A Language Designer's Perspective on Benchmarking Suites and Competitions

Brad Chamberlain
Chapel Team, Cray Inc.
June 2, 2017

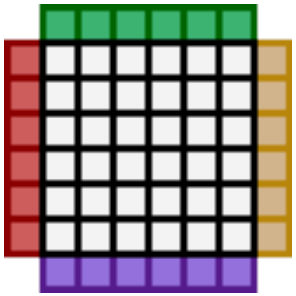
Safe Harbor Statement

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.

My Background

Education:

UNIVERSITY OF WASHINGTON
Computer Science & Engineering



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

Industry R&D: CRAY

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

CRAY CHAPEL

What is Chapel?

Chapel: A productive parallel programming language

- portable
- open-source
- a collaborative effort

Goals:

- Support general parallel programming
- Make parallel programming at scale far more productive

Motivation for Chapel

Q: Can a single language be…

…as programmable as Python?
…as fast as Fortran?
…as portable as C?
…as scalable as MPI?
…as generic and meta- as C++? (but using simpler notation?)
…as fun as <your favorite language here>?

A: We believe so.

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

A: Due to a lack of…

…long-term efforts
…resources
…community will
…developer/user co-design
…patience

Chapel is our attempt to change this

A few terminology notes for this talk…

Benchmark **= benchmarks, kernels, proxy apps, mini-apps, …**

- I don't want to get caught up in that terminological debate

Language **= any parallel programming model**

- whether a true language, an extension, a library, a pragma notation, …

"So you're designing an HPC language… how?"

- **Do something modest?**
  - challenging to create a sea change
    - likely to either result in hybrid programming models (e.g., MPI+X+Y)
    - or to not present an enticing cost::benefit ratio for switching (e.g., UPC?)
- **Do something big?**
  - potential for greater impact, but almost certain to take more time
- **closed-source or open-source?**
  - if closed, how to get co-design feedback early and often?
  - if open, how to keep audience's attention during development?
- **Chapel took the "go big (in the open) or go home" route**
  - currently suffers from "I knew you as an awkward kid" syndrome

Chapel Headlines: Which were you aware of?

- **Chapel is open-source and freely available**
- **Chapel is portable** (recent adds: AWS EC2, Docker, Windows 10, …)
- **Chapel has 14 full-time employees working on it at Cray**
  - and many other collaborators/contributors in the community
- **Chapel performance can now compete with, or beat, MPI and SHMEM**
- **Chapel has closed all major known compiler-introduced memory leaks**
- **Chapel now supports MPI+X execution**
- **Chapel supports unified access to MCDRAM on Intel Xeon Phi ("KNL")**
- **Chapel has nearly 200 webpages of modern, online documentation**
- **Chapel has a rich, growing library** (FFTW, BLAS, LAPACK, BigInt, …)
- **CHIUW, Chapel's 4th annual implementer and user workshop is today**

EMBRACE's theme and languages like Chapel

Getting Chapel's message out is clearly our challenge…

…but benchmarks play a big role in our ability to do so

Chapel Headlines (directly related to benchmarks)

- **Chapel is free**
- **Chapel is portable** (recent adds: AWS EC2, Docker, Windows 10, …)
- **Chapel has 14 full-time employees working on it at Cray**
  - and many other collaborators/contributors in the community
- **Chapel performance can now compete with, or beat, MPI and SHMEM**
- **Chapel has closed all major known compiler-introduced memory leaks**
- **Chapel now supports MPI+X execution**
- **Chapel supports unified access to MCDRAM on Intel Xeon Phi ("KNL")**
- **Chapel has nearly 200 webpages of modern, online documentation**
- **Chapel has a rich, growing library** (FFTW, BLAS, LAPACK, BigInt, …)
- **CHIUW, Chapel's 4th annual implementer and user workshop is today**

Chapel Headlines (indirectly related to benchmarks)

- **Chapel is free**
- **Chapel is portable** (recent adds: AWS EC2, Docker, Windows 10, …)
- **Chapel has 14 full-time employees working on it at Cray**
  - and many other collaborators/contributors in the community
- **Chapel performance can now compete with, or beat, MPI and SHMEM**
- **Chapel has closed all major known compiler-introduced memory leaks**
- **Chapel now supports MPI+X execution**
- **Chapel supports unified access to MCDRAM on Intel Xeon Phi ("KNL")**
- **Chapel has nearly 200 webpages of modern, online documentation**
- **Chapel has a rich, growing library** (FFTW, BLAS, LAPACK, BigInt, …)
- **CHIUW, Chapel's 4th annual implementer and user workshop is today**

EMBRACE's theme and languages like Chapel

Getting Chapel's message out is clearly our challenge…

…but benchmarks play a big role in our ability to do so

Benchmarks permit us (and users) to evaluate our progress

- relative to the status quo
- relative to other competing technologies

If you care about language innovation and adoption, you should care about benchmarks

And in addition, arenas for benchmark comparisons

What do I mean by an “arena”?

- **Essentially, a place for benchmark comparisons**
  - cross-language, cross-implementation, cross-architecture
- **Think of the top-500 as a performance-centric arena**
  - how can we expand this notion to include productivity, other concerns?
- **I’ll build on this definition as we go…**

Outline

- ✓ **Context**
  - ✓ Who I am
  - ✓ What Chapel is
  - ✓ Why I'm here
- ➢ **Survey of benchmark suites with which I have experience**
  - NPB, HPCC, DOE proxy apps, CLBG, PRK
    - What they are
    - What I've appreciated about them
    - Where they could be improved
- **Summary:** If I had resources to throw at benchmark suites…

Disclaimers

All of the following characterizations are my personal opinions—yours will likely differ.

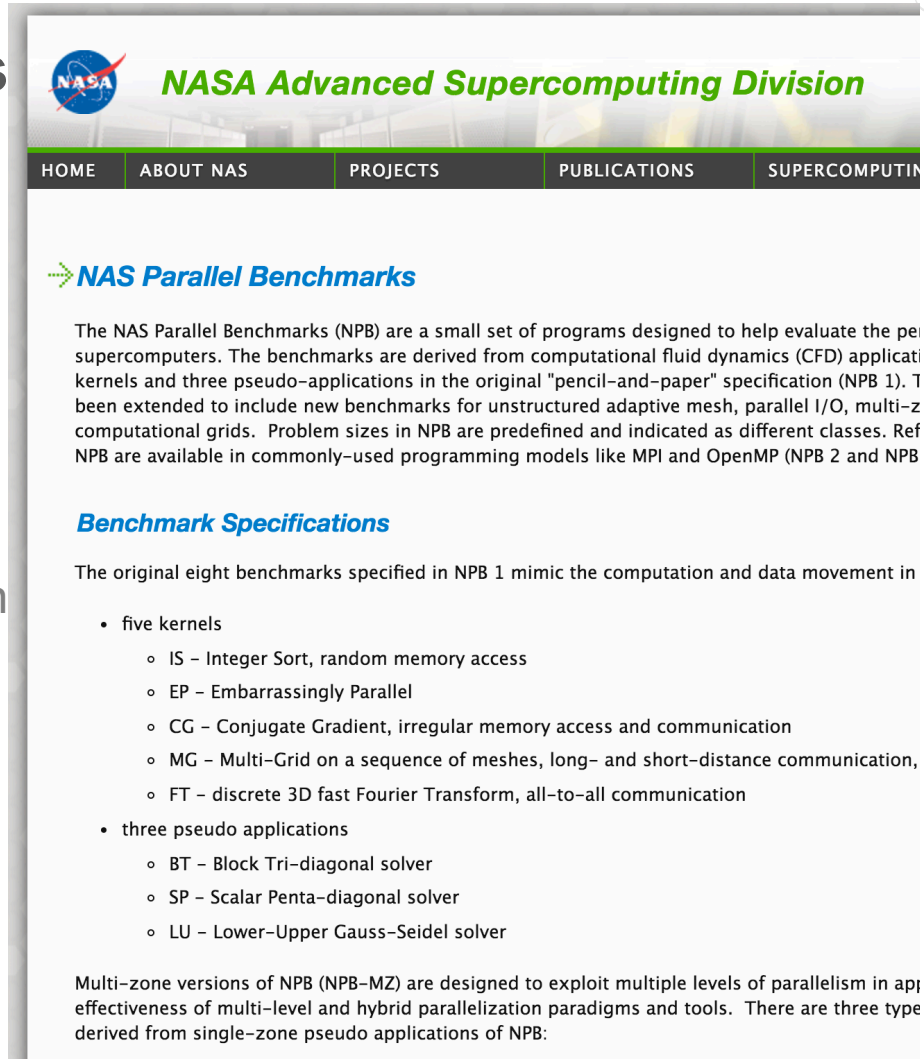
Also, my own opinions may be based on incomplete / incorrect information (for which I apologize).

The NAS Parallel Benchmark Suite (NPB)

(circa mid-to-late 1990's)

NPB: What it is

- 8 CFD-oriented benchmarks
  - paper-and-pencil descriptions
  - MPI, OpenMP implementations
    - (and others as well…)
  - Capture common HPC patterns
    - pleasingly parallel computations
    - data transpose
    - sparse mat-vect multiplication
    - stencils on hierarchical grids
    - bucket-exchange communication
    - …



NASA Advanced Supercomputing Division

HOME | ABOUT NAS | PROJECTS | PUBLICATIONS | SUPERCOMPUTIN

NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB) are a small set of programs designed to help evaluate the pe supercomputers. The benchmarks are derived from computational fluid dynamics (CFD) applicat kernels and three pseudo-applications in the original "pencil-and-paper" specification (NPB 1). T been extended to include new benchmarks for unstructured adaptive mesh, parallel I/O, multi-z computational grids. Problem sizes in NPB are predefined and indicated as different classes. Ref NPB are available in commonly-used programming models like MPI and OpenMP (NPB 2 and NPB

Benchmark Specifications

The original eight benchmarks specified in NPB 1 mimic the computation and data movement in

- five kernels
  - IS - Integer Sort, random memory access
  - EP - Embarrassingly Parallel
  - CG - Conjugate Gradient, irregular memory access and communication
  - MG - Multi-Grid on a sequence of meshes, long- and short-distance communication,
  - FT - discrete 3D fast Fourier Transform, all-to-all communication
- three pseudo applications
  - BT - Block Tri-diagonal solver
  - SP - Scalar Penta-diagonal solver
  - LU - Lower-Upper Gauss-Seidel solver

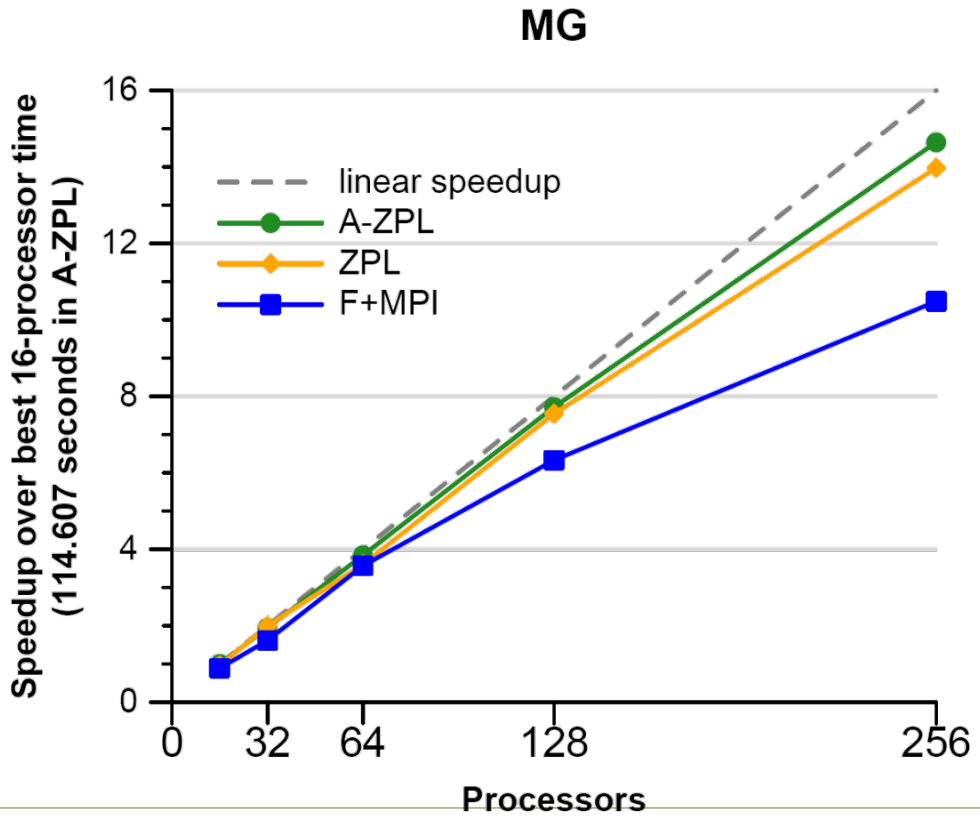
Multi-zone versions of NPB (NPB-MZ) are designed to exploit multiple levels of parallelism in app effectiveness of multi-level and hybrid parallelization paradigms and tools. There are three type derived from single-zone pseudo applications of NPB:

NPB: What it did well

- **Early example of what an HPC benchmark suite should be**
  - Well-designed and implemented
  - Reasonably well-documented
  - The basis for many evaluations of languages, systems, compilers

NPB: What it did well

- **Helped me graduate:**
  - supported comparison between ZPL and MPI for interesting patterns
  - sufficiently approachable for a graduate student to be successful with



