



Chapel Comes of Age: Productive Parallelism at Scale

Brad Chamberlain, Chapel Team, Cray Inc.

PNW PLSE Workshop

May 14, 2018



COMPUTE

STORE

ANALYZE



Chapel: Niche or Quiche?

Brad Chamberlain, Chapel Team, Cray Inc.

PNW PLSE Workshop

May 14, 2018

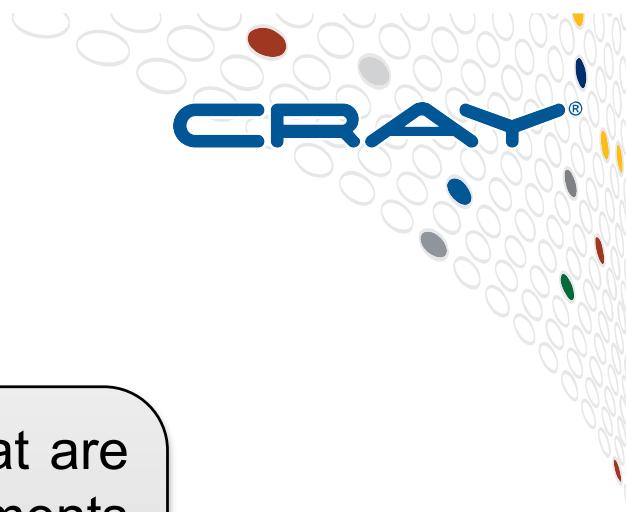


COMPUTE

STORE

ANALYZE

Safe Harbor Statement



This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.



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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel and Productivity



Chapel aims to be as...

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



COMPUTE

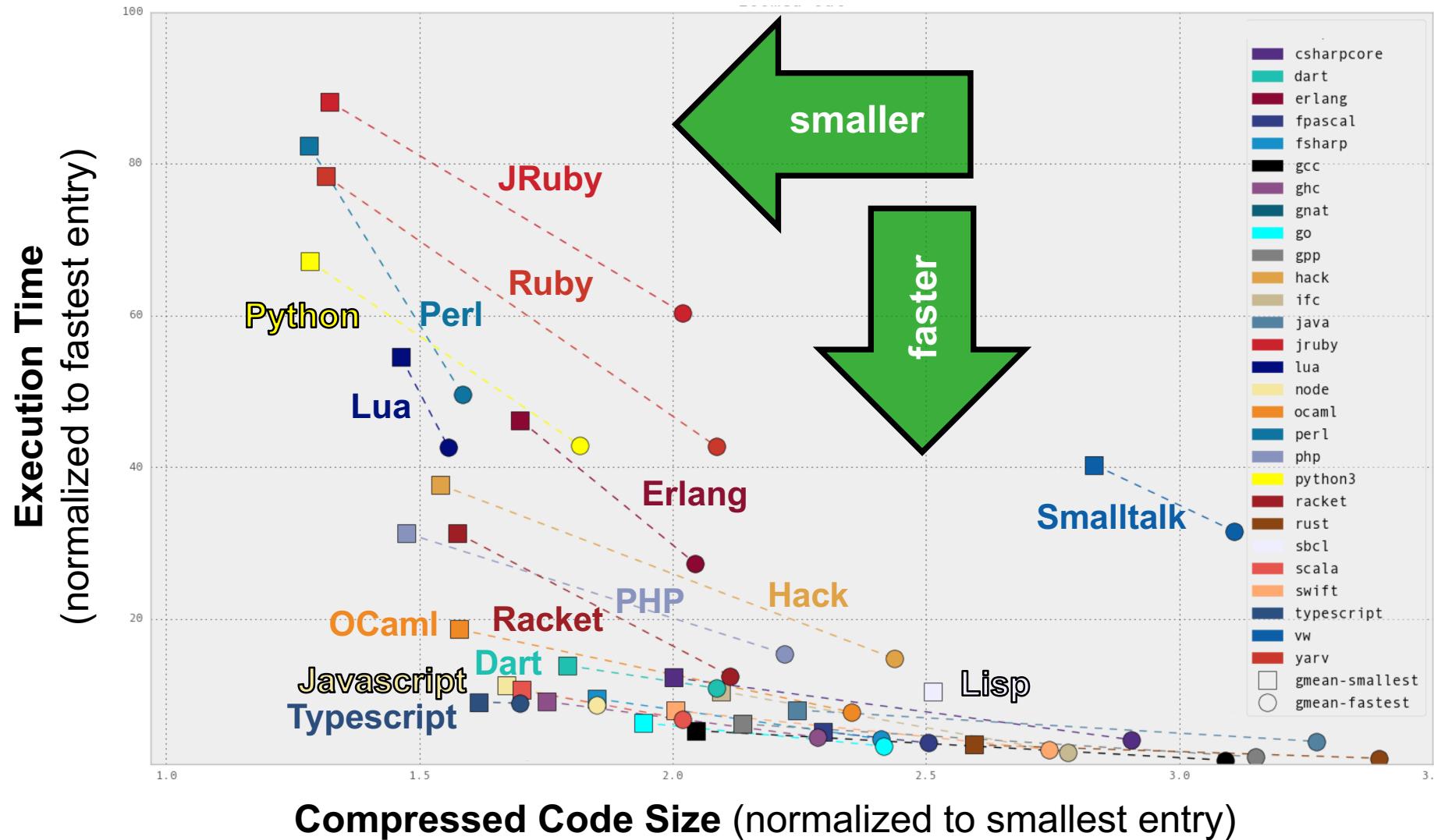
STORE

ANALYZE

Copyright 2018 Cray Inc.

CLBG Cross-Language Summary

(Oct 2017 standings)



COMPUTE

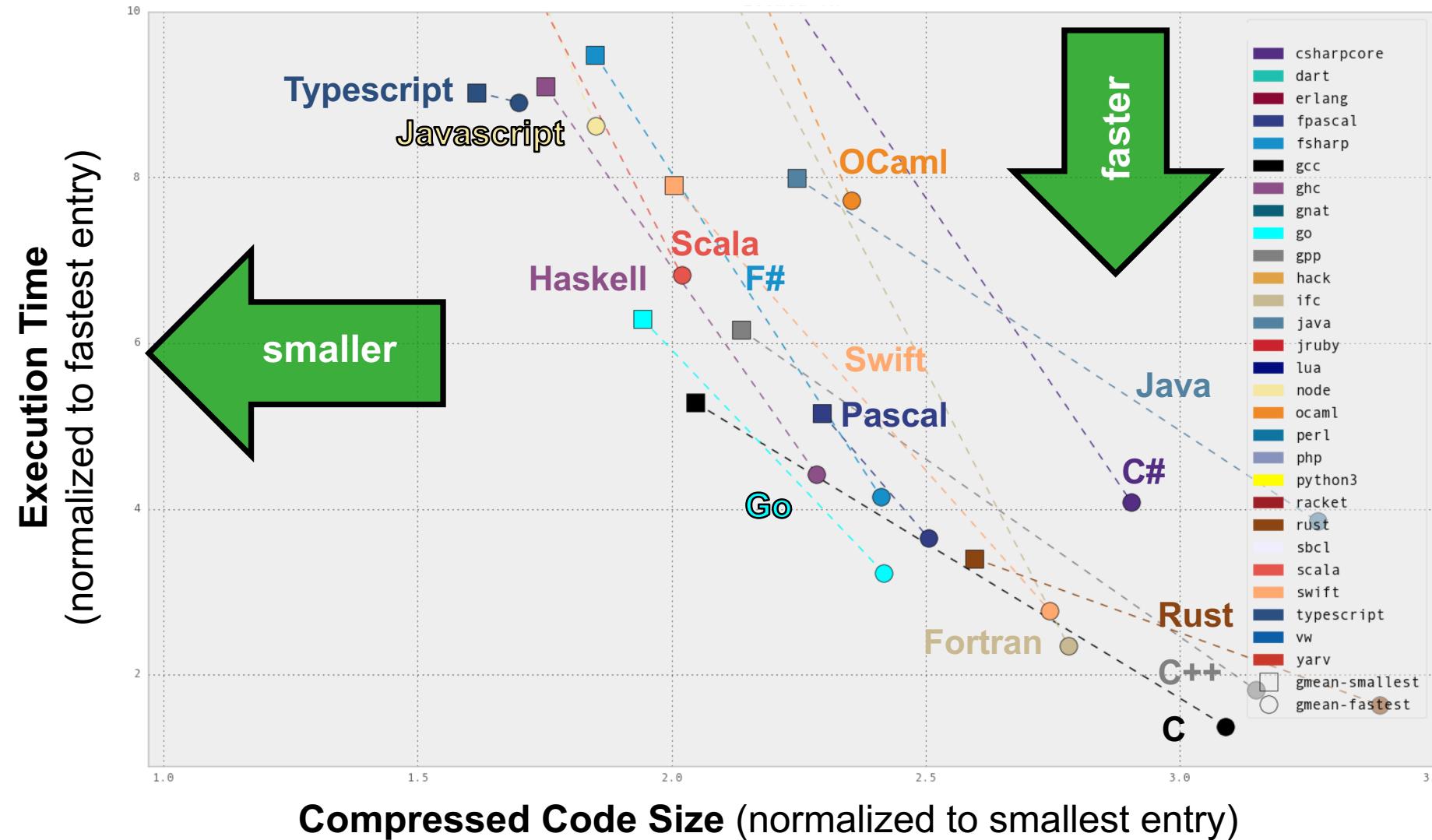
STORE

ANALYZE

Copyright 2018 Cray Inc.

CLBG Cross-Language Summary

(Oct 2017 standings, zoomed in)



COMPUTE

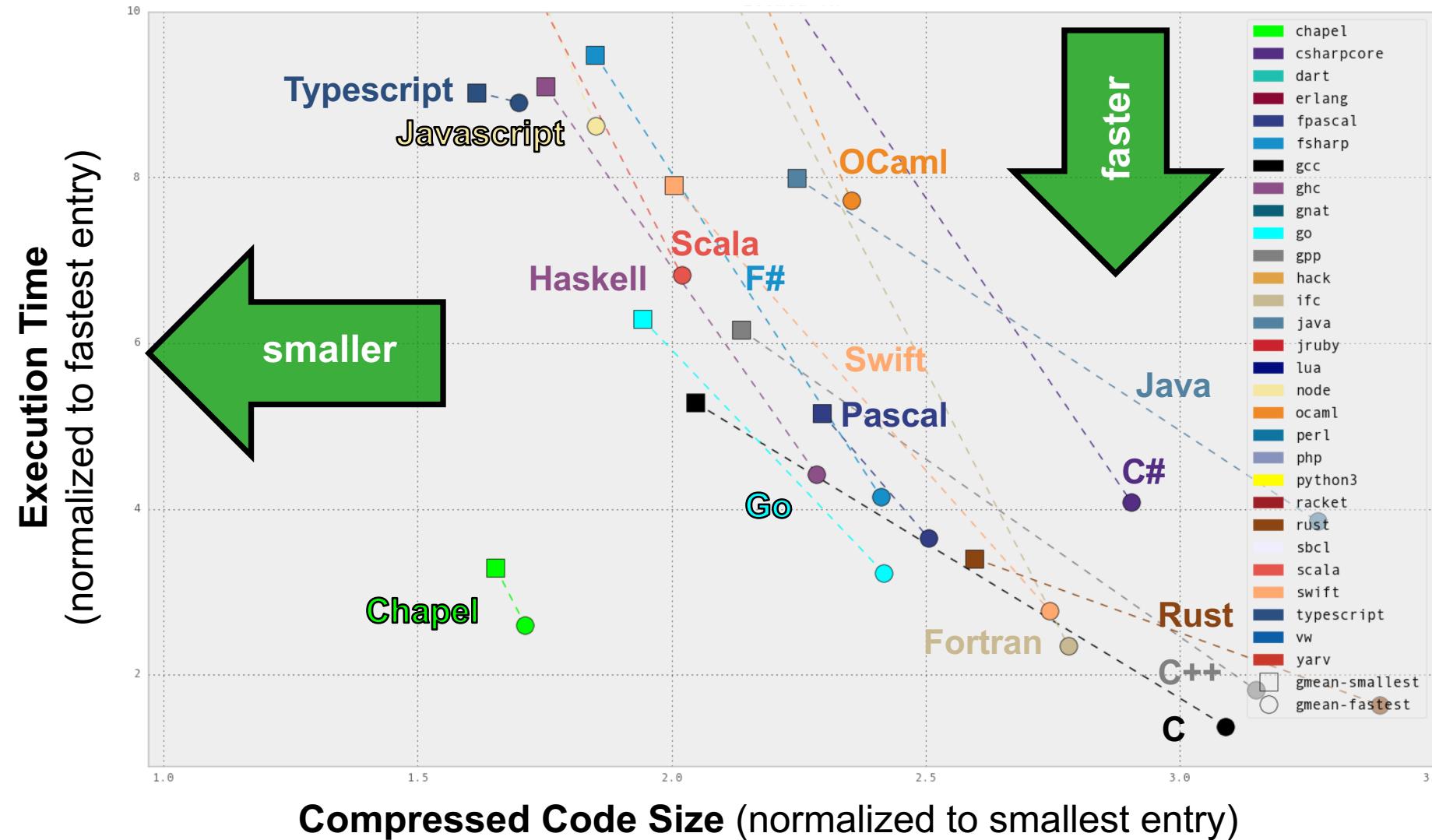
STORE

ANALYZE

Copyright 2018 Cray Inc.

