Entering the Fray

Chapel's Computer Language Benchmarks Game Entry

Brad Chamberlain, Ben Albrecht, Lydia Duncan, Ben Harshbarger, Elliot Ronaghan, Preston Sahabu, Mike Noakes, and Laura Delaney

CHIUW 2017, Orlando, FL

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.

CLBG: What it is

- **A suite of 13 "toy" benchmarks**
  - exercise key features like…
    …memory management
    …tasking and synchronization
    …arbitrary-precision math
    …vectorization
    …strings and regular expressions
  - single-node
  - serial, vectorizable, or multicore parallel

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

But wait…

- **This is IPDPS / HPC / Chapel…**

  …do we really care about a single-node benchmark suite?

- **Yes:**
  - success at the largest scales depends on good scalar performance
  - despite its focus on large-scale systems, Chapel is also intended for productive programming on workstations
  - several CLBG features match early user wishes
    - memory management
    - tasking and lightweight synchronization
    - arbitrary precision arithmetic
    - strings and regular expressions
    - vectorization
    - …
  - who doesn't enjoy a good game?

CLBG: What it is

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

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

Timeline

Feb 2016: Inquired about adding Chapel to the contest

Apr 2016: Chapel entries began to be accepted

Our approach:

- Submit codes that strive for performance without sacrificing elegance
- Submit codes that would serve as good models for learning the benchmark

May 2016: First program accepted

Sept 2016: Last program accepted, Chapel added to the site

CLBG: What it is

- **A suite of 13 "toy" benchmarks**
  - exercise key features like…
    …memory management
    …tasking and synchronization
    …arbitrary-precision math
    …vectorization
    …strings and regular expressions
  - single-node
  - serial, vectorizable, or multicore parallel
- **Imagine a 3D ragged matrix:**
  - with 13 benchmarks
    x ~2
    x a … sting
  - each entry contains:
    - source code
    - performance information
    - "code size"

Chapel added to site in September 2016

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

CLBG: What it is

- **A suite of 13 "toy" benchmarks**
  - exercise key features like…
    …memory management
    …tasking and synchronization
    …arbitrary-precision math
    …vectorization
    …strings and regular expressions
  - single-node
  - serial, vectorizable, or multicore parallel
- **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"

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?

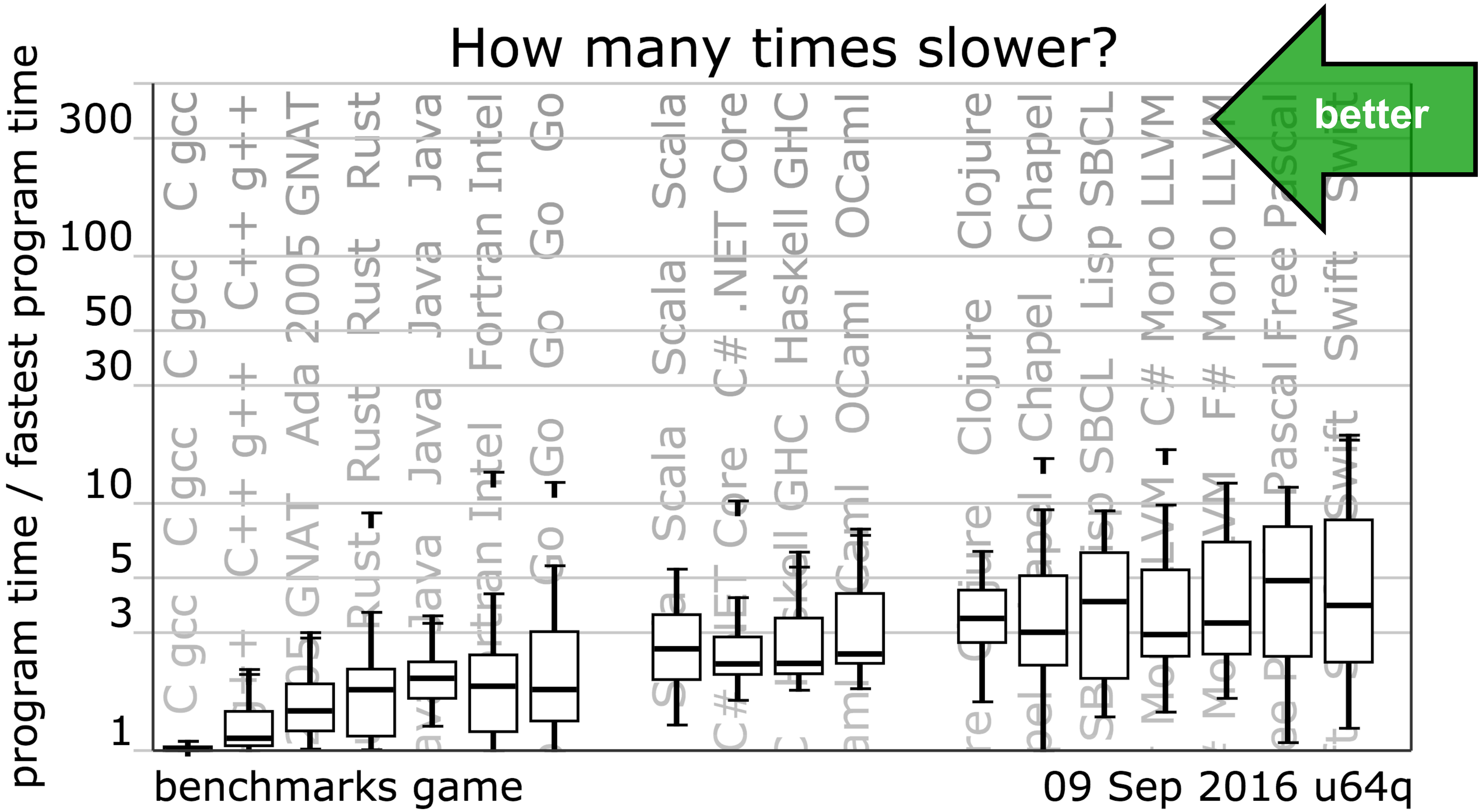
Ada C Chapel C# C++ Dart
Erlang F# Fortran Go Hack
Haskell Java JavaScript Lisp Lua
OCaml Pascal Perl PHP Python
Racket Ruby JRuby Rust Smalltalk
Swift TypeScript

{ for researchers } fast-faster-fastest

stories

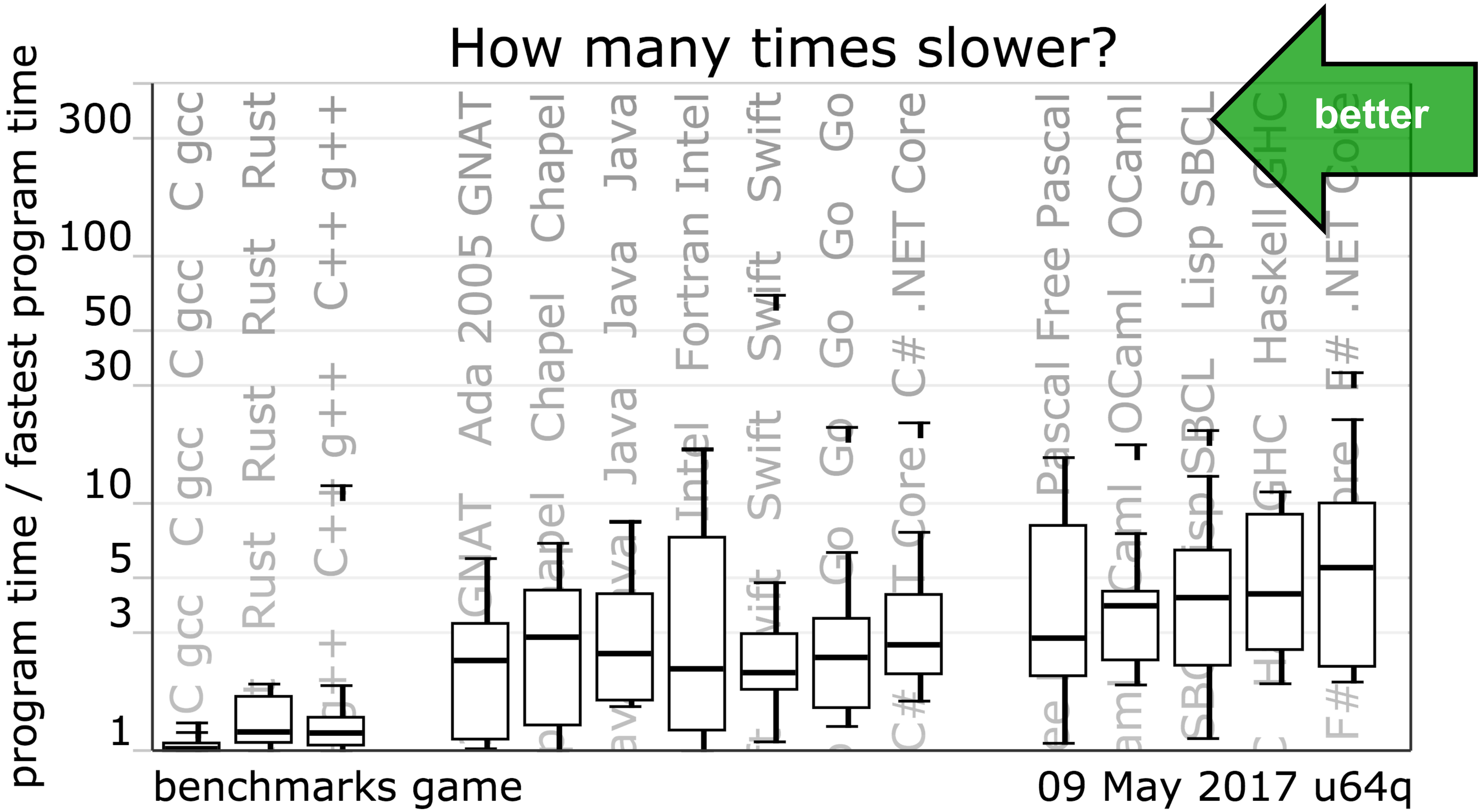
CLBG: Fast-faster-fastest graph (Sep 2016)