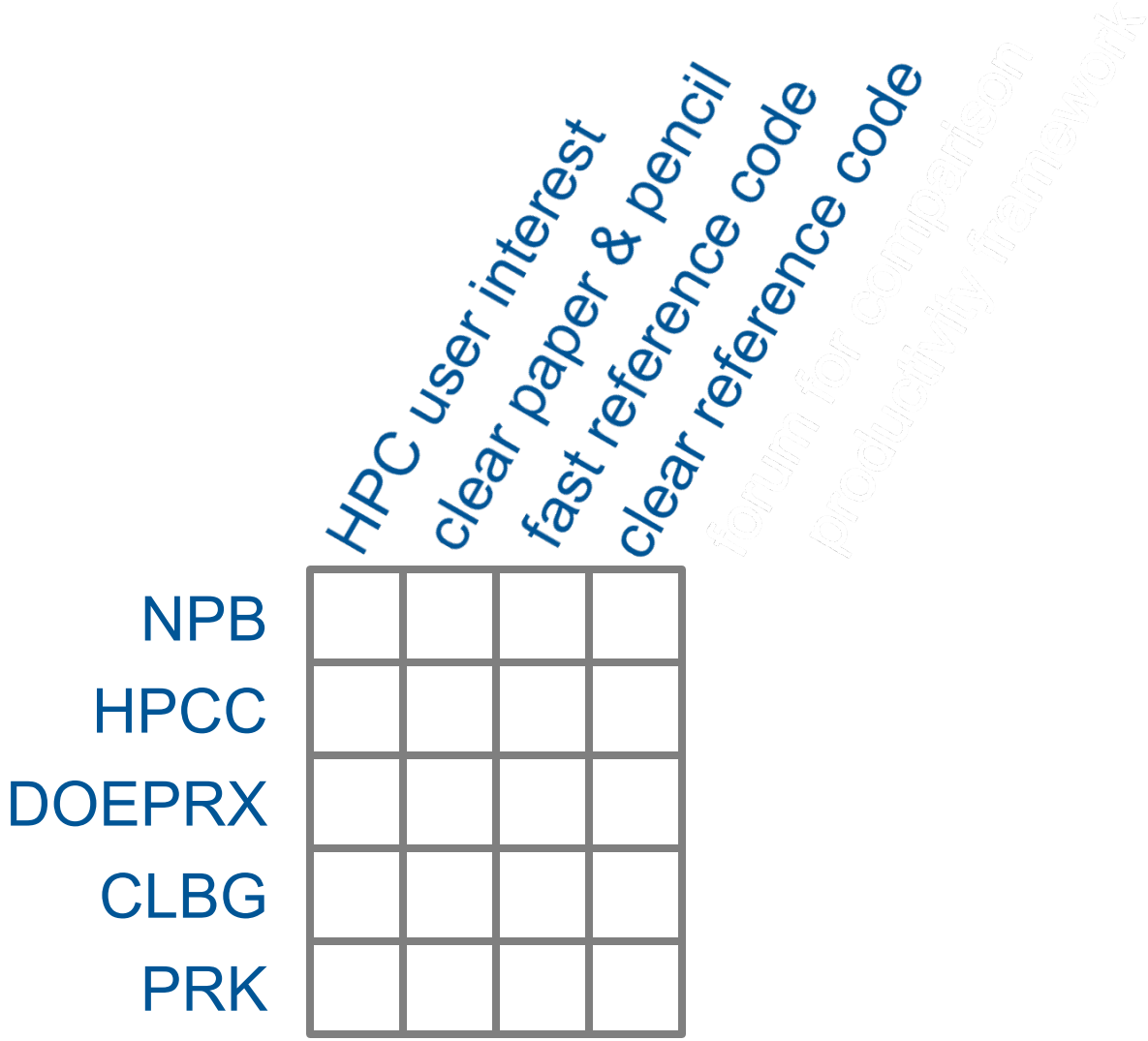
Rating Benchmark Suites (on a 7-point scale)

- **Would HPC Application Developers care about this?**
  - **key:** 1 = no; 4 = eh…; 7 = yes!
- **Does a (clear) paper and pencil description exist?**
  - **key:** 1 = no or completely unclear; 7 = yes, and it's crystal clear
    - Ideally, a paper and pencil description should not assume the reader can translate from math equations into HPC code
    - rather, it should talk in terms of data structures and access patterns
- **Does the suite include a fast reference version?**
- **Does the suite include a clear reference version?**
  - **key:** 1 = no, or it's not; 7 = yes and it is

Rating Benchmark Suites (on a 7-point scale)

- **Would HPC Application Developers care about this?**
  - **key:** 1 = no; 4 = eh…; 7 = yes!
  - NPB: 6-7 when written, 5-6 now?
- **Does a (clear) paper and pencil description exist?**
  - **key:** 1 = no or completely unclear; 7 = yes, and it's crystal clear
    - Ideally, a paper and pencil description should not assume the reader can translate from math equations into HPC code
    - rather, it should talk in terms of data structures and access patterns
  - NPB: 3 (too many equations, not enough data structures / CS)
- **Does the suite include a fast reference version?**
  - NPB: 7
- **Does the suite include a clear reference version?**
  - **key:** 1 = no, or it's not; 7 = yes and it is
  - NPB: 3 (it's not terrible, but also not particularly instructive)

Benchmark Suite Scorecard

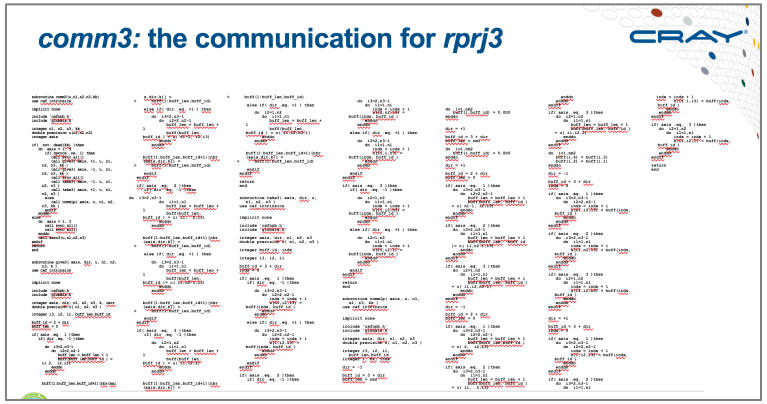


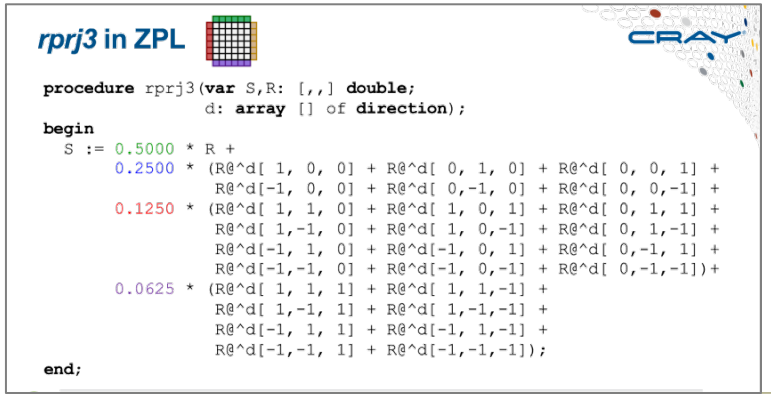
Benchmark Suite Scorecard

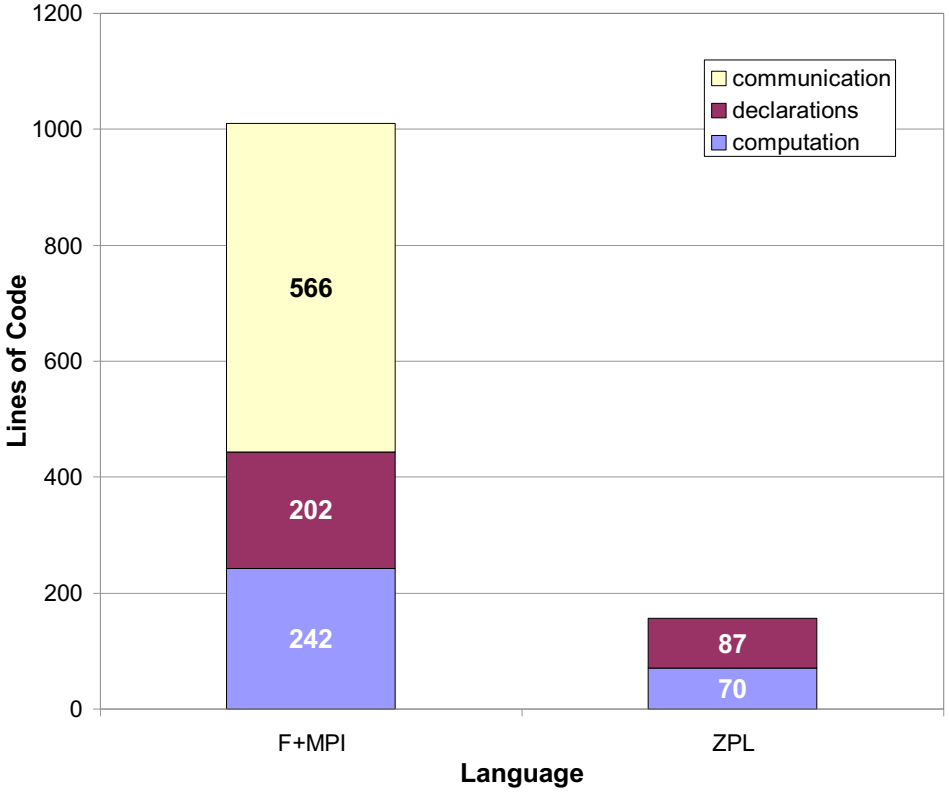
	HPC user interest	clear paper & pencil	fast reference code	clear reference code	forum for comparison	productivity framework
NPB	✓	~	✓	~		
HPCC						
DOEPRX						
CLBG						
PRK						

NPB: Where it fell short

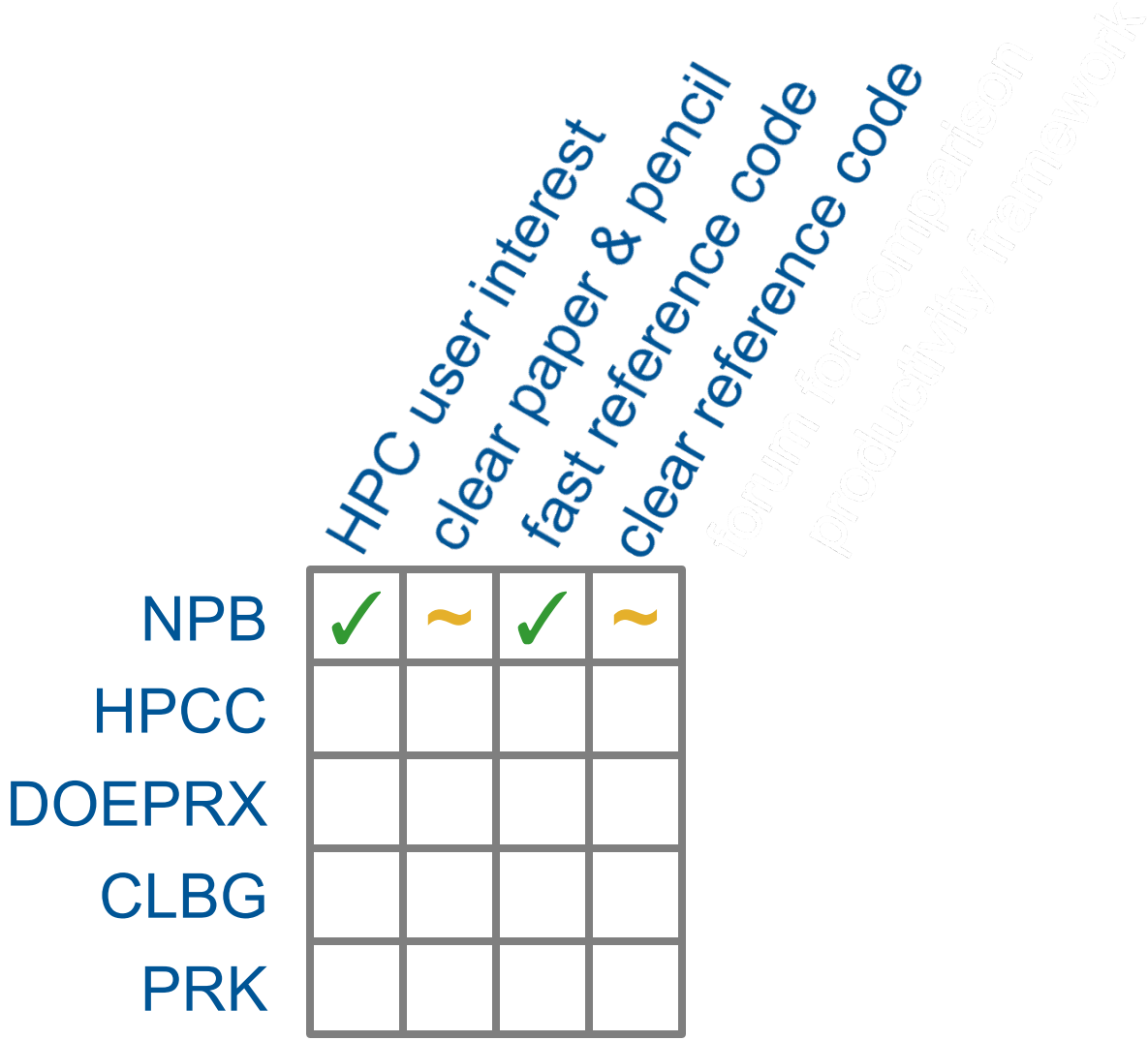
- No established competition for comparing performance
- No prescribed basis for comparing elegance / productivity
  - neither is a big surprise given its timing and HPC's performance focus