CLBG Cross-Language Summary

(Oct 2017 standings, zoomed in)



COMPUTE

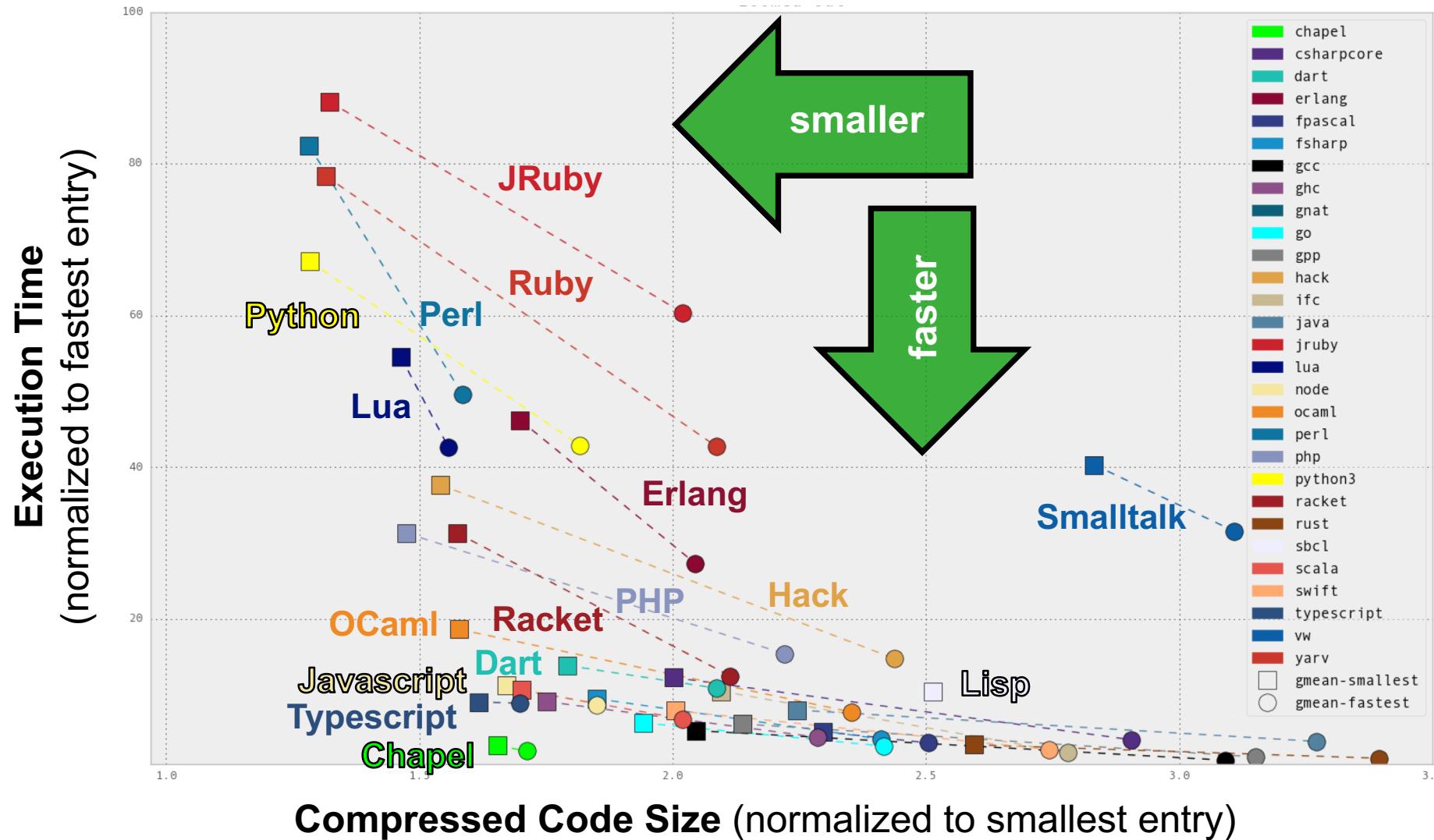
STORE

ANALYZE

Copyright 2018 Cray Inc.

CLBG Cross-Language Summary

(Oct 2017 standings)



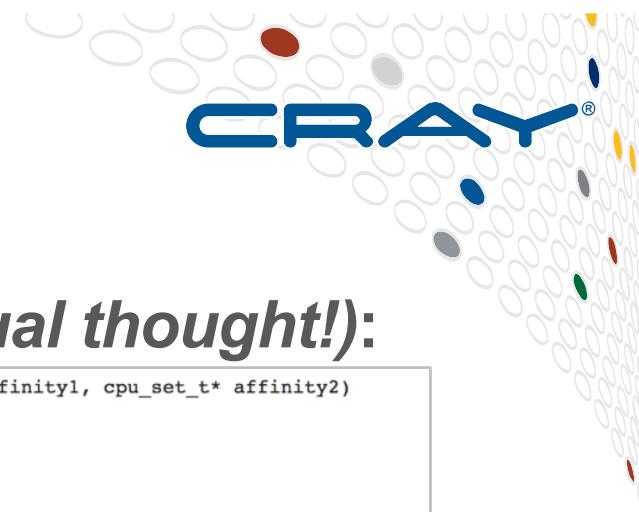
COMPUTE

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, ((i-1)%3):Color);
    const group2 = [i in 1..popSize2] new Chameneos(i, colors10[i]);

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

    print(group1);
    print(group2);

    for c in group1 do delete c;
    for c in group2 do delete c;
}

// Print the results of getNewColor() for all color pairs.
// proc printColorEquations() {
//     for c1 in Color do
//         for c2 in Color do
//             writeln(c1, " + ", c2, " -> ", getNewColor(c1, c2));
//             writeln();
// }

// Hold meetings among the population by creating a shared meeting
// place, and then creating per-chameneos tasks to have meetings.
// proc holdMeetings(population, numMeetings) {
//     const place = new MeetingPlace(numMeetings);

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

//     delete place;
}
```

excerpt from 1210.gz Chapel entry

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
    cpu_set_t active_cpus;
    FILE* f;
    char buf[2048];
    pos;
    cpu_idx;
    physical_id;
    core_id;
    cpu_cores;
    apic_id;
    cpu_count;
    i;

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

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

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

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

excerpt from 2863.gz C gcc entry



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

CLBG: Qualitative Code Comparisons



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

```
proc main() {
    printColorEquations();

    const group1 = [i in 1..PopSize1] new Chameneos(i, 0);
    const group2 = [i in 1..popSize2] new Chameneos(i, 0);

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

    print(group1);
    print(group2);

    for c in group1 do delete c;
    for c in group2 do delete c;
}

// Print the results of getNewColor() for all colors
// 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*

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

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

    delete place;
}

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

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

processor_str      = "processor";
processor_str_len = strlen(processor_str);
physical_id_str   = "physical id";
physical_id_str_len = strlen(physical_id_str);
core_id_str        = "core id";
n(core_id_str);
cores;
n(cpu_cores_str);
};


```

excerpt from 2863.gz C gcc entry



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

CLBG: Qualitative Code Comparisons



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

```
proc main() {  
  
    char const* core_id_str = "core id";  
    size_t core_id_str_len = strlen(core_id_str);  
    char const* cpu_cores_str = "cpu cores";  
    size_t cpu_cores_str_len = strlen(cpu_cores_str);  
  
    CPU_ZERO(&active_cpus);  
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);  
    cpu_count = 0;  
    for (i = 0; i != CPU_SETSIZE; i += 1)  
    {  
        if (CPU_ISSET(i, &active_cpus))  
        {  
            cpu_count += 1;  
        }  
    }  
  
    if (cpu_count == 1)  
    {  
        is_smp[0] = 0;  
        return;  
    }  
}
```

excerpt from 1210.gz Chapel entry

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)  
{  
    cpu_set_t active_cpus;  
    FILE* f;  
    char buf[2048];  
    pos;  
    cpu_idx;  
    physical_id;  
    core_id;  
    cpu_cores;  
    apic_id;  
    cpu_count;  
    i;  
  
    char const* processor_str = "processor";  
    size_t processor_str_len = strlen(processor_str);  
    physical_id_str = "physical id";  
    physical_id_str_len = strlen(physical_id_str);  
    core_id_str = "core id";  
    core_id_str_len = strlen(core_id_str);  
    cpu_cores_str = "cpu cores";  
    cpu_cores_str_len = strlen(cpu_cores_str);  
  
    CPU_ZERO(&active_cpus);  
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);  
    cpu_count = 0;  
    for (i = 0; i != CPU_SETSIZE; i += 1)  
    {  
        if (CPU_ISSET(i, &active_cpus))  
        {  
            cpu_count += 1;  
        }  
    }  
  
    if (cpu_count == 1)  
    {  
        is_smp[0] = 0;  
        return;  
    }  
  
    is_smp[0] = 1;  
    CPU_ZERO(affinity1);
```

excerpt from 2863.gz C gcc entry



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Excerpt from PNW PLSE Review

*“Chapel has been around for quite a while,
and it still seems like a niche language...”*



COMPUTE

| STORE |

ANALYZE

Copyright 2018 Cray Inc.



Chapel: “A Niche Language”?

Chapel is arguably niche in that it...
...was originally designed for HPC

Yet, Chapel’s chief concerns aren’t HPC-specific:

- performance
- programmability (cf. Python)
- parallelism (cf. multicore)
- distributed memory (cf. cloud computing)



COMPUTE

| STORE |

ANALYZE

Copyright 2018 Cray Inc.



Chapel: “A Niche Language”?

Chapel is arguably niche in that it...

...was originally designed for HPC

...has only a modest-sized community (so far)

Yet, we've historically discouraged its use in production...



COMPUTE

| STORE |

ANALYZE

Copyright 2018 Cray Inc.



Chapel: A Quiche Language!

The outsider's impression:



The reality, for most of Chapel's history:



COMPUTE

STORE

Copyright 2018 Cray Inc.

ANALYZE

Image sources: <https://www.themountaintinkitchen.com>, <https://xkcd.com/>



Chapel: A Quiche Language!

The outsider's impression:



Why aren't more people using
this delectable language?

Though recently, it's more like:



COMPUTE

STORE

Copyright 2018 Cray Inc.

ANALYZE

Image sources: <https://www.themountaintinkitchen.com>, <https://xkcd.com/>

Chapel: “Been Around for Quite Awhile”



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

- Research focus: ~6-7 FTEs
 - distinguish locality from parallelism
 - seamlessly mix data- and task-parallelism
 - support user-defined distributed arrays, parallel iterators

Chapel’s Adolescence: “the five-year push” (2013–2018)

- Development focus: ~13-14 FTEs
 - **performance and scalability**
 - **ecosystem:** documentation, libraries, tools, ...
 - **base language fixes:** OOP features, error-handling, strings, ...



