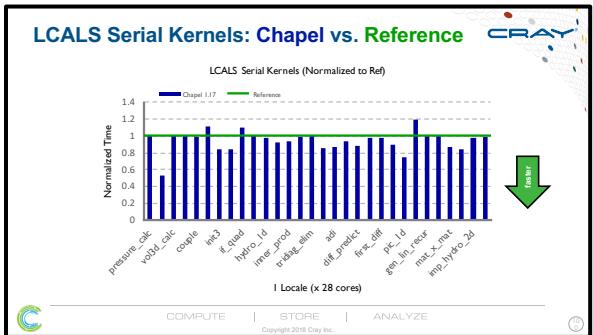
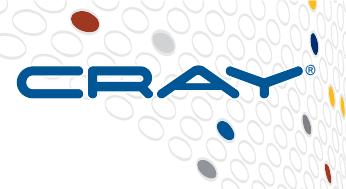
|

STORE

|

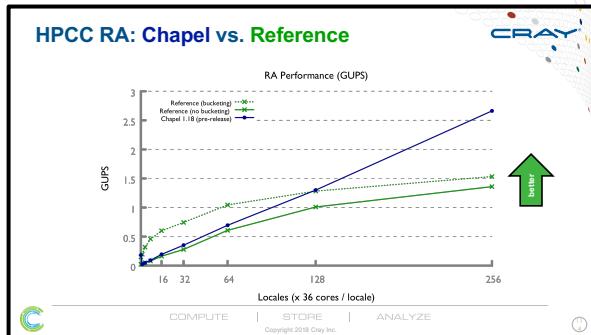
ANALYZE

HPC Patterns: Chapel vs. Reference



LCALS

HPCC RA

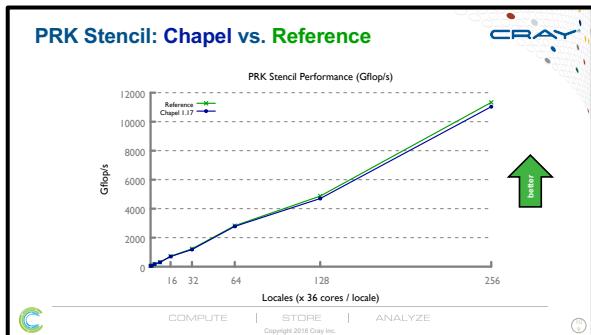
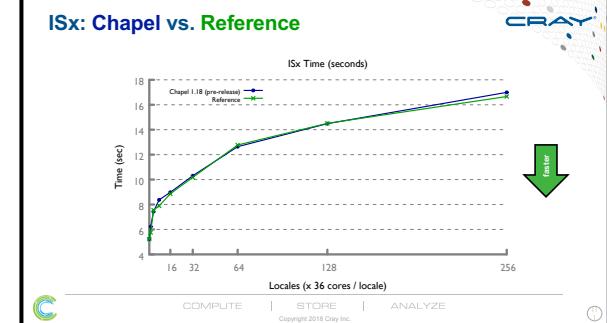
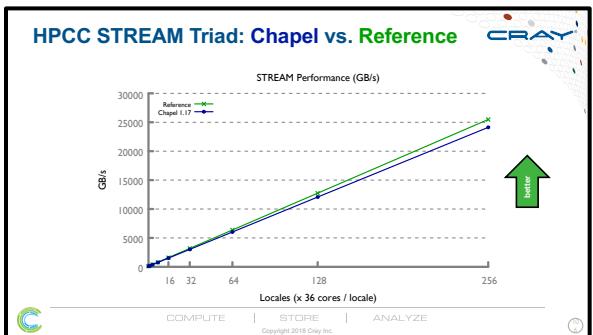