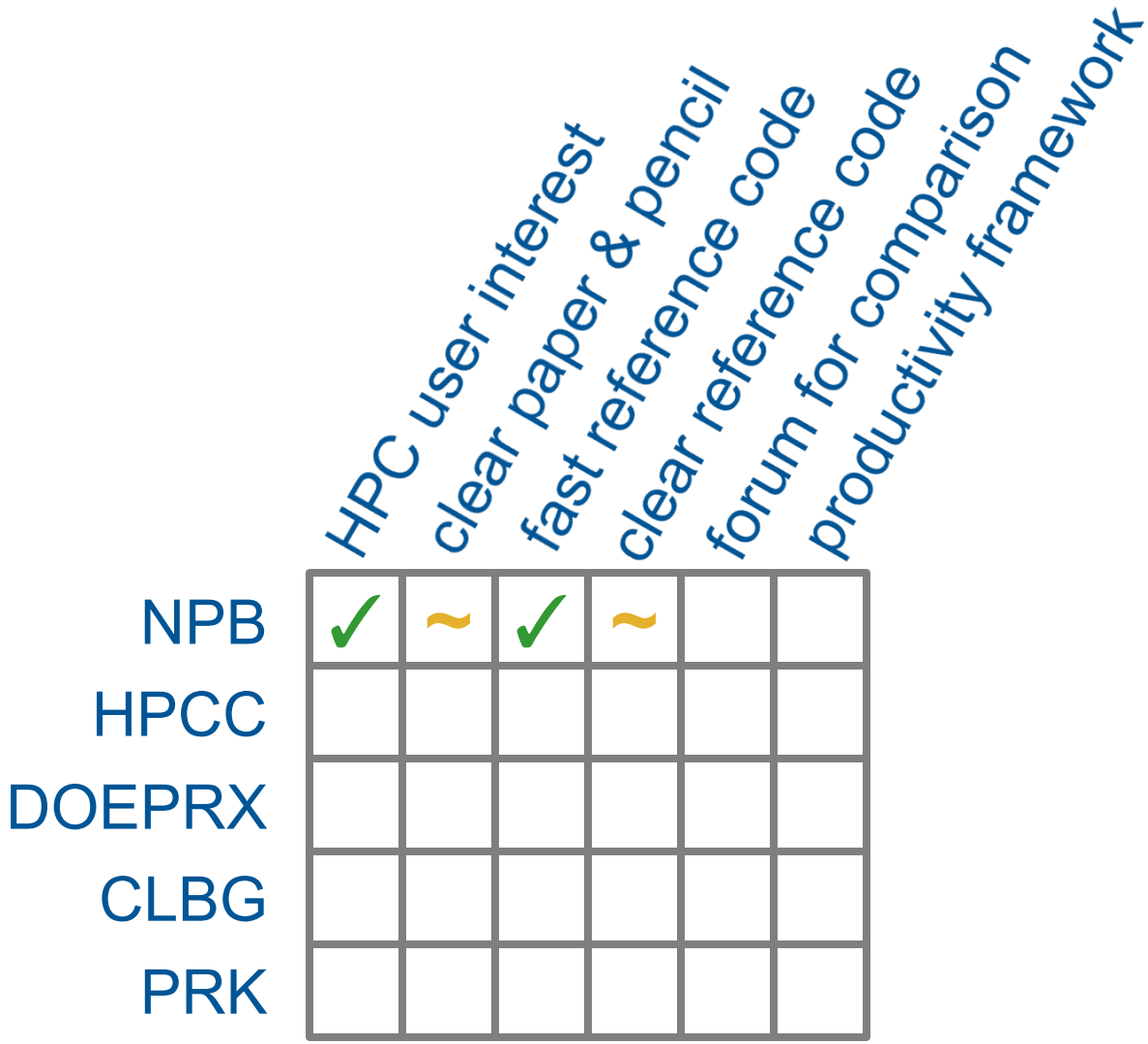




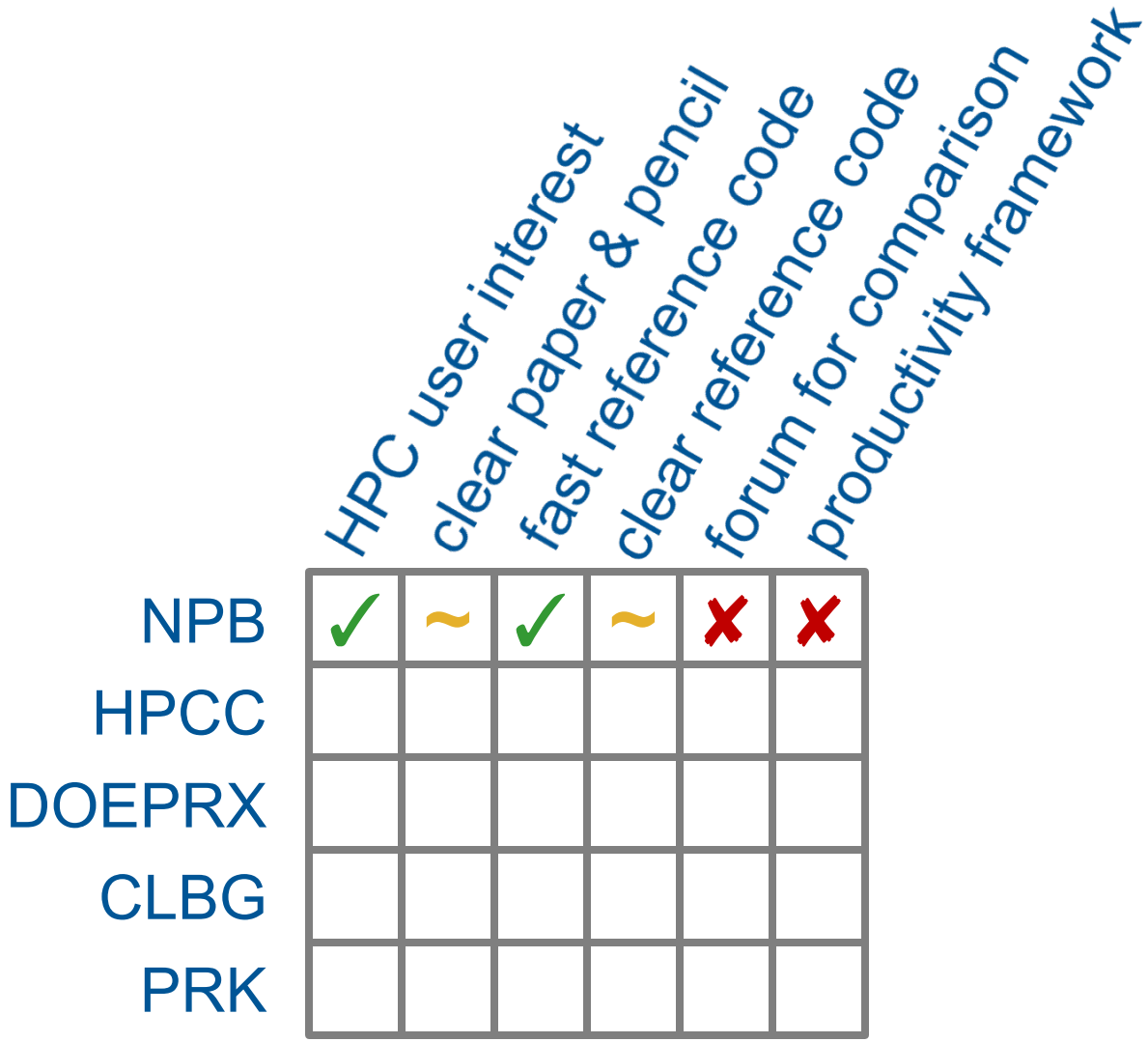
Benchmark Suite Scorecard



Benchmark Suite Scorecard



Benchmark Suite Scorecard



Other things one might want to rate…

- Size of codes? (effort to port to new languages)
- Arbitrarily scalable input sets?
- Self-verification of result?
- Was it designed more to measure communication (1) or computation (7)?

Other things one might want to rate…

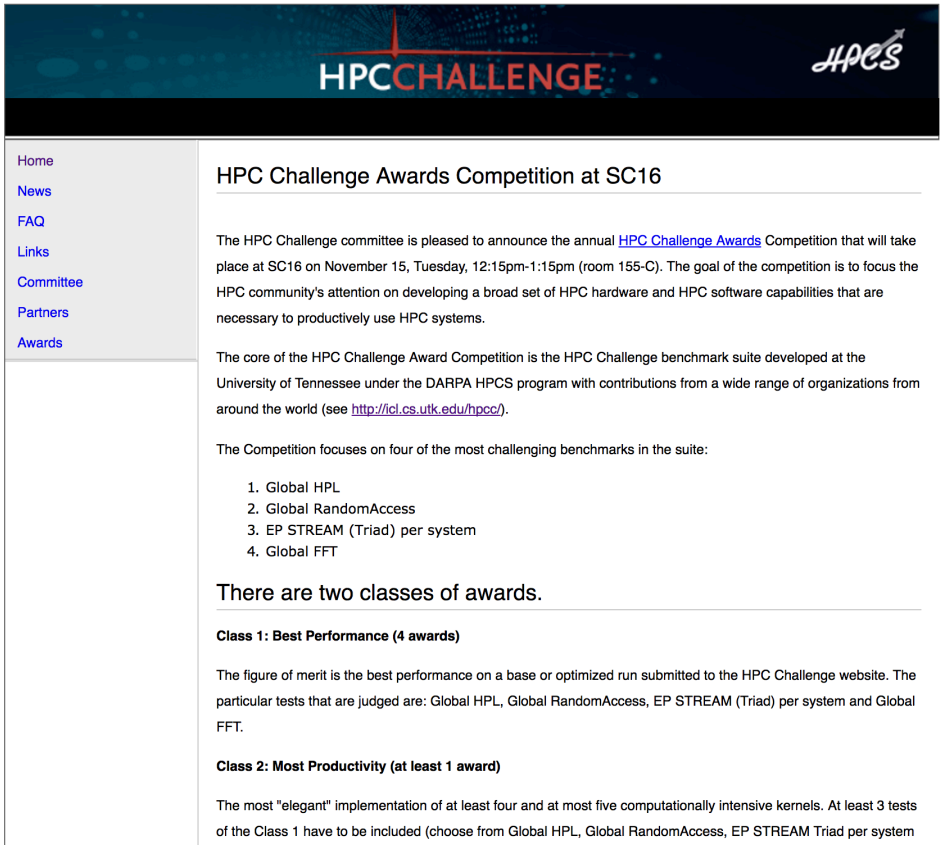
- Size of codes? (effort to port to new languages)
  - NPB: 5-6
- Arbitrarily scalable input sets?
  - NPB: 2
- Self-verification of result?
  - NPB: 7
- Was it designed more to measure communication (1) or computation (7)?
  - NPB: 5

(but I'm not as interested in these, personally, at least today)

The HPC Challenge Competition (HPCC)

HPCC: What it is

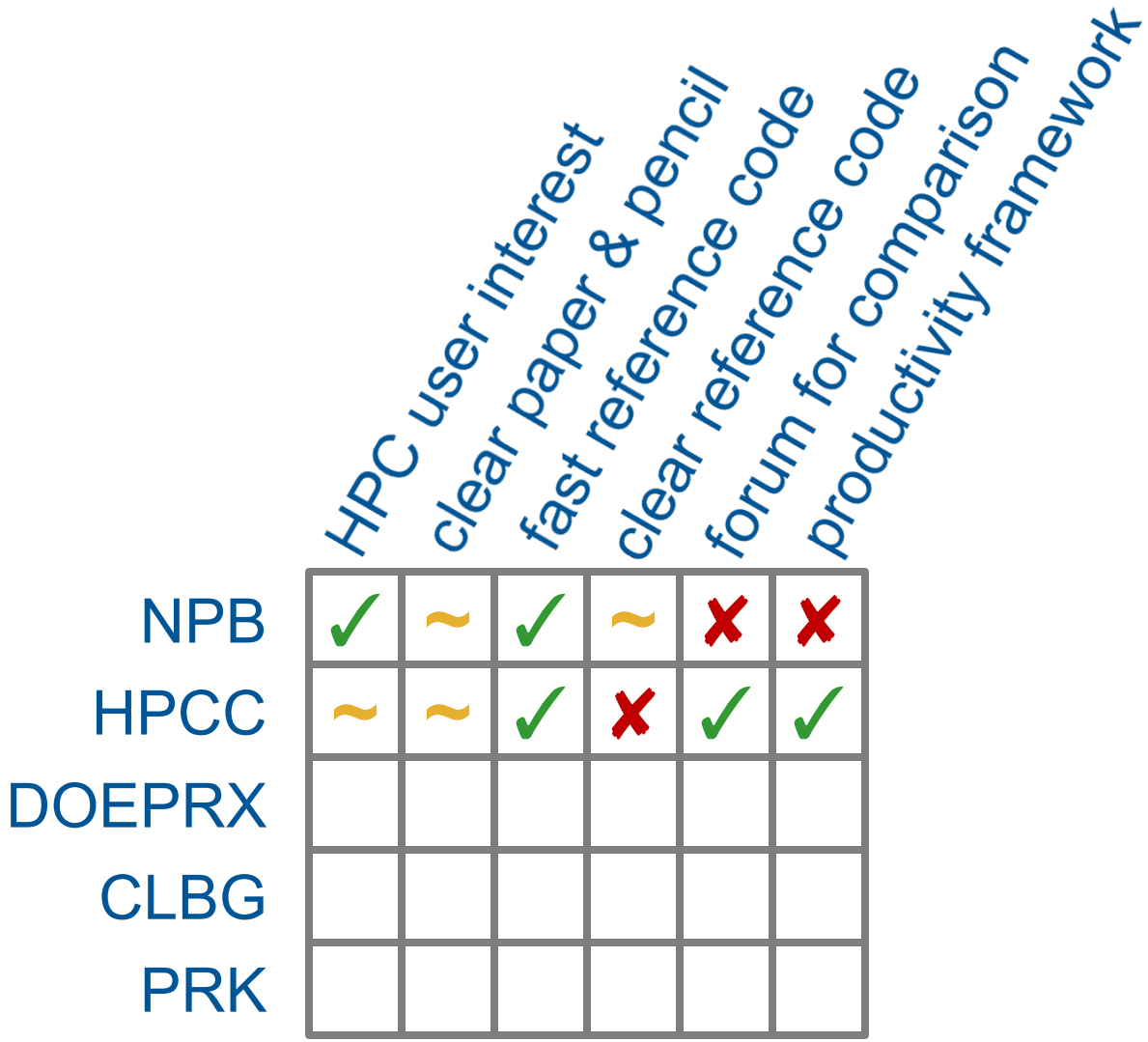
- **A benchmark suite and competition kicked off towards the start of the HPCS program**
  - class 1: perf only (boring!)
  - class 2: productivity
    - 50% performance
    - 50% elegance, judged by panel
  - 4 core computations:
    - Stream Triad (memory, EP)
    - Random Access (GUPS)
    - FFT (data transpose)
    - HPL (block-cyclic linear algebra)
  - over time, entrants could submit their own computations of interest as well…