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel: “Been Around for Quite Awhile”



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

- Research focus: ~6-7 FTEs
 - distinguish locality from parallelism
 - seamlessly mix data- and task-parallelism
 - support user-defined distributed arrays, parallel iterators

Chapel’s Adolescence: “the five-year push”

(2013–2018)

Then Now

- Development focus: ~13-14 FTEs
 - **performance and scalability**
 - **ecosystem:** documentation, libraries, tools, ...
 - **base language fixes:** OOP features, error-handling, strings, ...



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Chapel Ecosystem: Then vs. Now



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 image shows a terminal window with two panes. The left pane displays the "Chapel Language Specification Version 0.93" document, which includes sections for "Chapel Quick Reference" and "Expression Precedence and Associativity". The right pane shows a "Task Parallel Primer" example, which includes code snippets and comments explaining parallel tasks, begin statements, and cobegin statements. The terminal prompt at the bottom indicates the user is on a Cray system named "bradc".

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
writeIn("Hello, world");

2. Compile and run it:
\$ chpl hello.chpl
> ./out
Hello, world

Expression Precedence and Associativity*

Operators Uses

new (right)	constructor call
** (right)	exponentiation
reduce scan	reduction, scan, apply domain
mapped	map
(right)	logical and/biwise mention

Statements

```
if cond then stmt1(); else stmt2();  
if (~cond) then stmt1(); else stmt2();  
select expr {  
  when equiv1 do stmt1();  
  when equiv2 { (stmt2()); }  
  otherwise { (stmt3()); }  
}  
  
while condition do ...;  
while condition { ... }  
do { ... } while condition;  
for index in aggregate do ...;  
for index in aggregate { ... }  
label name for ...;  
break; or break outer;  
continue; or continue outer;
```

Procedures

```
proc barrier( real, i: imag): complex {  
  var c: complex = z + i;  
  return c;  
}  
proc foo(i) return i**2 + i + 1;
```

Formal Argument Intents

Intent	Semantics
in	consumed
out	produced
inout	produced and consumed
ref	passed by reference
const	passed by value or reference, but with local modifications disabled
blank	like ref for arrays, domains, syncs, singles; otherwise like const

Named Formal Arguments

```
proc foo(arg1: int, arg2: real) { ... }  
foo(arg2=3.14, arg1=2);
```

Default Values for Formal Arguments

```
proc foo(arg1: int, arg2: real = 3.14);  
foo(2);
```

File Edit Options Buffers Tools chpl Help

// Task Parallel Primer

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

```
config const n = 10;  
  
writeln("1: ## The begin statement #####");  
  
// The begin statement spawns a thread of execution that is independent  
// of the current (main) thread of execution.  
begin writeln("1: output from spawned task");  
  
// The main thread of execution continues on to the next statement.  
// There is no guarantee as to which statement will execute first.  
writeln("1: output from main task");  
  
writeln("2: ## The cobegin statement #####");  
  
// For more structured behavior, the cobegin statement can be used to  
// spawn a block of tasks, one for each statement. Control continues  
// after the cobegin block, but only after all the tasks within the  
// cobegin block have completed.  
cobegin {  
  writeln("2: output from spawned task 1");  
  writeln("2: output from spawned task 2");  
}  
  
// The output from within the cobegin statement will always precede the  
// following output from the main thread of execution.  
writeln("2: output from main task");  
  
writeln("3: ## The cobegin statement with nested begin statements #####");  
  
// If any begin statements are used within a cobegin statement,  
// the thread of execution does not a wait for those begin statements  
// to complete.  
cobegin {  
  begin writeln("3: output from spawned task 1");  
  begin writeln("3: output from spawned task 2");  
}  
  
-uu----F1 taskParallel.chpl Top L1 (Chapel/l Abbrev)-----  
Loading /users/bradc/chapel/highlight/emacs/22/chpl-mode.el (source)...done
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Documentation: Now



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

The screenshot displays a hierarchical web-based documentation system 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 for Quickstart Instructions, Using Chapel, Platform-Specific Notes, Technical Notes, and Tools. The "WRITING CHAPEL PROGRAMS" section includes links for 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 for Chapel Evolution and Archived Language Specifications.

The central page shows the "Chapel Documentation" header and a "View page source" link. It features a main content area titled "Compiling and Running Chapel" with a sub-section "Using Chapel". This section includes 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. A "Previous" button is located at the bottom of this sidebar.

On the right side, there are two more pages: "Using Chapel" and "Task Parallelism". The "Using Chapel" page has its own sidebar with links to Quickstart Instructions, Using Chapel, Platform-Specific Notes, Technical Notes, and Tools. The "Task Parallelism" page has its own sidebar with links to Primers, Language Basics, Iterators, Task Parallelism, Task Parallelism (with sub-links for Begin Statements, Cobegin Statements, Coforall Statements, Sync / Singles, Atomics, Locality, Data Parallelism, Library / Utilities), and Cobegin Statements. Each page also includes a "View page source" link.



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Libraries: Then

After HPCS: ~25 library modules

- documented via source comments, if at all:

The image shows two side-by-side terminal windows on a Mac OS X system. Both windows have a title bar "bradc — ssh bradc@troll.cray.com — bash". The left window displays the contents of the file "Random.chpl", which contains extensive multi-line comments explaining the implementation of a random number generator. The right window displays the contents of the file "Regexp.chpl", which contains extensive multi-line comments documenting the interface and usage of the qio_Regexp module.

```
// Copyright (c) 2004-2013, Cray Inc. (See LICENSE file for more details)

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



```
extern type qio_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:s\tring, 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_re\gexp_options_t);
extern proc qio_regexp_get_pattern(ref regexp:qio_regexp_t, ref pattern: 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):bo\ol;
--uu---F1 Regexp.chpl Top L1 (Chapel/l Abbrev)-----
```



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Libraries: Now

Now: ~60 library modules

- web-documented, many user-contributed

The documentation pages show lists of library modules:

- Standard Modules:**
 - 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:**
 - BLAS
 - Collection
 - Crypto
 - Curl
 - DistributedBag
 - DistributedDeque
 - DistributedIterators
 - FFTW
 - FFTW_MT
 - Futures
 - HDFS
 - HDFSSIterator
 - LAPACK
 - LinearAlgebra
 - MPI
 - Norm
 - OwnedObject
 - RangeChunk
 - RecordParser
 - Search
 - SharedObject
 - Sort
 - VisualDebug
 - ZMQ



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Tools: Then



After HPCS:

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



COMPUTE

| STORE |

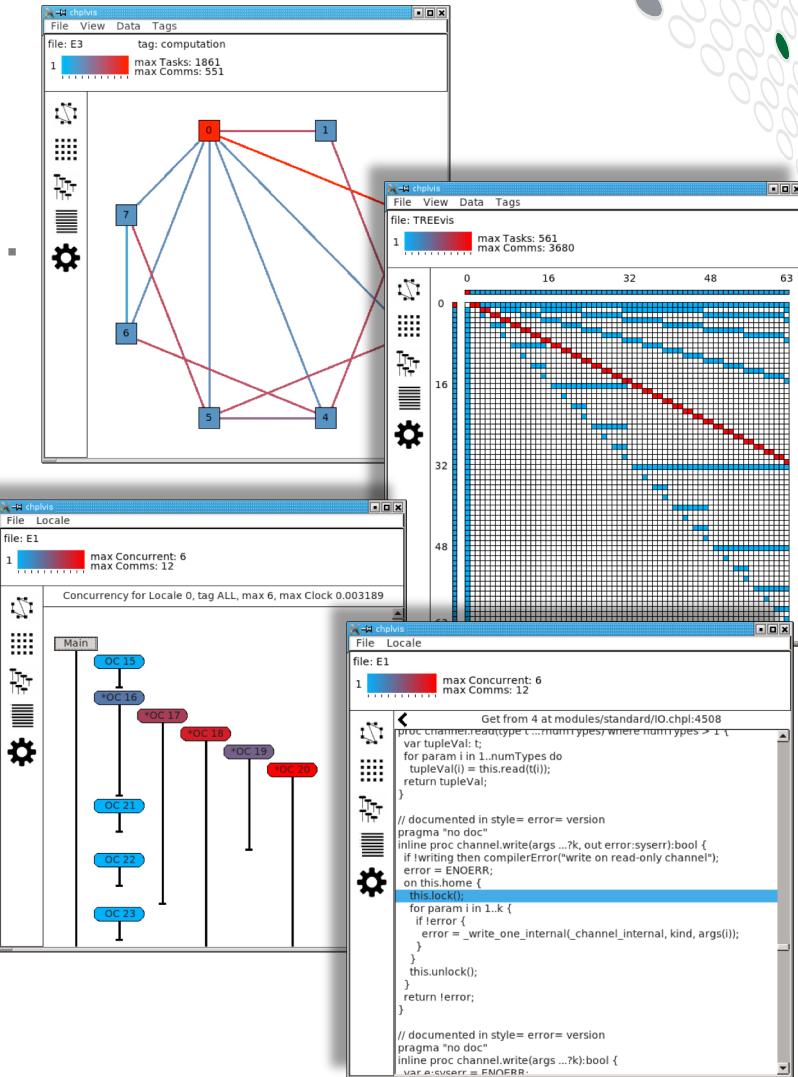
ANALYZE

Copyright 2018 Cray Inc.

Tools: Now

Now:

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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Chapel Performance: Then vs. Now



COMPUTE

|

STORE

|

ANALYZE

Copyright 2018 Cray Inc.

Performance Focus Areas during 5-year push



- Cleaner, simpler generated code
- NUMA sensitivity within multi-socket nodes
- Best-use of RDMA and NIC memory registration
- Reduced overheads in tasks, memory, communication
- Bulk transfer optimizations
- ...and much more...



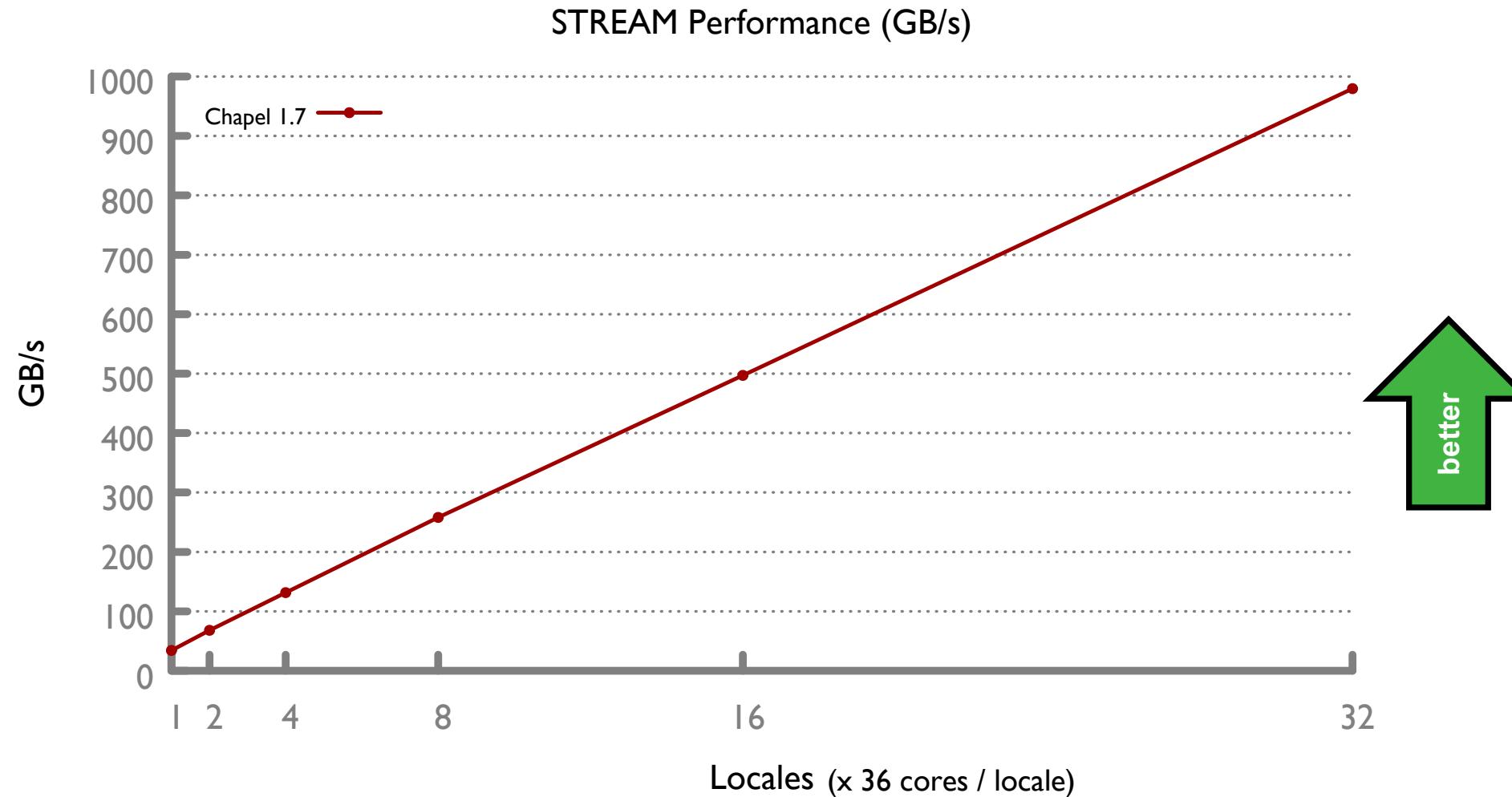
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