STREAM

Triad

PRK

Stencil



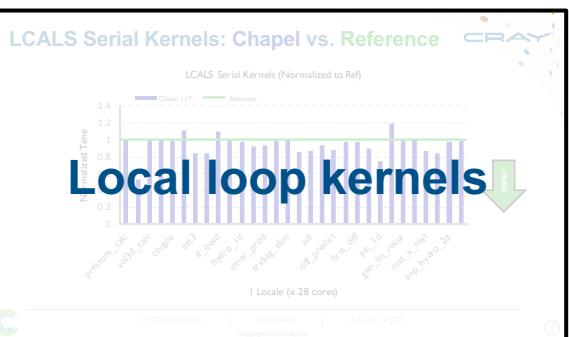
Nightly performance tickers online at:
<https://chapel-lang.org/perf-nightly.html>

COMPUTE

STORE

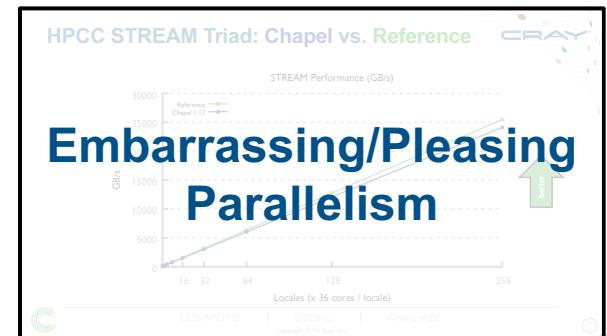
ANALY

HPC Patterns: Chapel vs. Reference



LCALS

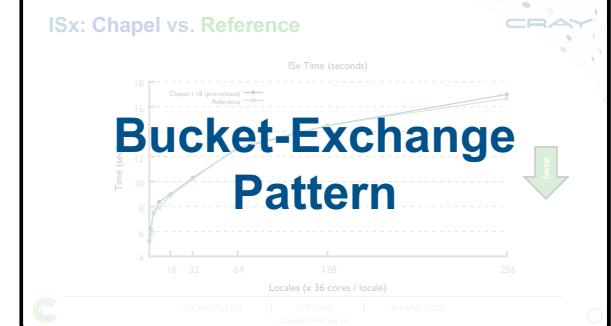
HPCC RA



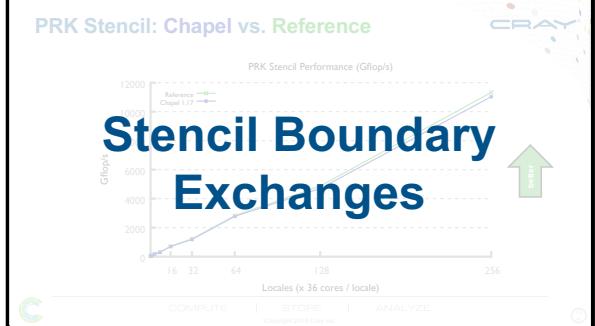
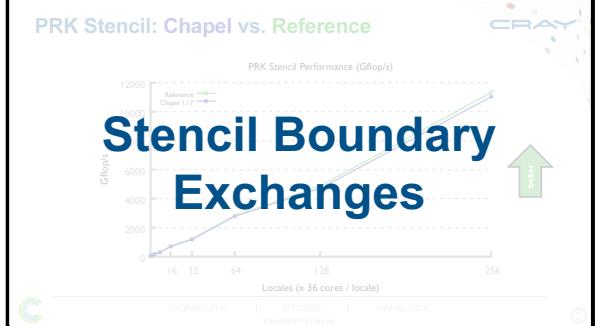
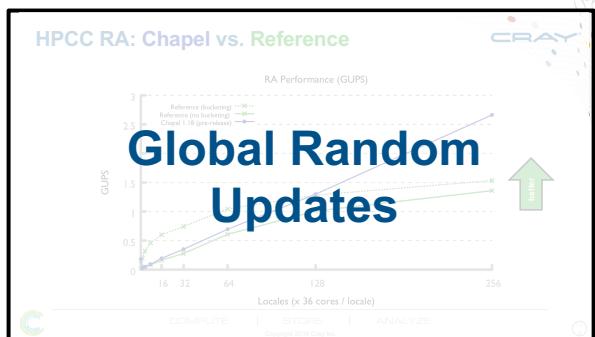
STREAM
Triad

