Striving for Productivity and Performance Portability

Brad Chamberlain, Chapel Team, Cray Inc.

Performance Portability in Extreme Scale Computing: Metrics Challenges, Solutions (Dagstuhl 17431)

October 26, 2017

If the HPC Community were to create a truly productive language… …would we ever know?

Brad Chamberlain, Chapel Team, Cray Inc.

Performance Portability in Extreme Scale Computing: Metrics Challenges, Solutions (Dagstuhl 17431)

October 26, 2017

If the HPC Community were to create a truly productive language… …how would we ever know?

Brad Chamberlain, Chapel Team, Cray Inc.

Performance Portability in Extreme Scale Computing: Metrics Challenges, Solutions (Dagstuhl 17431)

October 26, 2017

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.

Copyright 2017 Cray Inc.

My Background

Education:

- Earned Ph.D. from University of Washington CSE in 2001
  - focused on the ZPL data-parallel array language
- Remain associated with UW CSE as an Affiliate Professor

Industry R&D:

- Currently a Principal Engineer at Cray Inc.
- Technical lead / founding member of the Chapel project

Copyright 2017 Cray Inc.

Disclaimers

- **This talk focuses a lot on languages, due to my biases**
  - That said, many points likely apply to other HPC software…
- **I tend to use the term “language” in a very loose sense**
  - “A way of communicating your intent to the computer”
  - Think “programming model” or “programming notation” if you prefer

Copyright 2017 Cray Inc.

Do we have productive HPC languages?

Scenario 1: Brad leaves his HPC bubble to ask a colleague:

- ”I need to write a desktop program. You’re young and hip, what productive language should I use?”
  - what does it say that I didn’t simply use my HPC language for this…?
- likely responses: Python, Swift, Go, R, Matlab, …

Scenario 2: Colleague asks Brad:

- ”I need to write some general-purpose distributed memory code. You’ve worked in HPC for decades, what should I use?”
- my response: … *[awkward embarrassment for our community]*

Copyright 2017 Cray Inc.

HPC Language Characterizations

Current languages

- pragmatic
- bottom-up design
  - driven by system capabilities
  - mechanism-oriented
- alphabet soup
- NASCAR
- punk rock
- worthy of respect

Productive languages

- idealistic
- top-down design
  - driven by end-user needs
  - intent-oriented
- something more coherent
- tricked-out BMW i8
- classical symphony
- worth striving for

Copyright 2017 Cray Inc.

Productivity: Played Out?

- **There's some sense that productivity isn't "hot" anymore**
  - "Haven't we [solved | given up on] that by now?"

- **Arguably analogous to "peace"**
  - not particularly "new" or "hip" as a concept
  - reasonable reasons for skepticism about our ability to achieve it
    - for productivity, these are more social than technical, in my opinion
  - yet, clearly something to desire / strive for over the alternative

- **Personally, I prefer not to throw in the towel** (in either case)

Copyright 2017 Cray Inc.

Productivity and HPCS

(a brief history, from my perspective)

Copyright 2017 Cray Inc.

Chapel's Origins: HPCS

DARPA HPCS: High *Productivity* Computing Systems

- **Goal:** improve productivity by a factor of 10x
- **Timeframe:** Summer 2002 – Fall 2012
- **Three phases, five competitors:** Cray, HP, IBM, SGI, Sun
- Cray developed a new system architecture, network, software stack
  - (this became the very successful Cray® XC30™ Supercomputer Series)

…and a new programming language: Chapel

Copyright 2016 Cray Inc.

Productivity, as defined by HPCS

Productivity =

performance

+ programmability (readability, writability, maintainability, modifiability, tunability, …)

+ portability

+ robustness

A reasonable starting point… but how to measure 10x?

- particularly since most of these are hard to measure individually?

In phase 2, an independent team was created to define this

Copyright 2017 Cray Inc.

My "Zany Metrics" (an early brainstorming exercise)

"Zany" Metrics

DARPA HPCS

- Abstractness of Code
  - how much code must change if we…
    - change number of processors, shape of processor set?
    - change problem size?
    - make processors not divide problem size evenly?
    - make processor dimensions, problem size non-2^k?
    - switch dense arrays to sparse?
    - change an array's rank?
- Portability of Code
  - how much code must change…
    - to run on another vendor's machine?
    - to get performance satisfactory to that vendor?

CRAY

(source: HPCS Phase II Metrics Kickoff: 2003-8-5)

Copyright 2017 Cray Inc.

"Language Bingo"

Language Comparison

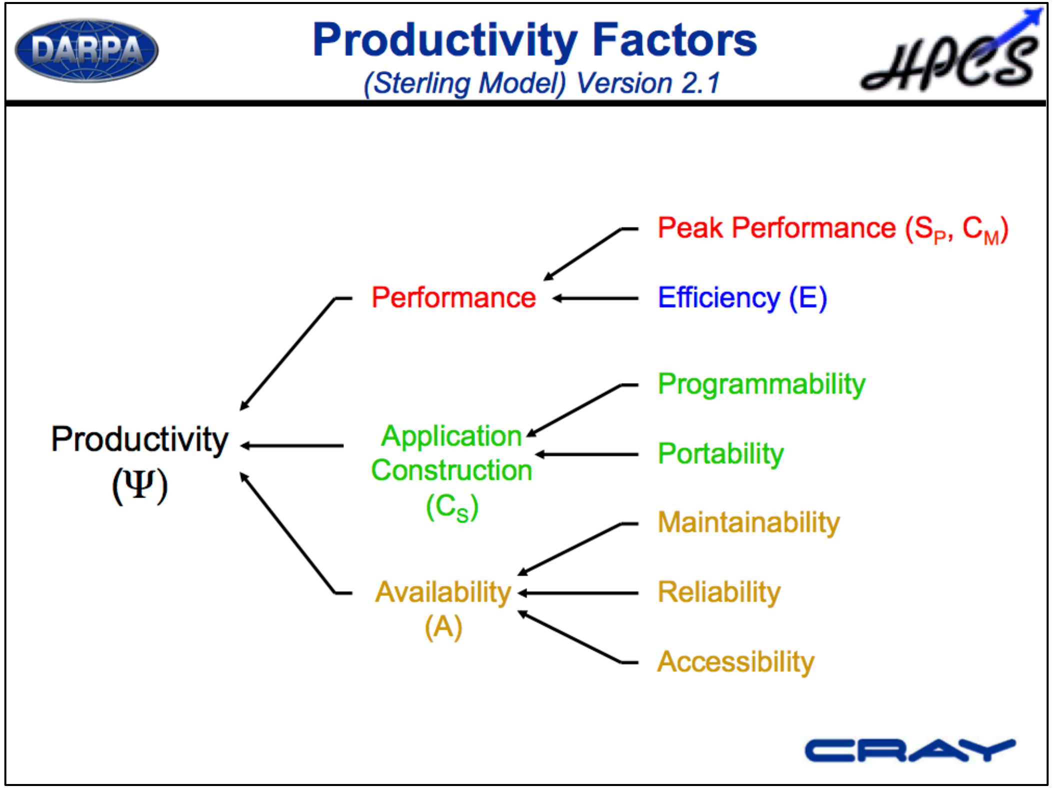
	MPI	SHMEM	Java	UPC	CAF	HPF	OpenMP	Fortran/ C
Performs Well	O	O	?	?	O	?	?	~
Portable	O	?	O	?	?	~	X	~
Performance Model	O	O	O	O	O	X	X	X
Global View	X	X	X	X	X	O	O	O
Post-scalar	~/X	~/X	O	X	~	~	~/X	~/X
Abstractions	X	X	O	~	~	X	X	X
Succinct	X	X	X	X	X	~	~	~
General	O	O	O	~	X	X	X	O

-- = no comment O = good ~ = so-so X = poor ? = unproven

(source: an early HPCS productivity meeting)

Copyright 2017 Cray Inc.

Sterling's Model of Productivity



(source: HPCS Phase II Metrics Kickoff: 2003-8-5)

Copyright 2017 Cray Inc.

Sterling's Model of Productivity

General Model of Productivity

DARPA HPCS

$R_i \equiv i^{th}$ result product

$T_i \equiv$ time to compute result R_i

$T_L \equiv$ total lifetime of machine

$T_V \equiv$ total overhead time of machine

$T_Q \equiv$ quiescent time of machine

$T_R \equiv$ working time of machine

$N_R \equiv$ total number of result products during T_L

$C_L \equiv$ all costs associated with machine during T_L

$C_{LS} \equiv$ application software costs during T_L

$C_{LO} \equiv$ costs of ownership during T_L

$C_M \equiv$ cost of procurement and initial installation

$C_{Si} \equiv$ cost of application software for result R_i

$\Psi \equiv$ productivity

$$T_R = \sum_i^{N_R} T_i$$

$$T_L = T_R + T_V + T_Q$$

$$R_L = \sum_i^{N_R} R_i$$

$$C_L = C_{LS} + C_M + C_{LO}$$

$$C_{LS} = \sum_i^{N_R} C_{Si}$$

$$\boxed{\Psi \equiv \frac{R_L}{C_L \times T_L}}$$

CRAY

Coincidence?!? I think not…

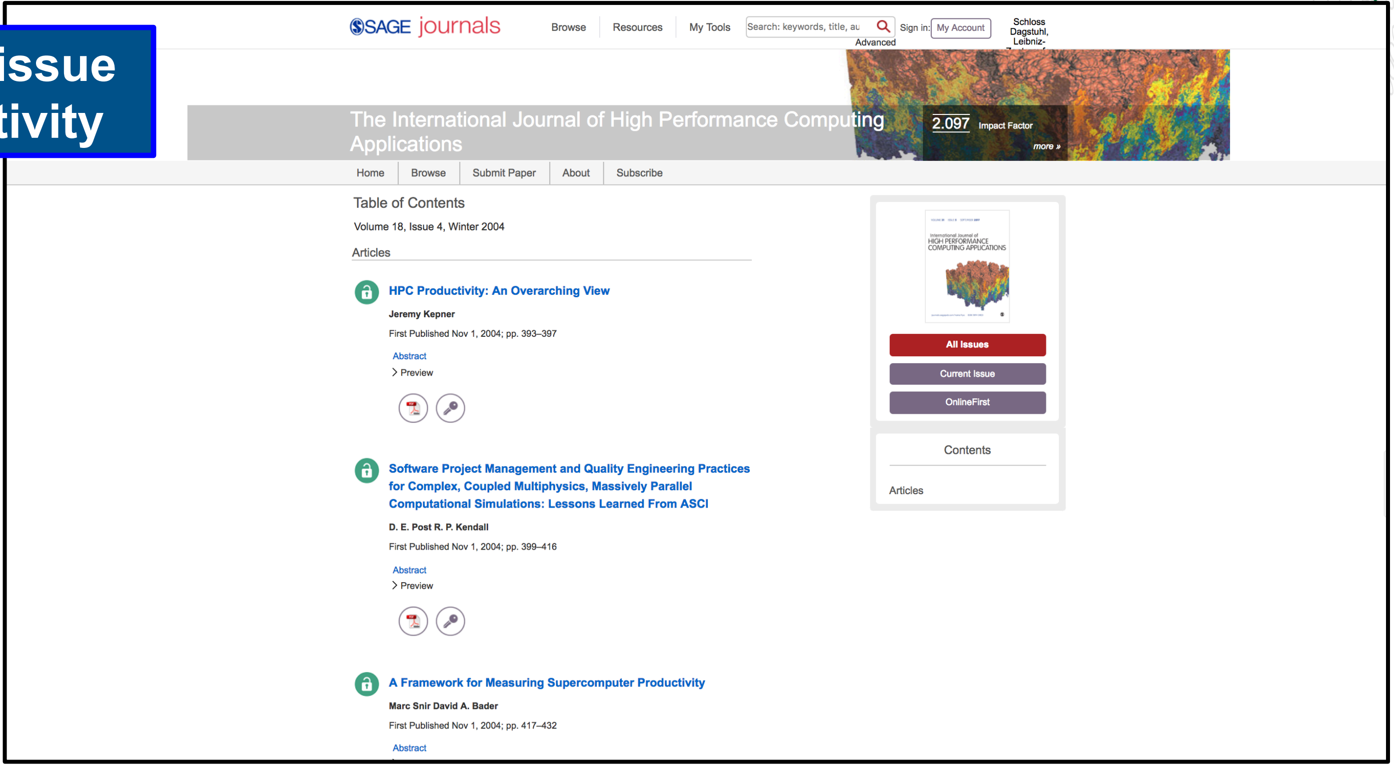
(source: HPCS Phase II Metrics Kickoff: 2003-8-5)

Copyright 2017 Cray Inc.