Site summary: relative performance (sorted by geometric mean)



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

Site summary: relative performance (sorted by geometric mean)



CLBG: Viewing per-benchmark 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					00% 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

Timeline

Feb 2016: Inquired about adding Chapel to the contest

Apr 2016: Chapel entries began to be accepted

Our approach:

- Submit codes that strive for performance without sacrificing elegance
- Submit codes that would serve as good models for learning the benchmark

May 2016: First program accepted

Sept 2016: Last program accepted, Chapel added to the site

Oct 2016: Upgraded to 1.14

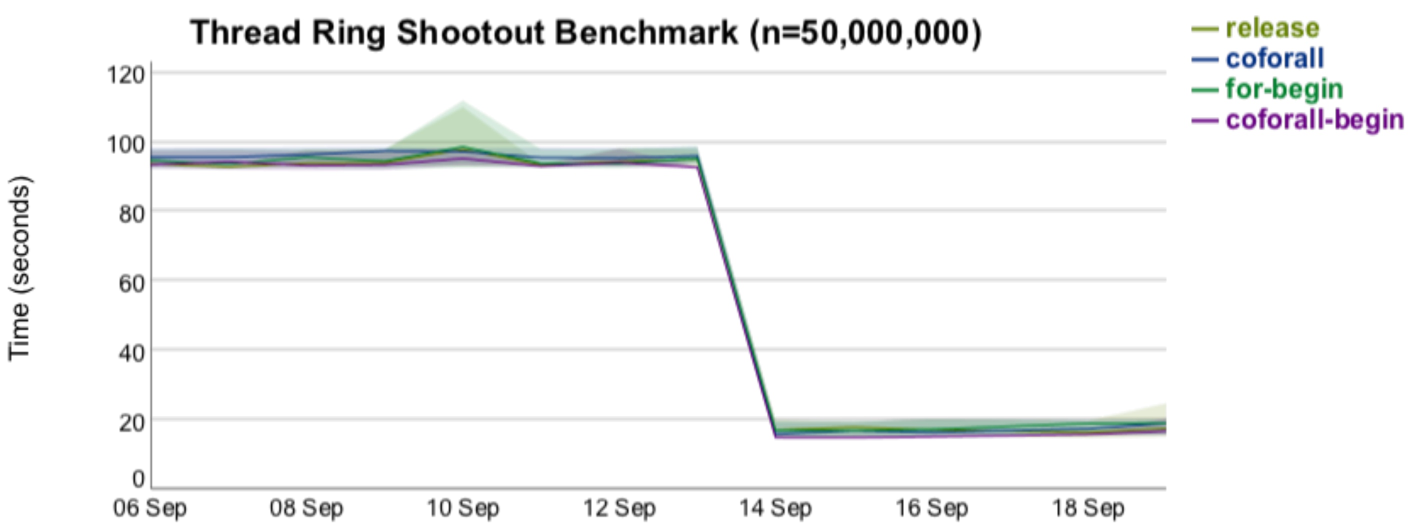
CLBG: Improvements due to 1.14

1.14 improved many benchmarks with no code changes:

- **thread-ring:** benefitted from qthread sync variable improvements
  - climbed ~16 slots ⇒ 5th fastest after Haskell, Go, F#, Scala
  - 1st most compact code followed by Ruby, Racket, Erlang, Ocaml, Python
- specifically, Chapel 1.14…

  …extended Qthreads sync vars to handle all Chapel operations

  …mapped Chapel sync vars directly to Qthreads sync vars (for simple types)

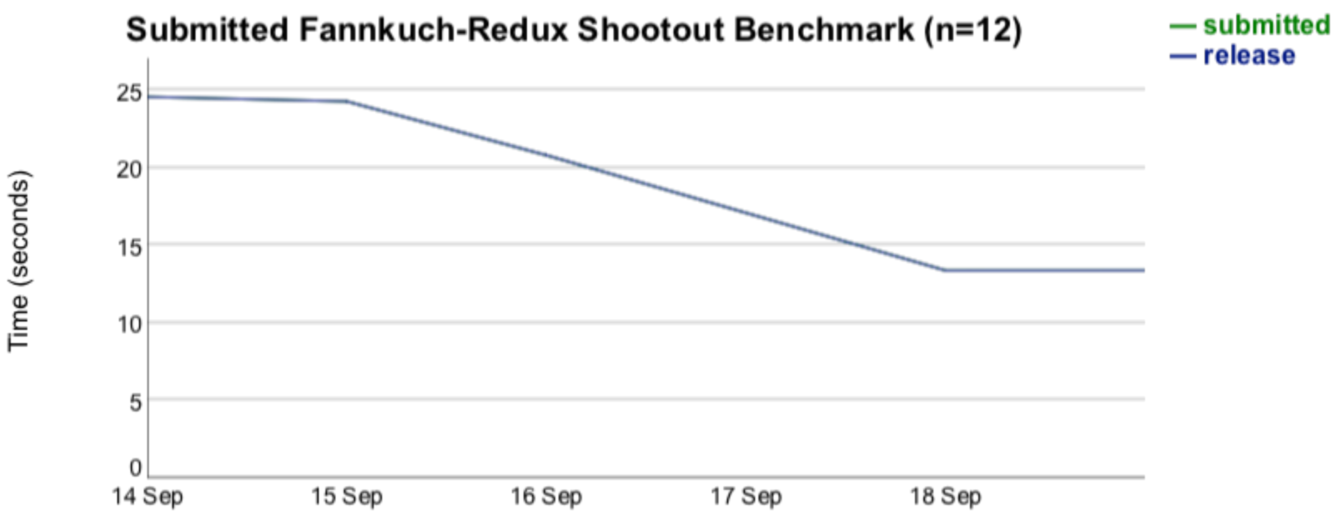


CLBG: Improvements due to 1.14

1.14 improved many benchmarks with no code changes:

- **fannkuch-redux:** benefitted from optimized array accesses
  - climbed from ~#22 to #6 in performance
  - ~1.5–2x more compact than most other top entries
- specifically, Chapel 1.14…

  …optimized an unnecessary multiply out of typical array accesses



  - this helped several other performance benchmarks as well
  - Chapel 1.15 made this optimization more precise and robust

CLBG: Improvements due to 1.14

1.14 improved many benchmarks with no code changes:

- **chameneos-redux:** benefitted from tasking improvements
  - climbed from ~#11 to #8 in terms of performance
- **binary-trees:** benefitted from jemalloc improvements
  - climbed ~2 performance slots as a result
  - still ~5x off from top entries which use explicit memory pools
- **n-body:** saw marginal improvements, but climbed ~17 slots
- **regex-dna, revcomp:** saw marginal improvements, climbed ~3 slots
- **meteor:** saw marginal improvements, climbed ~1 slot

Chapel CLBG Standings (Oct 17th)

- **8 / 13 programs in top-20 fastest:**
  - one #1 fastest:
    pidigits
  - 2 others in the top-5 fastest:
    meteor-contest
    thread-ring
  - 2 others in the top-10 fastest:
    chameneos-redux
    fannkuch-redux
  - 3 others in the top-20 fastest:
    binary-trees
    n-body
    spectral-norm
- **8 / 13 programs in top-20 smallest:**
  - two #1 smallest:
    n-body
    thread-ring
  - 2 others in the top-5 smallest:
    pidigits
    spectral-norm
  - 4 others in the top-20 smallest:
    chameneos-redux
    mandelbrot
    meteor-contest
    regex-dna

Chapel CLBG Standings (Apr 20th)

- **12 /13 programs in top-20 fastest:**
  - one #1 fastest:
    pidigits
  - 3 others in the top-5 fastest:
    chameneos-redux
    meteor-contest
    thread-ring
  - 3 others in the top-10 fastest:
    fannkuch-redux
    fasta
    mandelbrot
  - 5 others in the top-20 fastest:
    binary-trees
    k-nucleotide
    n-body
    regex-redux
    spectral-norm

- **8 / 13 programs in top-20 smallest:**
  - two #1 smallest:
    n-body
    thread-ring
  - 2 others in the top-5 smallest:
    pidigits
    spectral-norm
  - 1 other in the top-10 smallest:
    regex-redux
  - 3 others in the top-20 smallest:
    chameneos-redux
    mandelbrot
    meteor-contest

Timeline

Feb 2016: Inquired about adding Chapel to the contest

Apr 2016: Chapel entries began to be accepted

Our approach:

- Submit codes that strive for performance without sacrificing elegance
- Submit codes that would serve as good models for learning the benchmark

May 2016: First program accepted

Sept 2016: Last program accepted, Chapel added to the site

Oct 2016: Upgraded to 1.14

ongoing: Improved programs themselves in spare time

Apr 2017: Upgraded to 1.15

What's new with the CLBG since then?