ISx

PRK
Stencil



ISx



COMPUTE

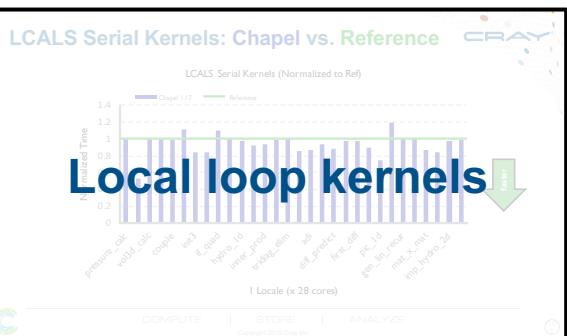
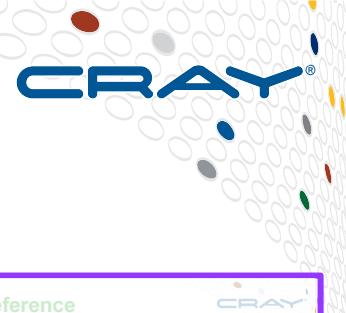
STORE

ANALYZE

Copyright 2018 Cray Inc.

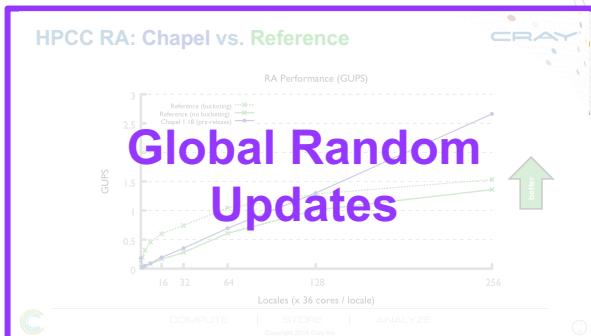
Nightly performance tickers online at:
<https://chapel-lang.org/perf-nightly.html>

HPC Patterns: Chapel vs. Reference



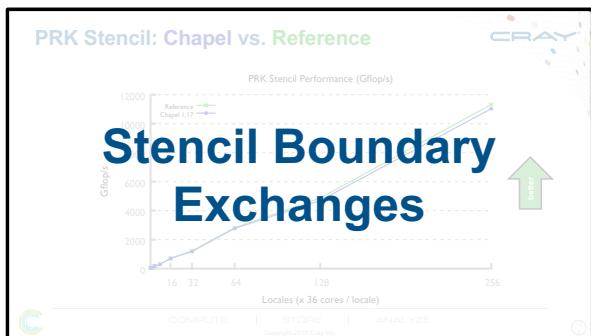
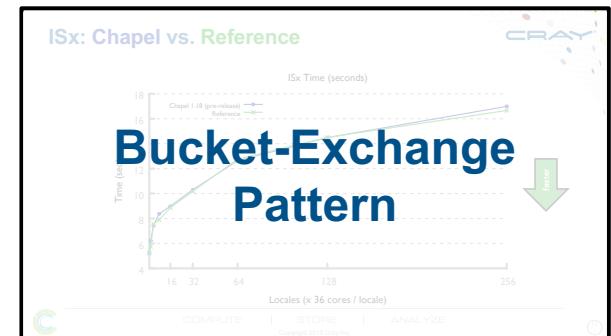
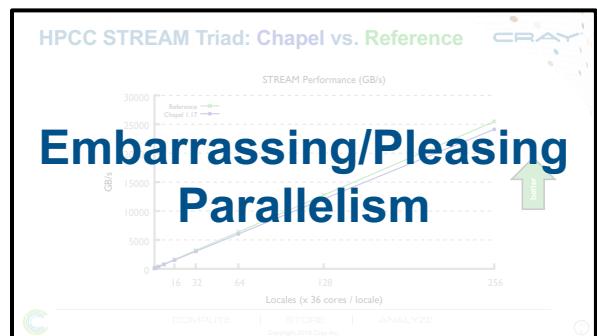
LCALS