Various Teams' Models of Productivity

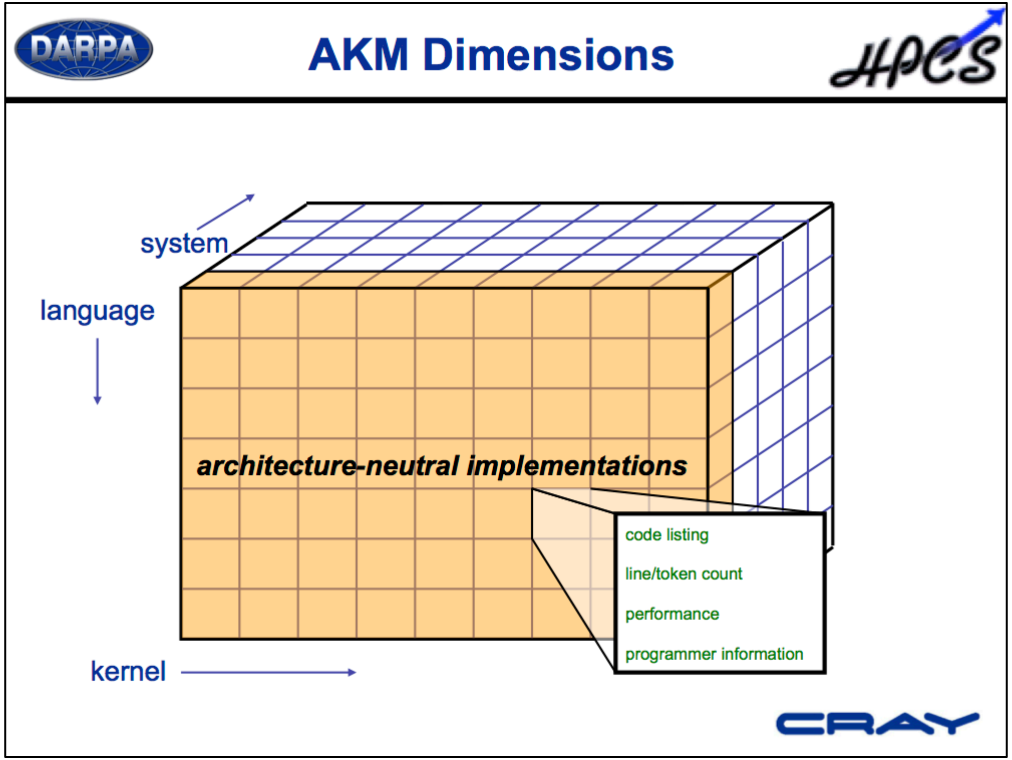
IJHPCA special issue on HPC productivity

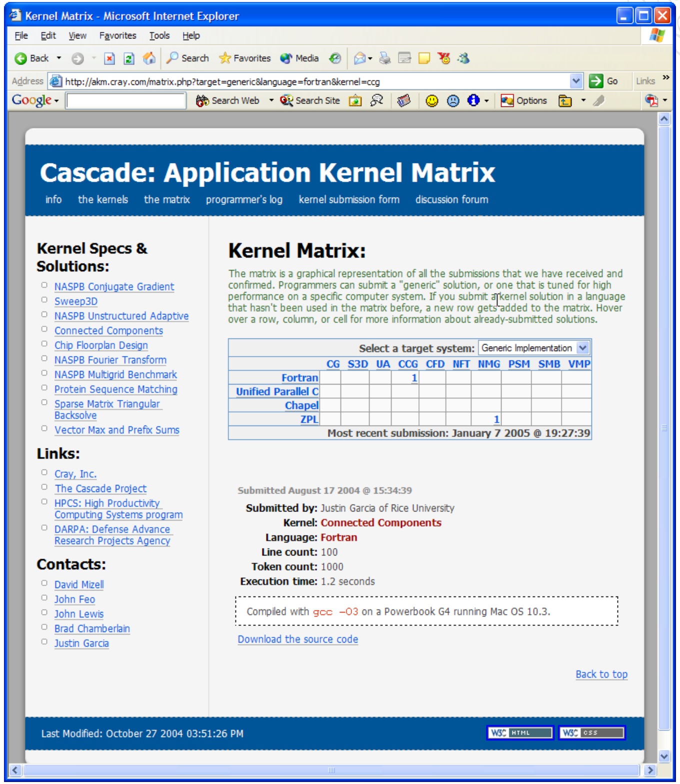
http://journals.sagepub.com/toc/hpcc/18/4



Copyright 2017 Cray Inc.

The Application Kernel Matrix




Kernel Matrix - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://akm.cray.com/matrix.php?target=generic&language=fortran&kernel=ccg

Cascade: Application Kernel Matrix

info | the kernels | the matrix | programmer's log | kernel submission form | discussion forum

Kernel Specs & Solutions:

- NASPB Conjugate Gradient
- Sweep3D
- NASPB Unstructured Adaptive
- Connected Components
- Chip Floorplan Design
- NASPB Fourier Transform
- NASPB Multigrid Benchmark
- Protein Sequence Matching
- Sparse Matrix Triangular Backsolve
- Vector Max and Prefix Sums

Links:

- Cray, Inc.
- The Cascade Project
- HPCS: High Productivity Computing Systems program
- DARPA: Defense Advance Research Projects Agency

Contacts:

- David Mizell
- John Feo
- John Lewis
- Brad Chamberlain
- Justin Garcia

Kernel Matrix:

The matrix is a graphical representation of all the submissions that we have received and confirmed. Programmers can submit a "generic" solution, or one that is tuned for high performance on a specific computer system. If you submit a kernel solution in a language that hasn't been used in the matrix before, a new row gets added to the matrix. Hover over a row, column, or cell for more information about already-submitted solutions.

Select a target system: Generic Implementation

	CG	S3D	UA	CCG	CFD	NFT	NMG	PSM	SMB	VMP
Fortran				1						
Unified Parallel C										
Chapel										
ZPL							1			

Most recent submission: January 7 2005 @ 19:27:39

Submitted August 17 2004 @ 15:34:39

Submitted by: Justin Garcia of Rice University
Kernel: Connected Components
Language: Fortran
Line count: 100
Token count: 1000
Execution time: 1.2 seconds

Compiled with gcc -O3 on a Powerbook G4 running Mac OS 10.3.

Download the source code

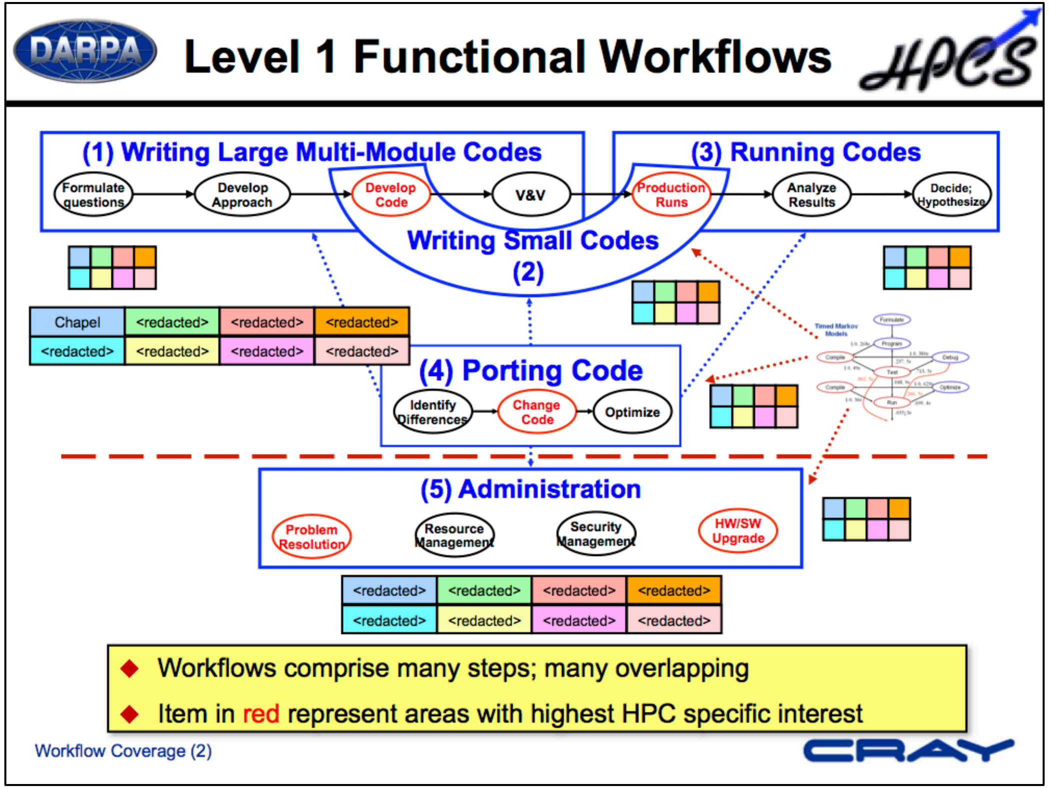
Back to top

Last Modified: October 27 2004 03:51:26 PM

(source: Marina del Mar SW Productivity Workshop, 2005?)

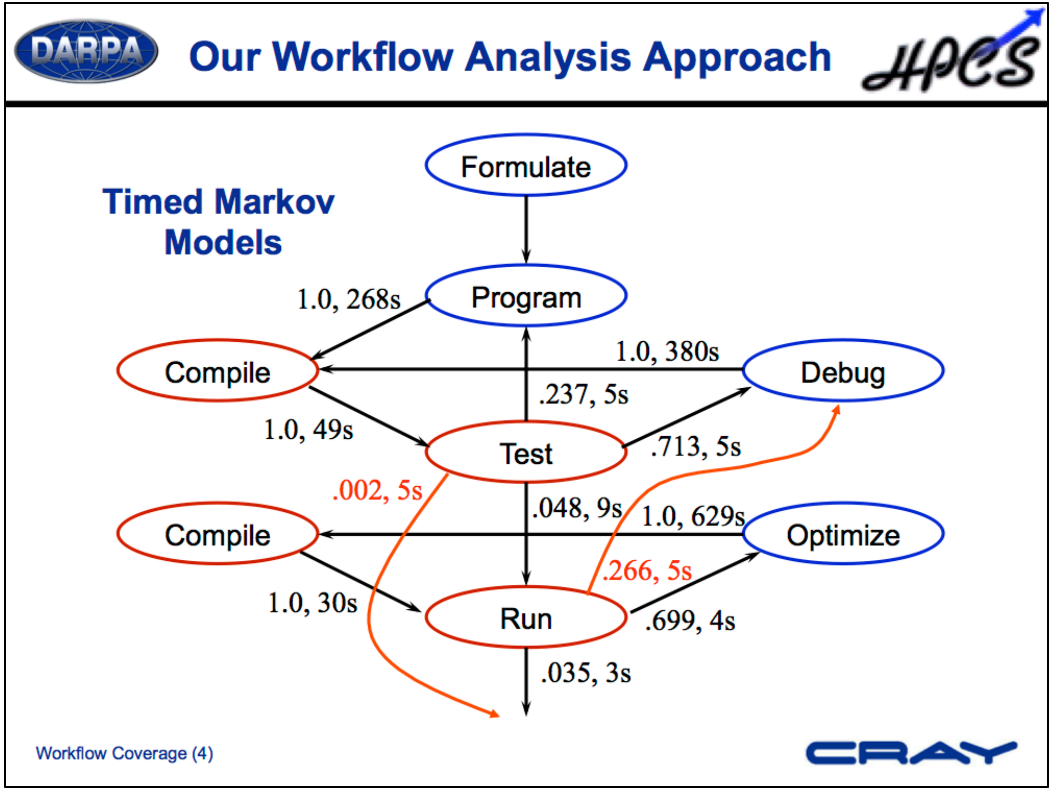
Copyright 2017 Cray Inc.

HPCS Workflows



(source: Cray government review)

Timed Markov Models

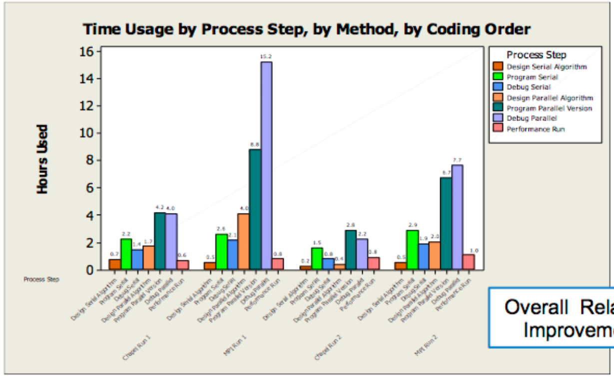


(source: Cray government review)

Copyright 2017 Cray Inc.