STREAM Triad Performance: Chapel Then



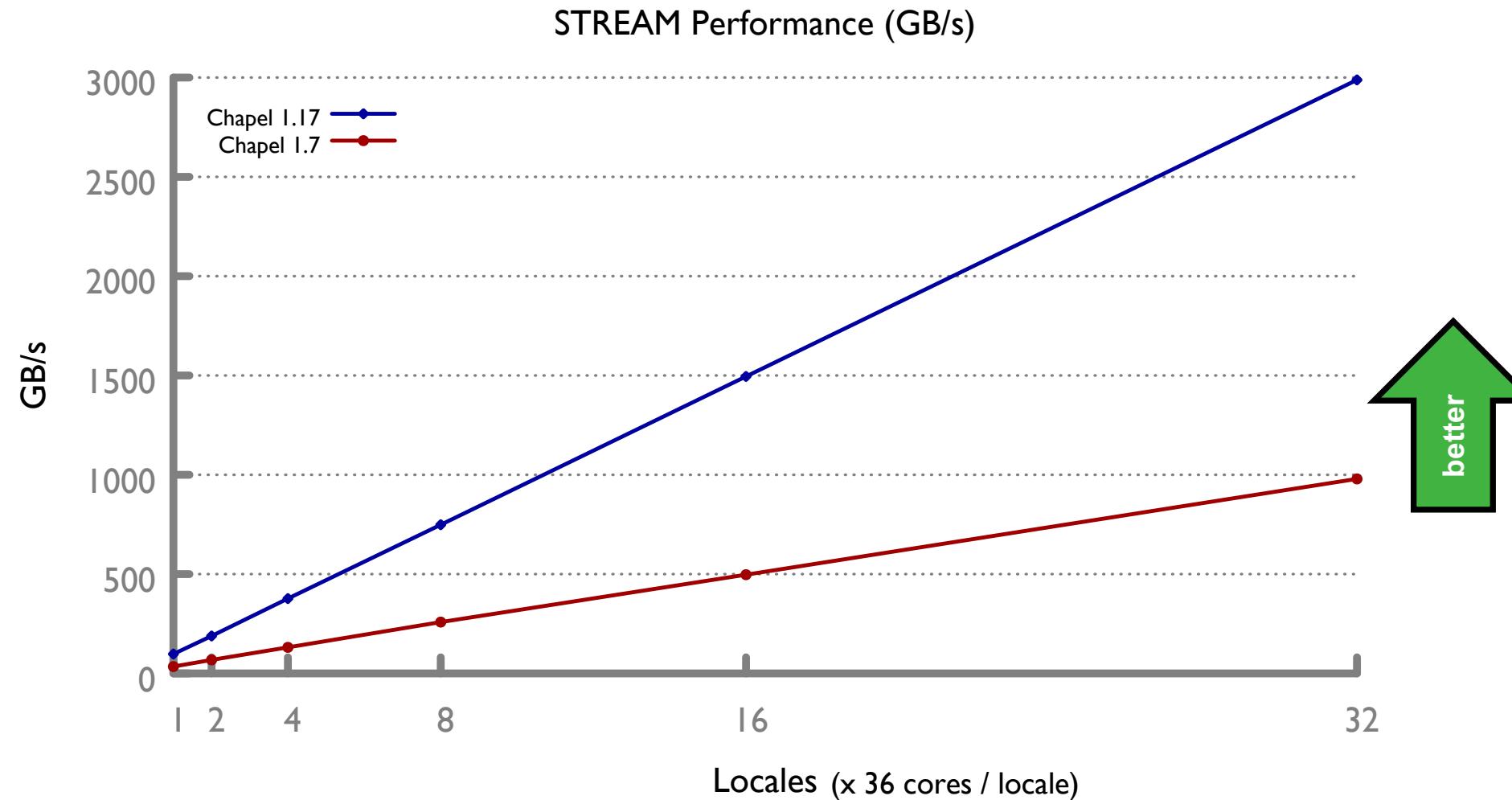
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

STREAM Triad Performance: Chapel Then vs. Now



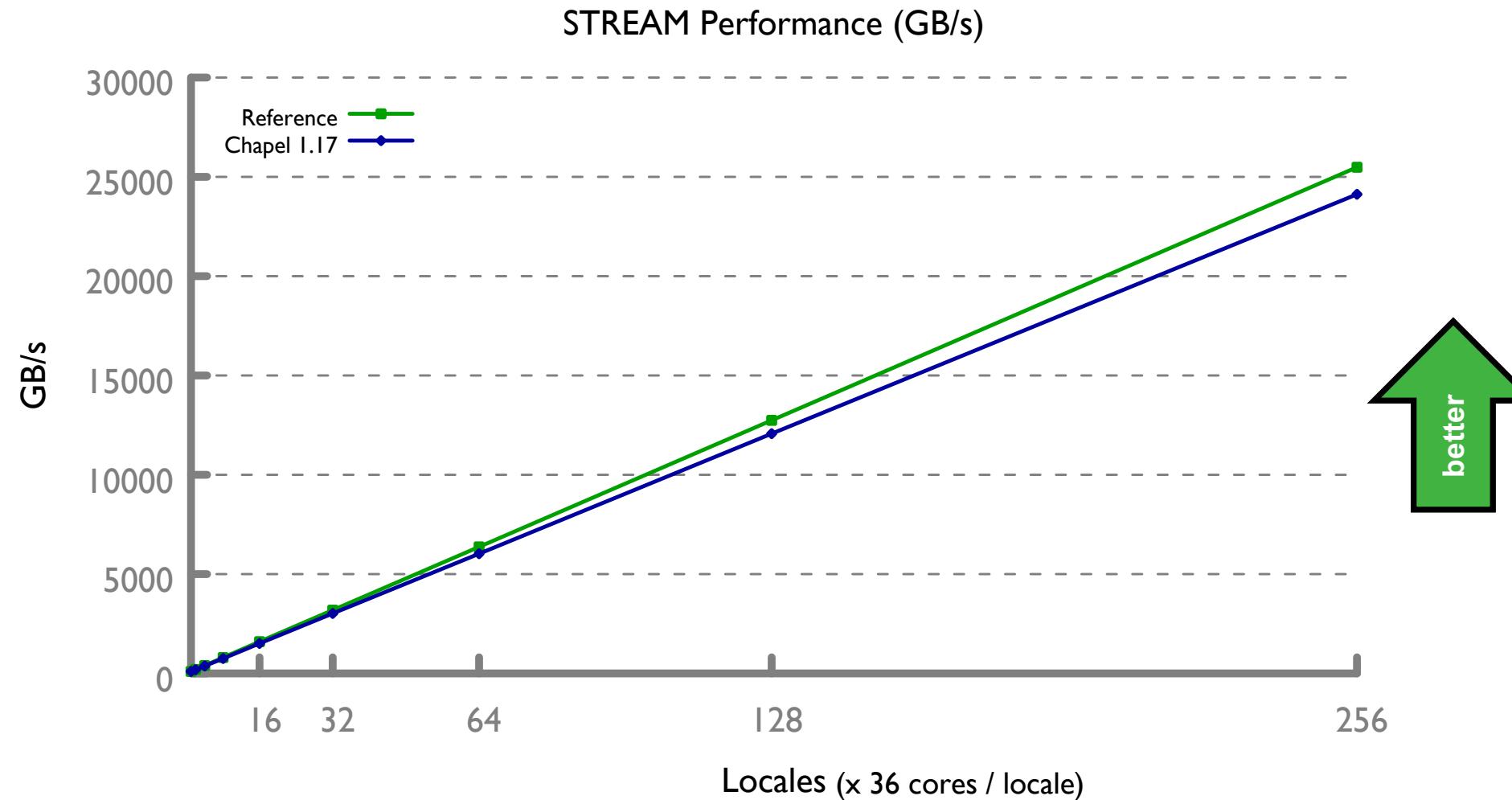
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

STREAM Triad Performance: Chapel Now vs. ref



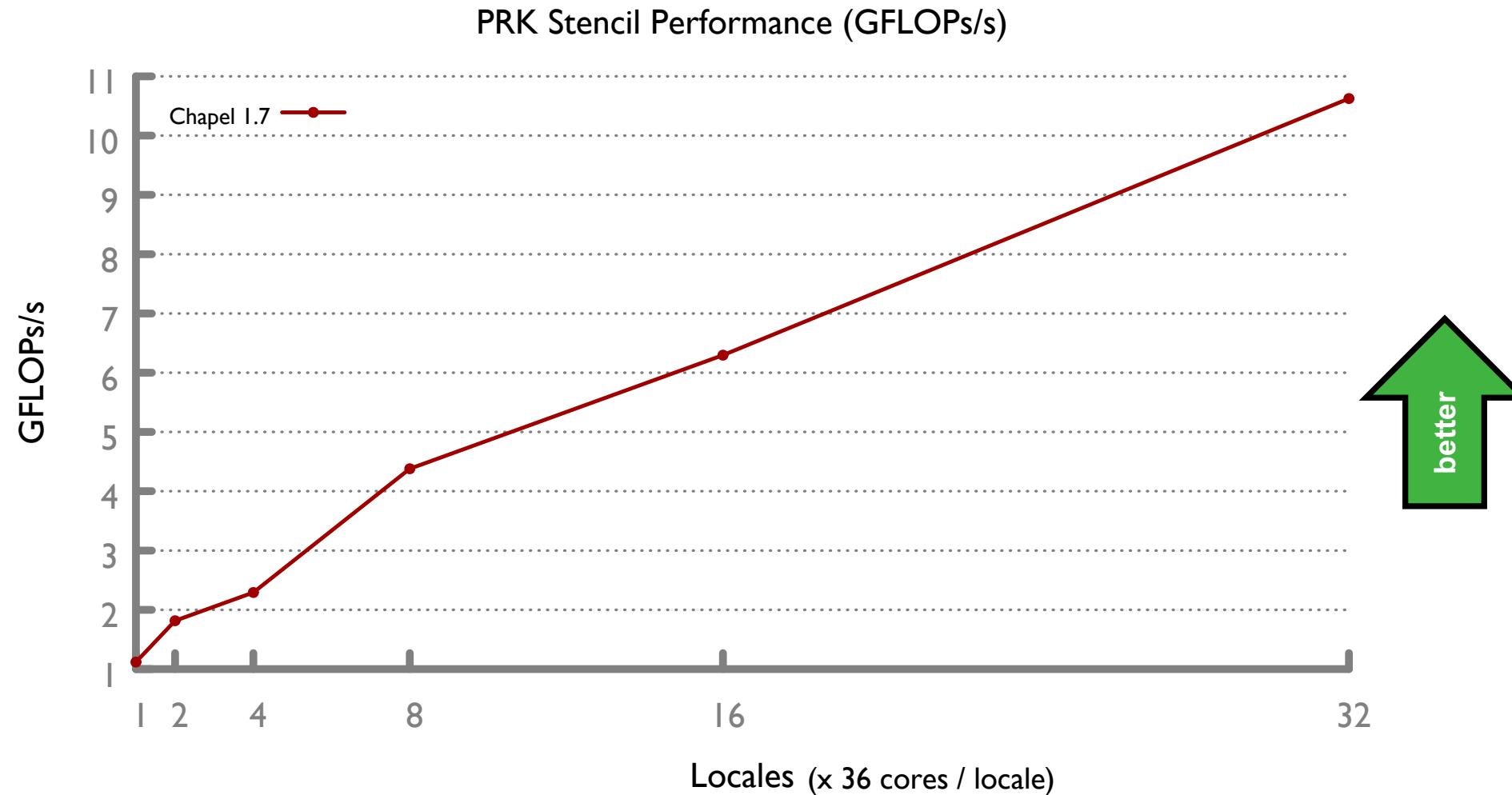
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