HPCC: What it did well

- Established an annual competition for benchmarking
- Focused attention on elegance in addition to performance

Benchmark Suite Scorecard



HPCC: Numerical Scoring

- **Would HPC Application Developers care about this?**
  - HPCC: 4 (comm idioms yes, computations, less so)
- **Does a (clear) paper and pencil description exist?**
  - HPCC: 5 (some—stream, ra—were clearer than others—hpl)
- **Does the suite include a fast reference version?**
  - HPCC: 7
- **Does the suite include a clear reference version?**
  - HPCC: 2 (monolithic, difficult to detangle code; some parts inscrutable)
- **Forum for comparison?**
  - HPCC: 7
- **Framework for evaluating productivity?**
  - HPCC 7

HPCC: Where it fell short

- **Many judges seemed to not spend much time on elegance**
  - in practice, might catch glimpses of code in 5-minute presentations
    - or not…
    - even when you did, 5-minutes is not enough time to make that call well
  - admittedly, SC is a busy time of year…
- **In early years, awarded separate perf and elegance awards**
  - disregarded the tension between those concerns
- **Difficult for public to process results after the fact**
  - code was not made available in a standard way
- **Once arbitrary codes added, couldn't make comparisons**
- **Lack of continuity from year to year…**

What Trend Do these Awards Suggest?

2006: Cilk wins "Best overall productivity"

Chapel and X10 take honorable mentions

2007: X10 and Python/Star-P win "most productive" awards

2008: Chapel, UPC+X10, Parallel Matlab tie for "most productive"

2009: Chapel wins "most productive"

2010: UPC+X10 win "most productive system"

CAF wins "most productive language"

2011: Chapel wins "most elegant language"

2012: Chapel wins "most elegant language"

2013: XcalableMP wins class 2

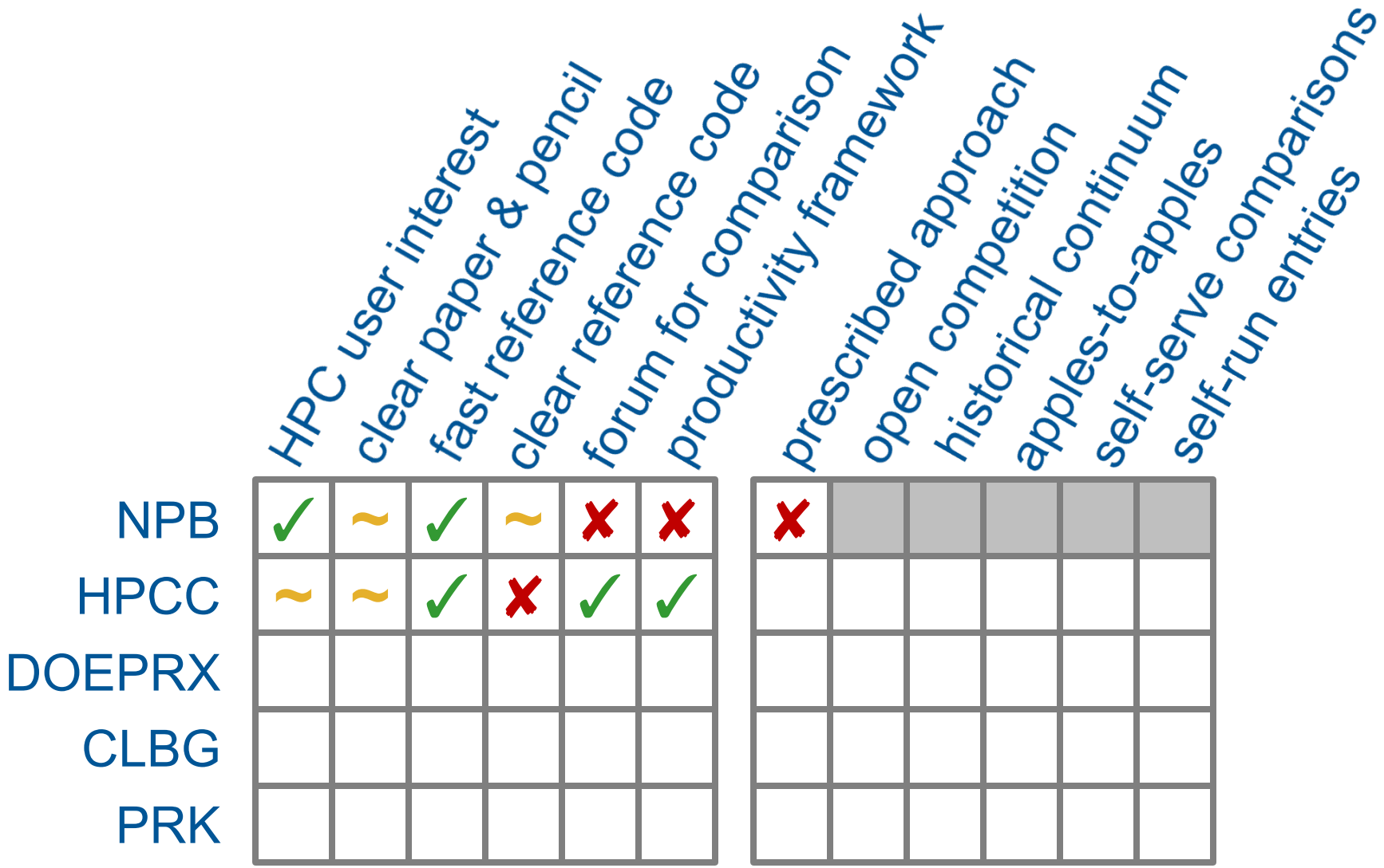
2014: PCJ wins "most elegant"

Ratings for Suites Supporting Comparisons

- **Is the approach prescribed / constrained (7) or not (1)?**
  - Why? Want to evaluate technologies over algorithmic cleverness
- **Is the competition open to anyone who wants to enter?**
- **Does the competition maintain continuity?**
  - Imagine if the top-500 required everyone to re-run every six months…
- **Does the competition use apples-to-apples comparisons?**
- **Can community members surf the results conveniently?**
  - In order to draw their own conclusions, make their own visualizations
- **Does the suite trivially support running it yourself?**
  - In order to reproduce results or obtain than on different systems

Benchmark Suite Scorecard

Benchmark Suite Scorecard



Benchmark Suite Scorecard

Comparison Ratings for HPCC

- Is the approach prescribed / constrained (7) or not (1)?
  - HPCC: 3
- Is the competition open to anyone who wants to enter?
  - HPCC: 7
- Does the competition maintain continuity?
  - HPCC: 1
- Does the competition use apples-to-apples comparisons?
  - HPCC: 2
- Can community members surf the results conveniently?
  - HPCC: 2
- Does the suite trivially support running it yourself?
  - HPCC: 1

Thoughts on improving the HPCC competition

- **Have entries carry over from year-to-year like top-500**
  - Or, run competition continually in real-time like the CLBG
- **Have judges devote time offline to evaluating elegance**
- **Re-unify set of benchmarks to study**
  - Perhaps introduce a new benchmark each year, retiring an old one?
- **Maintain submitted codes and results in a unified manner**

DOE Proxy Applications (DOEPRX)

DOE Proxy Apps: What they are

- **Benchmarks that are…**
  …large enough and realistic enough that experts value their results
  …yet tractable enough that non-experts can tackle them
- **Where's the screenshot?**
  - This is not a well-defined benchmark suite per se
  - More a style of benchmark that has been in vogue in recent years
  - As a result, no central repository (as far as I'm aware of…)

DOE Proxy Apps: What they do well

- As intended: Create tractable codes that matter

DOE Proxy Apps: Where they fall short

- **They present something of a moving target:**
  - There are *lots* of them, including *apparent* redundant instances
  - Many seem to go through phases of being more or less fashionable
  - Each requires a fair amount of effort to port and tune
  - **The challenge:** How is a modest-sized team to invest its time?
- **Aforementioned lack of centralized suite**
  - Keeping tabs on several / all of them requires lots of effort
- **No established forums for comparison**

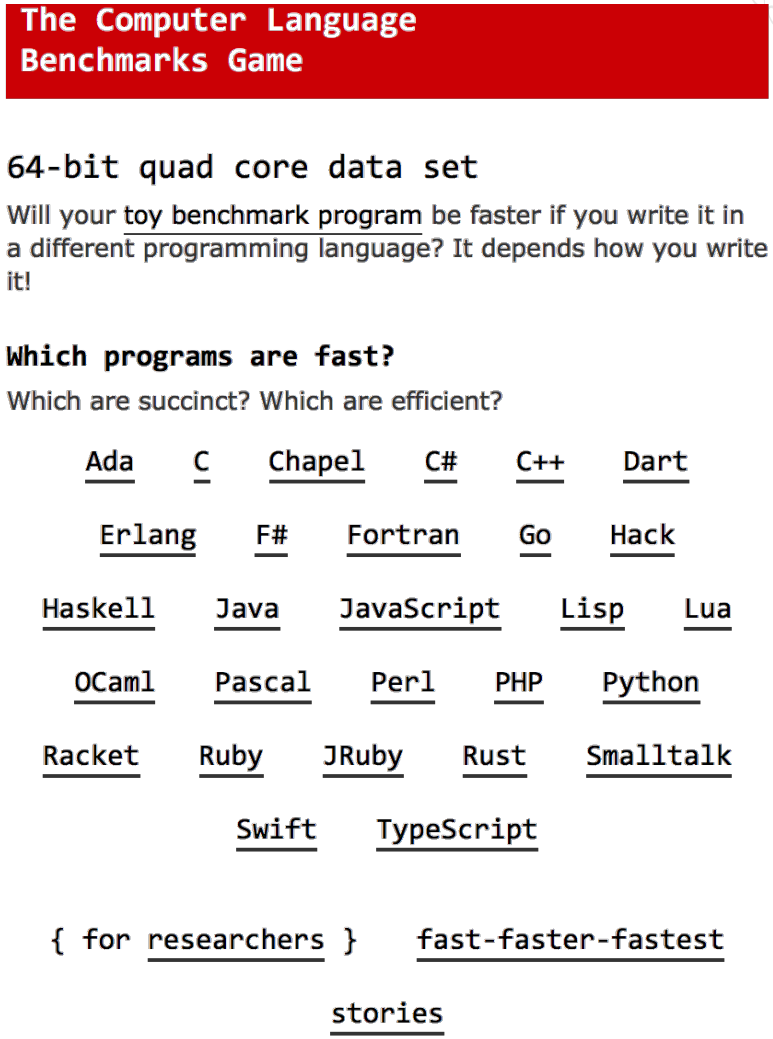
Benchmark Suite Scorecard

Benchmark Suite Scorecard

The Computer Language Benchmarks Game (CLBG)

CLBG: What it is

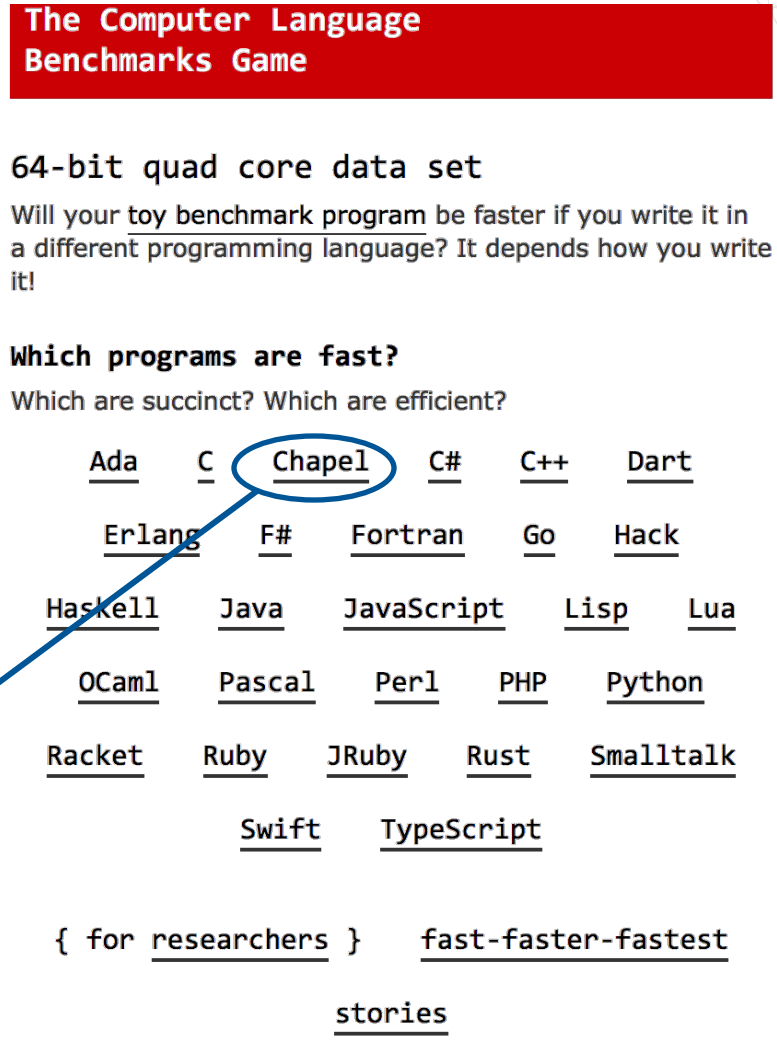
- **A suite of 13 "toy" benchmarks**
  - single-node
  - serial, vectorizable, or multicore parallel
  - exercise key features like…
    …memory management
    …tasking and synchronization
    …arbitrary-precision math
    …vectorization
    …strings and regular expressions
- **Imagine a 3D ragged matrix:**
  - with 13 benchmarks
    x ~28 languages
    x as many impls as are interesting
  - each entry contains:
    - source code
    - performance information
    - "code size"