User Studies: Quantitative Evaluation

Workflow 2 – Productivity Improvement Summary



Overall Relative Productivity Improvement 2.3 to 2.6 X

- Debug Parallel Code
  - Programmers created less bugs when programming in Chapel
    - Global view of data simplifies programming
- Program Parallel
  - Fewer changes needed when going from serial to parallel
    - Multi-resolution of parallelism makes it easier to go from serial to parallel
- Designing Parallel Algorithm
  - MPI programs frequently require restructuring of the serial C code, finding right MPI lib routine

(source: Cray government review)

Copyright 2017 Cray Inc.

User Studies: Qualitative Evaluation

"The biggest feature from a broad perspective for me was domains. Especially for scientific codes, it is invaluable to be able to define the couple problem domains you're working with. It makes it trivial to change the size or layout or distribution if you decide you need to, it helps guarantee that all of your different arrays match up. ***A 3D rectangular grid is infinitely more clear in Chapel with domains than in C****, where you have to figure out how they laid it out (is it one giant array? what is the major dimension? x? z? y?)."*

*"****I loved not having to think as hard*** *about offsets and counts for the parallel version of the code* ***in Chapel, as opposed to the MPI version****, where I almost always had to chase down two or three indexing errors."*

"Lastly, ***I'm a huge huge fan of the type inference used in Chapel.*** *I like that I don't have to specify types everywhere--they can just be inferred from how I'm using them, but if I mess something up, the compiler catches it."*

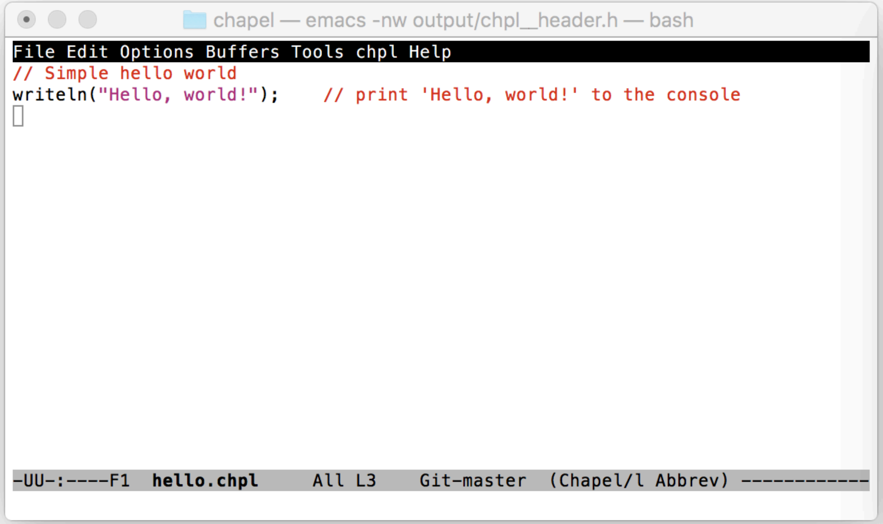
Copyright 2017 Cray Inc.

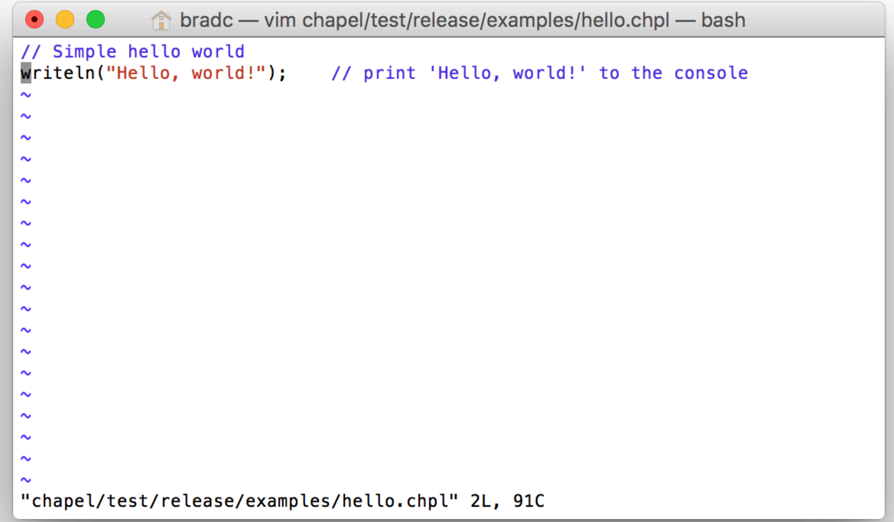
Summary: Many Useful Concepts/Techniques…

BUT…

Copyright 2017 Cray Inc.

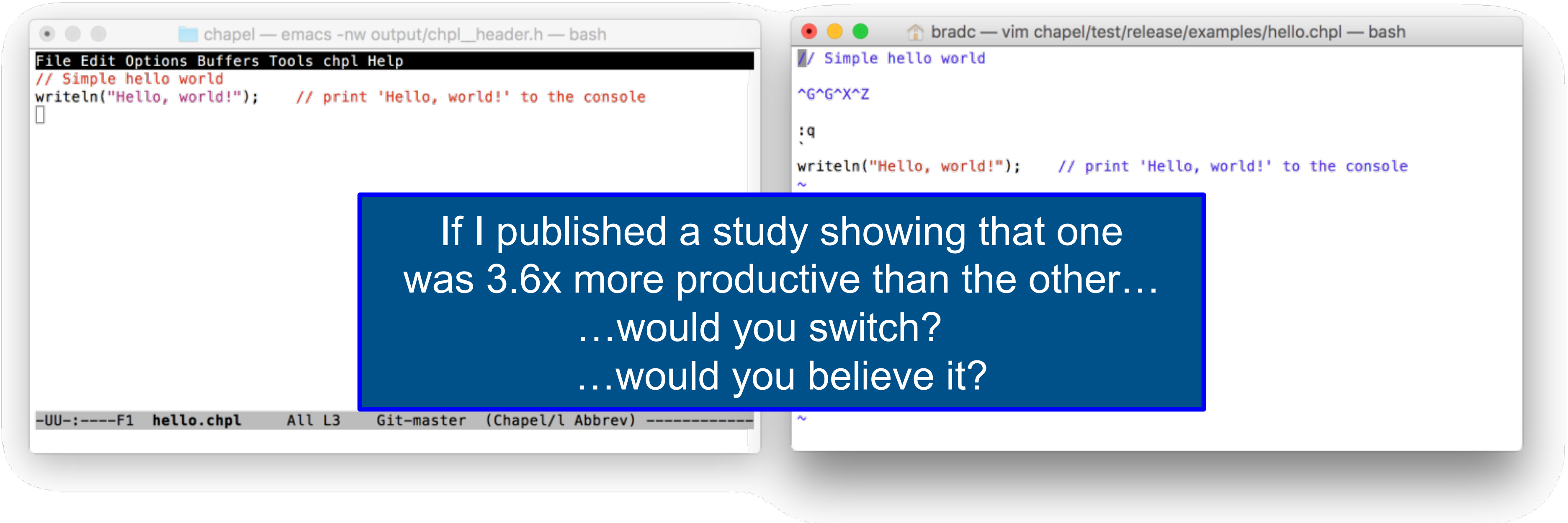
Which is more productive?





Copyright 2017 Cray Inc.

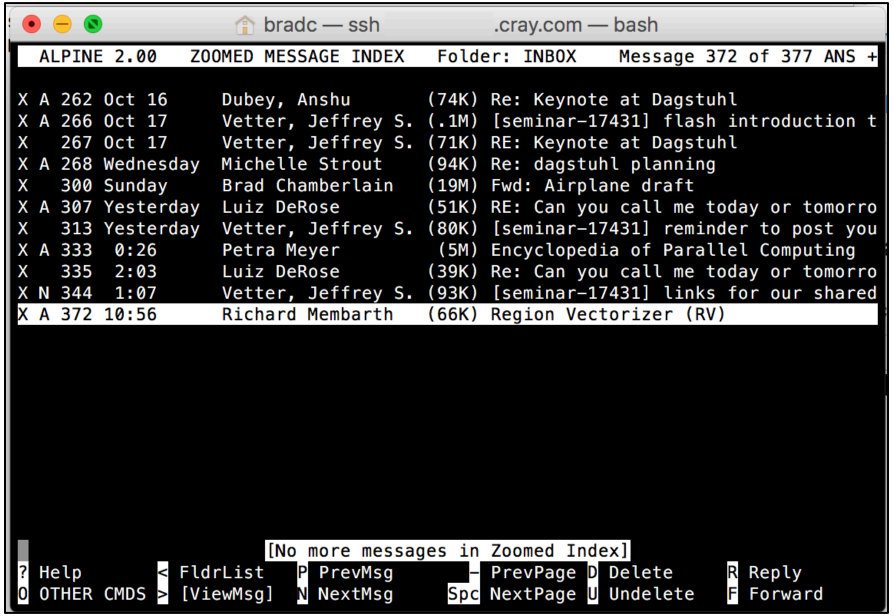
Which is more productive? (tipping my hand)

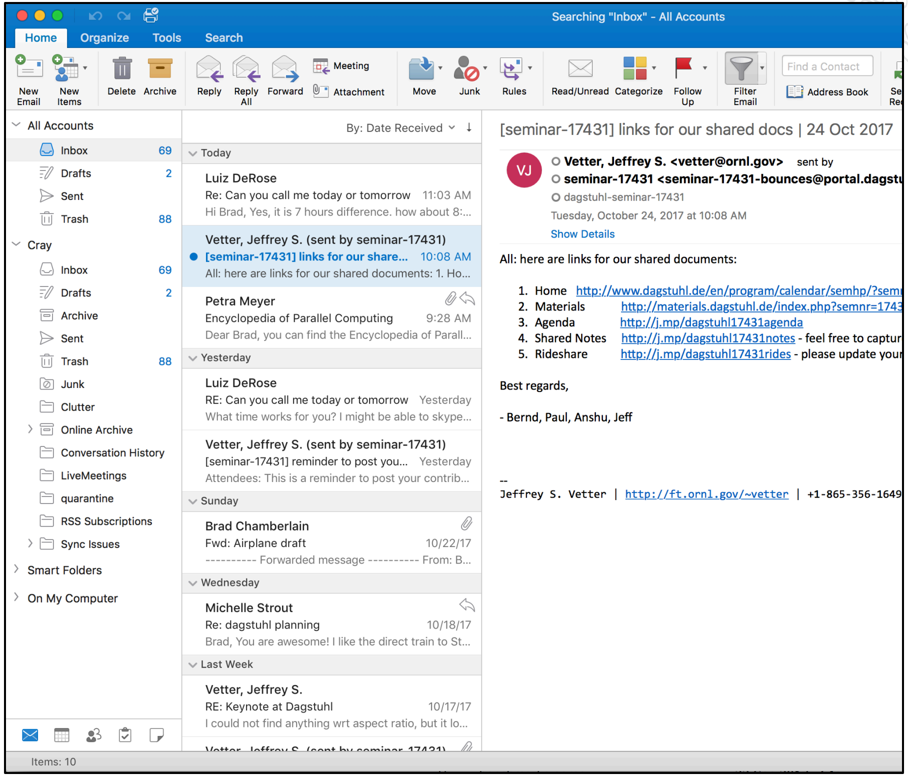


If I published a study showing that one
was 3.6x more productive than the other…
…would you switch?
…would you believe it?

Copyright 2017 Cray Inc.

Which is more productive?





Copyright 2017 Cray Inc.

Productivity: Your Mileage May Vary

- **Productivity is a highly personal, social phenomenon**
  - "I'll know it when I see it"
- **To that end, our evaluations should be social, not analytic**
  - Support personal weighing of tradeoffs
  - Allocate time and spaces for evaluating potential solutions

Copyright 2017 Cray Inc.

Poll: How many are familiar with the Computer Language Benchmarks Game?

Copyright 2017 Cray Inc.

Computer Language Benchmarks Game (CLBG)

The Computer Language Benchmarks Game

64-bit quad core data set

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

Which programs are fast?

Which are succinct? Which are efficient?