HPCC RA



STREAM
Triad

PRK
Stencil



COMPUTE

STORE

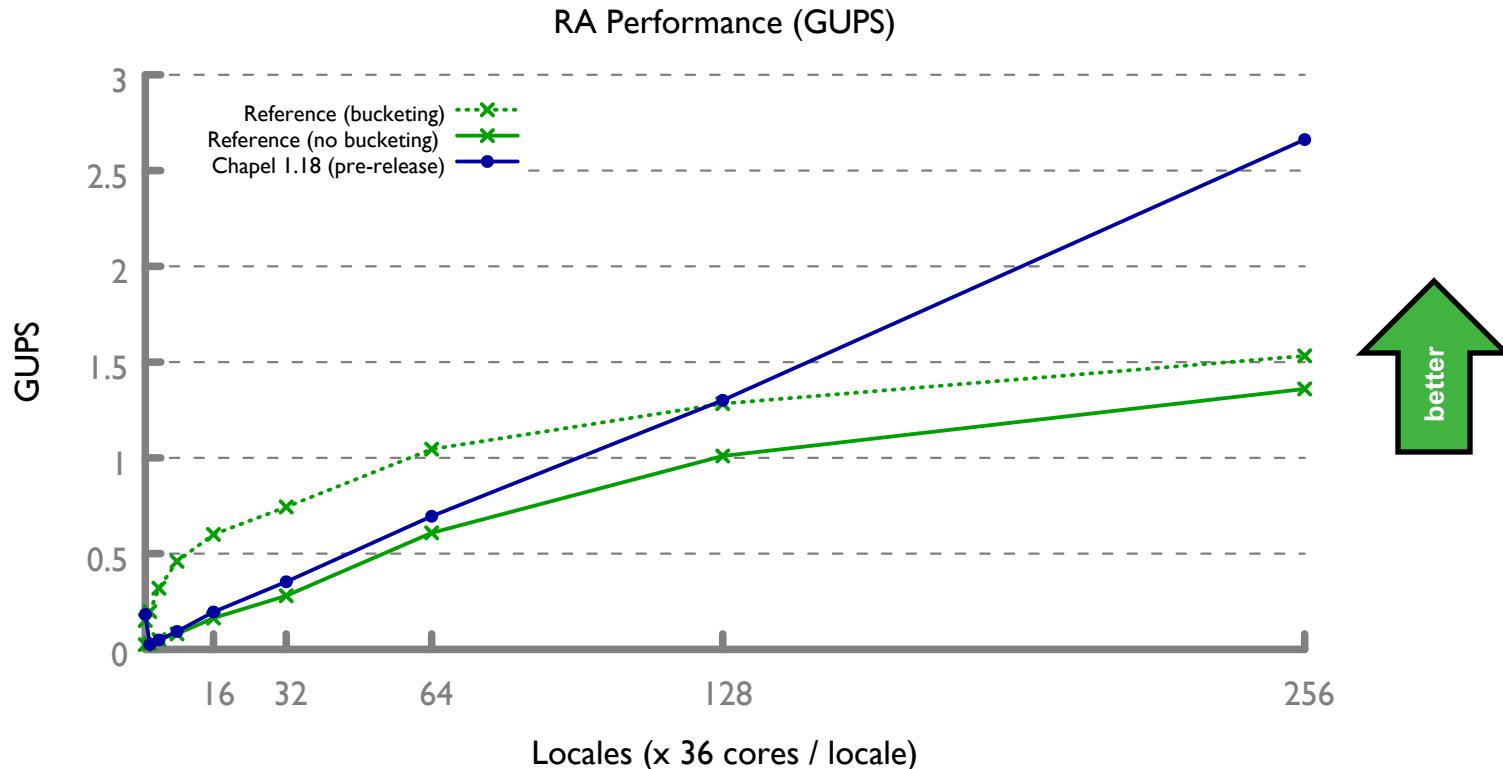
ANALYZE

Copyright 2018 Cray Inc.