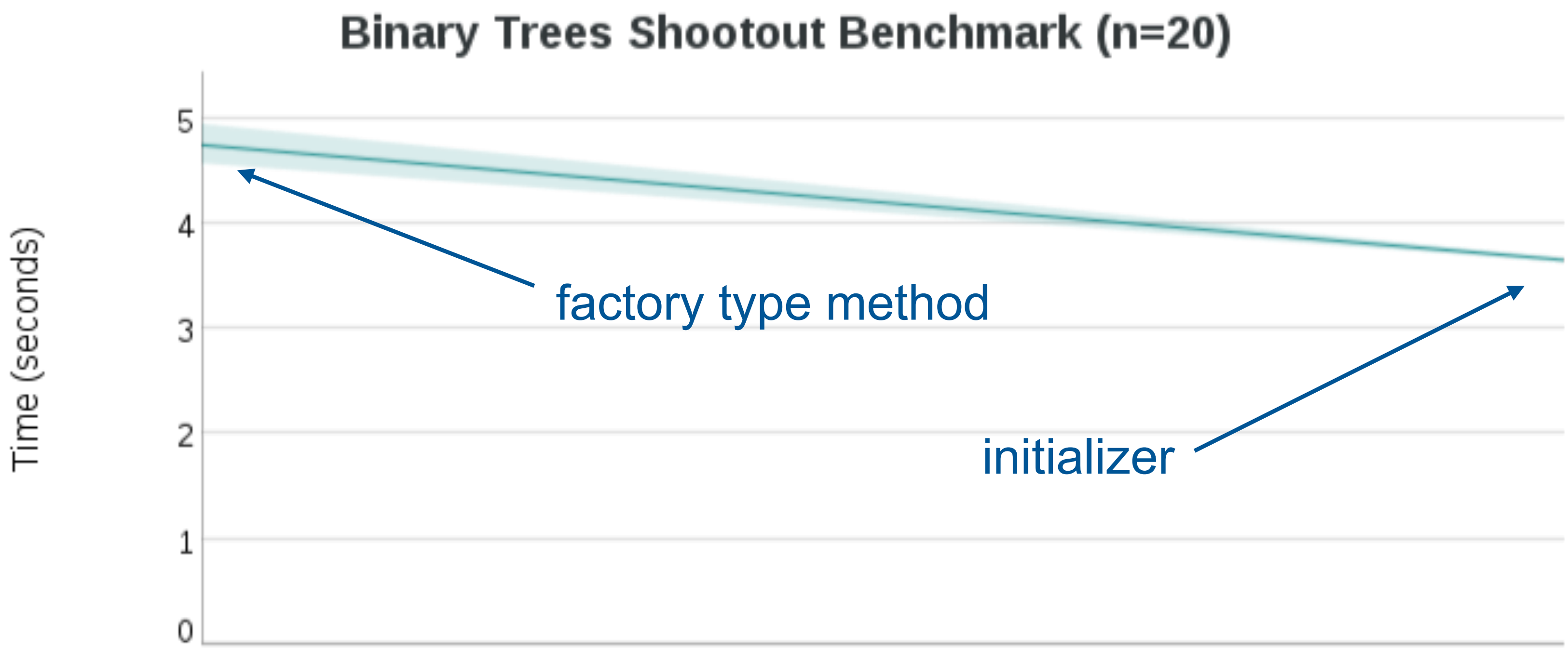
- **Two programs changed their official definitions:**

  binary-trees:
  - improved checksum to avoid false positives at 1/2, 1/4, 1/8 the memory
  - eliminated per-node data field
  - changed what trees are allocated and freed, slightly
  - increased the problem size

  regex:
  - changed the regular expression used
  - renamed the test to regex-redux
  - several versions are not currently passing due to these changes
    - our current standings may be due in part to this

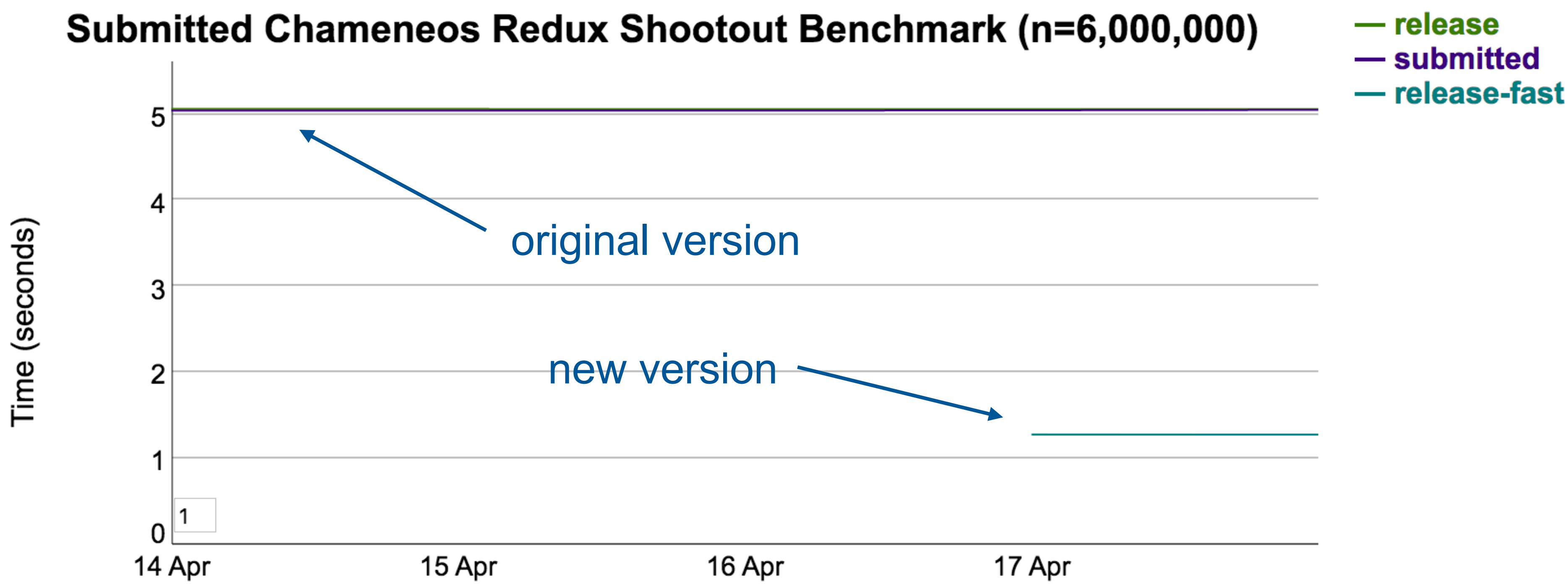
What's new with the Chapel CLBG entries?

- **We've submitted some new versions:**
  binary-trees: used an initializer rather than a factory type method



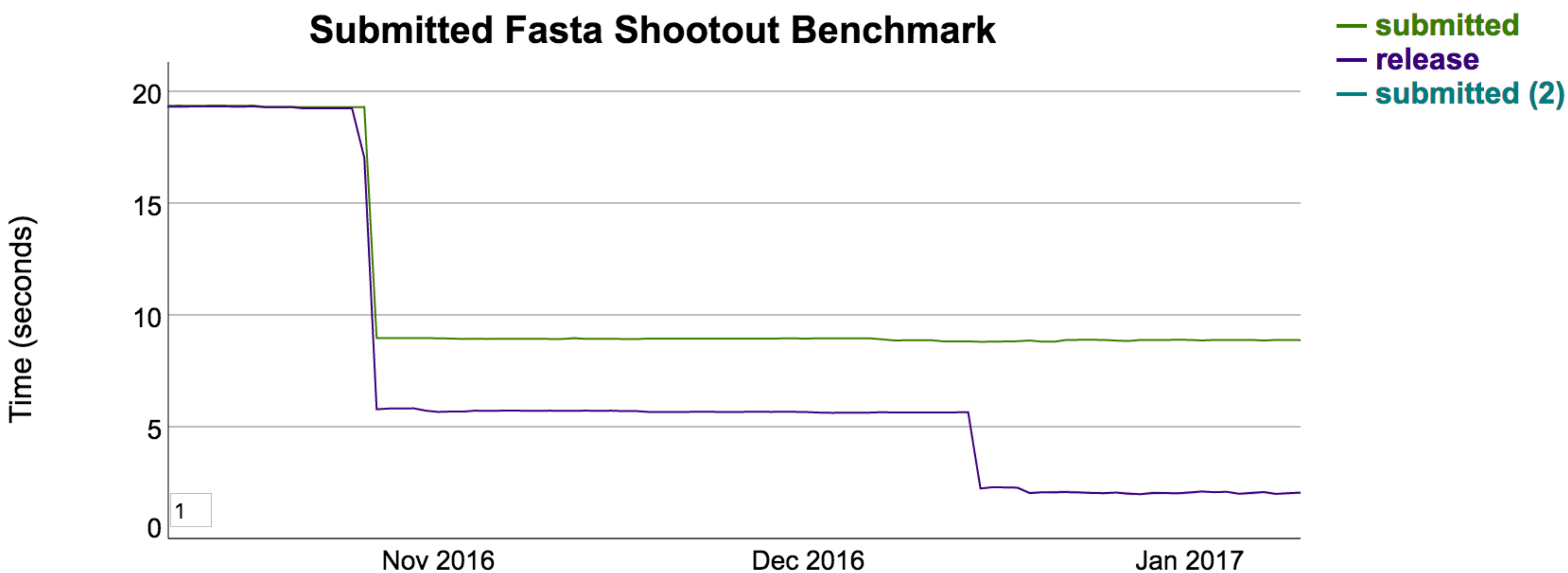
What's new with the Chapel CLBG entries?

- **We've submitted some new versions:**
  - **binary-trees:** used an initializer rather than a factory type method
  - **chameneos-redux:** increased parallelism and tuned a spin-wait



What's new with the Chapel CLBG entries?