Ada C Chapel C# C++ Dart

Erlang F# Fortran Go Hack

Haskell Java JavaScript Lisp Lua

OCaml Pascal Perl PHP Python

Racket Ruby JRuby Rust Smalltalk

Swift TypeScript

{ for researchers } fast-faster-fastest

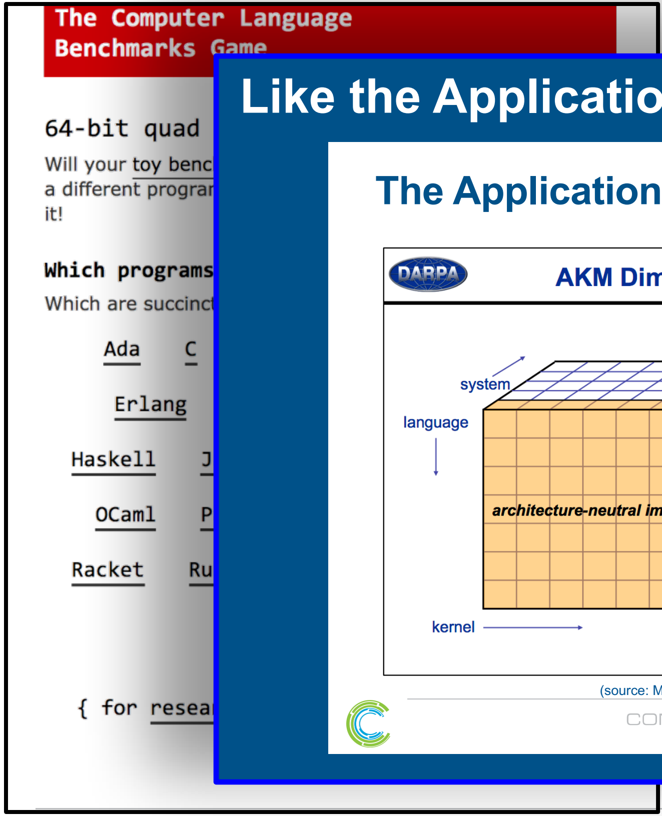
stories

Website supporting cross-language comparisons

- 13 toy benchmark programs x ~28 languages x several entries
  - exercise key computational idioms
  - specific approach prescribed

Copyright 2017 Cray Inc.

Computer Language Benchmarks Game (CLBG)



Website supporting cross-

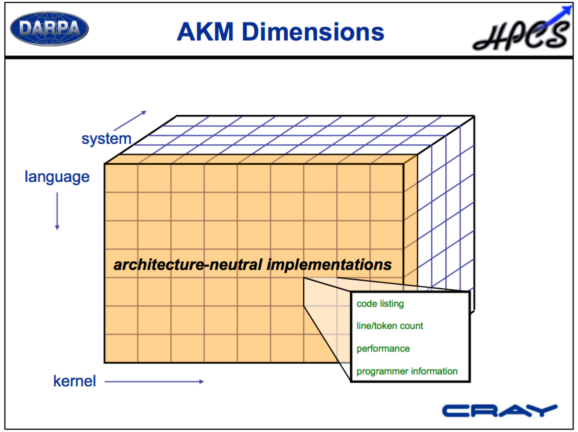
x

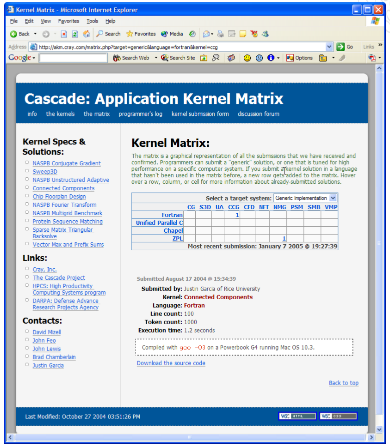
entations

dioms

Like the Application Kernel Matrix in several respects…

The Application Kernel Matrix





(source: Marina del Mar SW Productivity Workshop, 2005?)

COMPUTE | STORE | ANALYZE

Copyright 2017 Cray Inc.

128

Copyright 2017 Cray Inc.

Computer Language Benchmarks Game (CLBG)

The Computer Language Benchmarks Game

64-bit quad core data set

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

Which programs are fast?

Which are succinct? Which are efficient?

Ada C Chapel C# C++ Dart

Erlang F# Fortran Go Hack

Haskell Java JavaScript Lisp Lua

OCaml Pascal Perl PHP Python

Racket Ruby JRuby Rust Smalltalk

Swift TypeScript

{ for researchers } fast-faster-fastest

stories

Website supporting cross-language comparisons

- 13 toy benchmark programs x ~28 languages x many implementations
  - exercise key computational idioms
  - specific approach prescribed

Take results with a grain of salt

- your mileage may vary

That said, it is one of the only such games in town…

Copyright 2017 Cray Inc.

Computer Language Benchmarks Game (CLBG)

The Computer Language Benchmarks Game

64-bit quad core data set

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

Which programs are fast?

Which are succinct? Which are efficient?

Ada C Chapel C# C++ Dart

Erlang F# Fortran Go Hack

Haskell Java JavaScript Lisp Lua

OCaml Pascal Perl PHP Python

Racket Ruby JRuby Rust Smalltalk

Swift TypeScript

{ for researchers } fast-faster-fastest

stories

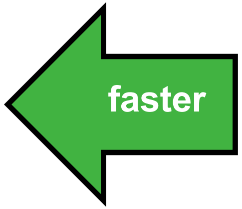
Chapel's approach to the CLBG:

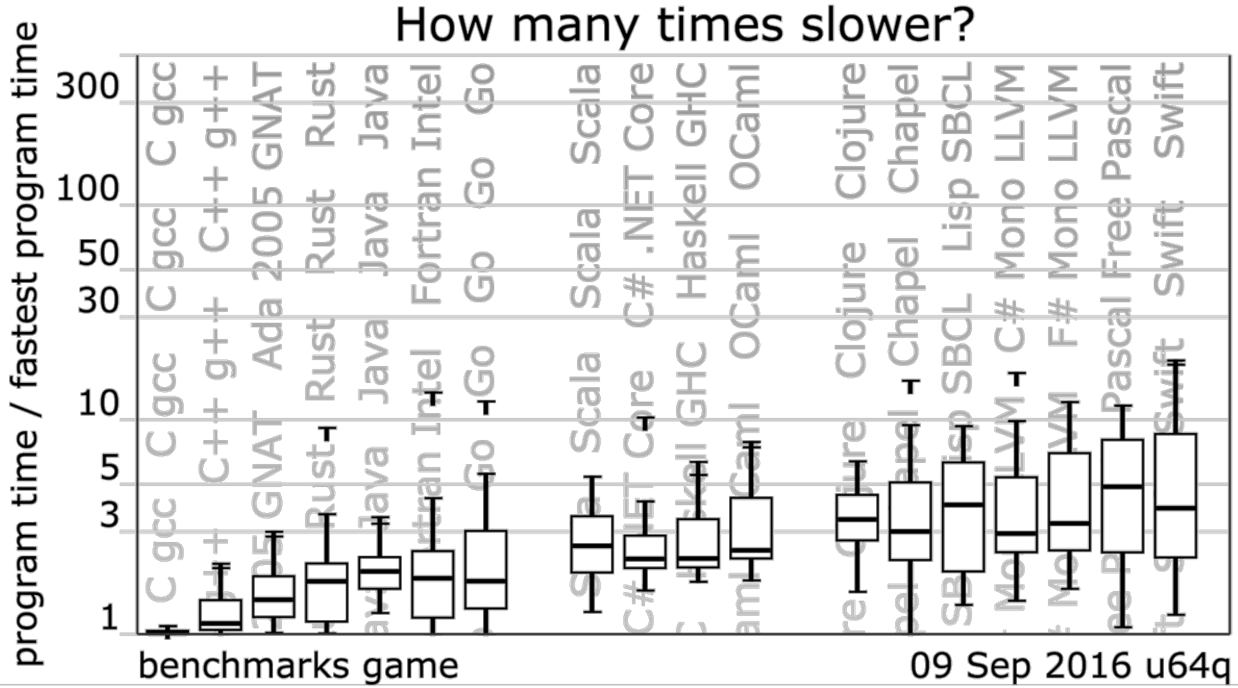
- striving for elegance over heroism
  - ideally: "Want to learn how program *xyz* works? Read the Chapel version."

Copyright 2017 Cray Inc.

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

Relative performance, sorted by geometric mean

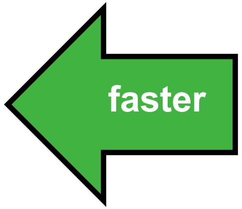


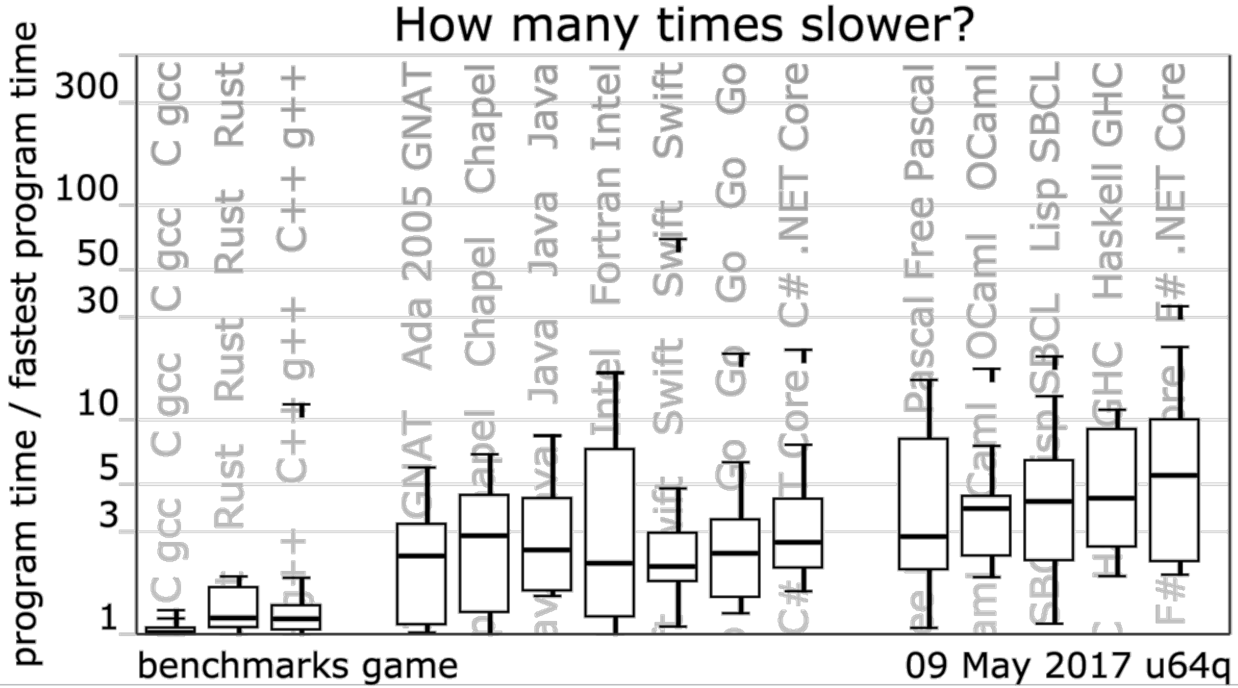


Copyright 2017 Cray Inc.

CLBG: Fast-faster-fastest graph (May 2017)

Relative performance, sorted by geometric mean

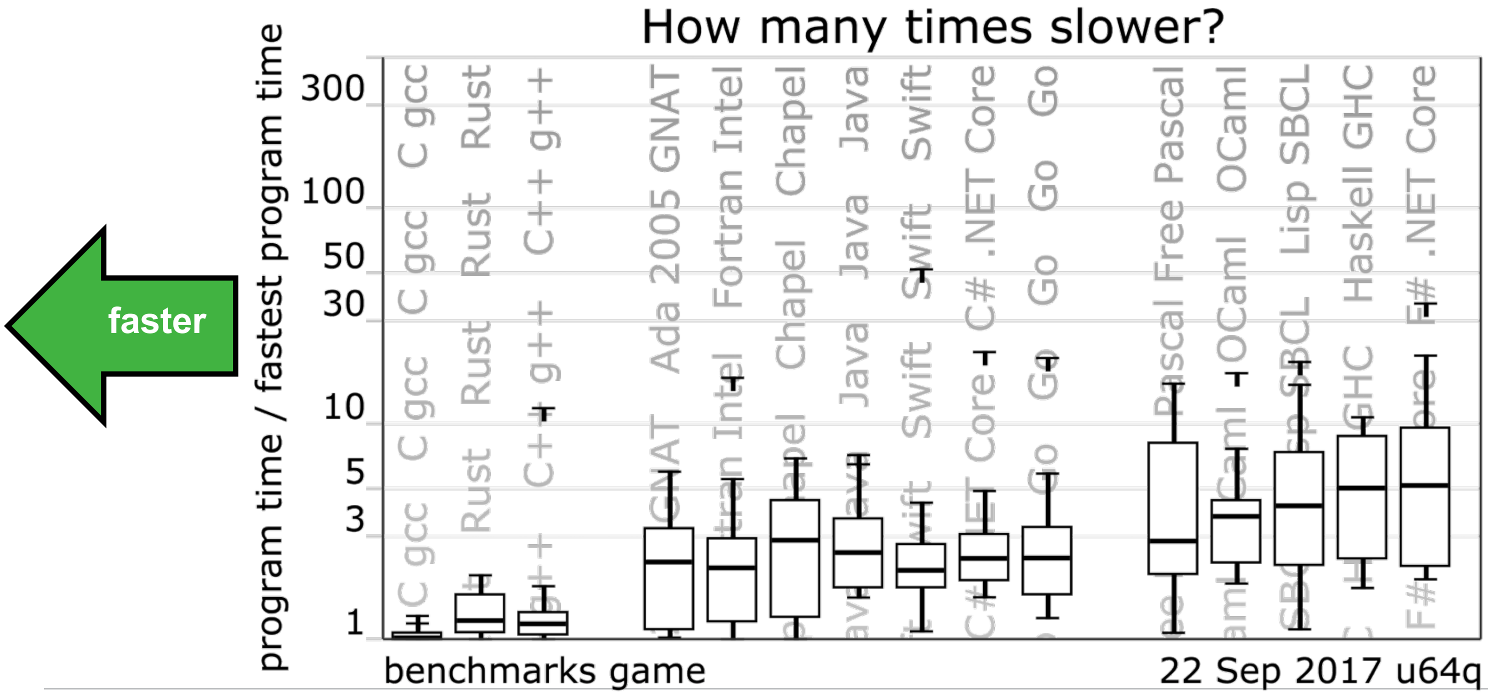




Copyright 2017 Cray Inc.