CLBG: What it is

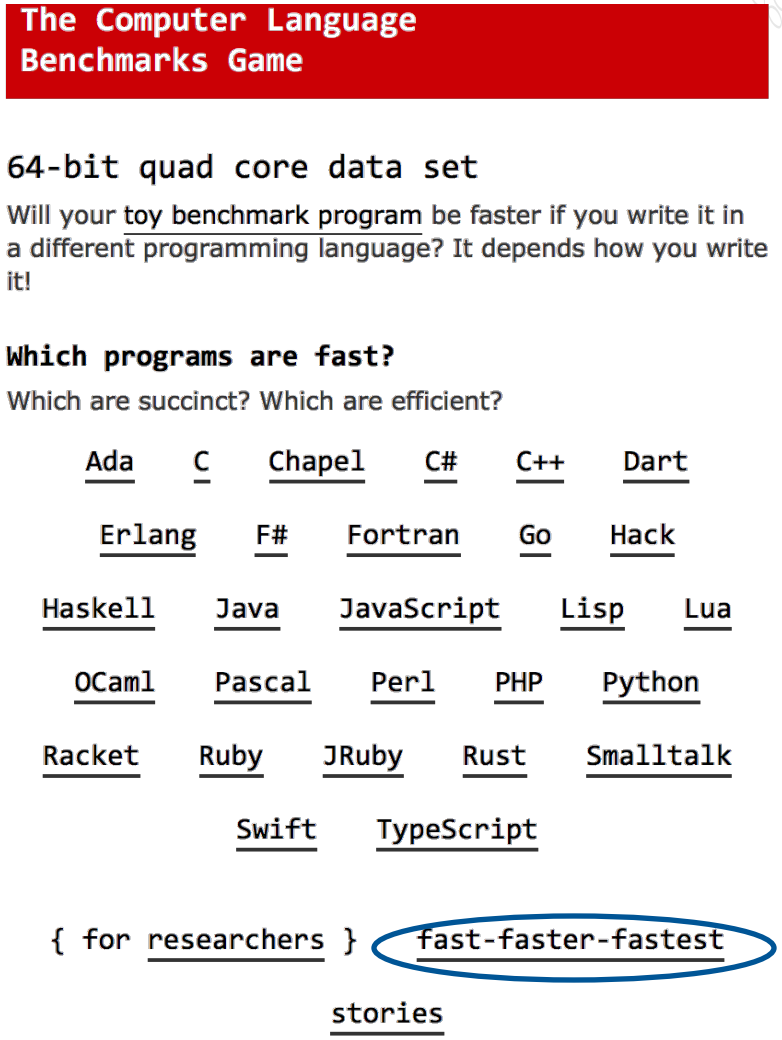
- **A suite of 13 "toy" benchmarks**
  - single-node
  - serial, vectorizable, or multicore parallel
  - exercise key features like…
    …memory management
    …tasking and synchronization
    …arbitrary-precision math
    …vectorization
    …strings and regular expressions
- **Imagine a 3D ragged matrix:**
  - with 13 benchmarks
    x
    x g
  - each entry contains:
    - source code
    - performance information
    - "code size"

Chapel entries have been accepted since ~IPDPS 2016



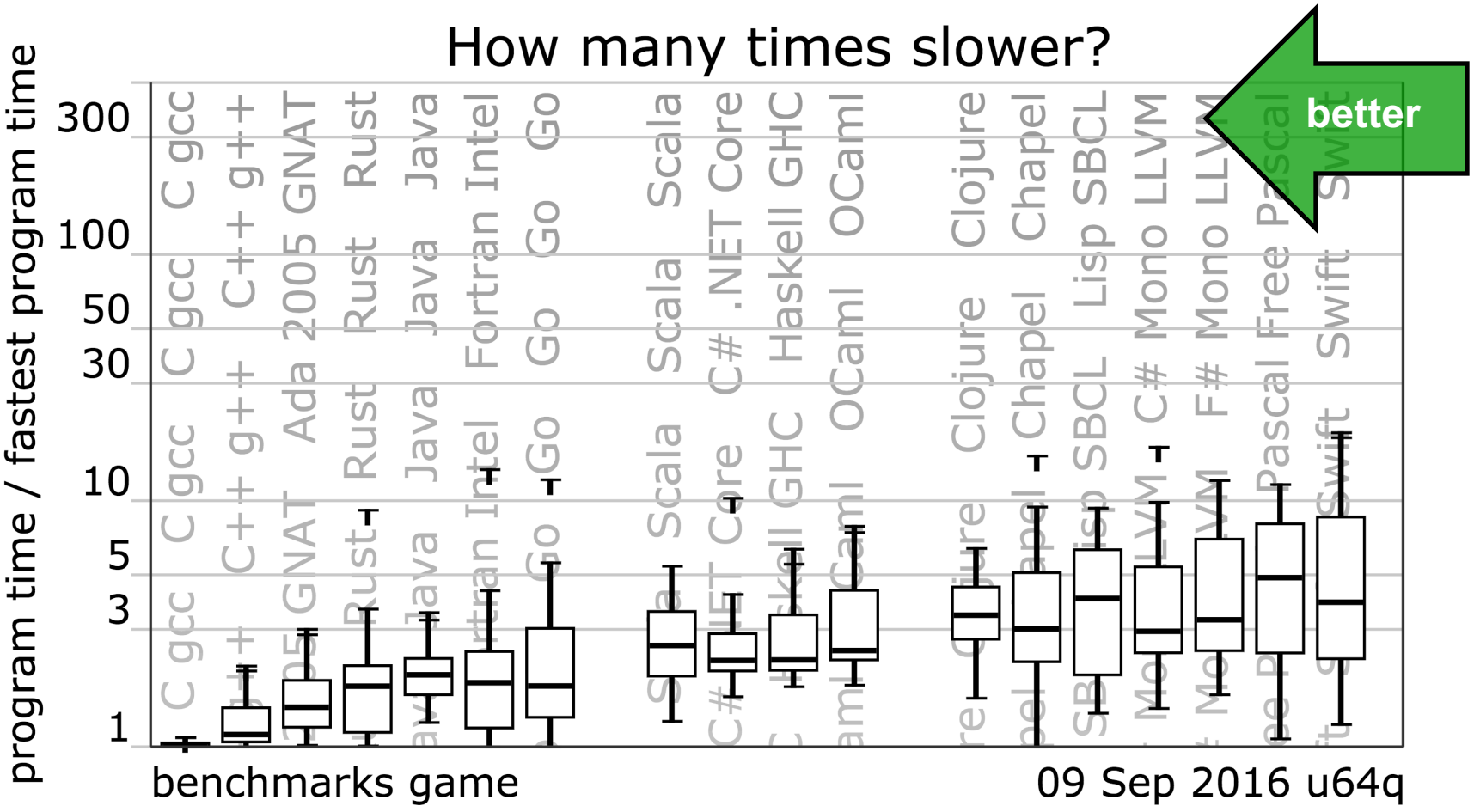
CLBG: What it is

- **A suite of 13 "toy" benchmarks**
  - single-node
  - serial, vectorizable, or multicore parallel
  - exercise key features like…
    …memory management
    …tasking and synchronization
    …arbitrary-precision math
    …vectorization
    …strings and regular expressions
- **Imagine a 3D ragged matrix:**
  - with 13 benchmarks
    x ~28 languages
    x as many impls as are interesting
  - each entry contains:
    - source code
    - performance information
    - "code size"



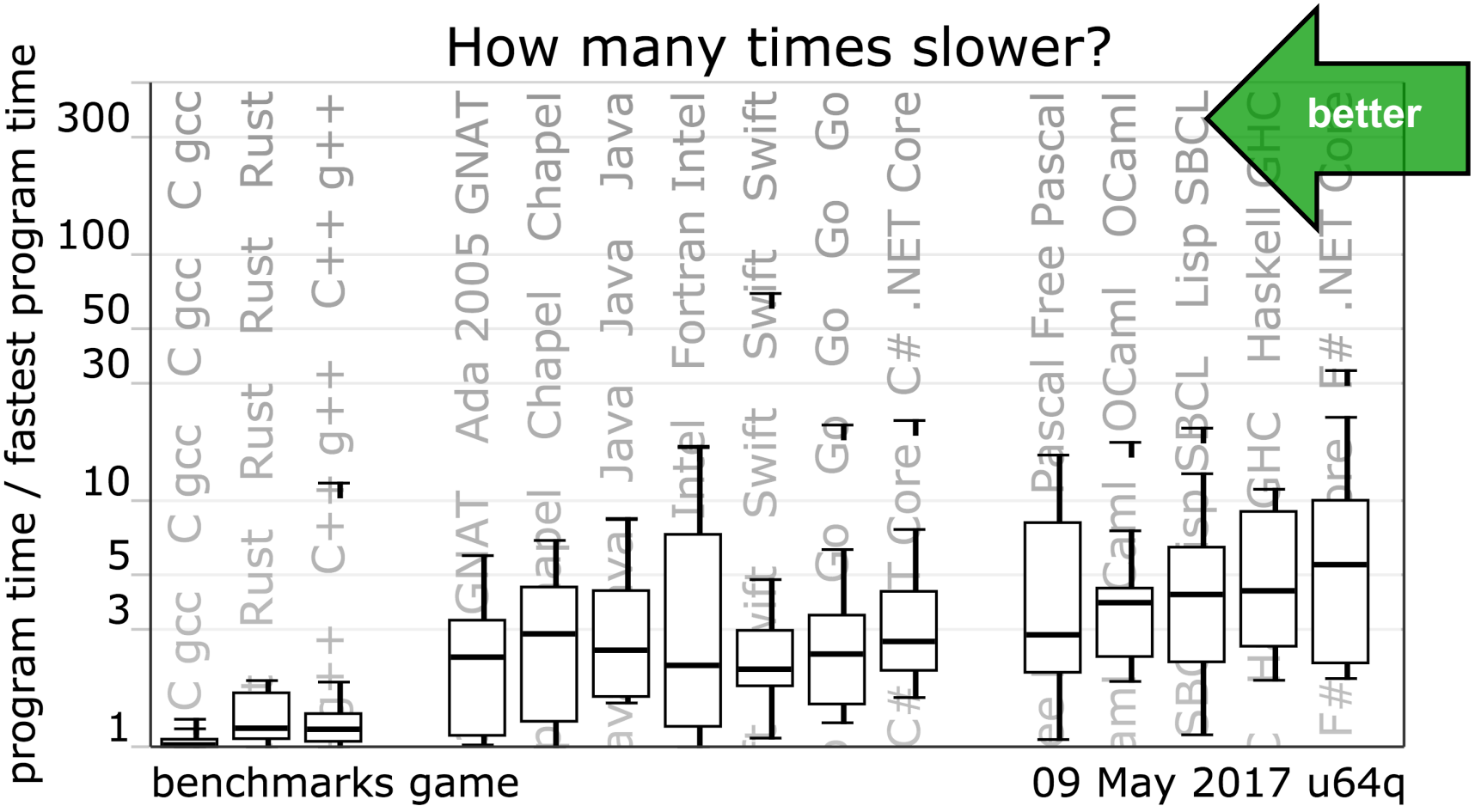
CLBG: Fast-faster-fastest graph (Sep 2016)

Site summary: relative performance (sorted by geometric mean)



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

Site summary: relative performance (sorted by geometric mean)



CLBG: Sorting Results

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

The Computer Language Benchmarks Game

chameneos-redux

description

program source code, command-line and measurements

×	source	secs	mem	gz	cpu	cpu load
1.0	**C** gcc #5	**0.60**	820	2863	2.37	100% 100% 98% 100%
1.2	**C++** g++ #5	**0.70**	3,356	1994	2.65	100% 100% 91% 92%
1.7	**Lisp** SBCL #3	**1.01**	55,604	2907	3.93	97% 96% 99% 99%
2.3	**Chapel** #2	**1.39**	76,564	1210	5.43	99% 99% 98% 99%
3.3	**Rust** #2	**2.01**	56,936	2882	7.81	97% 98% 98% 98%
5.6	C++ g++ #2	3.40	1,880	2016	11.88	100% 51% 100% 100%
6.8	Chapel	4.09	66,584	1199	16.25	100% 100% 100% 100%
8.0	**Java** #4	**4.82**	37,132	1607	16.73	98% 98% 54% 99%
8.5	**Haskell** GHC	**5.15**	8,596	989	9.26	79% 100% 2% 2%
10	Java	6.13	53,760	1770	8.78	42% 45% 41% 16%
10	Haskell GHC #4	6.34	6,908	989	12.67	99% 100% 2% 1%
11	**C#** .NET Core	**6.59**	86,076	1400	22.96	99% 82% 78% 91%
11	**Go**	**6.90**	832	1167	24.19	100% 96% 56% 100%
13	Go #2	7.59	1,384	1408	27.65	91% 99% 99% 78%
13	Java #3	7.94	53,232	1267	26.86	54% 96% 98% 94%