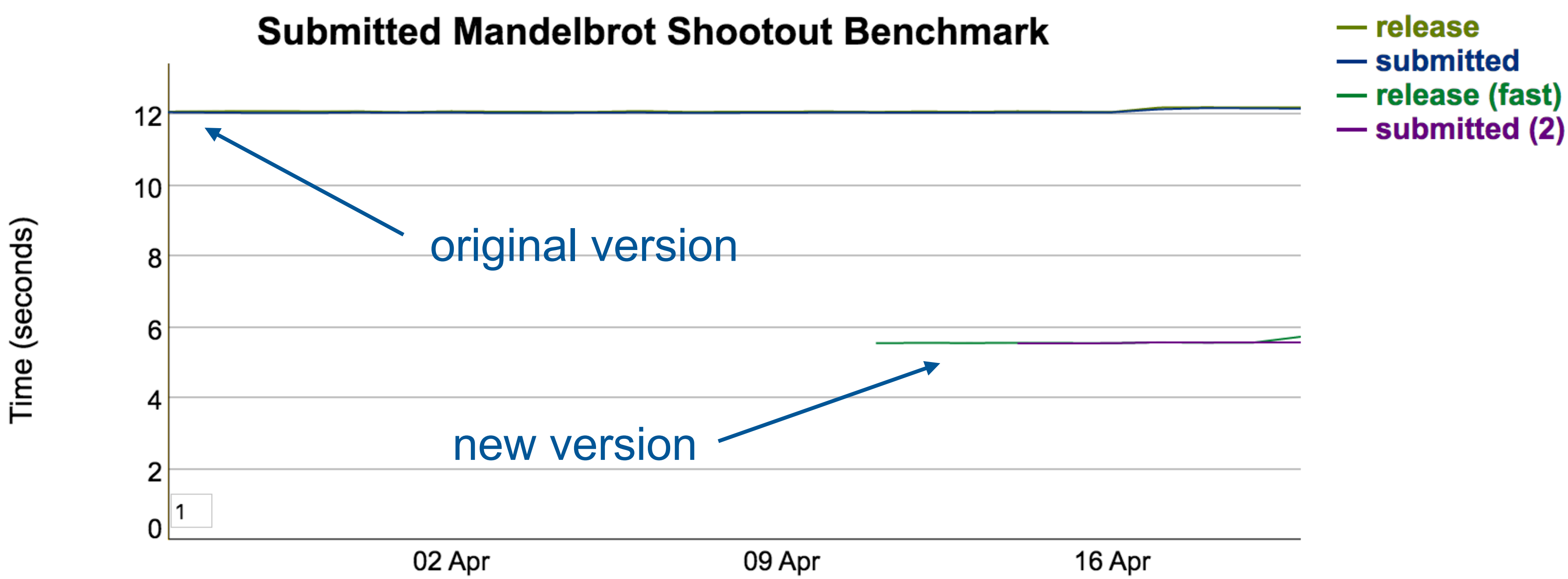
- **We've submitted some new versions:**
  - **binary-trees:** used an initializer rather than a factory type method
  - **chameneos-redux:** increased parallelism and tuned a spin-wait
  - **fasta:** implemented a parallel version and tuned for clarity and speed
    - also, changed some 'var' declarations due to const-checking improvements



What's new with the Chapel CLBG entries?

- **We've submitted some new versions:**
  binary-trees: used an initializer rather than a factory type method
  chameneos-redux: increased parallelism and tuned a spin-wait
  fasta: implemented a parallel version and tuned for clarity and speed
  - also, changed some 'var' declarations due to const-checking improvements

  mandelbrot: accelerated by hoisting values and using tuples of values



What's new with the Chapel CLBG entries?

- **We've submitted some new versions:**

  binary-trees: used an initializer rather than a factory type method

  chameneos-redux: increased parallelism and tuned a spin-wait

  fasta: implemented a parallel version and tuned for clarity and speed
  - also, changed some 'var' declarations due to const-checking improvements

  mandelbrot: accelerated by hoisting values and using tuples of values

  meteor-fast: fixed a race condition caused by array memory changes
  - textbook example of an array being used by a 'begin' task

  pidigits: submitted a version that uses 'bigint's
  - currently the #1 fastest version, and also quite elegant

- **Note that some of these changes followed the 1.15 release**
  - As such, not all are found in examples/benchmarks/shootout/ for 1.15

CLBG: Comparing Pairs of Languages

Can also compare languages pair-wise (for 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

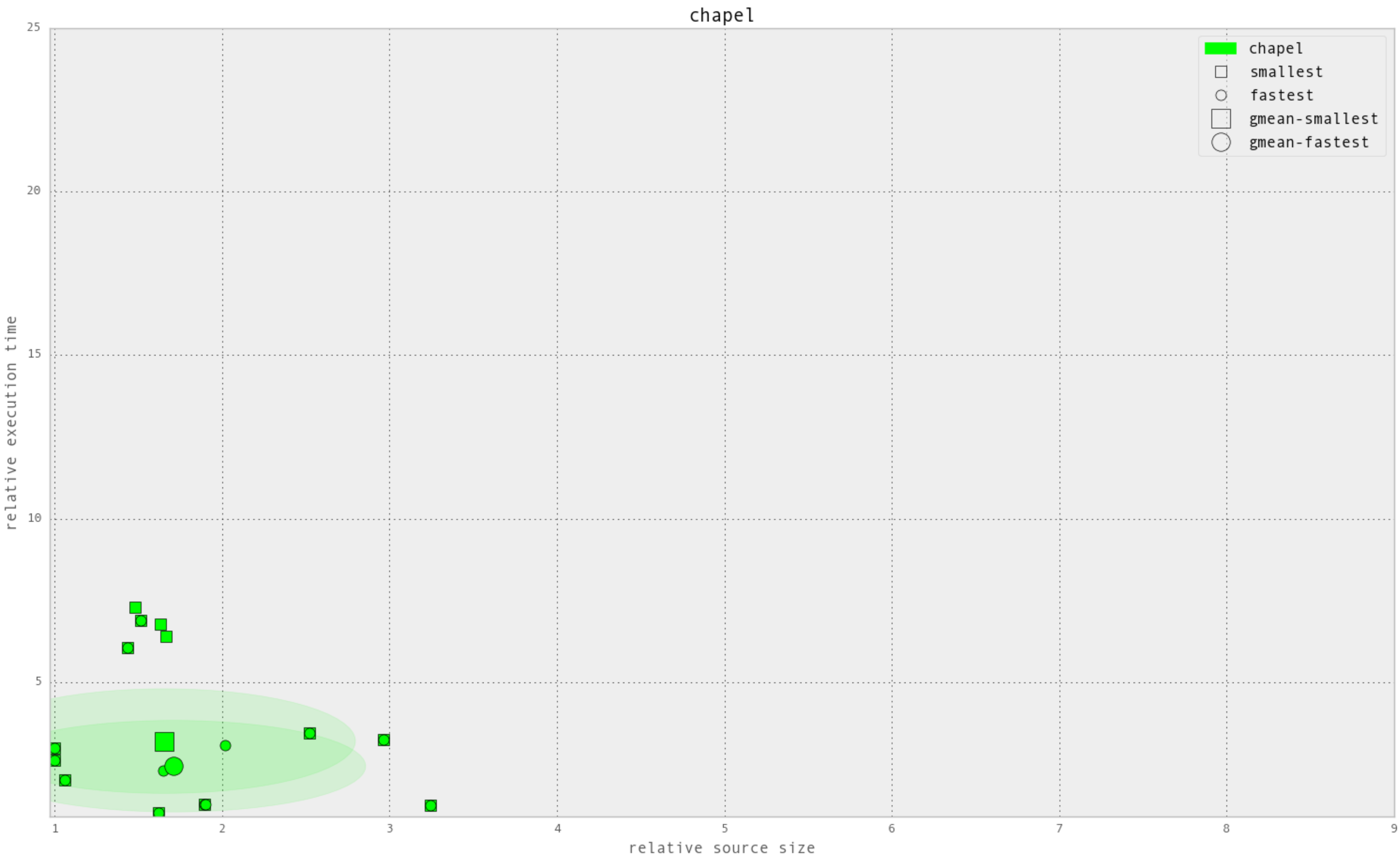
source	secs	mem	gz	cpu	cpu load
Chapel	**11.38**	46,056	728	45.18	100% 99% 99% 100%

Happily, all the data is open!