CLBG: Fast-faster-fastest graph (Sept 2017)

Relative performance, sorted by geometric mean



Copyright 2017 Cray Inc.

CLBG: Website

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

The Computer Language Benchmarks Game

pidigits

description

program source code, command-line and measurements

×	source	secs	mem	gz	cpu	cpu load
1.0	**Chapel** #2	**1.62**	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

×	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	477,092	**478**	3.12	0% 100% 2% 1%
1.8	Lua #5					% 1% 1% 99%
1.8	**Pascal** Free Pas					% 100% 1% 1%
1.9	**Lisp** SBCL #3					% 1% 100% 1%
1.9	PHP #3					0% 0% 0% 1%

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

Copyright 2017 Cray Inc.

CLBG: Website

Can also compare languages pair-wise:

- but only sorted by execution speed…

The Computer Language Benchmarks Game

Chapel programs versus Fortran Intel

all other Chapel programs & measurements

by benchmark task performance

k-nucleotide

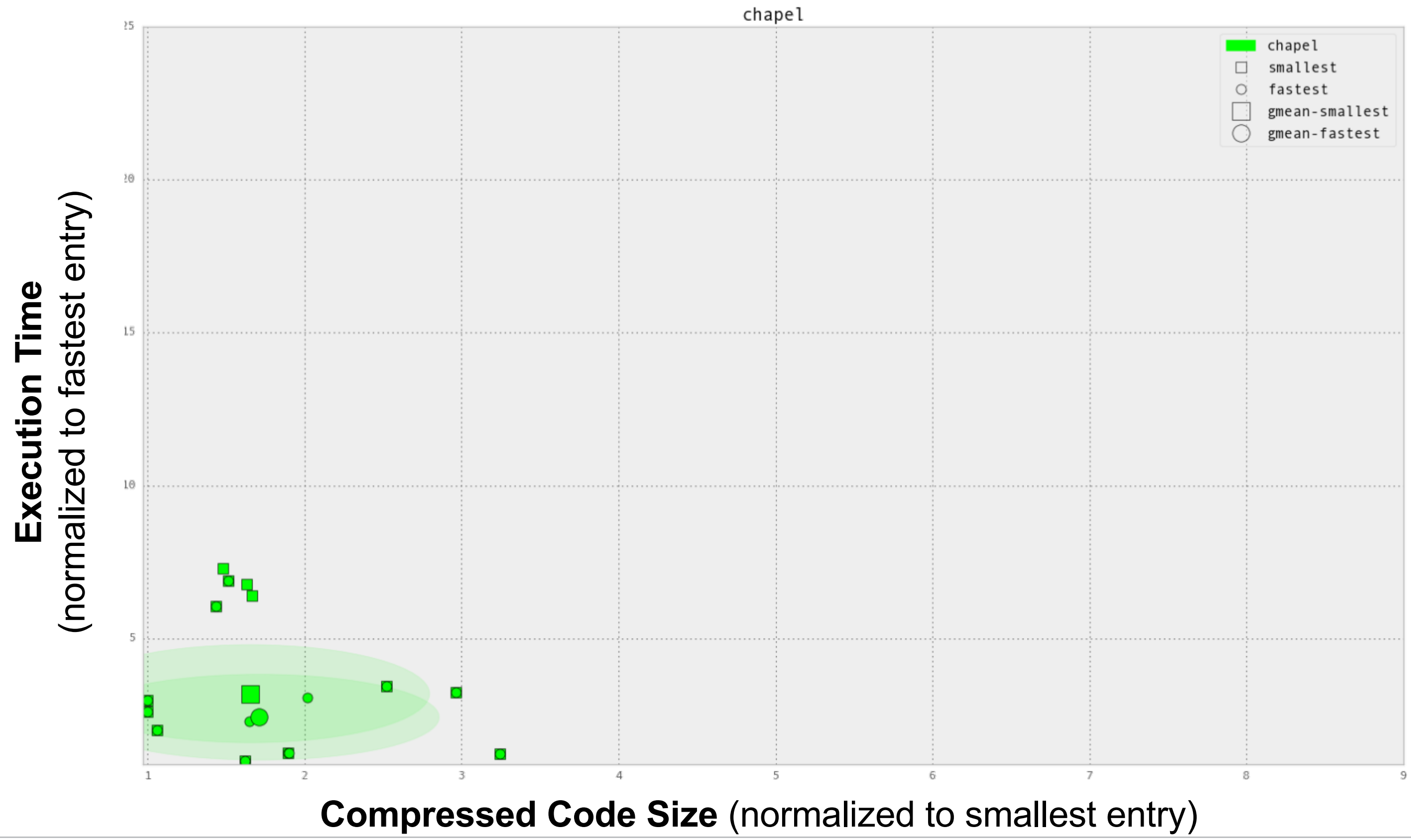
source	secs	mem	gz	cpu	cpu load
Chapel	**16.69**	350,432	1063	62.96	100% 92% 93% 93%
Fortran Intel	87.62	203,604	2238	87.57	1% 0% 100% 0%

fasta

source	secs	mem	gz	cpu	cpu load
Chapel	**1.71**	52,184	1392	5.90	99% 82% 83% 82%
Fortran Intel	2.53	8	1327	2.53	0% 1% 0% 100%

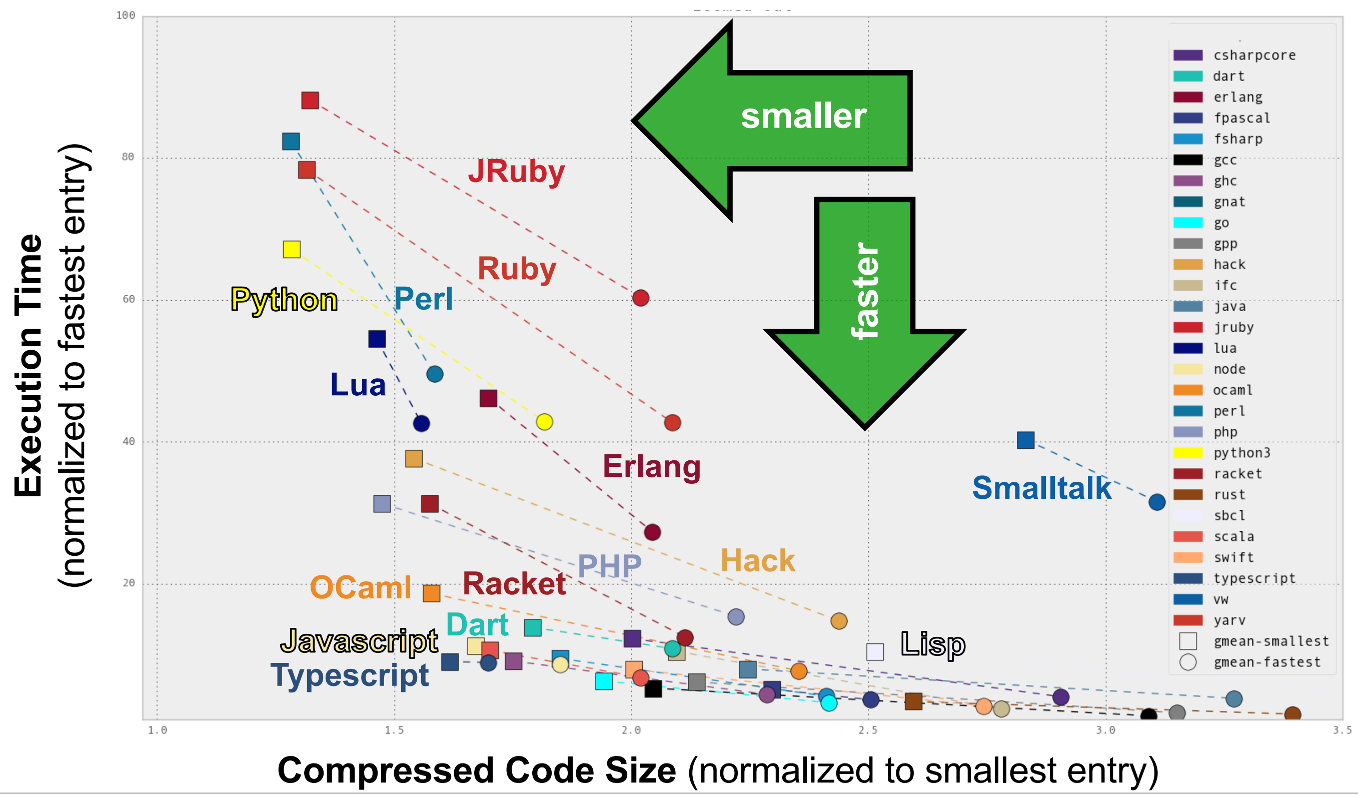
Copyright 2017 Cray Inc.

Scatter plots of CLBG code size x speed



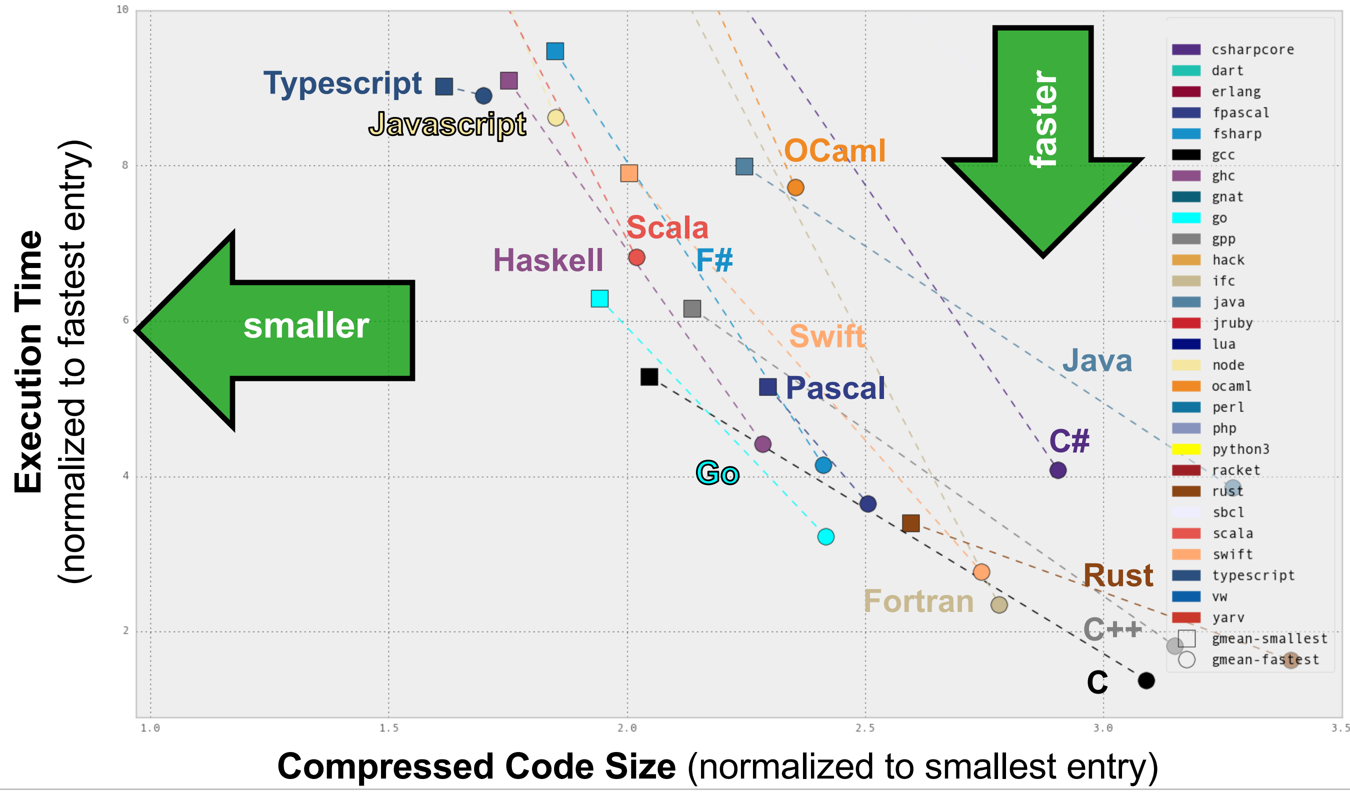
Copyright 2017 Cray Inc.