PRK Stencil Performance: Chapel Then



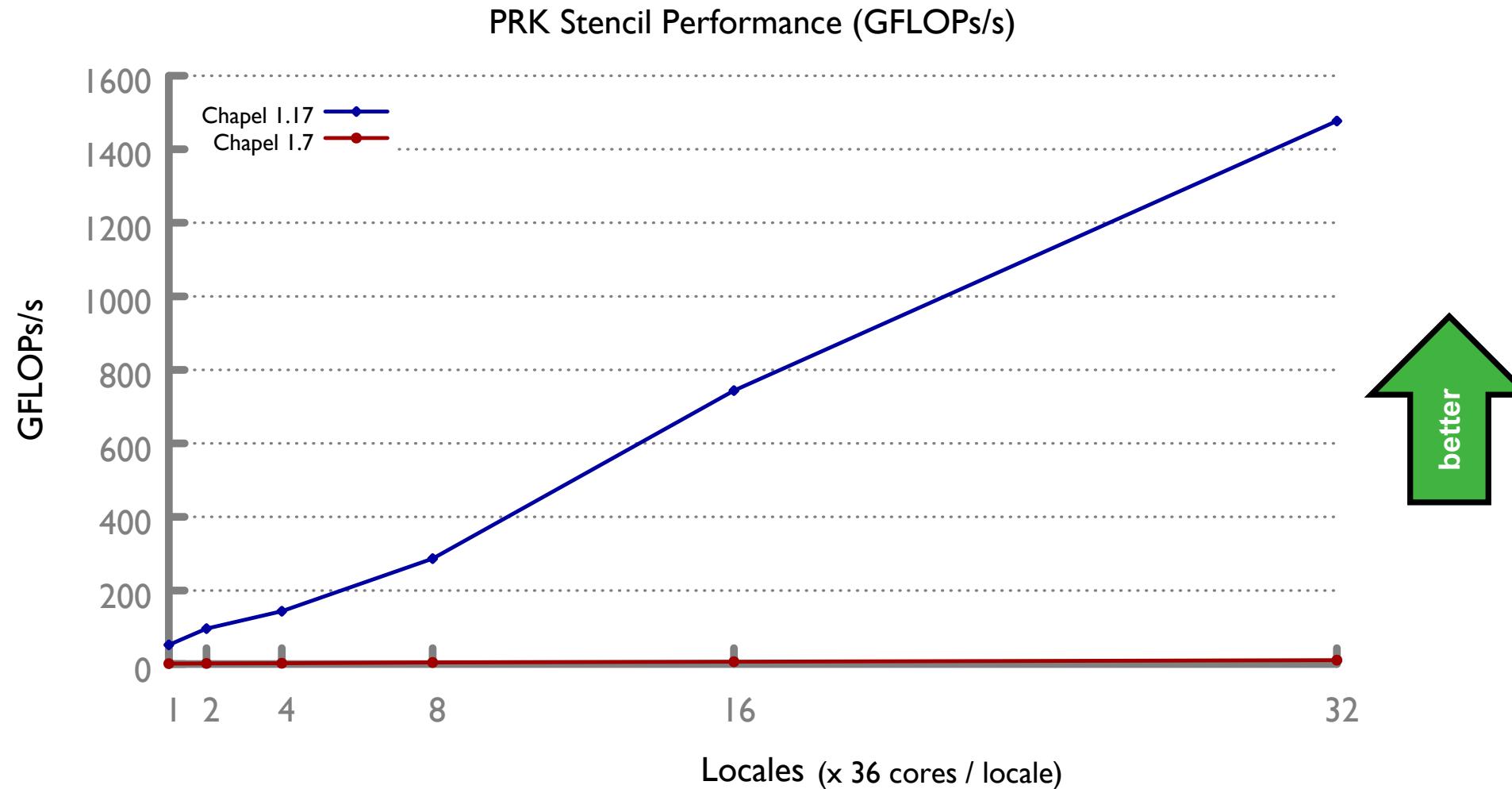
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

PRK Stencil Performance: Chapel Then vs. Now



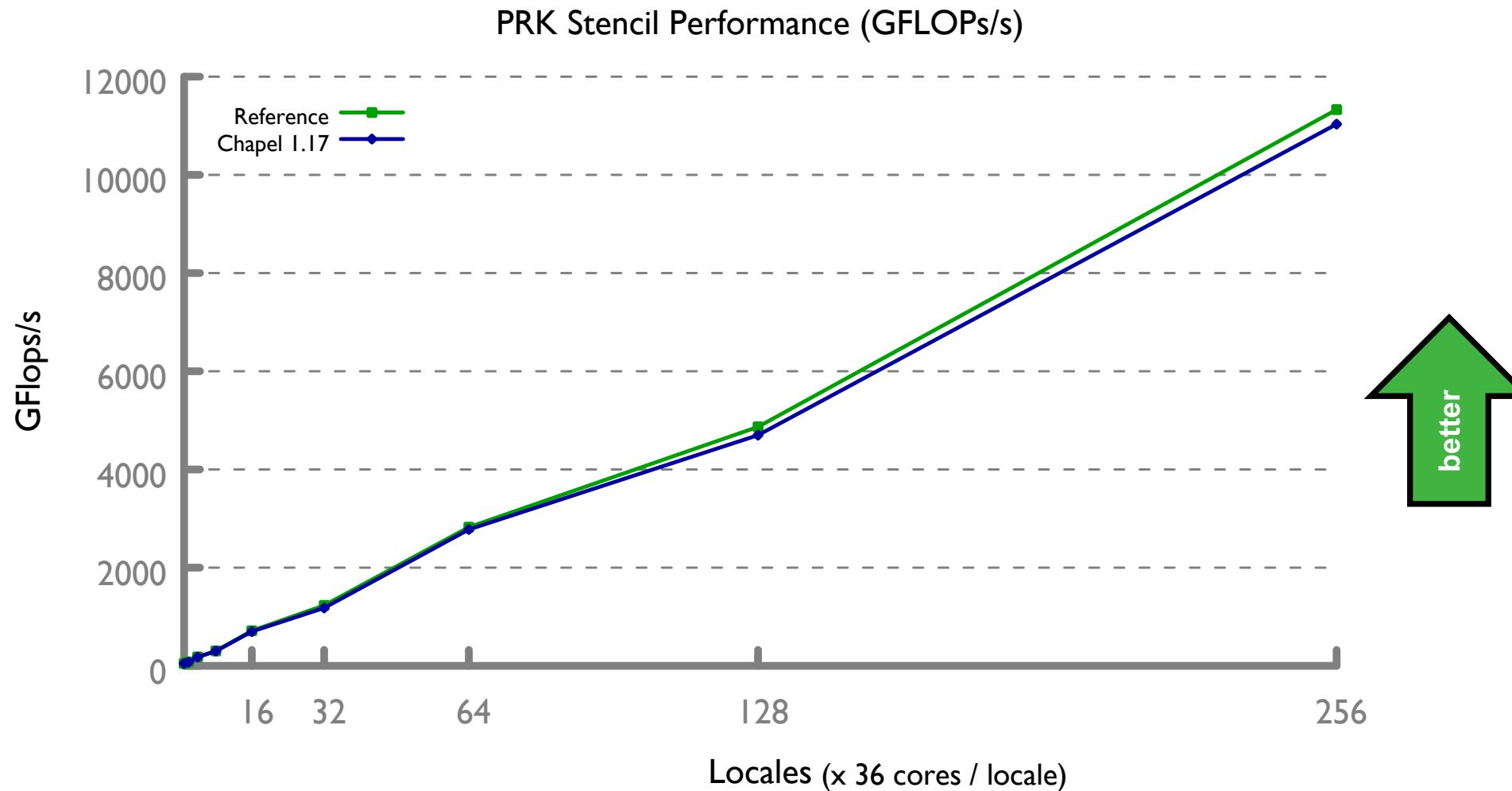
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

PRK Stencil Performance: Chapel Now vs. ref



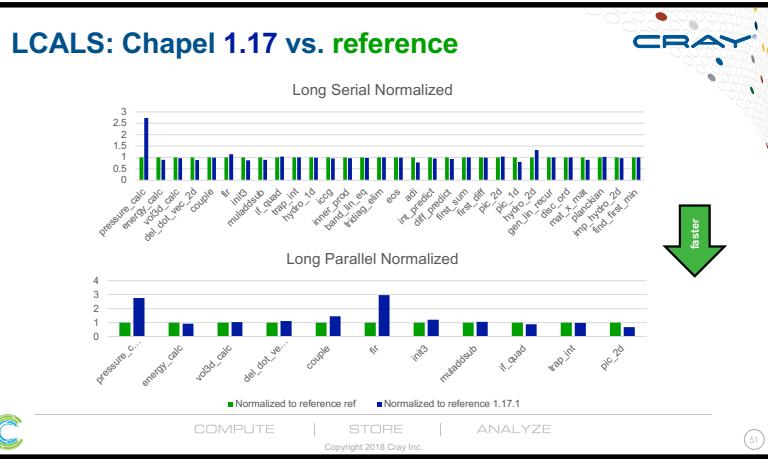
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