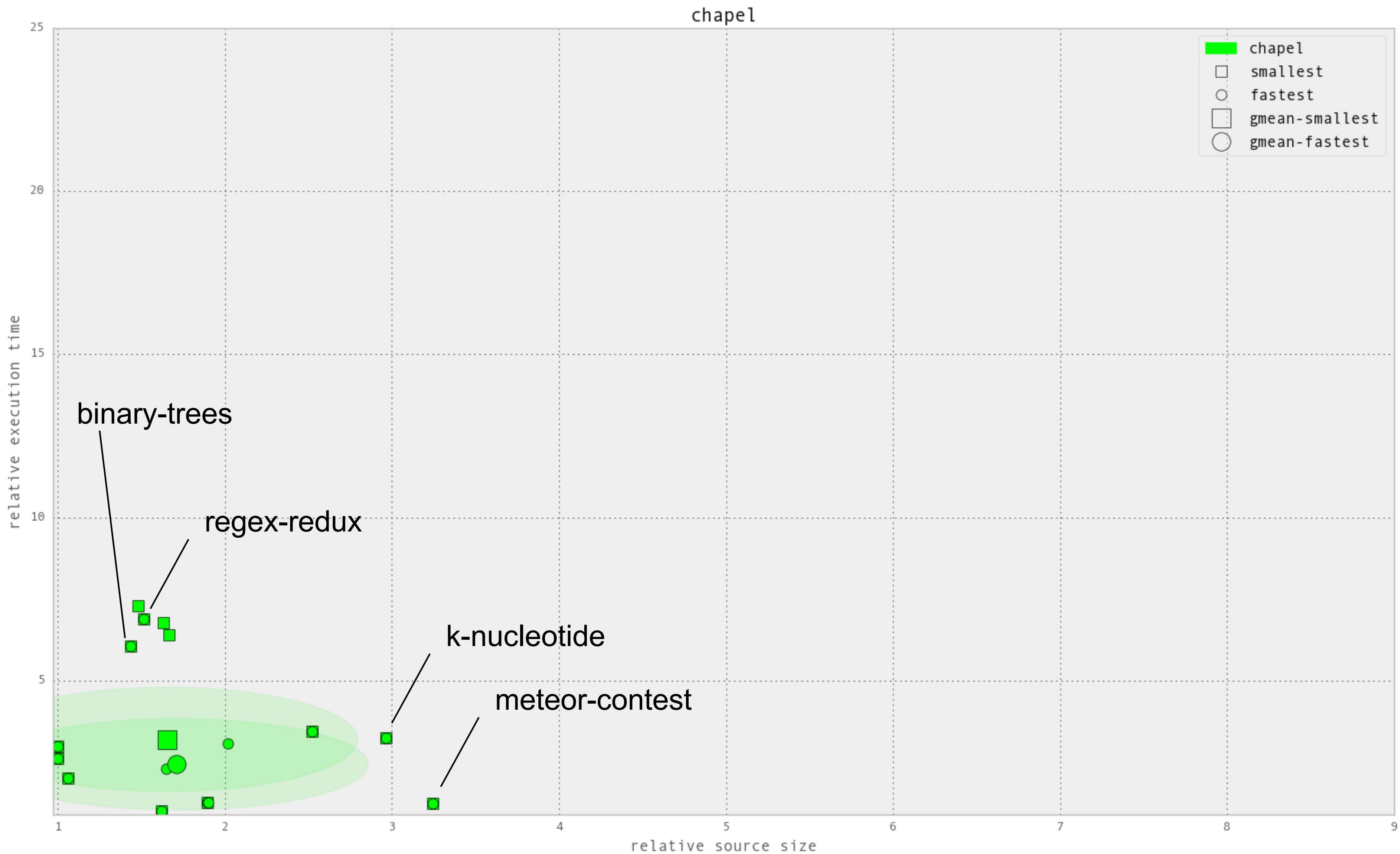
CLBG Scatter Plots

- **The following graphs use the CLBG's normalized ratios**
  - Graphs were created using April 20th data (current at time of creation)
    - things have continued to be in flux again since that date…

Chapel entries (Apr 2017)



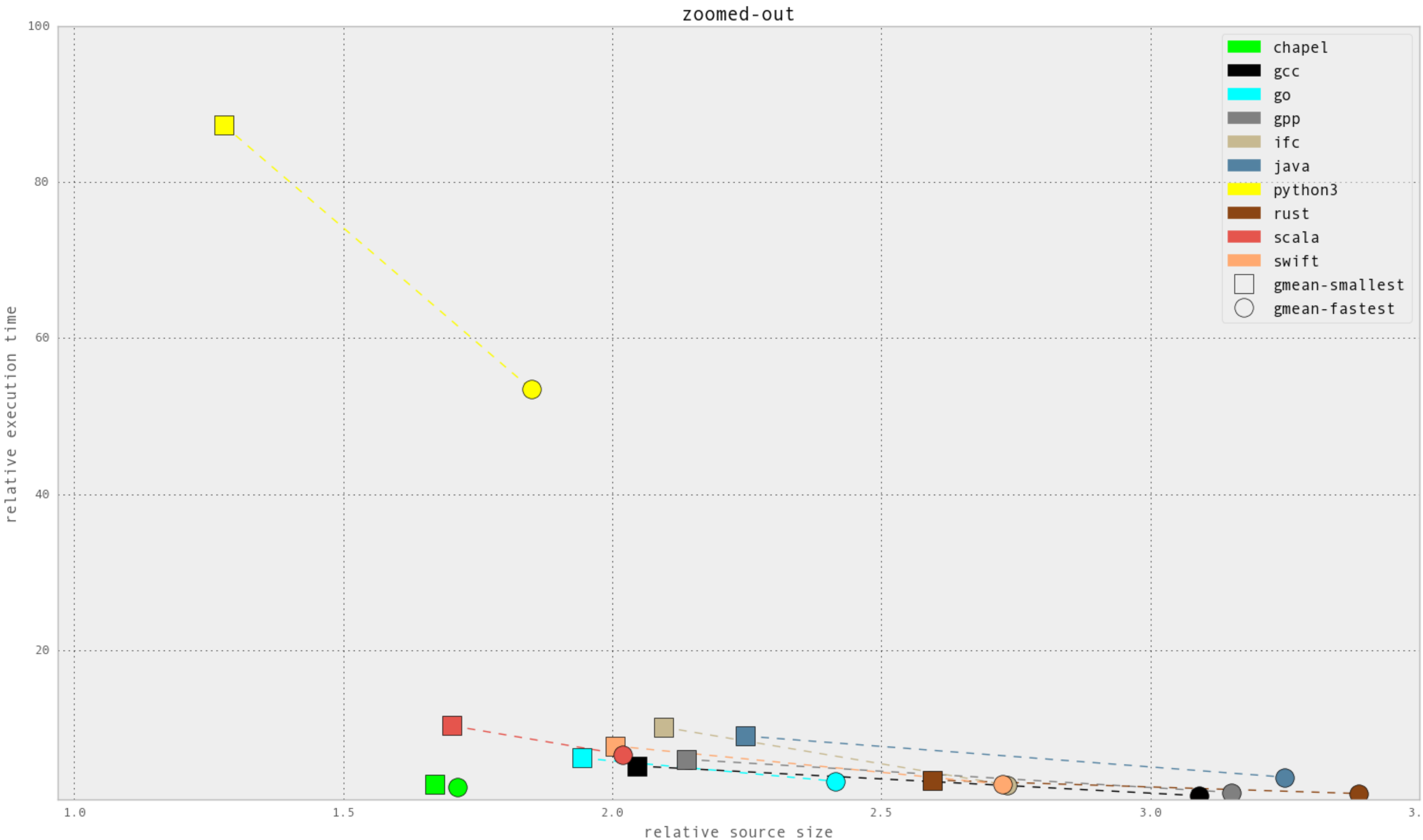
Chapel entries (Apr 2017, noting outliers)



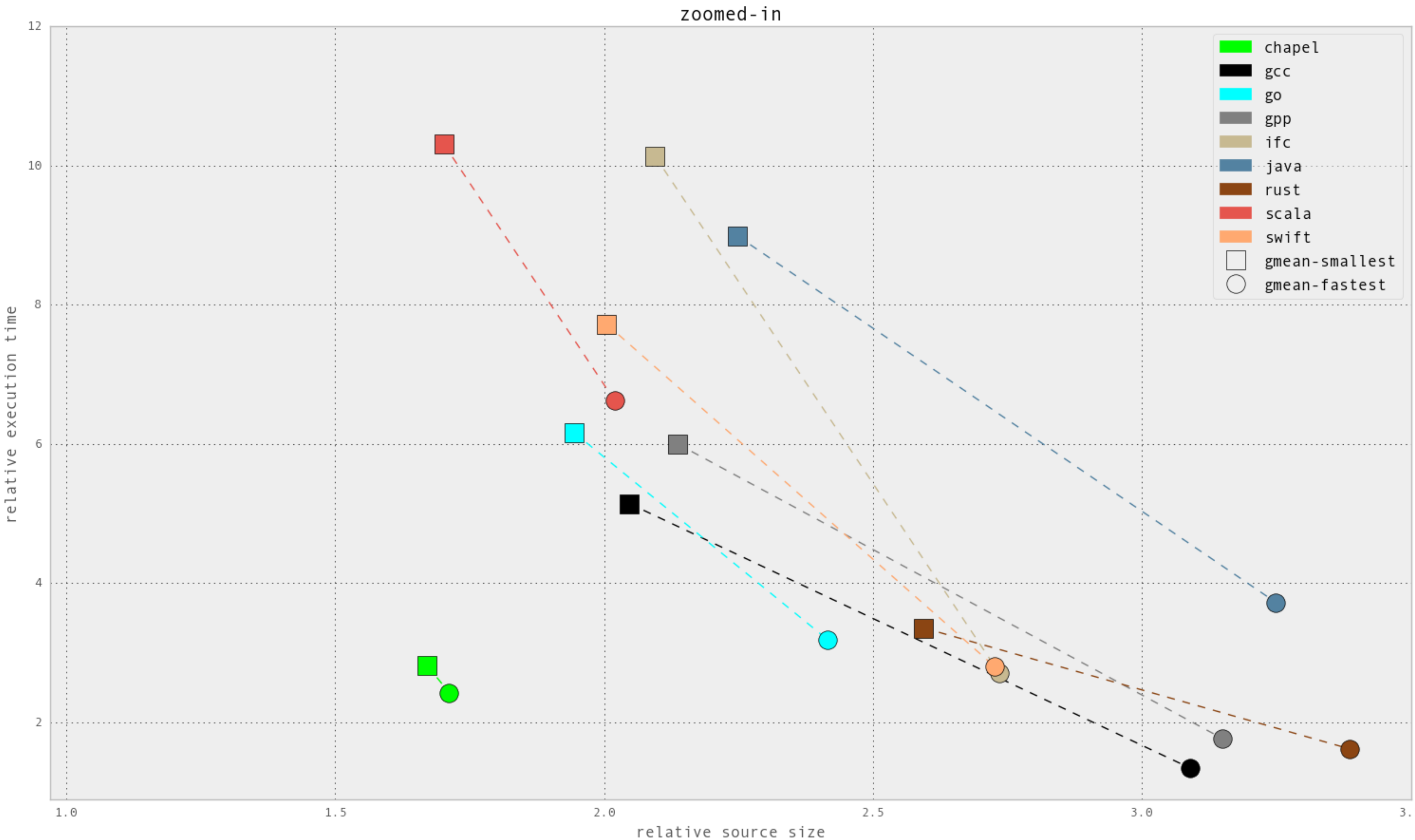
Chapel vs. 9 other languages

Chapel vs. 9 other languages (zoomed out)

Cross-Language Summary



Cross-Language Summary (no Python)