CLBG Language Cross-Language Summary
(Oct 2017 standings)



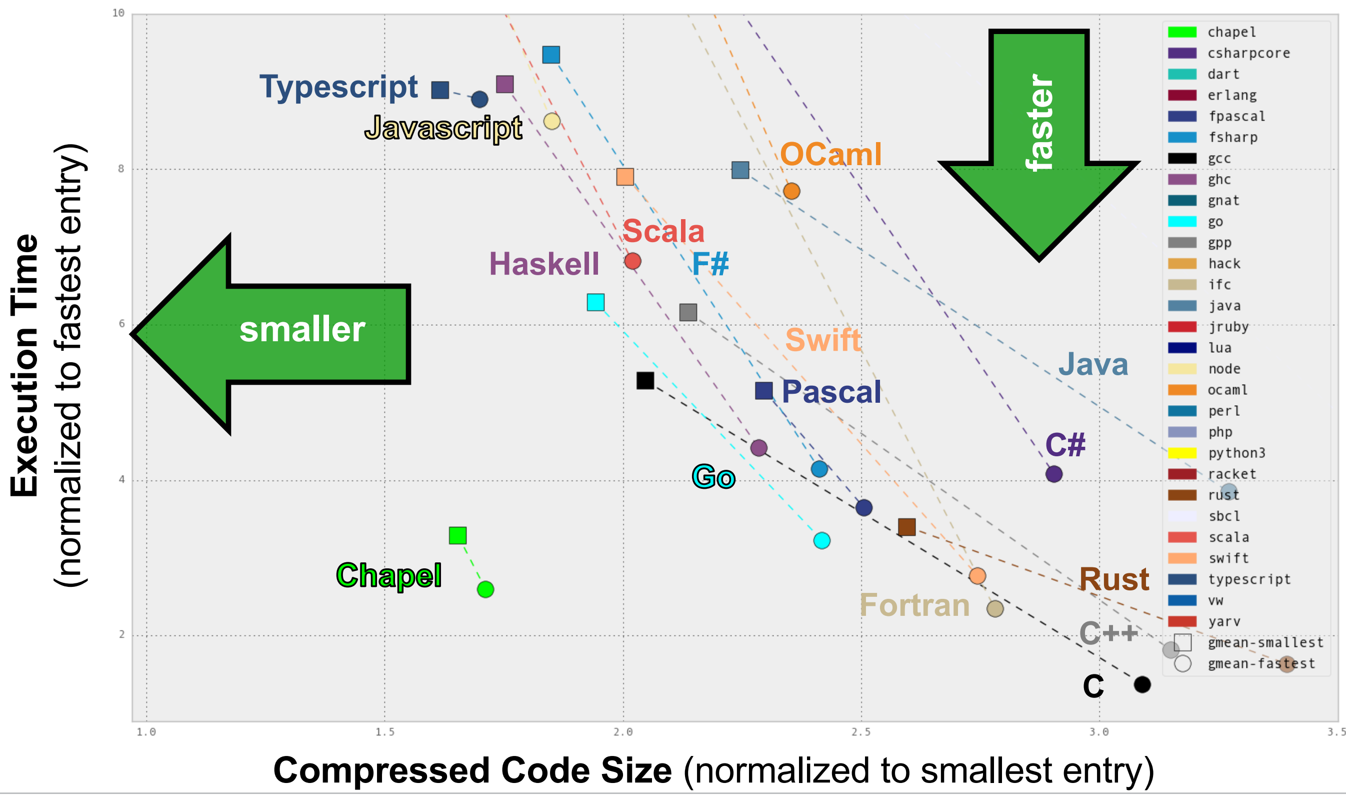
Copyright 2017 Cray Inc.

CLBG Language Cross-Language Summary
(Oct 2017 standings, zoomed in)



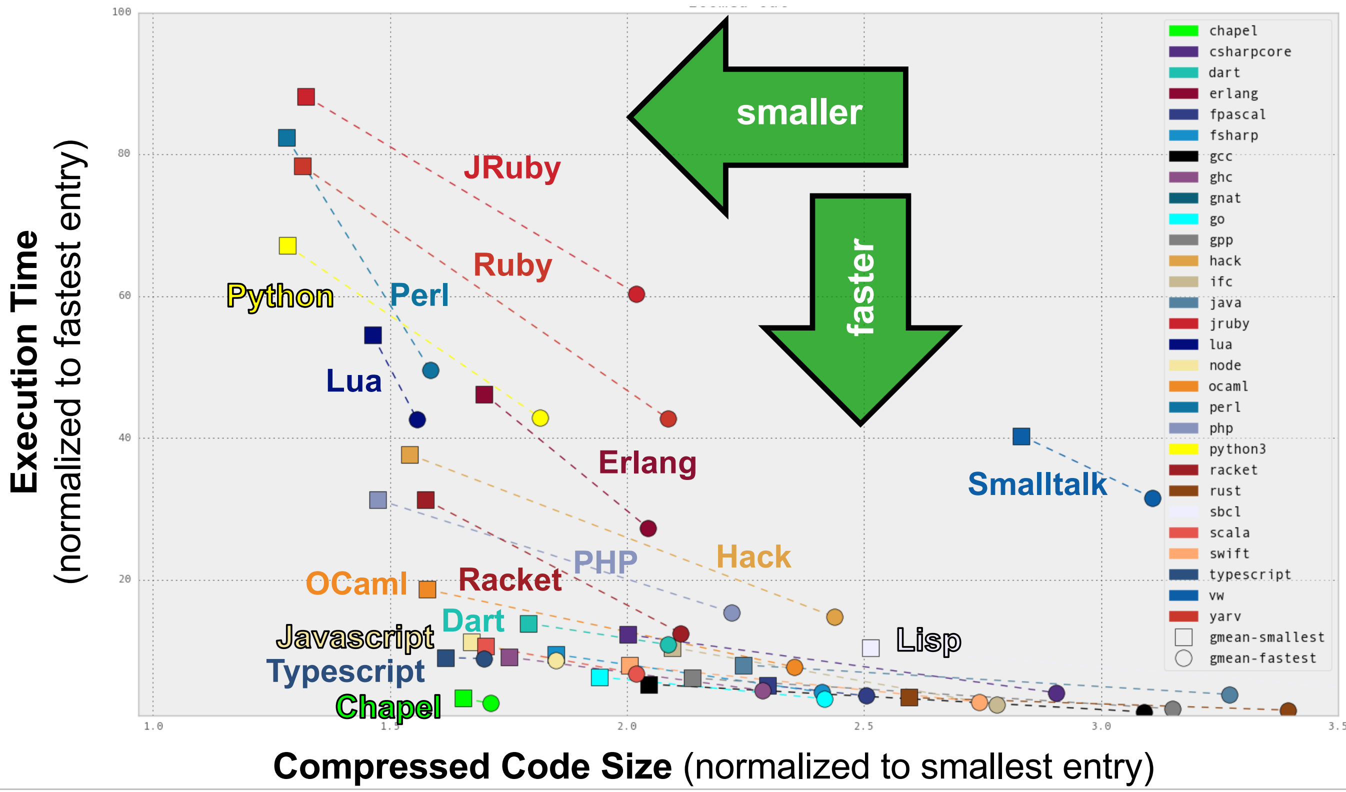
Copyright 2017 Cray Inc.

CLBG Language Cross-Language Summary
(Oct 2017 standings, zoomed in)



Copyright 2017 Cray Inc.

CLBG Language Cross-Language Summary
(Oct 2017 standings)



Copyright 2017 Cray Inc.

CLBG: Qualitative Comparisons

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

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

  const group1 = [i in 1..popSize1] new Chameneos(i, ((i-1)%3):Color);
  const group2 = [i in 1..popSize2] new Chameneos(i, colors10[i]);

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

  print(group1);
  print(group2);

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


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


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

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

  delete place;
}
```

excerpt from 1210 gz Chapel entry

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

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

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

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

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

excerpt from 2863 gz C gcc entry

Copyright 2017 Cray Inc.

CLBG: Qualitative Comparisons

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

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

  const group1 = [i in 1..popSize1] new Chameneos(i, (
  const group2 = [i in 1..popSize2] new Chameneos(i, c

  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 pa
//
proc printColorEquations() {
  for c1 in Color do
    for c2 in Color do
      writeln(c1, " + ", c2, " -> ", getNewColor(c1, c
  writeln();
}

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

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

  delete place;
}
```

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

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

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

  delete place;
}
```

excerpt from 1210 gz Chapel entry

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

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

excerpt from 2863 gz C gcc entry

Copyright 2017 Cray Inc.

CLBG: Qualitative Comparisons

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

```
proc main() {
```

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

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

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

excerpt from 1210 gz Chapel entry

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

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

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

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

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

excerpt from 2863 gz C gcc entry

Copyright 2017 Cray Inc.

Benchmark Suite Scorecard (EMBRACE 2017)

	HPC user interest	clear paper & pencil	fast reference code	clear reference code	forum for comparison	productivity framework	prescribed approach	open competition	historical continuum	apples-to-apples	self-serve comparisons	self-run entries
NPB												
HPCC												
DOE PROXY APPS												
CLBG	✘	✘	✓	✓	✓	✓	✓	~	✓	✓	✓	✓
PRK												

Copyright 2017 Cray Inc.

Benchmark Suite Scorecard (EMBRACE 2017)

	HPC user interest	clear paper & pencil	fast reference code	clear reference code	forum for comparison	productivity framework	prescribed approach	open competition	historical continuum	apples-to-apples	self-serve comparisons	self-run entries
NPB	✓	~	✓	~	✗	✗	✗					
HPCC	~	~	✓	✗	✓	✓	~	✓	✗	✗	✗	✗
DOE PROXY APPS	✓	~	✓	~	✗	✗	~					
CLBG	✗	✗	✓	✓	✓	✓	✓	~	✓	✓	✓	✓
PRK	~	✓	✓	?	~	✗	✓	✓				✓

Copyright 2017 Cray Inc.

My Vision for Advancing Productive Alternatives

1. Establish an ongoing, online language bake-off

- The “HPC Computer Language Benchmarks Game”
- Support personal comparisons between technologies
- **Challenges:**
  - Finding someone to be the enthusiastic host and benevolent dictator
  - What systems to run on? and whose?
  - What benchmarks? Intel PRK suite?
  - How to manage entries? What rules? How reviewed?
  - Website design and implementation

Copyright 2017 Cray Inc.

Scalable Parallel Programming Concerns

Q: What should parallel programmers focus on?

A: *Serial Code:* Software engineering and performance

Parallelism: What should execute simultaneously?

Locality: Where should those tasks execute?

Mapping: How to map the program to the system?

For portable performance?

Separation of Concerns: (drink!)

Copyright 2017 Cray Inc.

Inspiration from Kathy Yelick (UC Berkeley, LBNL)

Why Consider New Languages at all?

- **Do we need a language? And a compiler?**
  - If higher-level syntax is needed for productivity
    - We need a language
  - If static analysis is needed to help with correctness
    - We need a compiler (front-end)
  - If static optimizations are needed to get performance
    - We need a compiler (back-end)

(Source: HPCS productivity workshop panel, ~2004?)

Copyright 2017 Cray Inc.

Poll: Familiar with Chapel?

Copyright 2017 Cray Inc.

Poll: Familiar with Chapel?
Have opinions about Chapel?

Copyright 2017 Cray Inc.

Poll: Familiar with Chapel?
Have opinions about Chapel?
Have downloaded / tried Chapel?

Copyright 2017 Cray Inc.

Poll: Familiar with Chapel?
Have opinions about Chapel?
Have downloaded / tried Chapel?
Within the past 12–18 months?

Copyright 2017 Cray Inc.

What is Chapel?

Chapel: A productive parallel programming language

- portable
- open-source
- a collaborative effort

Goals:

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

Copyright 2017 Cray Inc.

What does "Productivity" mean to you?

Recent Graduates:

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

Seasoned HPC Programmers:

"that sugary stuff that I don't need because I ~~was born to suffer~~"
want full control to ensure performance"

Computational Scientists:

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

Chapel Team:

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

Copyright 2017 Cray Inc.

Chapel and Productivity

Chapel strives to be…

…as programmable as Python

…as fast as Fortran

…as scalable as MPI (or SHMEM or UPC)

…as portable as C

…as flexible as C++

…as fun as [your favorite programming language]

Copyright 2017 Cray Inc.

The Chapel Team at Cray (May 2017)

Copyright 2017 Cray Inc.

Chapel Community R&D Efforts

HERIOT WATT UNIVERSITY

AMD

THE GEORGE WASHINGTON UNIVERSITY WASHINGTON, DC

WESTERN WASHINGTON UNIVERSITY

東京大学 THE UNIVERSITY OF TOKYO

Colorado State University

RICE

UNIVERSITY OF MARYLAND

BERKELEY LAB

Lawrence Berkeley National Laboratory

Lawrence Livermore National Laboratory

Sandia National Laboratories

DEPARTMENT OF DEFENSE UNITED STATES OF AMERICA

Yale

(and several others…)

http://chapel.cray.com/collaborations.html

Copyright 2017 Cray Inc.

The Challenge

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

A: ~~Technical challenges?~~

- while they exist, we don’t think this is the main issue…

A: Due to a lack of…

…long-term efforts

…resources

…community will

…co-design between developers and users

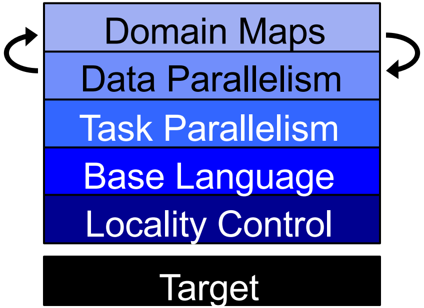
…patience

Chapel is our attempt to reverse this trend

Copyright 2017 Cray Inc.

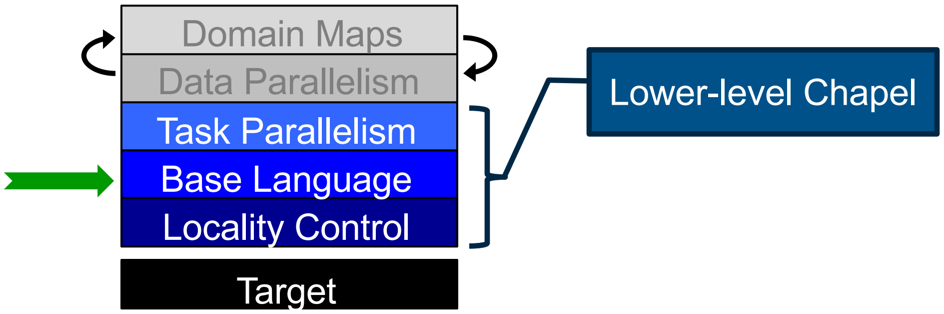
Chapel language feature areas

Chapel language concepts



Copyright 2017 Cray Inc.

Base Language



Copyright 2017 Cray Inc.

Base Language Features, by example

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

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