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



HPCC RA: Chapel vs. Reference

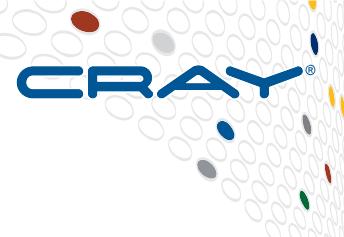


COMPUTE

STORE

ANALYZE

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++;
        }
    }
    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

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,
          MPI_IProbe(MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
          MPI_TEST_RECV, &status);
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;
```

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 & (TABSIZE-1)] ^= Ran;
*      }
*/
```

```
    /* our done messages */
    proc_count = 0 ; proc_count < tparams.NumProcs ; ++proc_count)
    if (proc_count == tparams.MyProc) { tparams.finish_req(tparams.MyProc) =
        MPI_REQUEST_NULL; continue; }
    /* end garbage - who cares, no one will look at it */
    Isend(&Ran, 0, tparams.dtype64, proc_count, FINISHED_TAG,
```

```
    localBufferSize, &status);
```

```
    statuses;
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Libraries



~60 library modules

- web-documented, many user-contributed

The screenshot shows the "Standard Modules" page of the Chapel Documentation. The top navigation bar includes links for "Docs", "Standard Modules", and "View page source". A search bar is also present. The main content area is titled "Standard Modules" and contains a brief description of what standard modules are. It lists several standard 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.

The screenshot shows the "Package Modules" page of the Chapel Documentation. The top navigation bar includes links for "Docs", "Package Modules", and "View page source". A search bar is also present. The main content area is titled "Package Modules" and contains a brief description of what package modules are. It lists several package modules: AllLocalesBarriers, BLAS, Buffers, Collection, Crypto, Curl, DistributedBag, DistributedDeque, DistributedIterators, FFTW, FFTW_MT, Futures, HDFS, HDFSSIterator, LAPACK, LinearAlgebra, MPI, Norm, OwnedObject, RangeChunk, RecordParser, ReplicatedVar, Search, SharedObject, Sort, VisualDebug, and ZMQ.



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

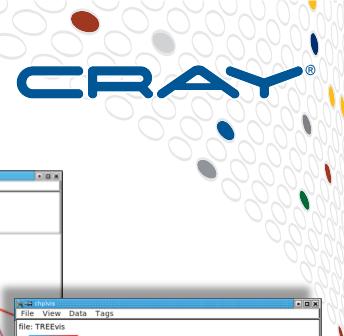
Distributed Computing: DistributedIterators, DistributedBag,
DistributedDeque, Block, Cyclic, Block-Cyclic, ...

File Systems: FileSystem, Path, HDFS

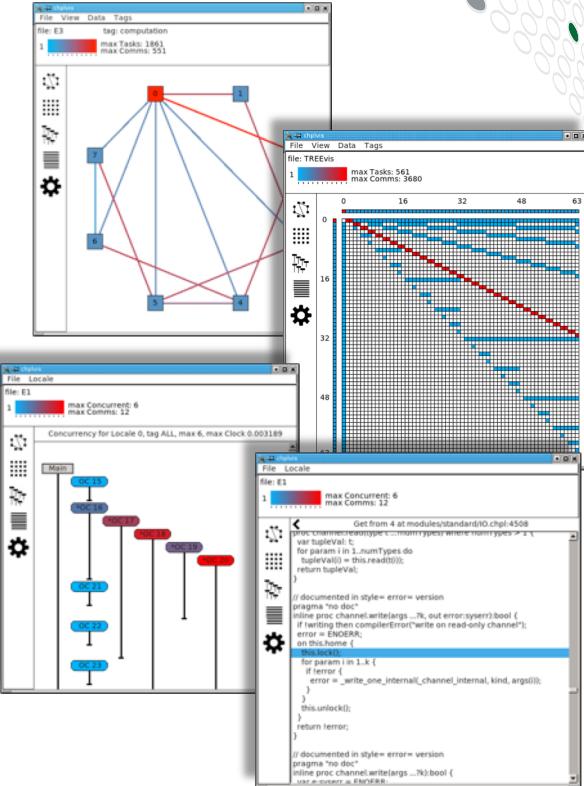
Others: BigInteger, BitOps, Crypto, Curl, DateTime, Random,
Reflection, Regexp, Search, Sort, Spawn, ...



Chapel Tools



- **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