HPC Patterns: Chapel Now vs. reference



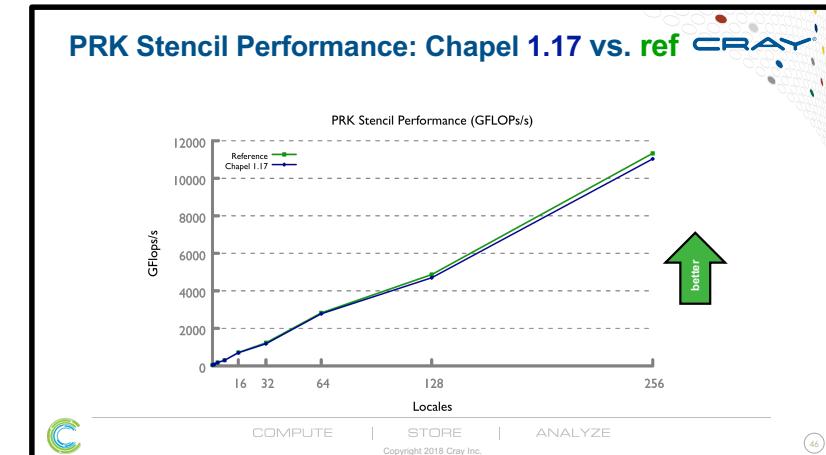
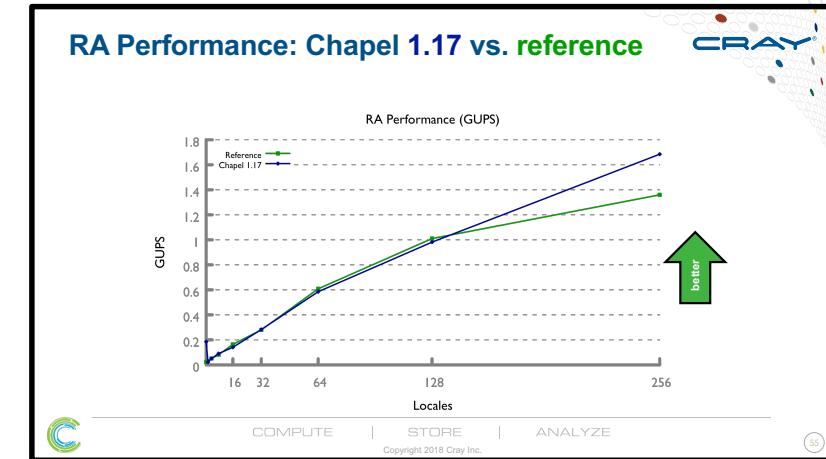
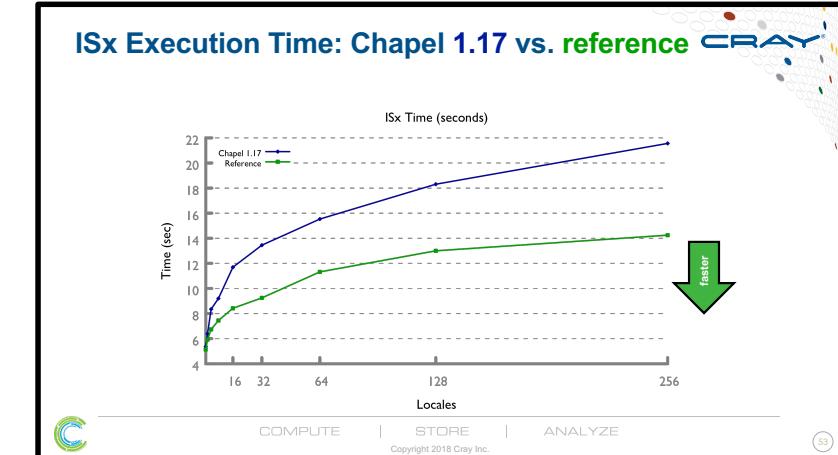
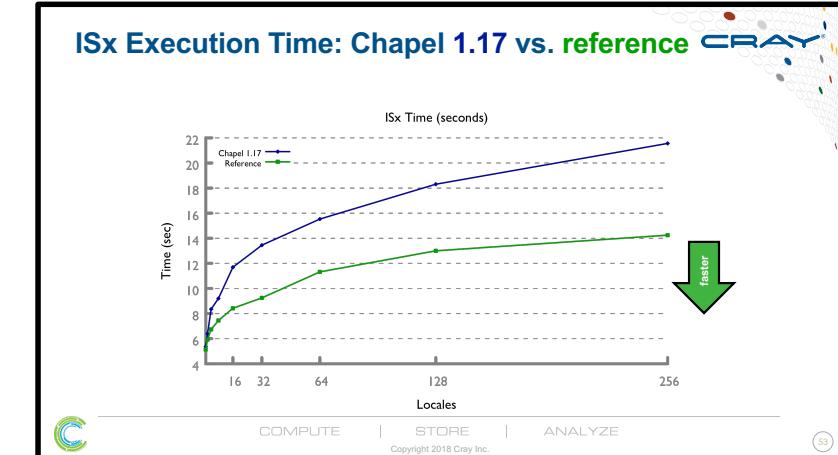
LCALS

HPCC RA

STREAM
Triad

ISx

PRK
Stencil



COMPUTE

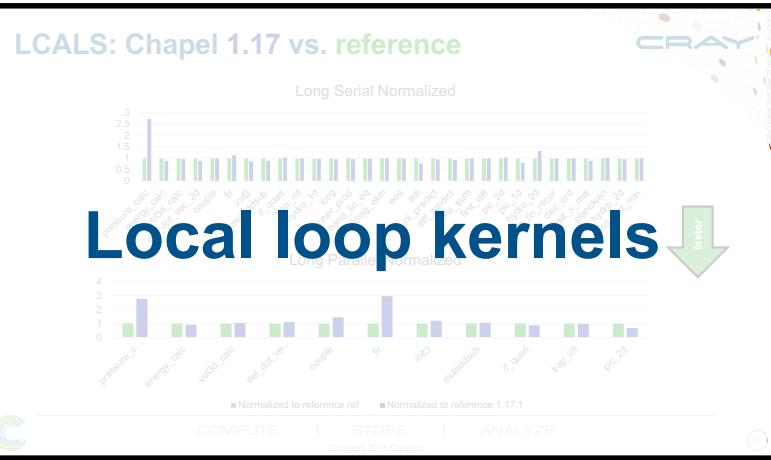
STORE

ANALYZE

Copyright 2018 Cray Inc.

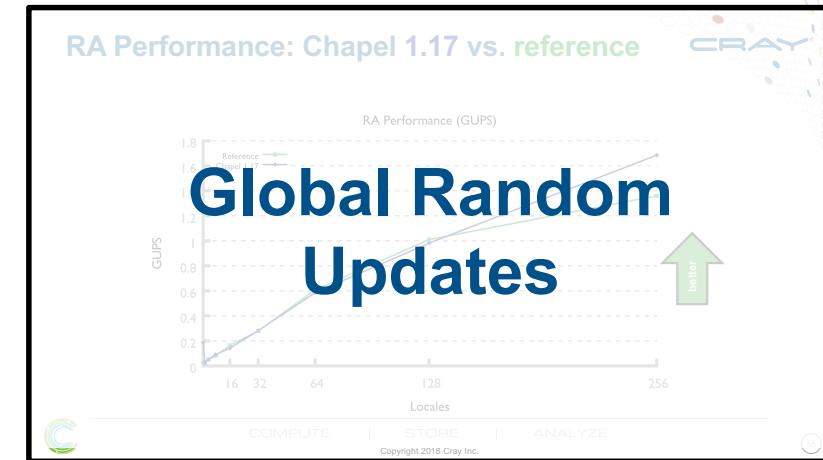
Nightly performance tickers online at:
<https://chapel-lang.org/perf-nightly.html>

HPC Patterns: Chapel Now vs. reference



LCALS

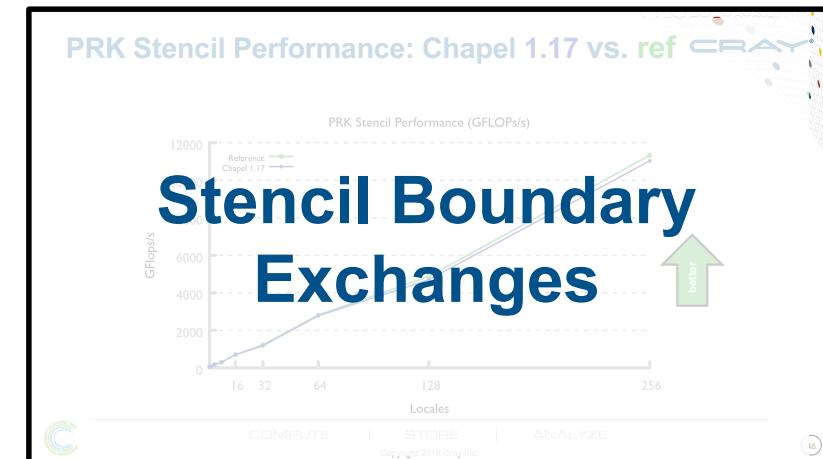
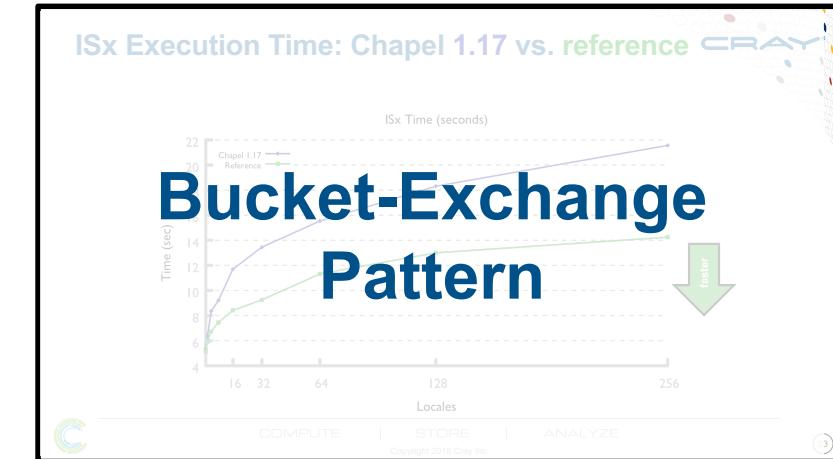
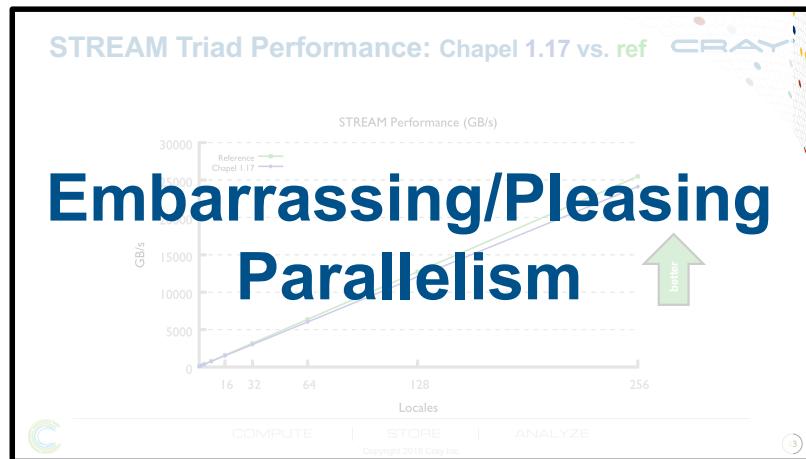
HPCC RA



STREAM
Triad

ISx

PRK
Stencil



COMPUTE

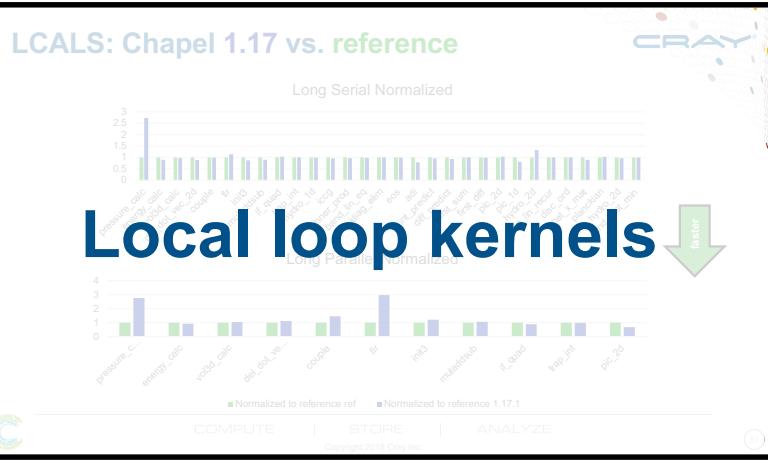
STORE

ANALYZE

Copyright 2018 Cray Inc.