```
config const n = 10;

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

```
0
1
1
2
3
5
8
…
```


Copyright 2017 Cray Inc.

Base Language Features, by example

Modern iterators

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

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

```
config const n = 10;

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

```
0
1
1
2
3
5
8
…
```


Copyright 2017 Cray Inc.

Base Language Features, by example

Configuration declarations
(to avoid command-line argument parsing)
`./a.out --n=1000000`

```
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
…
```


Copyright 2017 Cray Inc.

Base Language Features, by example

Static type inference for:
- arguments
- return types
- variables

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

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

```
config const n = 10;

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

```
0
1
1
2
3
5
8
…
```


Copyright 2017 Cray Inc.

Base Language Features, by example

Zippered iteration

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

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

```
config const n = 10;

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

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


Copyright 2017 Cray Inc.

Base Language Features, by example

Range types and operators

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

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

```
config const n = 10;

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

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


Copyright 2017 Cray Inc.

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
…
```


Copyright 2017 Cray Inc.

Base Language Features, by example

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

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

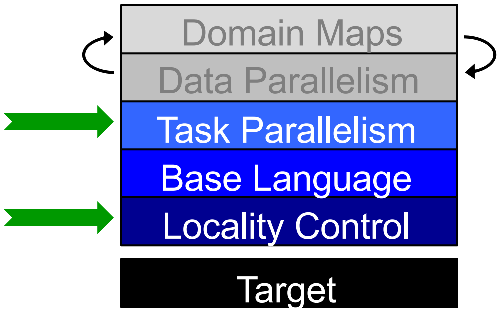
```
config const n = 10;

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

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


Copyright 2017 Cray Inc.

Task Parallelism and Locality Control



Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

taskParallel.chpl

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

```
prompt> chpl taskParallel.chpl -o taskParallel
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
```


Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

Abstraction of System Resources

taskParallel.chpl

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

```
prompt> chpl taskParallel.chpl -o taskParallel
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
```


Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

High-Level
Task Parallelism

taskParallel.chpl

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

```
prompt> chpl taskParallel.chpl -o taskParallel
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
```


Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

taskParallel.chpl

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

Control of Locality/Affinity

```
prompt> chpl taskParallel.chpl -o taskParallel
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
```


Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

Abstraction of System Resources

taskParallel.chpl

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

```
prompt> chpl taskParallel.chpl -o taskParallel
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
```


Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

High-Level
Task Parallelism

taskParallel.chpl

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

```
prompt> chpl taskParallel.chpl -o taskParallel
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
```


Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

Not seen here:

Data-centric task coordination via atomic and full/empty vars

taskParallel.chpl

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

```
prompt> chpl taskParallel.chpl -o taskParallel
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
```


Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

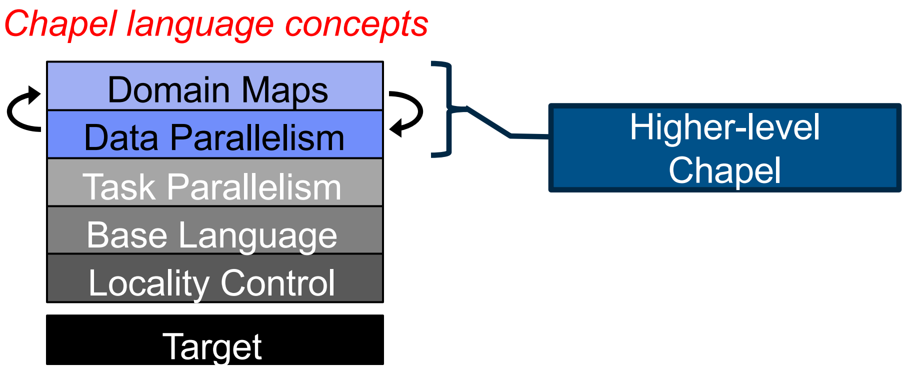
taskParallel.chpl

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

```
prompt> chpl taskParallel.chpl -o taskParallel
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
```


Copyright 2017 Cray Inc.

Higher-Level Features



Copyright 2017 Cray Inc.

Data Parallelism, by example

dataParallel.chpl

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

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

```
prompt> chpl dataParallel.chpl -o dataParallel
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
```


Copyright 2017 Cray Inc.

Data Parallelism, by example

Domains (Index Sets)

dataParallel.chpl

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

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

```
prompt> chpl dataParallel.chpl -o dataParallel
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
```


Copyright 2017 Cray Inc.

Data Parallelism, by example

Arrays

dataParallel.chpl

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

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

```
prompt> chpl dataParallel.chpl -o dataParallel
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
```


Copyright 2017 Cray Inc.

Data Parallelism, by example

Data-Parallel Forall Loops

dataParallel.chpl

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

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

```
prompt> chpl dataParallel.chpl -o dataParallel
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
```


Copyright 2017 Cray Inc.

Distributed Data Parallelism, by example

Domain Maps
(Map Data Parallelism to the System)

dataParallel.chpl

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

```
prompt> chpl dataParallel.chpl -o dataParallel
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
```


Copyright 2017 Cray Inc.

Distributed Data Parallelism, by example

dataParallel.chpl

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

```
prompt> chpl dataParallel.chpl -o dataParallel
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
```


Copyright 2017 Cray Inc.

Distributed Data Parallelism, by example

magic?

HPF-like?

descriptive?

Not in the slightest...

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 -o dataParallel
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
```


Copyright 2017 Cray Inc.

Distributed Data Parallelism, by example

Chapel's prescriptive approach:

```
forall (i,j) in D do…
```

⇒ invoke and inline D's default parallel iterator

- defined by D's type / domain map

default domain map

- create task per local core
- block indices across tasks

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 -o dataParallel
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
```


Copyright 2017 Cray Inc.

Distributed Data Parallelism, by example

Chapel's prescriptive approach:

```
forall (i,j) in D do…
```

⇒ invoke and inline D's default parallel iterator

- defined by D's type / domain map

default domain map

cyclic domain map

on each target locale…

- create task per core
- block local indices across tasks

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 -o dataParallel
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
```


Copyright 2017 Cray Inc.

Distributed Data Parallelism, by example

Chapel's prescriptive approach:

```
forall (i,j) in D do…
```

What if I don't like D's iteration strategy?

domain map

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;
```

- Write and call your own parallel iterator:

```
forall (i,j) in myParIter(D) do…
```

- Or, use a different domain map:

```
var D = {1..n, 1..n} dmapped Block(…);
```

- Or, write and use your own domain map:

```
var D = {1..n, 1..n} dmapped MyDomMap(…);
```

Domain Maps specify…
…mapping of indices to locales
…layout of domains / arrays in memory
…parallel iteration strategies
…core operations on arrays / domains

Copyright 2017 Cray Inc.

Chapel and Performance Portability

- **Avoid locking key policy decisions into the language**
  - Array memory layout?
  - Sparse storage format?
  - Parallel loop policies?
  - Abstract node architecture?

Copyright 2017 Cray Inc.

Chapel and Performance Portability

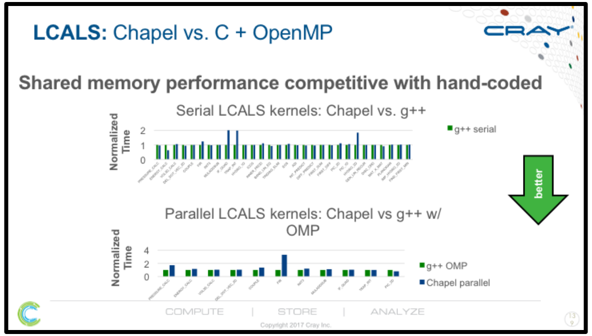
- **Avoid locking key policy decisions into the language**
  - Array memory layout? **not defined by Chapel**
  - Sparse storage format? **not defined by Chapel**
  - Parallel loop policies? **not defined by Chapel**
  - Abstract node architecture? **not defined by Chapel**
- **Instead, permit users to specify these in Chapel itself**
  - support performance portability through…

    …a separation of concerns (drink!)

    …abstractions—known to the compiler, and therefore optimizable
  - goal: to make Chapel a future-proof language

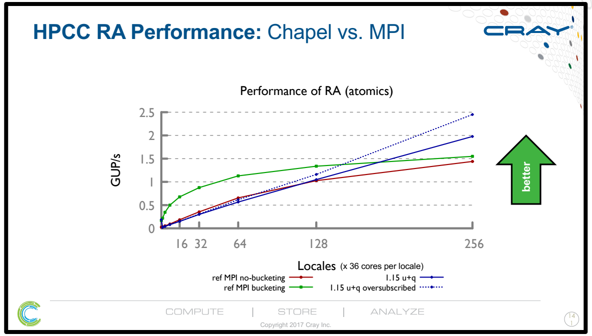
Copyright 2017 Cray Inc.

Chapel Performance: Increasingly Competitive
(novel for Chapel; new within the past 12–18 months)



LCALS

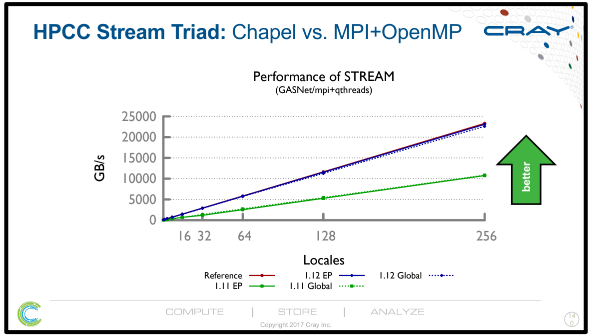
HPCC RA

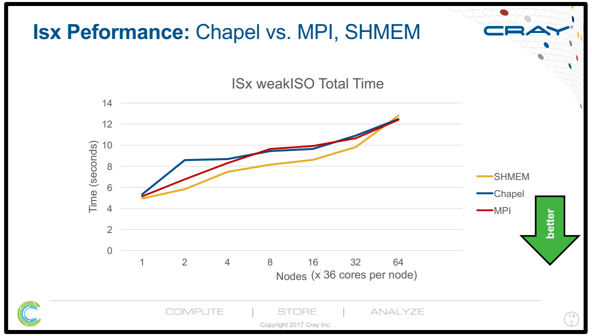


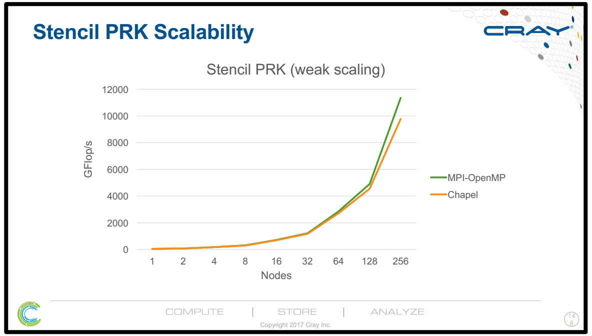
STREAM Triad

ISx

PRK Stencil







Nightly performance tickers online at:
http://chapel.sourceforge.net/perf/

Copyright 2017 Cray Inc.