COMPUTE

STORE

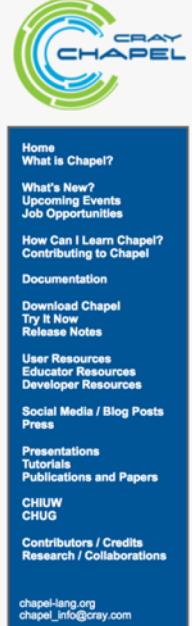
ANALYZE

Chapel Central



<https://chapel-lang.org>

- downloads
- presentations
- papers
- resources
- documentation



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} mapped 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?](#)

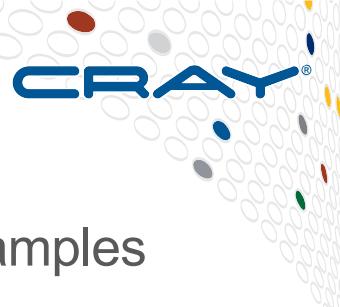


COMPUTE

STORE

ANALYZE

Chapel Online Documentation



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

The screenshot displays the Chapel Online Documentation website with a dark theme. The main navigation bar includes links for "Docs", "Chapel Documentation", "version 1.17", "View page source", and a search bar labeled "Search docs". The left sidebar contains a navigation tree with sections like "COMPILING AND RUNNING CHAPEL", "WRITING CHAPEL PROGRAMS", and "LANGUAGE HISTORY". The main content area shows the "Chapel Documentation" page, which includes sections for "Compiling and Running Chapel", "Writing Chapel Programs", "Language History", and "Using Chapel". Each section has its own sub-navigation bar and content. A separate panel on the right shows examples of "Task Parallelism", "Begin Statements", and "Cobegin Statements" with code snippets and explanations.

Chapel Documentation

version 1.17 ▾

Docs » Chapel Documentation

View page source

Search docs

COMPILING AND RUNNING CHAPEL

- Quickstart Instructions
- Using Chapel
- Platform-Specific Notes
- Technical Notes
- Tools

WRITING CHAPEL PROGRAMS

- Quick Reference
- Hello World Variants
- Primers
- Language Specification
- Built-in Types and Functions
- Standard Modules
- Package Modules
- Standard Layouts and Distributions
- Chapel Users Guide (WIP)

LANGUAGE HISTORY

- Chapel Evolution
- Archived Language Specifications

Chapel Documentation

version 1.17 ▾

Docs » Using Chapel

View page source

Search docs

COMPILING AND RUNNING CHAPEL

- Quickstart Instructions
- Using Chapel
- Platform-Specific Notes
- Technical Notes
- Tools

WRITING CHAPEL PROGRAMS

- Quick Reference
- Hello World Variants
- Primers
- Language Specification
- Built-in Types and Functions
- Standard Modules
- Package Modules
- Standard Layouts and Distributions
- Chapel Users Guide (WIP)

LANGUAGE HISTORY

- Chapel Evolution
- Archived Language Specifications

Using Chapel

Contents:

- 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
- Reporting Chapel Issues

Platform-Specific Notes

Technical Notes

© Copyright 2018, Cray Inc.

Task Parallelism

View taskParallel.chpl on GitHub

This primer illustrates Chapel's parallel tasking features, namely the `begin`, `cobegin`, and `cotask` statements.