Chapel CLBG Standings as of Apr 20th

- **12 /13 programs in top-20 fastest:**
  - one #1 fastest:
    pidigits
  - 3 others in the top-5 fastest:
    chameneos-redux
    meteor-contest
    thread-ring
  - 3 others in the top-10 fastest:
    fannkuch-redux
    fasta
    mandelbrot
  - 5 others in the top-20 fastest:
    binary-trees
    k-nucleotide
    n-body
    regex-redux
    spectral-norm
- **8 / 13 programs in top-20 smallest:**
  - two #1 smallest:
    n-body
    thread-ring
  - 2 others in the top-5 smallest:
    pidigits
    spectral-norm
  - 1 other in the top-10 smallest:
    regex-redux
  - 3 others in the top-20 smallest:
    chameneos-redux
    mandelbrot
    meteor-contest

Comparing Chapel vs. C Chameneos

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 4th-place 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 1st-place C gcc entry

Comparing Chapel vs. C Chameneos

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

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

  const group1 = [i in 1..popSize1] new Ch
  const group2 = [i in 1..popSize2] new Ch

  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
  writeln();
}


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

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

  delete place;
}
```


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


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

                                                   [2048];
                                                   ;
                                                   _idx;
                                                   sical_id;
                                                   e_id;
                                                   _cores;
                                                   c_id;
                                                   _count;

    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);

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


```
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 4th-place Chapel entry   *excerpt from 2863 gz 1st-place C gcc entry*

Comparing Chapel vs. C Chameneos

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);
```

```
char const*                core_id_str          =
size_t                     core_id_str_len      =
char const*                cpu_cores_str        =
size_t                     cpu_cores_str_len    =

CPU_ZERO(&active_cpus);
sched_getaffinity(0, sizeof(active_cpus), &active_
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;
}
```

```
  const place = new MeetingPlace(numMeetings);

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

  delete place;
}
```

```
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 1210 gz 4th-place Chapel entry

excerpt from 2863 gz 1st-place C gcc entry

Chapel CLBG Standings as of Apr 20th

- **12 /13 programs in top-20 fastest:**
  - one #1 fastest:
    pidigits
  - 3 others in the top-5 fastest:
    chameneos-redux
    meteor-contest
    thread-ring
  - 3 others in the top-10 fastest:
    fannkuch-redux
    fasta
    mandelbrot
  - 5 others in the top-20 fastest:
    binary-trees
    k-nucleotide
    n-body
    regex-redux
    spectral-norm

- **8 / 13 programs in top-20 smallest:**
  - two #1 smallest:
    n-body
    thread-ring
  - 2 others in the top-5 smallest:
    pidigits
    spectral-norm
  - 1 other in the top-10 smallest:
    regex-redux
  - 3 others in the top-20 smallest:
    chameneos-redux
    mandelbrot
    meteor-contest

Comparing Chapel vs. C pidigits

```
use BigInteger;

config const n = 50;          // Compute n digits of pi, 50 by default

proc main() {
  param digitsPerLine = 10;

  // Generate n digits, printing them in groups of digitsPerLine
  for (d, i) in genDigits(n) {
    write(d);
    if i % digitsPerLine == 0 then
      writeln("\t:", i);
  }

  // Pad out any trailing digits for the final line
  if n % digitsPerLine then
    writeln(" " * (digitsPerLine - n % digitsPerLine), "\t:", n);
}


iter genDigits(numDigits) {
  var numer, denom: bigint = 1,
      accum, tmp1, tmp2: bigint;

  var i, k = 1;
  while i <= numDigits {
    nextTerm(k);
    k += 1;
    if numer <= accum {
      const d = extractDigit(3);
      if d == extractDigit(4) {
        yield(d, i);
        eliminateDigit(d);
        i += 1;
      }
    }
  }

  proc nextTerm(k) {
    const k2 = 2 * k + 1;

    accum.addmul(numer, 2);
    accum *= k2;
    denom *= k2;
    numer *= k;
  }

  proc extractDigit(nth) {
    tmp1.mul(numer, nth);
    tmp2.add(tmp1,accum);
    tmp1.div_q(tmp2, denom);

    return tmp1: int;
  }

  proc eliminateDigit(d) {
    accum.submul(denom, d);
    accum *= 10;
    numer *= 10;
  }
```

excerpt from 423 gz 1st-place Chapel entry

```
#include <stdio.h>
#include <stdlib.h>
#include <gmp.h>

mpz_t tmp1, tmp2, acc, den, num;
typedef unsigned int ui;

ui extract_digit(ui nth) {
   // joggling between tmp1 and tmp2, so GMP won't have to use temp buffers
   mpz_mul_ui(tmp1, num, nth);
   mpz_add(tmp2, tmp1, acc);
   mpz_tdiv_q(tmp1, tmp2, den);

   return mpz_get_ui(tmp1);
}

void eliminate_digit(ui d) {
   mpz_submul_ui(acc, den, d);
   mpz_mul_ui(acc, acc, 10);
   mpz_mul_ui(num, num, 10);
}

void next_term(ui k) {
   ui k2 = k * 2U + 1U;

   mpz_addmul_ui(acc, num, 2U);
   mpz_mul_ui(acc, acc, k2);
   mpz_mul_ui(den, den, k2);
   mpz_mul_ui(num, num, k);
}

int main(int argc, char **argv) {
   ui d, k, i;
   int n = atoi(argv[1]);

   mpz_init(tmp1);
   mpz_init(tmp2);

   mpz_init_set_ui(acc, 0);
   mpz_init_set_ui(den, 1);
   mpz_init_set_ui(num, 1);

   for (i = k = 0; i < n;) {
      next_term(++k);
      if (mpz_cmp(num, acc) > 0)
         continue;

      d = extract_digit(3);
      if (d != extract_digit(4))
         continue;

      putchar('0' + d);
      if (++i % 10 == 0)
         printf("\t:%u\n", i);
      eliminate_digit(d);
   }

   return 0;
}
```

excerpt from 448 gz 4th-place C gcc entry

Comparing Chapel vs. C pidigits

```
use BigInteger;

config const n = 50;          // Compute n digits of pi, 50 by default

proc main() {
  param digitsPerLine = 10;

  // Generate n digits, printing them in groups of digitsPerLine
  for (d, i) in genDigits(n) {
    write(d);
    if i % digitsPerLine == 0 then
      writeln("\t:", i);
  }

  // Pad out any trailing digits for the final line
  if n % digitsPerLine then
    writeln(" " * (digitsPerLine - n % digitsPerLine), "\t:", n);
}


iter genDigits(numDigits) {
  var numer, denom: bigint = 1,
      accum, tmp1, tmp2: bigint;

  var i, k = 1;
  while i <= numDigits {
    nextTerm(k);
    k += 1;
    if numer <= accum {
      const d = extractDigit(3);
      if d == extractDigit(4) {
        yield(d, i);
        eliminateDigit(d);
        i += 1;
      }
    }
  }

  proc nextTerm(k) {
    const k2 = 2 * k + 1;

    accum.addmul(numer, 2);
    accum *= k2;
    denom *= k2;
    numer *= k;
  }

  proc extractDigit(nth) {
    tmp1.mul(numer, nth);
    tmp2.add(tmp1,accum);
    tmp1.div_q(tmp2, denom);

    return tmp1: int;
  }

  proc eliminateDigit(d) {
    accum.submul(denom, d);
    accum *= 10;
    numer *= 10;
  }
```


```
proc nextTerm(k) {
  const k2 = 2 * k + 1;

  accum.addmul(numer, 2);
  accum *= k2;
  denom *= k2;
  numer *= k;
}
```


```
#include <stdio.h>
#include <stdlib.h>
#include <gmp.h>

mpz_t tmp1, tmp2, acc, den, num;
typedef unsigned int ui;

emp buffers

int main(int argc, char **argv) {
   ui d, k, i;
   int n = atoi(argv[1]);

   mpz_init(tmp1);
   mpz_init(tmp2);

   mpz_init_set_ui(acc, 0);
   mpz_init_set_ui(den, 1);
   mpz_init_set_ui(num, 1);

   for (i = k = 0; i < n;) {
      next_term(++k);
      if (mpz_cmp(num, acc) > 0)
         continue;

      d = extract_digit(3);
      if (d != extract_digit(4))
         continue;

      putchar('0' + d);
      if (++i % 10 == 0)
         printf("\t:%u\n", i);
      eliminate_digit(d);
   }

   return 0;
}
```

excerpt from 423 gz 1st-place Chapel entry

excerpt from 448 gz 4th-place C gcc entry

Comparing Chapel vs. C pidigits

```chapel
use BigInteger;

config const n = 50;          // Compute n digits of pi, 50 by default

proc main() {
  param digitsPerLine = 10;

  // Generate n digits, printing them in groups of digitsPerLine
  for (d, i) in genDigits(n) {
    write(d);
    if i % digitsPerLine == 0 then
      writeln("\t:", i);
  }

  // Pad out any trailing digits for the final line
  if n % digitsPerLine then
    writeln(" " * (digitsPerLine - n % digitsPerLine), "\t:", n);
}


iter genDigits(numDigits) {
  var numer, denom: bigint = 1,
      accum, tmp1, tmp2: bigint;

  var i, k = 1;
```

```c
void next_term(ui k) {
   ui k2 = k * 2U + 1U;

   mpz_addmul_ui(acc, num, 2U);
   mpz_mul_ui(acc, acc, k2);
   mpz_mul_ui(den, den, k2);
   mpz_mul_ui(num, num, k);
}
```

```chapel
    tmp1.div_q(tmp2, denom);

    return tmp1: int;
  }

  proc eliminateDigit(d) {
    accum.submul(denom, d);
    accum *= 10;
    numer *= 10;
  }
```