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

HPC Patterns: Chapel Now vs. reference

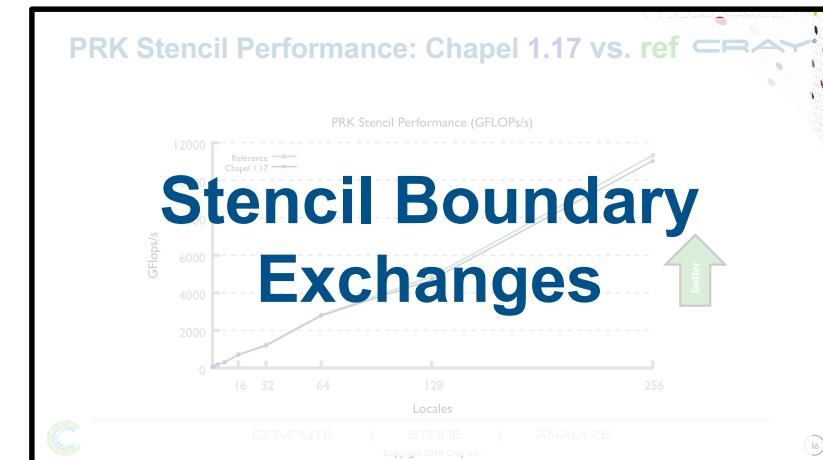
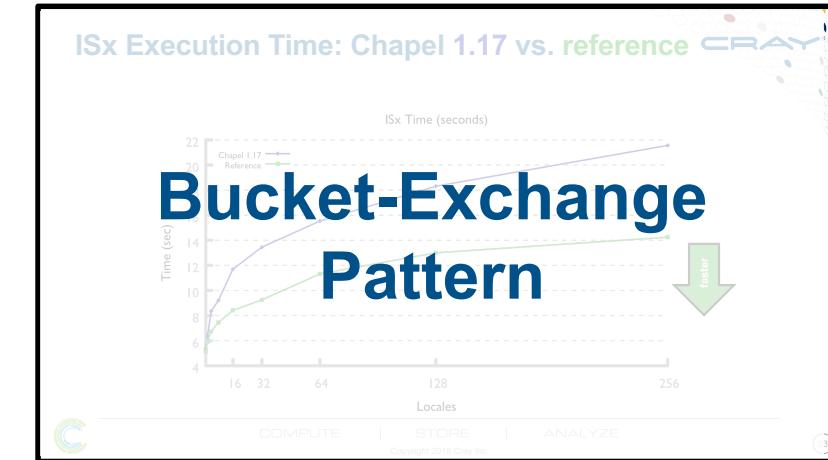
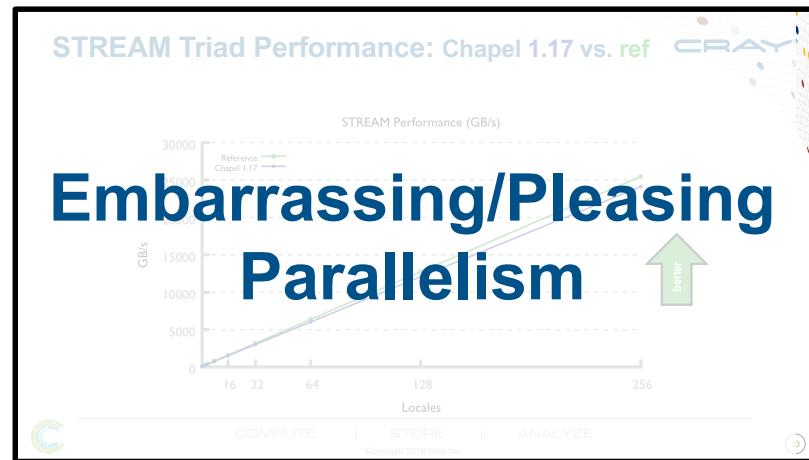
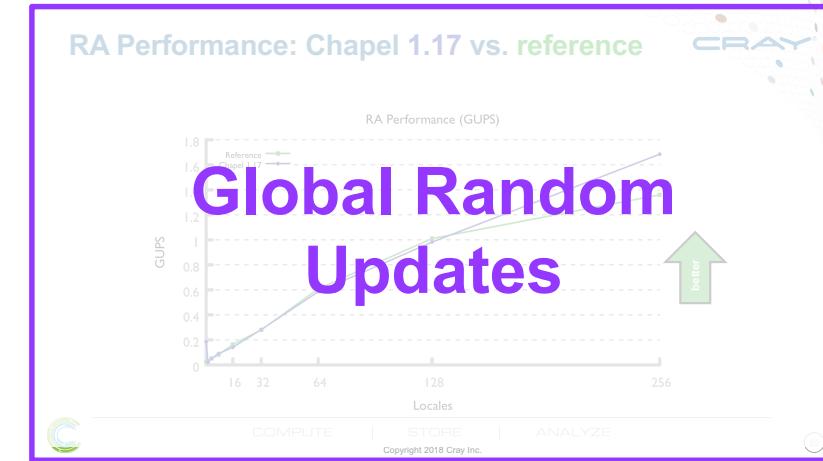


LCALS

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 Random Access Kernel: MPI



```

/* Perform updates to main table. The scalar equivalent is:
 *
 * for (i=0; i<NUPDATE; i++) {
 *   Ran = (Ran << 1) ^ ((s64Int) Ran < 0) ? POLY : 0;
 *   Table[Ran & (TABSIZE-1)] ^= Ran;
 * }
 */
MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
while (i < SendCnt) {
    /* receive messages */
    do {
        MPI_Test(&inreq, &have_done, &status);
        if (have_done) {
            if (status.MPI_TAG == UPDATE_TAG) {
                MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
                bufferBase = 0;
                for (j=0; j < recvUpdates; j++) {
                    inmsg = LocalRecvBuffer[bufferBase+j];
                    LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                                 tparams.GlobalStartMyProc;
                    HPCC_Table[LocalOffset] ^= inmsg;
                }
            } else if (status.MPI_TAG == FINISHED_TAG) {
                NumberReceiving--;
            } else
                MPI_Abort( MPI_COMM_WORLD, -1 );
            MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                      MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
        }
    } while (have_done && NumberReceiving > 0);
    if (pendingUpdates < maxPendingUpdates) {
        Ran = (Ran << 1) ^ ((s64Int) Ran < ZERO64B ? POLY : ZERO64B);
        GlobalOffset = Ran & (tparams.TableSize-1);
        if ( GlobalOffset < tparams.Top)
            WhichPe = ( GlobalOffset / (tparams.MinLocalTableSize + 1) );
        else
            WhichPe = ( (GlobalOffset - tparams.Remainder) /
                         tparams.MinLocalTableSize );
        if (WhichPe == tparams.MyProc) {
            LocalOffset = (Ran & (tparams.TableSize - 1)) -
                         tparams.GlobalStartMyProc;
            HPCC_Table[LocalOffset] ^= Ran;
        }
    }
}

```

```

    } else {
        HPCC_InsertUpdate(Ran, WhichPe, Buckets);
        pendingUpdates++;
    }
    i++;
}
else {
    MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
    if (have_done) {
        outreq = MPI_REQUEST_NULL;
        pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                             &peUpdates);
        MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
                  UPDATE_TAG, MPI_COMM_WORLD, &outreq);
        pendingUpdates -= peUpdates;
    }
}
/* send remaining updates in buckets */
while (pendingUpdates > 0) {
    /* receive messages */
    do {
        MPI_Test(&inreq, &have_done, &status);
        if (have_done) {
            if (status.MPI_TAG == UPDATE_TAG) {
                MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
                bufferBase = 0;
                for (j=0; j < recvUpdates; j++) {
                    inmsg = LocalRecvBuffer[bufferBase+j];
                    LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                                 tparams.GlobalStartMyProc;
                    HPCC_Table[LocalOffset] ^= inmsg;
                }
            } else if (status.MPI_TAG == FINISHED_TAG) {
                /* we got a done message. Thanks for playing... */
                NumberReceiving--;
            } else
                MPI_Abort( MPI_COMM_WORLD, -1 );
            MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                      MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
        }
    } while (have_done && NumberReceiving > 0);
}

```

```

MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
if (have_done) {
    outreq = MPI_REQUEST_NULL;
    pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                         &peUpdates);
    MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
              UPDATE_TAG, MPI_COMM_WORLD, &outreq);
    pendingUpdates -= peUpdates;
}
/* send our done messages */
for (proc_count = 0 ; proc_count < tparams.NumProcs ; ++proc_count) {
    if (proc_count == tparams.MyProc) { tparams.finish_req[tparams.MyProc] =
                                         MPI_REQUEST_NULL; continue; }
    /* send garbage - who cares, no one will look at it */
    MPI_Isend(&Ran, 0, tparams.dtype64, proc_count, FINISHED_TAG,
              MPI_COMM_WORLD, tparams.finish_req + proc_count);
}
/* Finish everyone else up... */
while (NumberReceiving > 0) {
    MPI_Wait(&inreq, &status);
    if (status.MPI_TAG == UPDATE_TAG) {
        MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
        bufferBase = 0;
        for (j=0; j < recvUpdates; j++) {
            inmsg = LocalRecvBuffer[bufferBase+j];
            LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                         tparams.GlobalStartMyProc;
            HPCC_Table[LocalOffset] ^= inmsg;
        }
    } else if (status.MPI_TAG == FINISHED_TAG) {
        /* we got a done message. Thanks for playing... */
        NumberReceiving--;
    } else
        MPI_Abort( MPI_COMM_WORLD, -1 );
    MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
              MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
}
MPI_Waitall( tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);

```



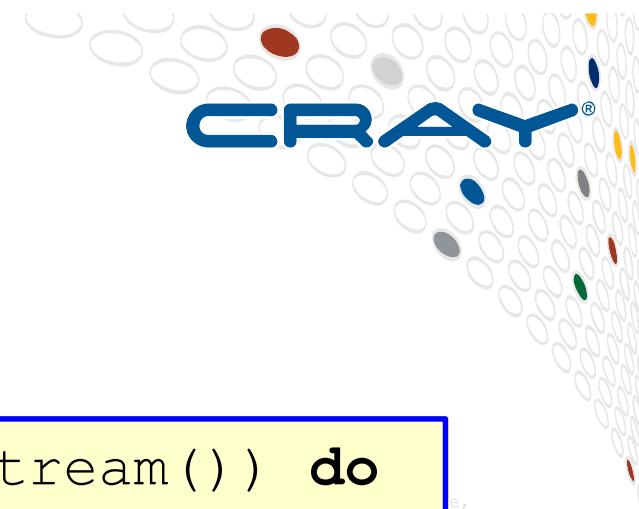
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

HPCC Random Access Kernel: MPI



```
/* Perform updates to main table. The scalar equivalent is:  
 *  
 * for (i=0; i<NUPDATE; i++) {  
 *   Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);  
 *   Table[Ran & (TABSIZ-1)] ^= Ran;  
 * }  
  
MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype,  
          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, &recvUpdates);  
          bufferBase = 0;  
        }  
      }  
    }  
  }  
  /* 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 */  
    Isend(&Ran, 0, tparams.dtype64, proc_count, FINISHED_TAG,  
          statuses);
```