```
config const n = 10; // used for the coforall loop
```

Begin Statements

The `begin` statement spawns a thread of execution that is independent of the current (main) thread of execution.

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

The main thread of execution continues on to the next statement. There is no guarantee as to which statement will execute first.

```
writeln("1: output from main task");
```

Cobegin Statements



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel for Python Programmers

Developed by Simon Lund



The screenshot shows a documentation page for "Chapel for Python Programmers". The left sidebar contains a navigation menu with links to "Getting Started", "Language Basics", "Parallelism", "NumPy", "Batteries", "Keywords", "Pythonic Module", "Python and Chapel", "Miscellaneous Notes", and "If Chapel had a band". The main content area has a header "Chapel for Python Programmers" and a subtitle "Subtitle: How I Learned to Stop Worrying and Love the Curlybracket.". The text discusses the benefits of Chapel over Python, mentioning packages like Bohrium, Cython, distarray, mpi4py, threading, multiprocessing, NumPy, Numba, and NumExpr. It also notes the potential performance trade-offs when implementing low-level methods in C/C++ and binding them to Python.

Docs » Chapel for Python Programmers [Edit on GitHub](#)

Chapel for Python Programmers

Subtitle: How I Learned to Stop Worrying and Love the Curlybracket.

So, what is Chapel and why should you care? We all know that Python is the best thing since sliced bread. Python comes with batteries included and there is nothing that can't be expressed with Python in a short, concise, elegant, and easily readable manner. But, if you find yourself using any of these packages - [Bohrium](#), [Cython](#), [distarray](#), [mpi4py](#), [threading](#), [multiprocessing](#), [NumPy](#), [Numba](#), and/or [NumExpr](#) - you might have done so because you felt that Python's batteries needed a recharge.

You might also have started venturing deeper into the world of curlybrackets. Implementing low-level methods in C/C++ and binding them to Python. In the process you might have felt that you gained performance but lost your

<https://chapel-for-python-programmers.readthedocs.io/>



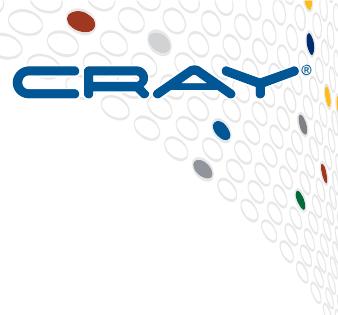
COMPUTE

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/>



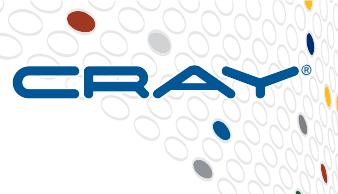
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Community



Questions Developer Jobs Tags Users [chapel]

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

6 votes 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 by Tahmangia 385 1 10

6 votes 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 angular asked Apr 23 at 23:15 by angular 33 3

47 views

6 votes Is there any `writeln()` format specifier for a bool?
I looked at the `writeln()` 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 writeln() ...`
chapel asked Nov 11 '17 at 22:21 by casseler

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

This repository Search Pull requests Issues Marketplace Gist

chapel-lang / chapel

Code Issues 292 Pull requests 26 Projects 0 Settings Insights

Filters Is:issue is:open Labels Milestones

292 Open 77 Closed

Implement "bounded-coforall" optimization for remote coforalls area: Compiler type: Performance
#6357 opened 13 hours ago by ronawho

Consider using processor atomics for remote coforalls EndCount area: Compiler type: Performance
#6356 opened 13 hours ago by ronawho 0 of 6

make uninstall area: BTR type: Feature Request
#6353 opened 14 hours ago by mpf

make check doesn't work with ./configure area: BTR
#6352 opened 16 hours ago by mpf

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 casseler

Remove chpl_comm_make_progress area: Runtime easy type: Design
#6349 opened a day ago by sunyunchai

Runtime error after make on Linux Mint area: BTR user issue
#6348 opened a day ago by denindiana

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

GITTER

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

Where communities thrive

FREE FOR COMMUNITIES

JOIN OVER 88K PEOPLE JOIN OVER 88K COMMUNITIES CREATE YOUR OWN COMMUNITY EXPLORE MORE COMMUNITIES

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
```