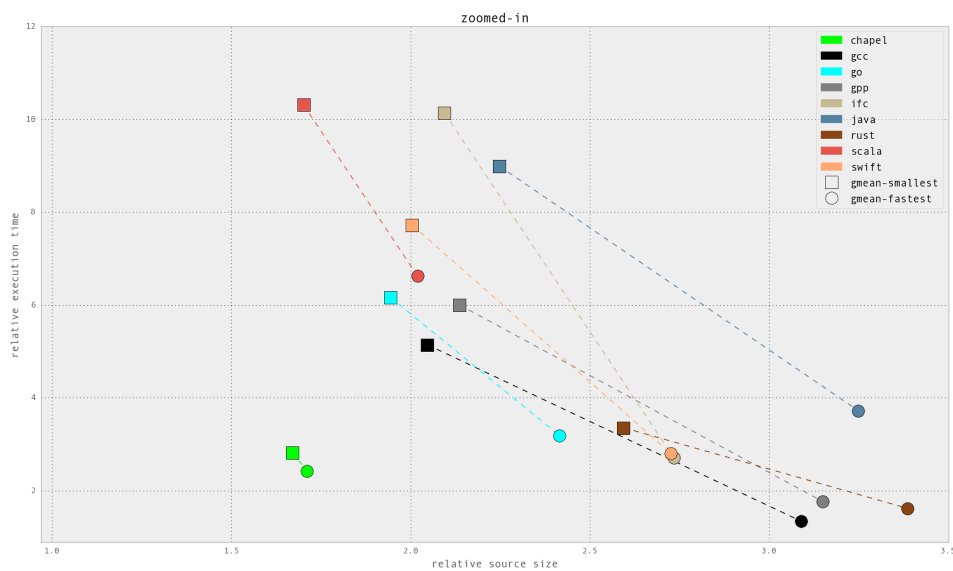
```c
#include <stdio.h>
#include <stdlib.h>
#include <gmp.h>

mpz_t tmp1, tmp2, acc, den, num;
typedef unsigned int ui;

ui extract_digit(ui nth) {
   // joggling between tmp1 and tmp2, so GMP won't have to use temp buffers
   mpz_mul_ui(tmp1, num, nth);
   mpz_add(tmp2, tmp1, acc);
   mpz_tdiv_q(tmp1, tmp2, den);

   return mpz_get_ui(tmp1);
}

void eliminate_digit(ui d) {
   mpz_submul_ui(acc, den, d);
   mpz_mul_ui(acc, acc, 10);
   mpz_mul_ui(num, num, 10);
}

void next_term(ui k) {
   ui k2 = k * 2U + 1U;

   mpz_addmul_ui(acc, num, 2U);
   mpz_mul_ui(acc, acc, k2);
   mpz_mul_ui(den, den, k2);
   mpz_mul_ui(num, num, k);
}

int main(int argc, char **argv) {
   ui d, k, i;
   int n = atoi(argv[1]);

   mpz_init(tmp1);
   mpz_init(tmp2);

   mpz_init_set_ui(acc, 0);
   mpz_init_set_ui(den, 1);
   mpz_init_set_ui(num, 1);

   for (i = k = 0; i < n;) {
      next_term(++k);
      if (mpz_cmp(num, acc) > 0)
         continue;

      d = extract_digit(3);
      if (d != extract_digit(4))
         continue;

      putchar('0' + d);
      if (++i % 10 == 0)
         printf("\t:%u\n", i);
      eliminate_digit(d);
   }

   return 0;
}
```

excerpt from 423 gz 1st-place Chapel entry

excerpt from 448 gz 4th-place C gcc entry

CLBG: Next Steps

- **Additional Performance Improvements**
  - Improve vectorization support
  - Optimize idioms used by string-related benchmarks
    - strings, associative domains/arrays, byte arrays
  - Support memory pools?
- **How to shine a light on these qualitative comparisons?**
  - Chapel blog articles?



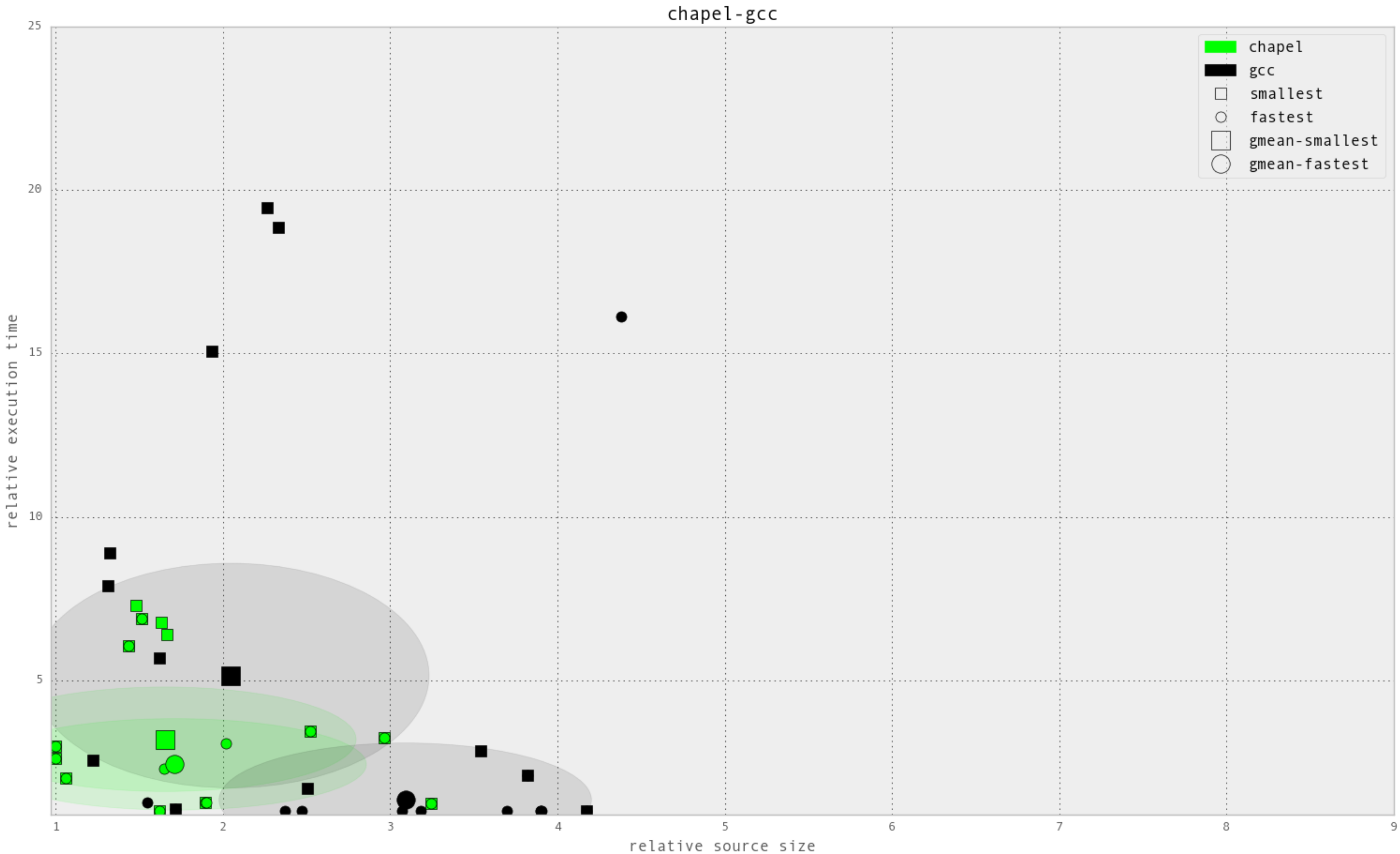
CLBG: Next Major Steps

- **How can we create a similar competition within HPC? (where “we” == “the HPC community”, not Chapel)**
  - multi-language
  - ongoing
  - open
  - addictive
- **Intel Parallel Research Kernels (PRK) as a possible basis**
  - My EMBRACE talk this morning has related thoughts

Questions?