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;  
 * }  
 */
```



COMPUTE

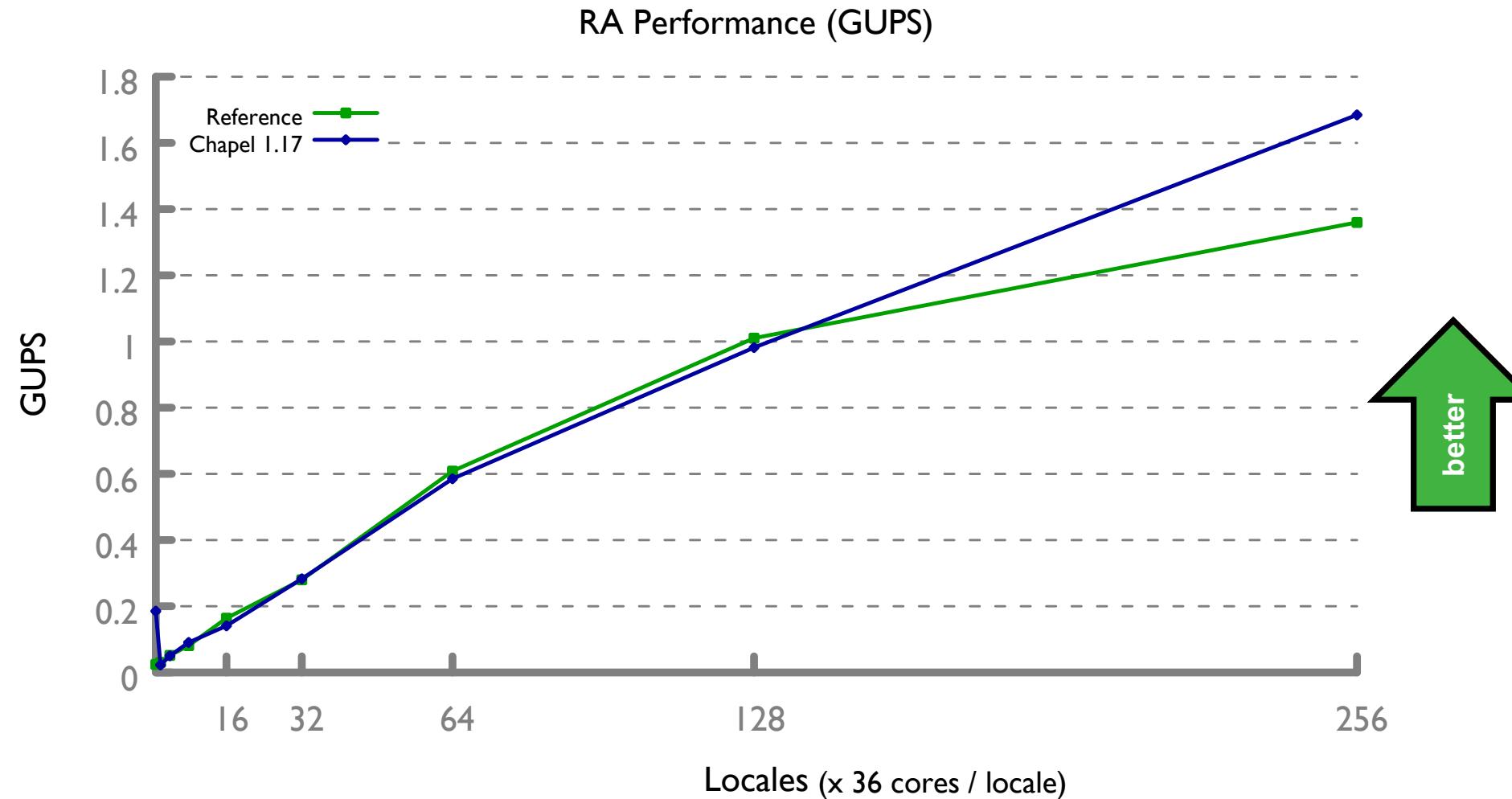
STORE

ANALYZE

Copyright 2018 Cray Inc.

RA Performance: Chapel Now vs. reference

CRAY®



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

What's Next?



CHIUW 2018: The 5th annual Chapel Implementers and Users Workshop

- Vancouver BC, Friday May 25th
- Details: <https://chapel-lang.org/CHIUW2018.html>

Chapel's college years: plans for 2018-2021

- Further Performance and Scalability Improvements
- Libfabric/OFI Support
- GPU Support
- Cloud Support
- Chapel AI



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



The Chapel Team at Cray (May 2018)



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Community Partners



Lawrence Berkeley
National Laboratory



Yale

(and several others...)

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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Summary



- Chapel's made huge progress over the past five years
- Ready for use in production*
- Open to collaborations
 - Plenty of research questions remain



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Chapel Resources



COMPUTE

STORE

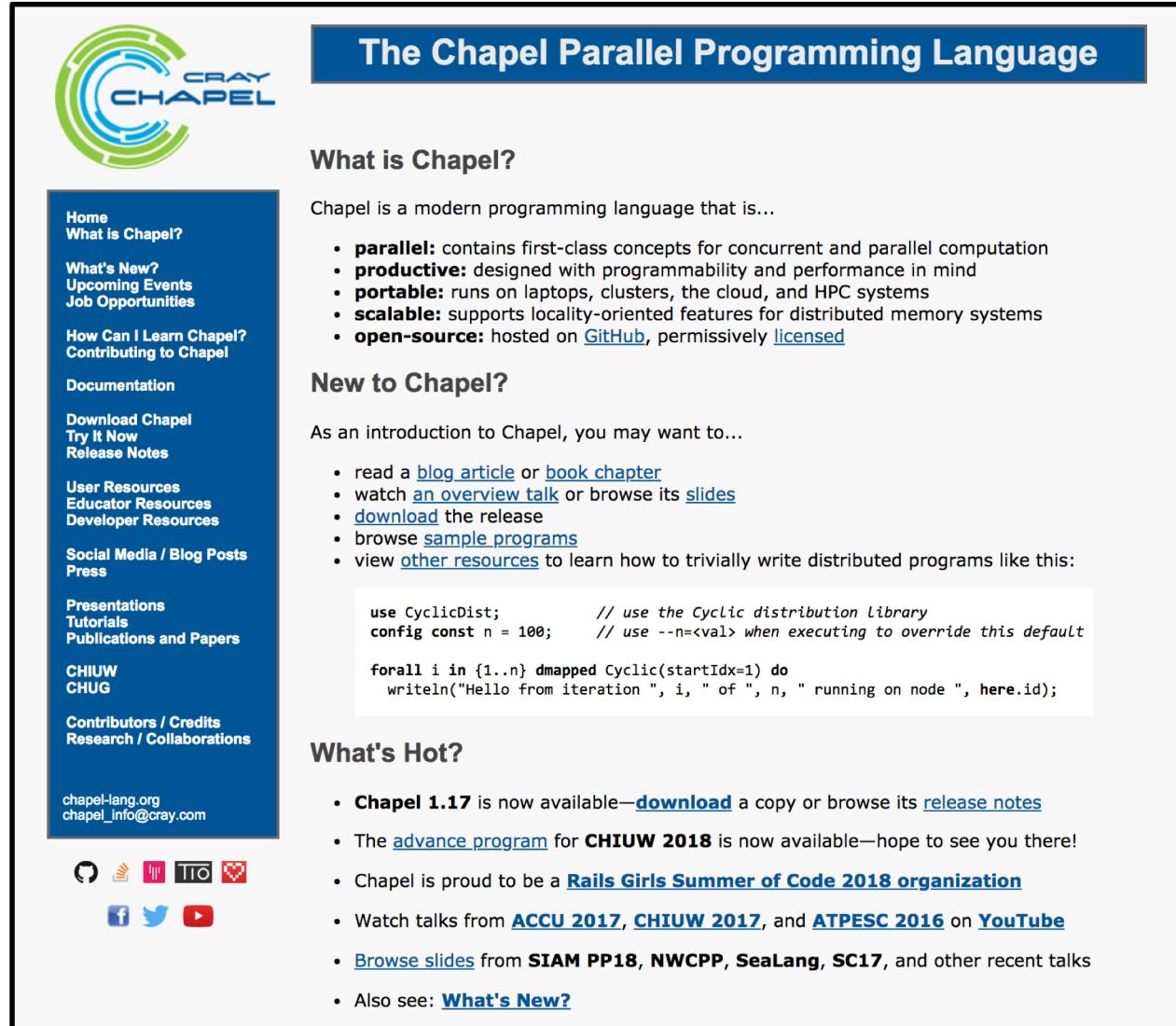
ANALYZE

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 and the text "The Chapel Parallel Programming Language". The main content area is titled "What is Chapel?" and describes Chapel as a modern programming language. It lists several characteristics: parallel, productive, portable, scalable, and open-source. Below this is a section titled "New to Chapel?", which provides links to blog articles, overview talks, sample programs, and other resources. A code snippet in Chapel syntax is shown, demonstrating a cyclic distribution library. The footer contains links to social media and email, and a "What's Hot?" section listing recent news items.

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?](#)



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 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 discusses an interview with Brad Chamberlain.
- Facebook:** Shows the Chapel Programming Language page with 72 subscribers. It features a chart comparing program times across various languages and environments.
- YouTube:** Shows the Chapel Parallel Programming Language channel with 72 subscribers. It contains a playlist of videos, including a CHI'17 keynote by Jonathan Dursi and a PyCon UK 2017 talk by Russel Winder.
- Custom Channel:** Shows a custom channel page for Chapel Parallel Programming Language with 72 subscribers. It includes a video thumbnail for the CHI'17 keynote and another for the PyCon UK 2017 talk.



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 panels:

- Stack Overflow Tagged Questions:** Shows the 'Tagged Questions' page for the 'chapel' tag. It lists three questions: "Tuple Concatenation in Chapel", "Is there a way to use non-scalar values in functions with what?", and "Is there any writef() format specifier for a bool?". Each question has a summary, votes, answers, and a list of tags.
- Github Issues:** Shows the 'Issues' tab for the 'chapel-lang / chapel' repository. It displays a list of 292 open issues, including titles like "Implement 'bounded-coforall' optimization for remote coforalls", "Consider using processor atomics for remote coforalls EndCount", and "make uninstall". Each issue includes a brief description, author, and timestamp.
- Gitter Chat:** Shows a Gitter channel for 'chapel-lang/chapel'. It features a purple header with the text 'Where communities thrive'. The chat log shows messages from users Brian Dolan (@buddha314) and Michael Ferguson (@mppf) about array syntax and references, with code snippets provided.
- Community Stats:** A purple page titled 'GITTER' with the subtitle 'Where communities thrive'. It highlights 'FREE FOR COMMUNITIES' with statistics: 'JOIN OVER 800K+ PEOPLE' and 'JOIN OVER 90K+ COMMUNITIES'. It also says 'CREATE YOUR OWN COMMUNITY' and 'EXPLORE MORE COMMUNITIES'.



COMPUTE

STORE

ANALYZE

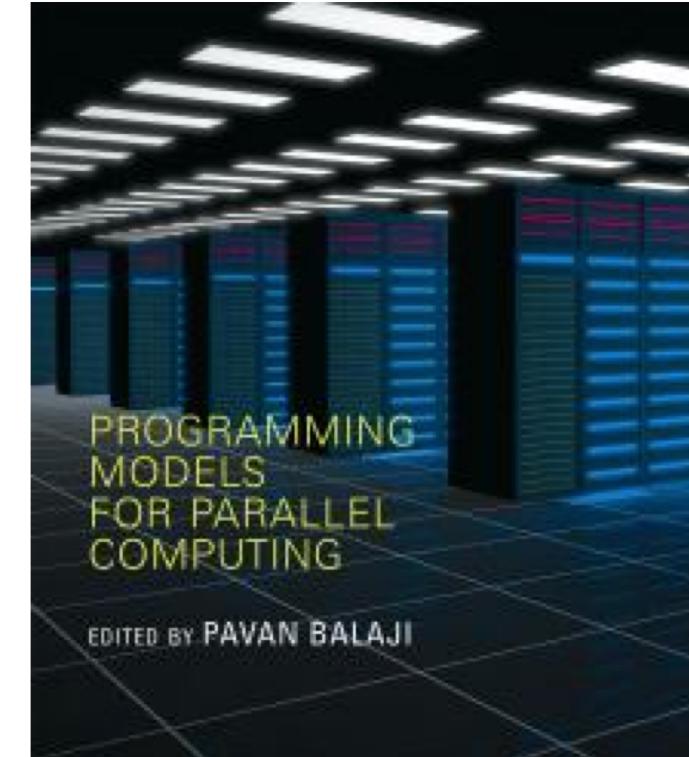
Copyright 2018 Cray Inc.

Suggested Reading (healthy attention spans)



Chapel chapter from *Programming Models for Parallel Computing*

- a detailed overview of Chapel's history, motivating themes, features
- published by MIT Press, November 2015
- edited by Pavan Balaji (Argonne)
- chapter is also available [online](#)



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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Suggested Reading (short attention spans)



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

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

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

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

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

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

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

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

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

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

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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Where to..

Submit bug reports:

[GitHub issues for chapel-lang/chapel](#): public bug forum

chapel_bugs@cray.com: for reporting non-public bugs

Ask User-Oriented Questions:

[StackOverflow](#): when appropriate / other users might care

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

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

Discuss Chapel development

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

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

Discuss Chapel's use in education

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

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



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Questions?



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

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

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



CRAY
THE SUPERCOMPUTER COMPANY