Brian Dolan @buddha314 oh, got it, thanks! May 09 14:41

Michael Ferguson @mpff May 09 14:42

```
proc f(xr) { /* xr refers to the actual argument */ }
proc g(ln xr) { /* xr is a copy of the actual argument */ }
var A[1..10] int;
f(A);
g(A);
```

Brian Dolan @buddha314 isn't there a proc f(ref arr) {} as well? May 09 14:43

Michael Ferguson @mpff May 09 14:55

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.

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

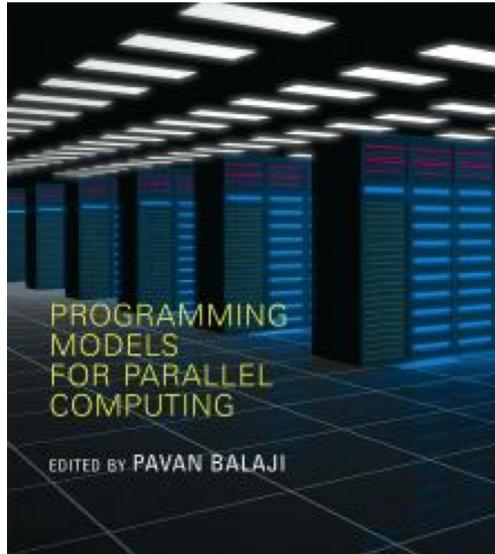
ANALYZE

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

Wrap-up



COMPUTE

|

STORE

|

ANALYZE



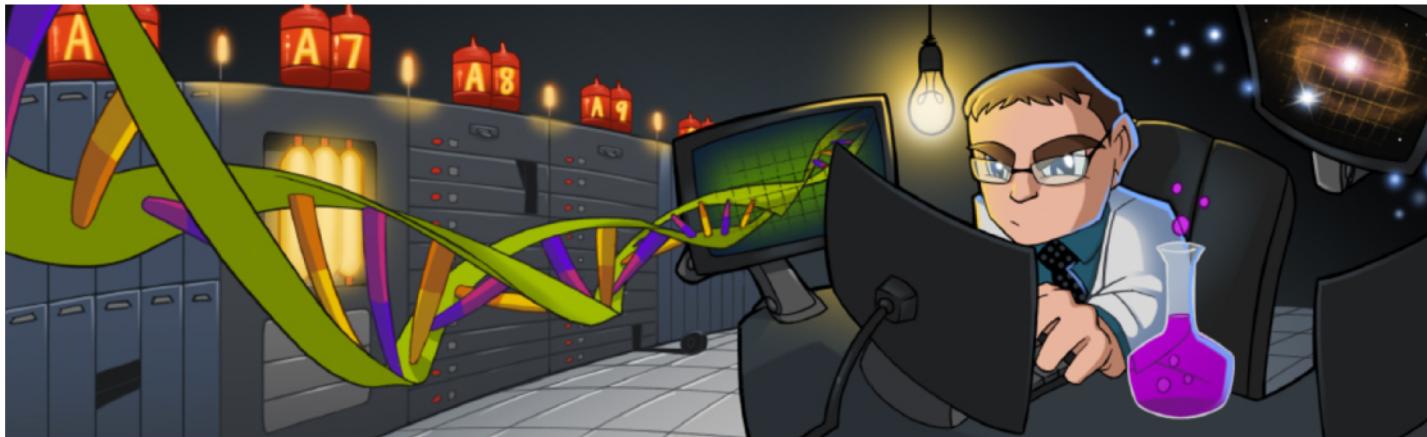
The Chapel language offers a unique combination of productivity, performance, and parallelism

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



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*



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

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.



CRAY