The Computer Language Benchmarks Game

chameneos-redux

description

program source code, command-line and measurements

×	source	secs	mem	gz	cpu	cpu load
1.0	**Erlang**	58.90	28,668	**734**	131.19	62% 60% 51% 53%
1.0	**Erlang** HiPE	59.39	25,784	**734**	131.58	60% 56% 56% 54%
1.1	**Perl** #4	5 min	14,084	**785**	7 min	40% 40% 29% 28%
1.1	**Racket**	5 min	132,120	**791**	5 min	1% 0% 0% 100%
1.1	Racket #2	175.88	116,488	842	175.78	100% 1% 1% 0%
1.2	**Python 3** #2	236.84	7,908	**866**	5 min	24% 48% 27% 45%
1.3	**Ruby**	90.52	9,396	**920**	137.53	35% 35% 35% 34%
1.3	**Ruby** JRuby	48.78	628,968	**928**	112.15	65% 60% 49% 58%
1.3	**Go** #5	11.05	832	**957**	32.48	75% 74% 75% 73%
1.3	**Haskell** GHC #4	6.34	6,908	**989**	12.67	99% 100% 2% 1%
1.3	Haskell GHC	5.15	8,596	989	9.26	79% 100% 2% 2%
1.6	**OCaml** #3					32% 38% 37% 39%
1.6	Go					100% 96% 56% 100%
1.6	**Chapel**					0% 100% 100% 100%
1.6	Chapel #2					99% 99% 98% 99%

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

CLBG: Comparing Pairs of Languages

Can also compare languages pair-wise (performance only):

The Computer Language Benchmarks Game

Chapel programs versus Go

all other Chapel programs & measurements

by benchmark task performance

regex-redux

source	secs	mem	gz	cpu	cpu load
Chapel	**10.02**	1,022,052	477	19.68	99% 72% 14% 12%
Go	29.51	352,804	798	61.51	77% 49% 43% 40%

binary-trees

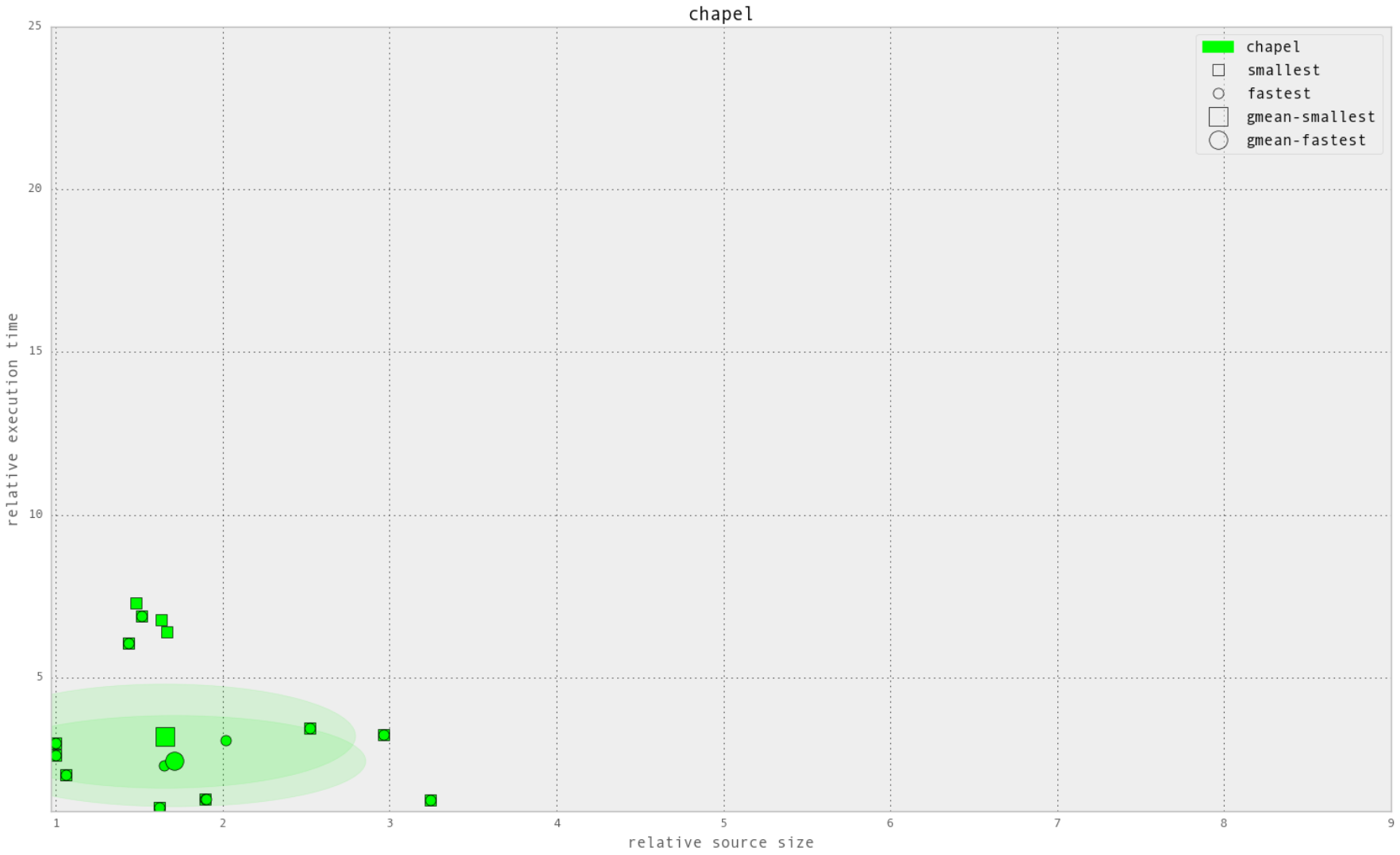
source	secs	mem	gz	cpu	cpu load
Chapel	**14.32**	324,660	484	44.15	100% 58% 78% 75%
Go	34.77	269,068	654	132.04	95% 97% 95% 95%

fannkuch-redux

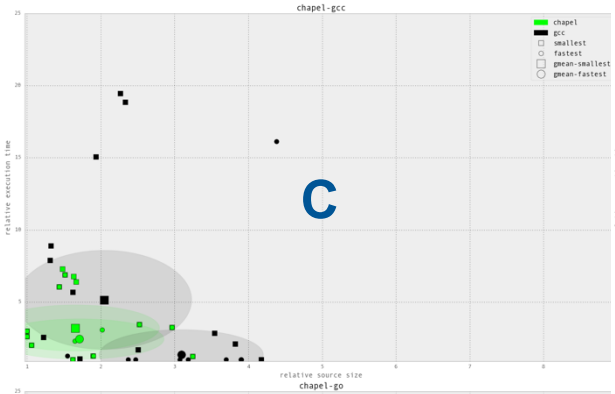
source	secs	mem	gz	cpu	cpu load
Chapel	**11.38**	46,056	728	45.18	100% 99% 99% 100%

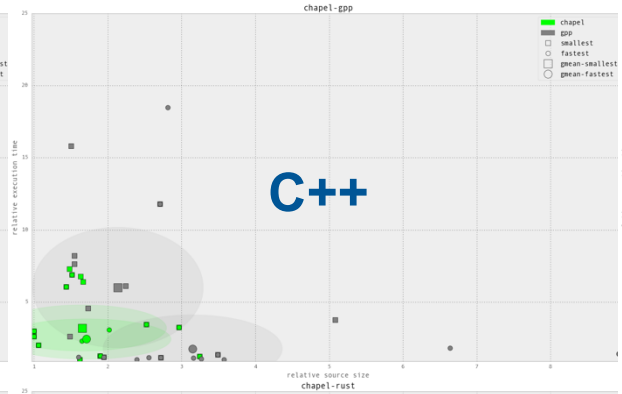
But happily, all the data is open source!

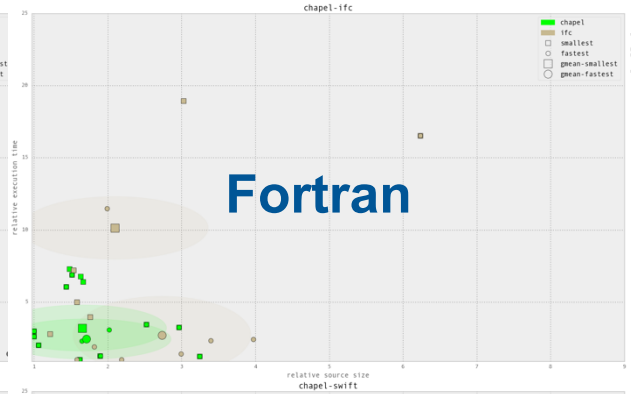
Chapel entries: normalized perf & size (Apr 2017)