Crossing the Rapids

Copyright 2017 Cray Inc.
source: http://feelgrafix.com/813578-free-stream-wallpaper.html

My Vision for Advancing Productive Alternatives

1. Establish an ongoing online bake-off
2. **Create forums for apps-languages pair-programming**
   - e.g., host a “speed-dating” Dagstuhl Seminar
     - *n* productive language groups
     - *n* apps groups looking for alternatives
       - where *n* = 3–5?
     - session 0: everyone gives lightning summaries of their language / app
     - sessions 1–n: rotate apps x language groups

meanwhile, we’re interested in doing this anytime

(so, send open-minded apps groups our way)

Copyright 2017 Cray Inc.

excerpt 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 for solving scientific problems 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

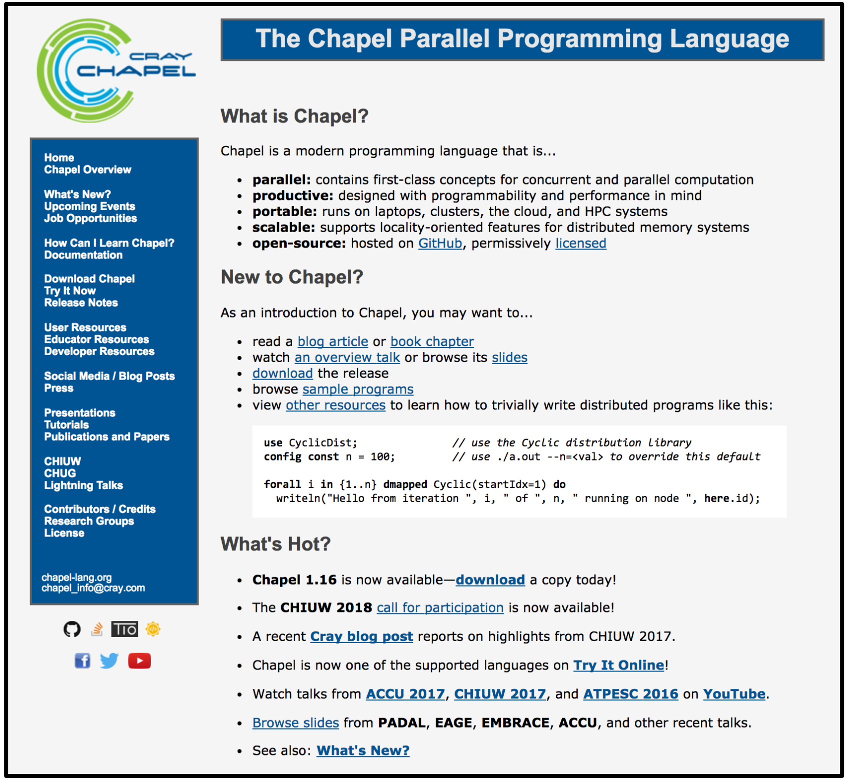
http://chapel.cray.com/presentations.html / https://www.youtube.com/watch?v=xj0rwdLOR4U

Copyright 2017 Cray Inc.

Chapel Resources

Copyright 2017 Cray Inc.

Chapel Central: http://chapel-lang.org



The Chapel Parallel Programming Language

Home
Chapel Overview

What's New?
Upcoming Events
Job Opportunities

How Can I Learn Chapel?
Documentation

Download Chapel
Try It Now
Release Notes

User Resources
Educator Resources
Developer Resources

Social Media / Blog Posts
Press

Presentations
Tutorials
Publications and Papers

CHIUW
CHUG
Lightning Talks

Contributors / Credits
Research Groups
License

chapel-lang.org
chapel_info@cray.com

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 ./a.out --n=<val> 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 1.16** is now available—**download** a copy today!
- The **CHIUW 2018** call for participation is now available!
- A recent **Cray blog post** reports on highlights from CHIUW 2017.
- Chapel is now one of the supported languages on **Try It Online**!
- Watch talks from **ACCU 2017**, **CHIUW 2017**, and **ATPESC 2016** on **YouTube**.
- Browse slides from **PADAL**, **EAGE**, **EMBRACE**, **ACCU**, and other recent talks.
- See also: **What's New?**

Copyright 2017 Cray Inc.

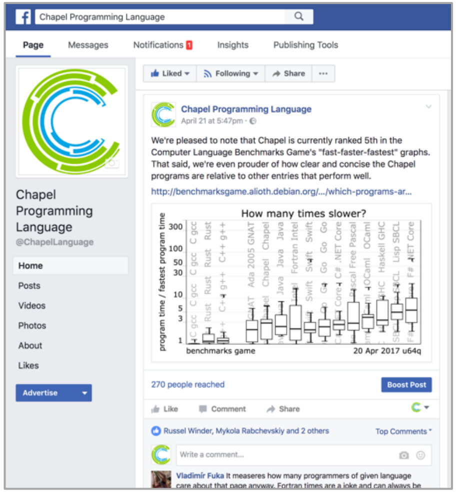
How to Stalk Chapel

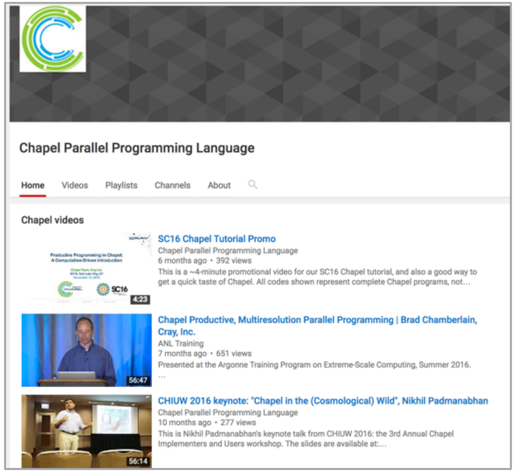
http://facebook.com/ChapelLanguage

http://twitter.com/ChapelLanguage

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

chapel-announce@lists.sourceforge.net



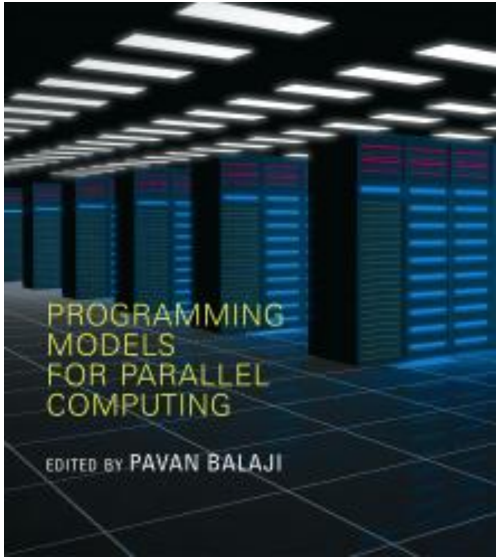


Copyright 2017 Cray Inc.

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 now also available online



Other Chapel papers/publications available at **http://chapel-lang.org/papers.html**

Copyright 2017 Cray Inc.

Suggested Reading (short attention spans)

CHIUW 2017: Surveying the Chapel Landscape, Cray Blog, July 2017.

- *a run-down of recent events*

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.

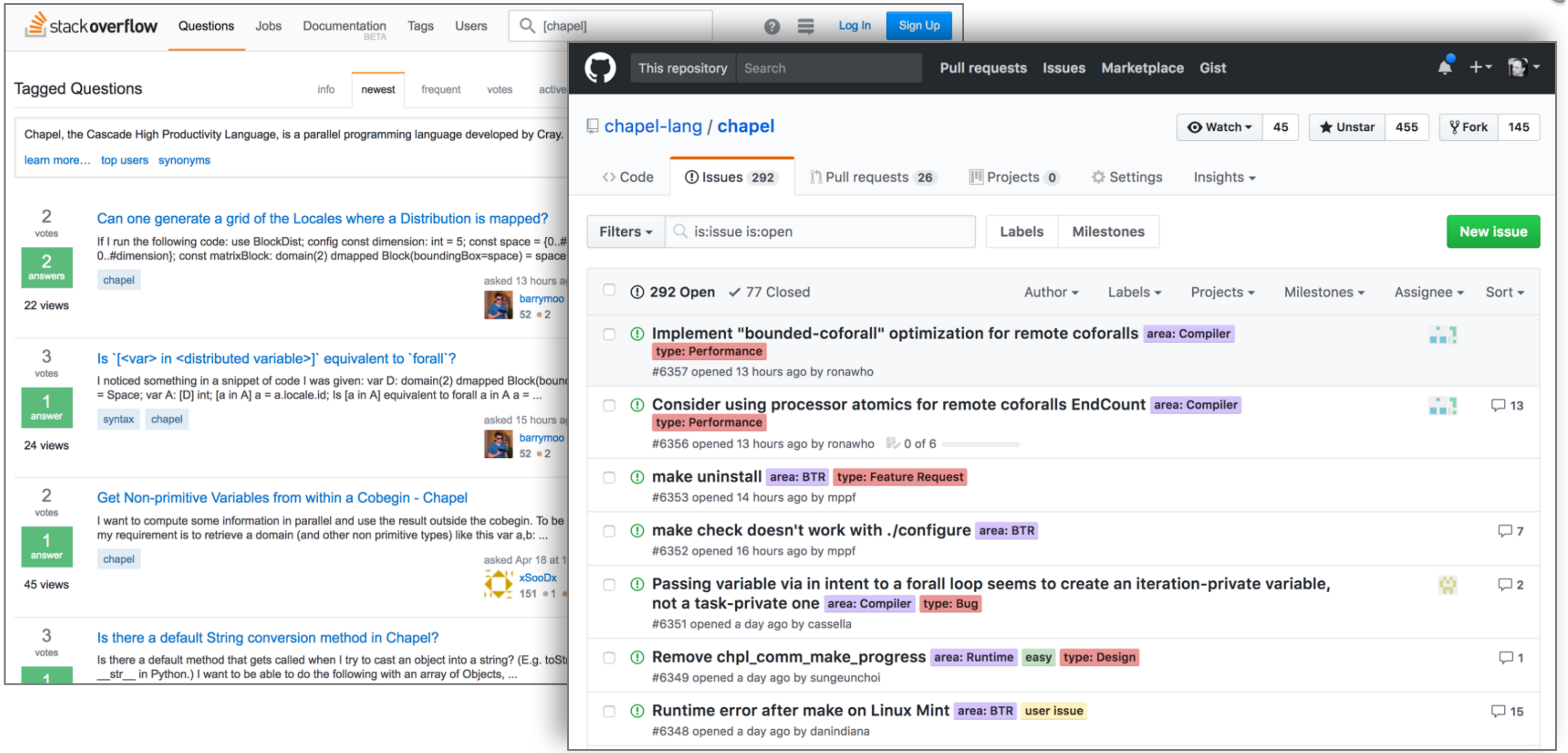
- *a series of articles answering common questions about why we are pursuing Chapel in spite of the inherent challenges*

[Ten] Myths About Scalable Programming Languages, IEEE TCSC Blog (index available on chapel.cray.com "blog articles" page), Apr-Nov 2012.

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

Copyright 2017 Cray Inc.

Chapel StackOverflow and GitHub Issues



Copyright 2017 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
- #chapel-users (irc.freenode.net): user-oriented IRC channel
- chapel-users@lists.sourceforge.net: user discussions

Discuss Chapel development

- chapel-developers@lists.sourceforge.net: developer discussions
- #chapel-developers (irc.freenode.net): developer-oriented IRC channel

Discuss Chapel's use in education

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

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

Copyright 2017 Cray Inc.

Questions?

Copyright 2017 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.

Copyright 2017 Cray Inc.

CRAY CHAPEL

CRAY

THE SUPERCOMPUTER COMPANY