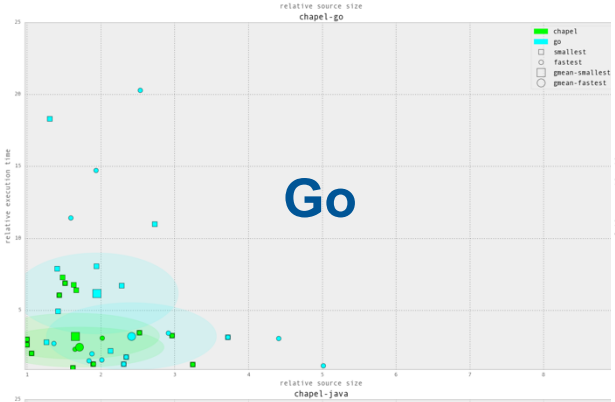


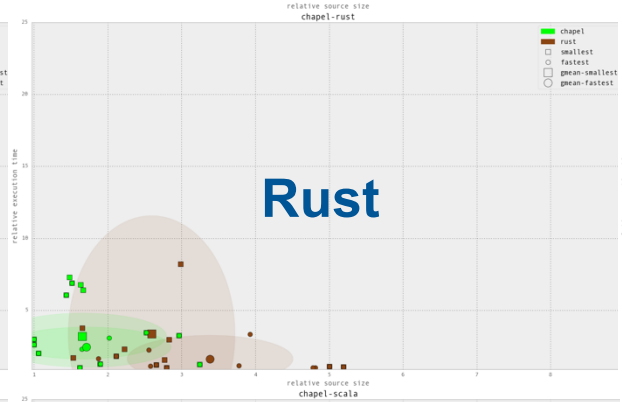
Chapel vs. 9 other languages

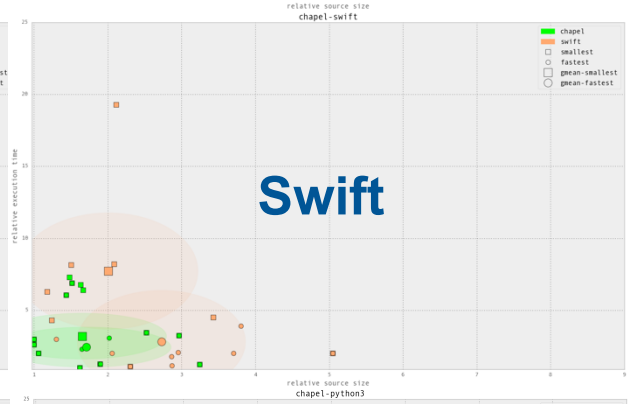


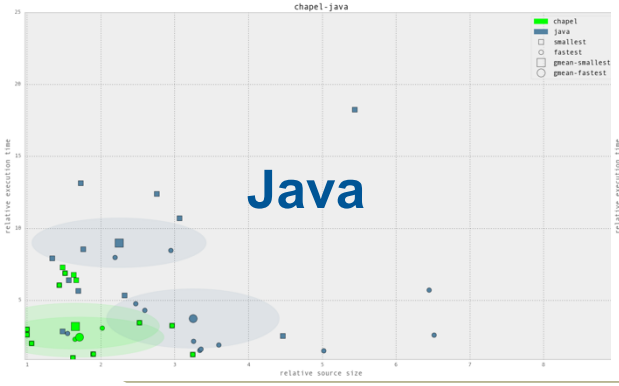


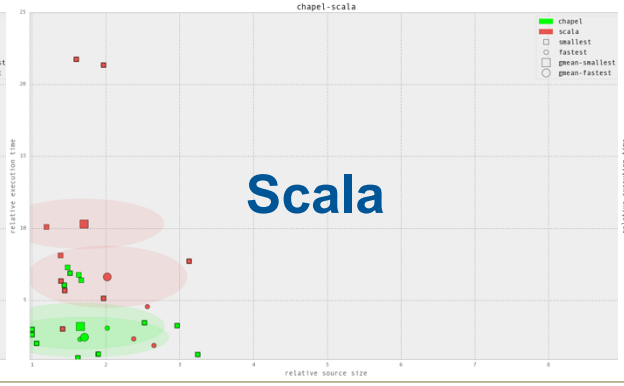


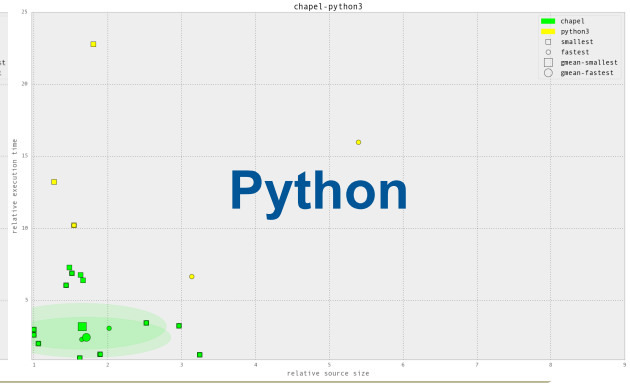




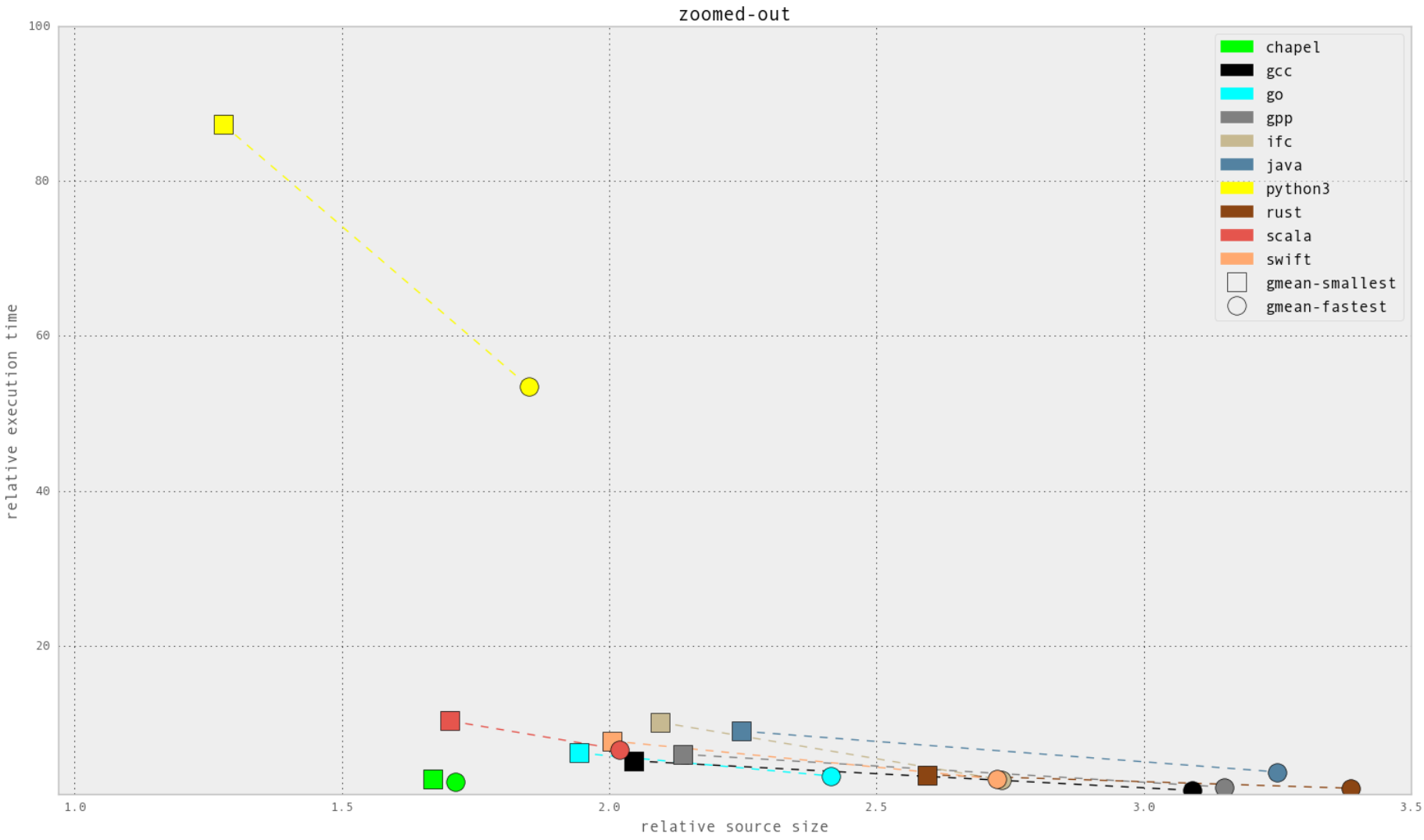




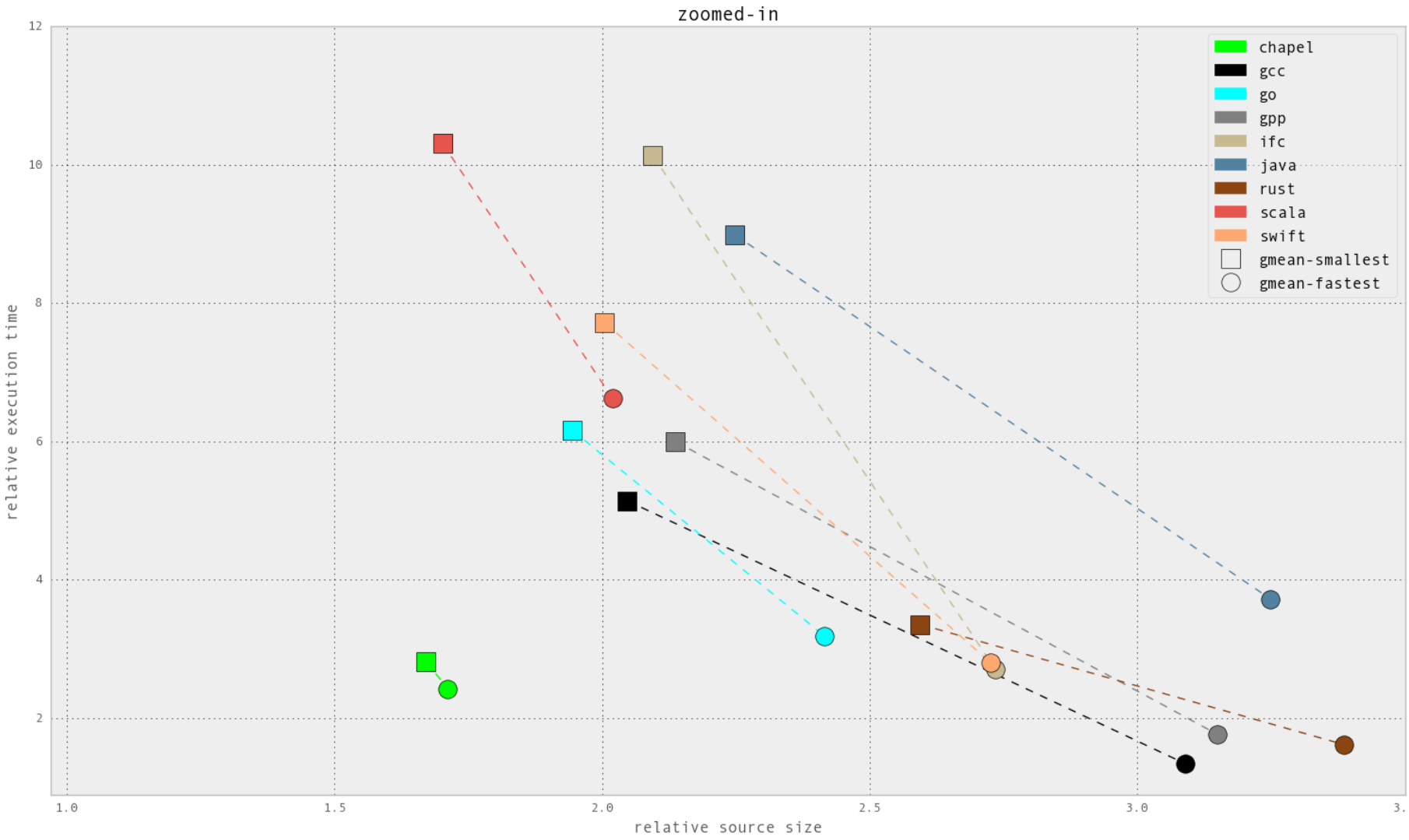




Cross-Language Summary



Cross-Language Summary (no Python)



CLBG: Website

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 #2 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 #5 entry

CLBG: What it does well

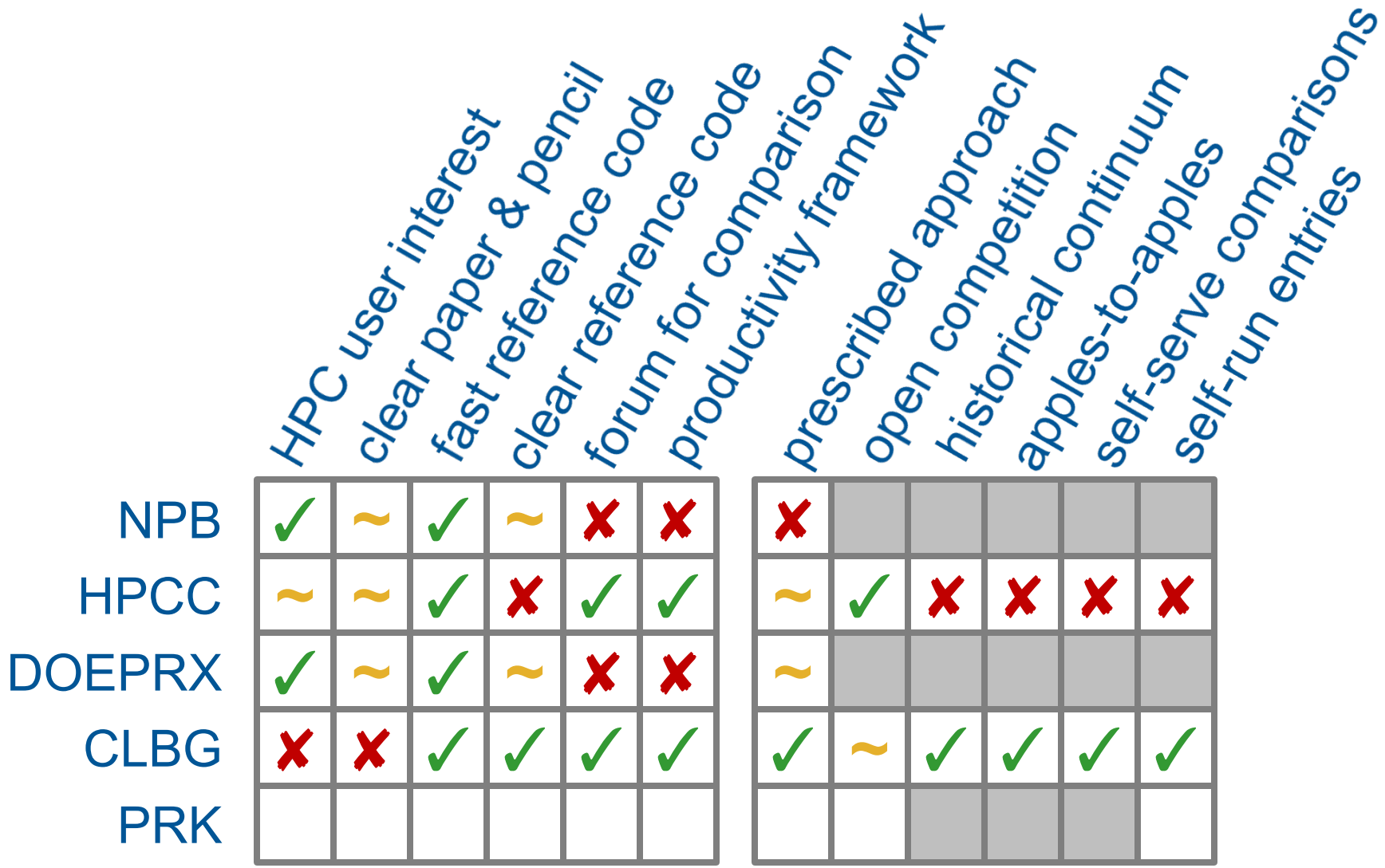
- Engages the community, drives interest and chatter
- Incredibly active in terms of steady stream of submissions

CLBG: Where it falls short (for HPC)

- Single-node only
- Only some overlap with HPC computational idioms

Benchmark Suite Scorecard

Benchmark Suite Scorecard



CLBG: Summary

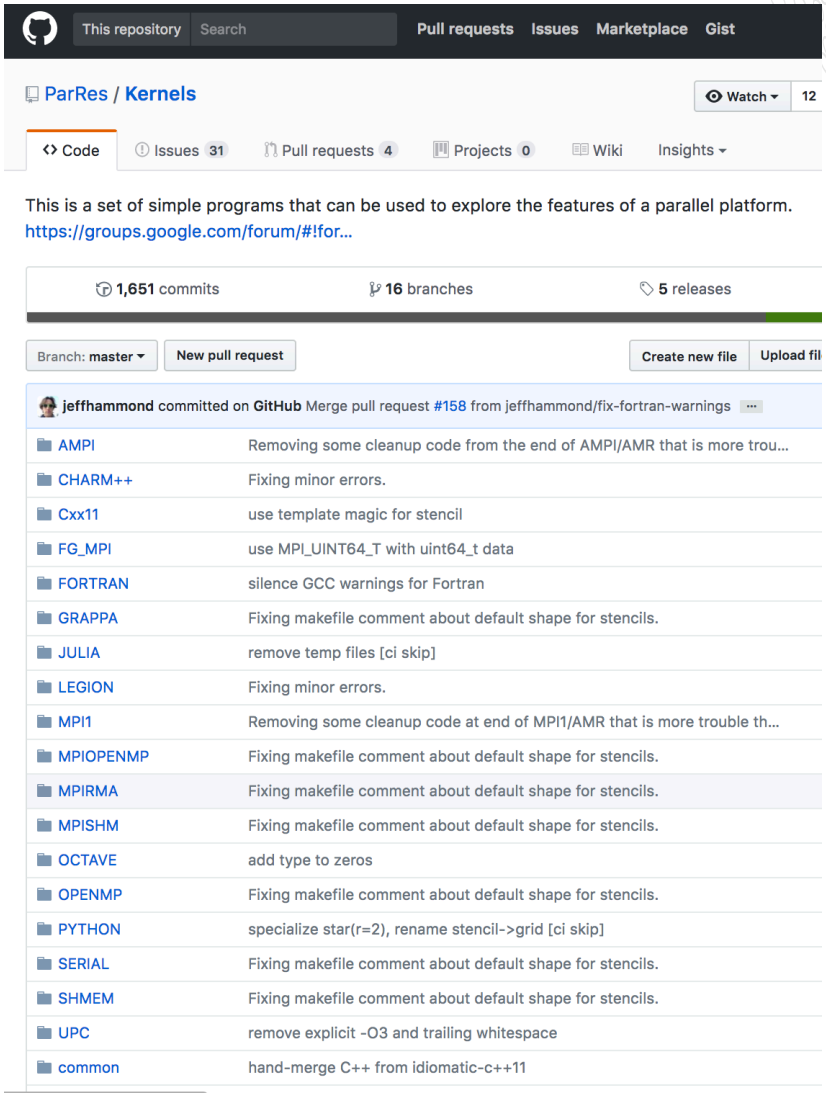
- **HPC would benefit greatly from something like the CLBG**
  - a good, engaging challenge to encourage innovation
  - online, continual, open, surfable
- **This would be far from trivial, though…**
  - what system(s) would be used for the evaluation?
  - what benchmarks?
  - need a reasonably neutral party to arbitrate questions
    - e.g., “Does doing xyz violate the prescribed approach?”
  - level of effort required to keep all the necessary software up-to-date
  - …
- **Yet, doing something would beat doing nothing**
  - top-500 and CLBG as examples of this
    - neither is perfect, yet each contributes something of value to the community

If interested in more, see my 4:20pm talk at CHIUW today

The Intel ParRes Kernels (PRK)

PRK: What it is

- **A suite of ~12 parallel kernels**
  - designed to expose perf bottlenecks
  - example kernels:
    - …stencil
    - …sparse matrix-vector
    - …particle-in-cell pattern
    - …wavefront-style computation
    - …transpose
    - …dgemm
  - serial, parallel, and distrib. versions
  - hosted on GitHub
    - ~18 languages represented
    - uses Travis to preserve quality



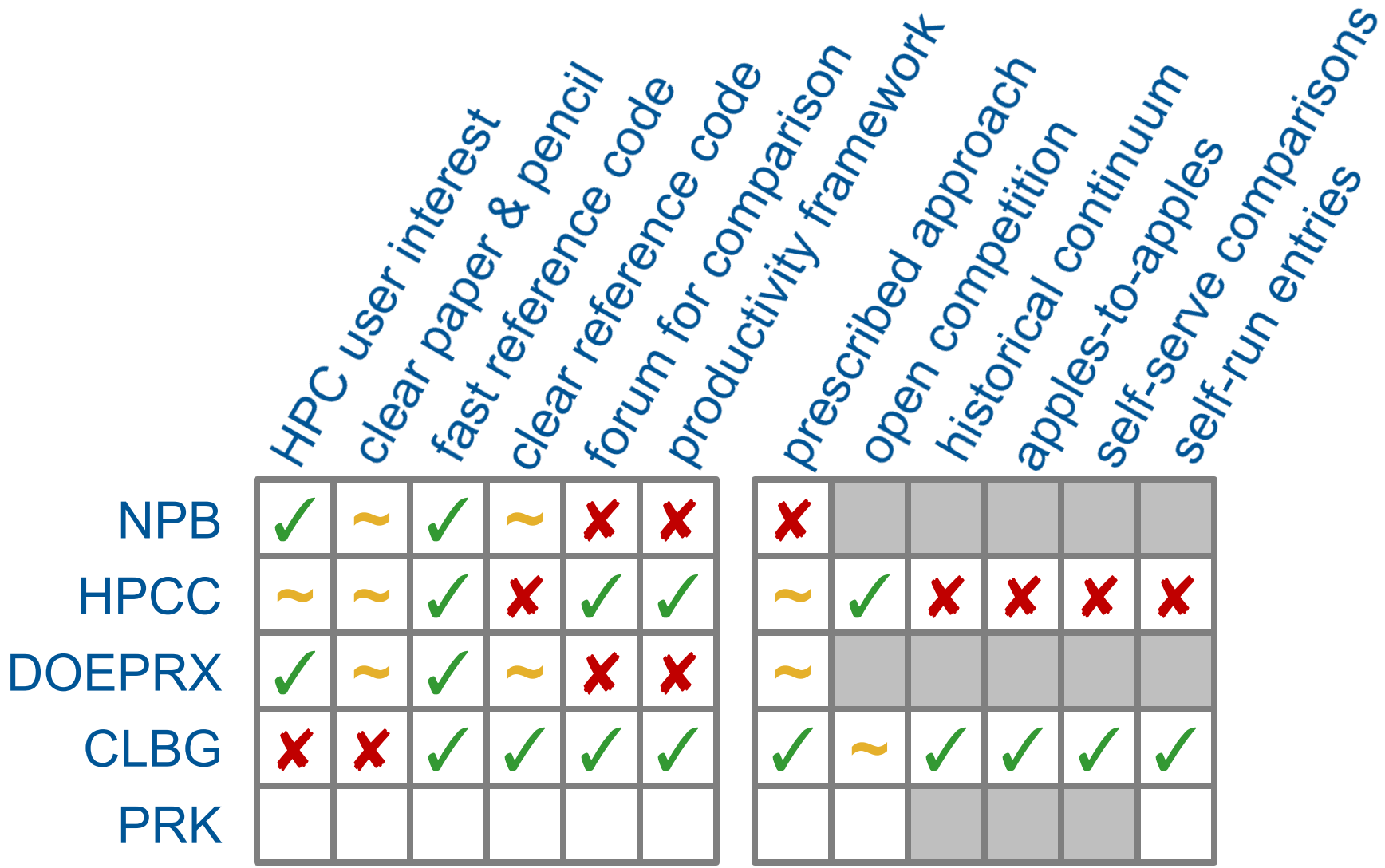
PRK: What it does well

- Establishes a set of basis vectors for real applications

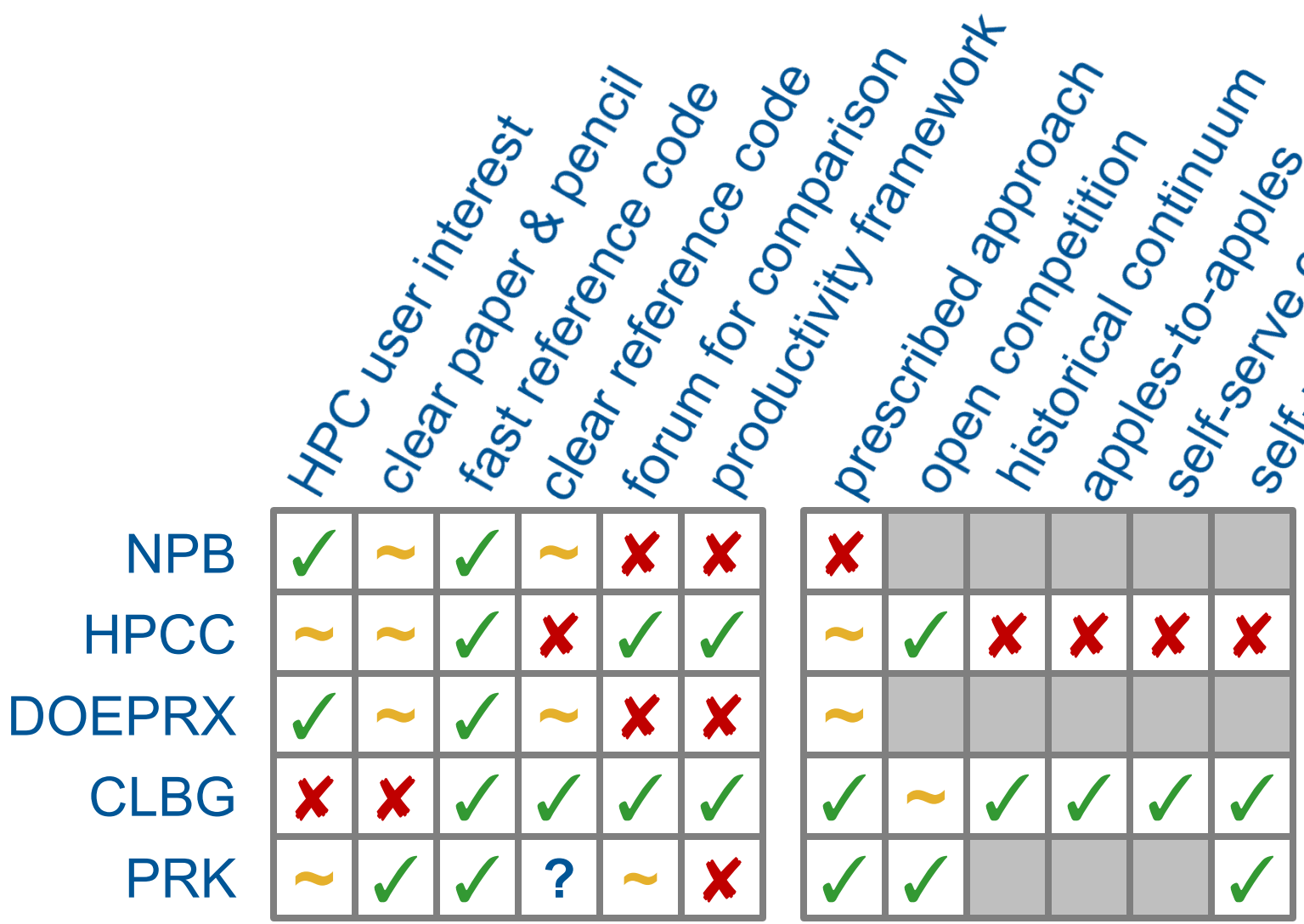
PRK: Where it falls short (at present)

- **Not a lot of uptake or interest as of yet**
  - Not for lack of interest among its curators
- **No formal competition or arena**
  - Yet, framework exists for running codes automatically

Benchmark Suite Scorecard



Benchmark Suite Scorecard



Wrap-up

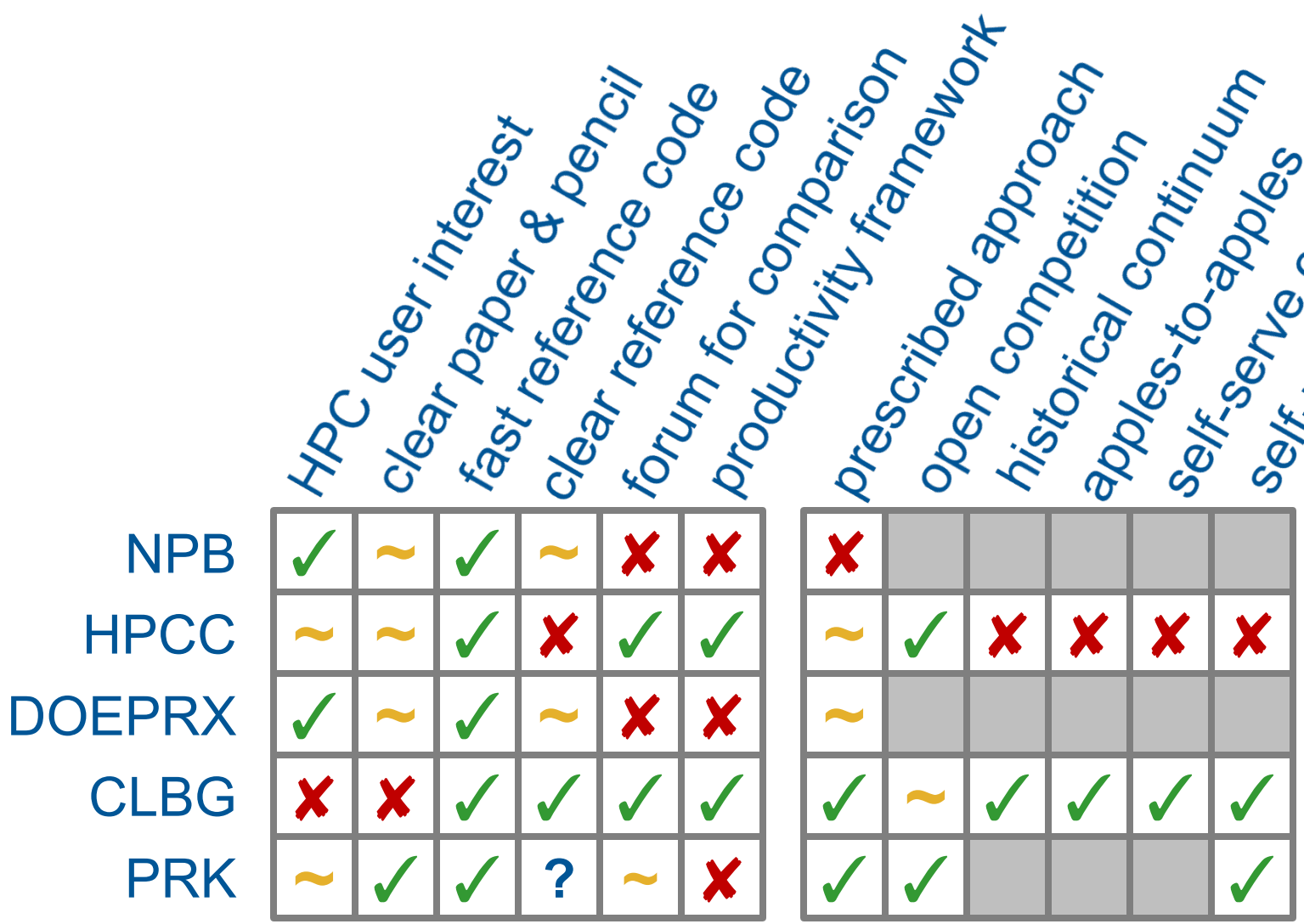
In Summary

Scoring existing HPC benchmark suites, each has a distinct approach and set of characteristics…

A Proposal:

- Create a group to curate a CLBG-style arena for the PRK
  - e.g., a DOE lab with access to supercomputer resources
  - or an academic group granted time on DOE resources
- Can we create something as viral and engaging for HPC as the CLBG is for mainstream programmers?

Benchmark Suite Scorecard



Questions?

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.

CRAY

THE SUPERCOMPUTER COMPANY