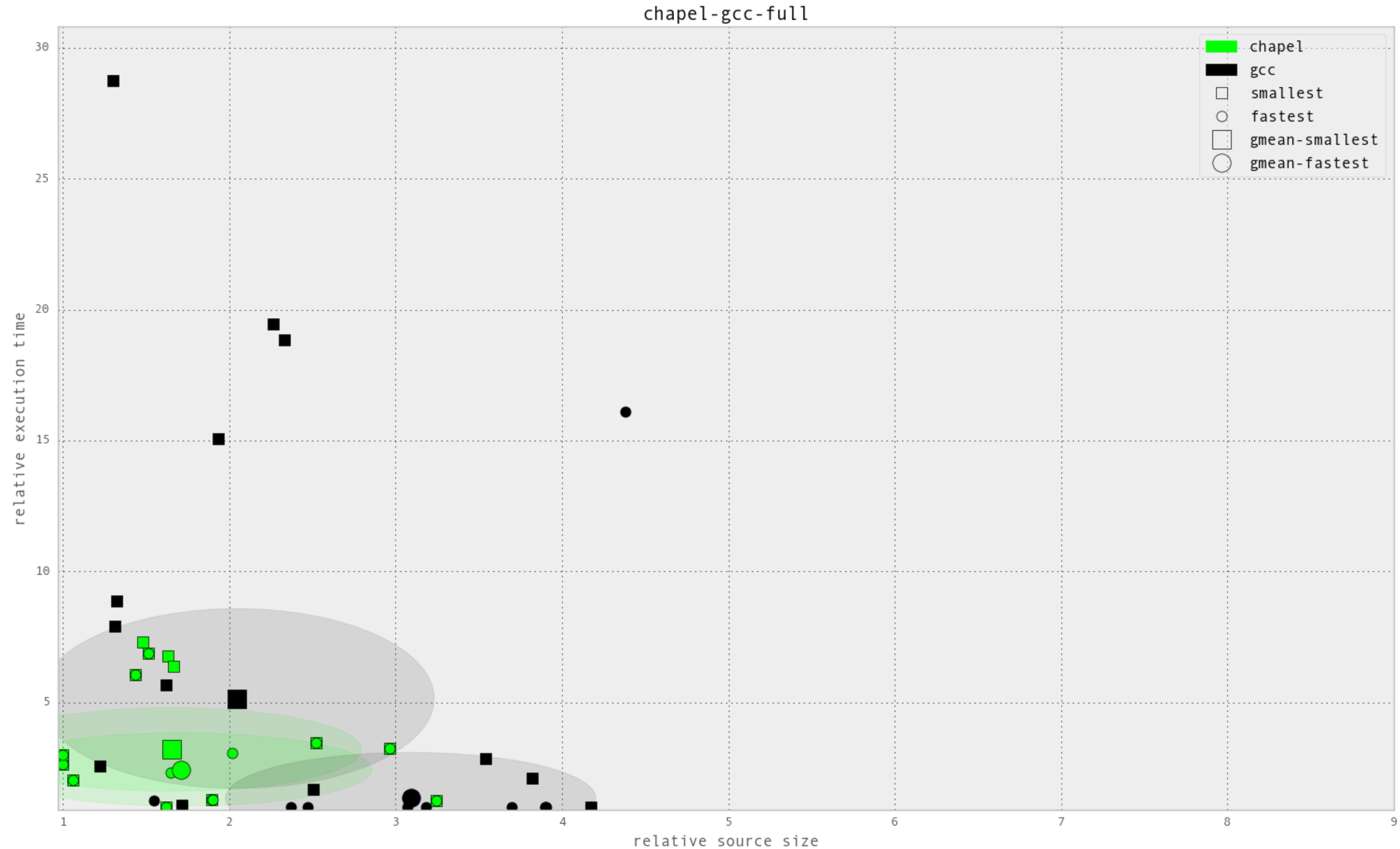
CLBG Scatter Plots

Chapel vs. C

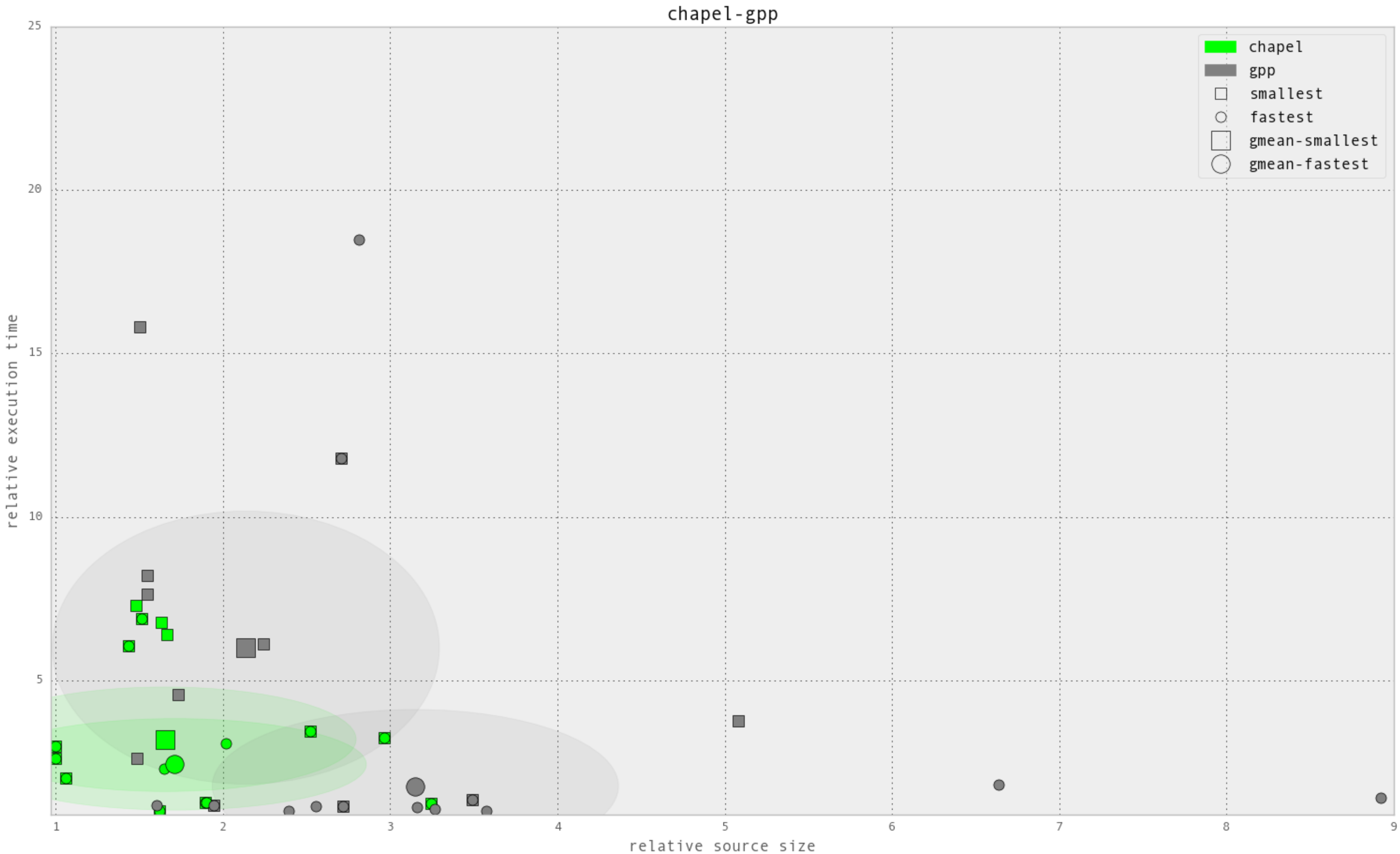


chapel-gcc
chapel
gcc
smallest
fastest
gmean-smallest
gmean-fastest
relative execution time
relative source size
25
20
15
10
5
1
2
3
4
5
6
7
8
9

Chapel vs. C (zoomed out)

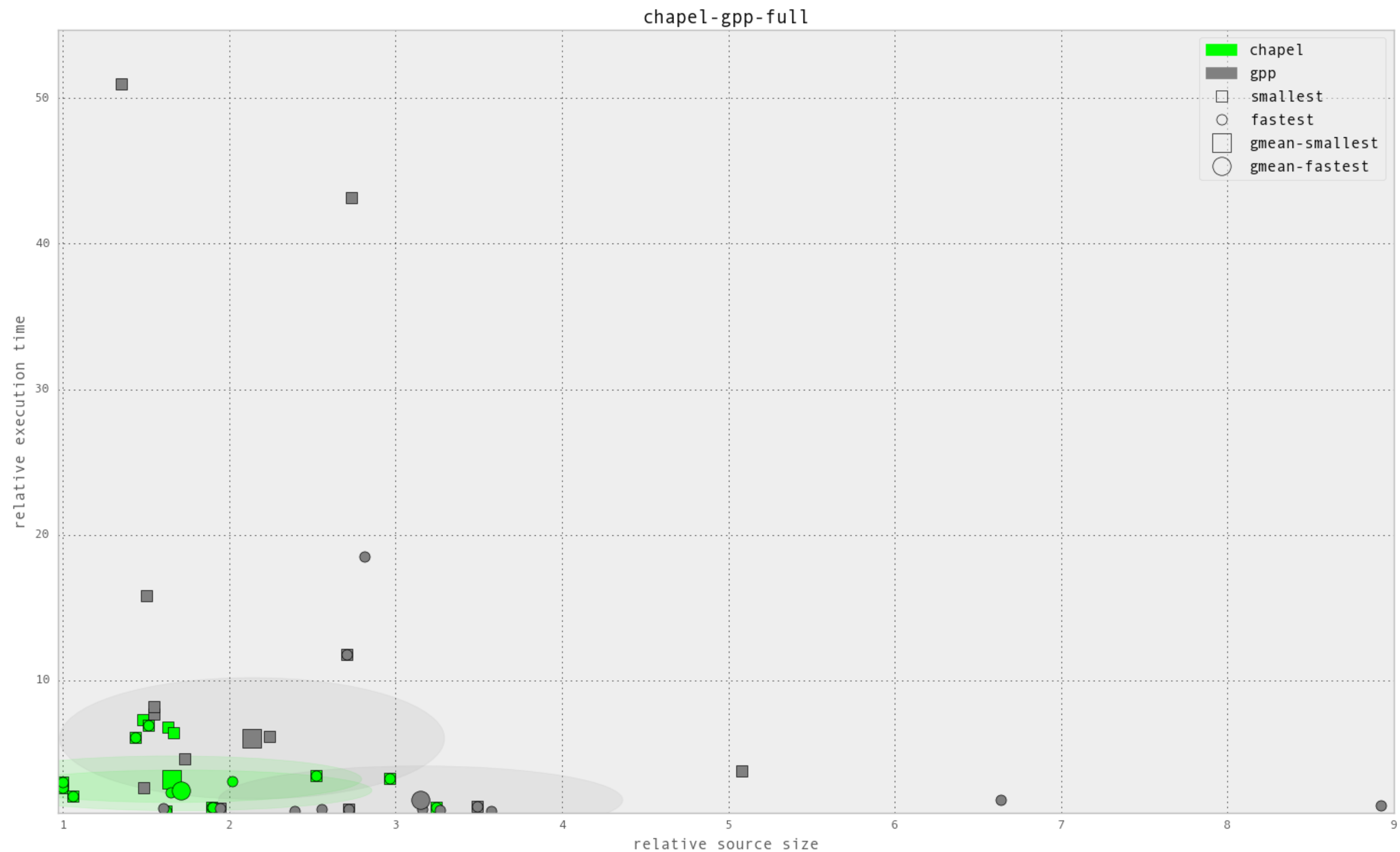


Chapel vs. C++

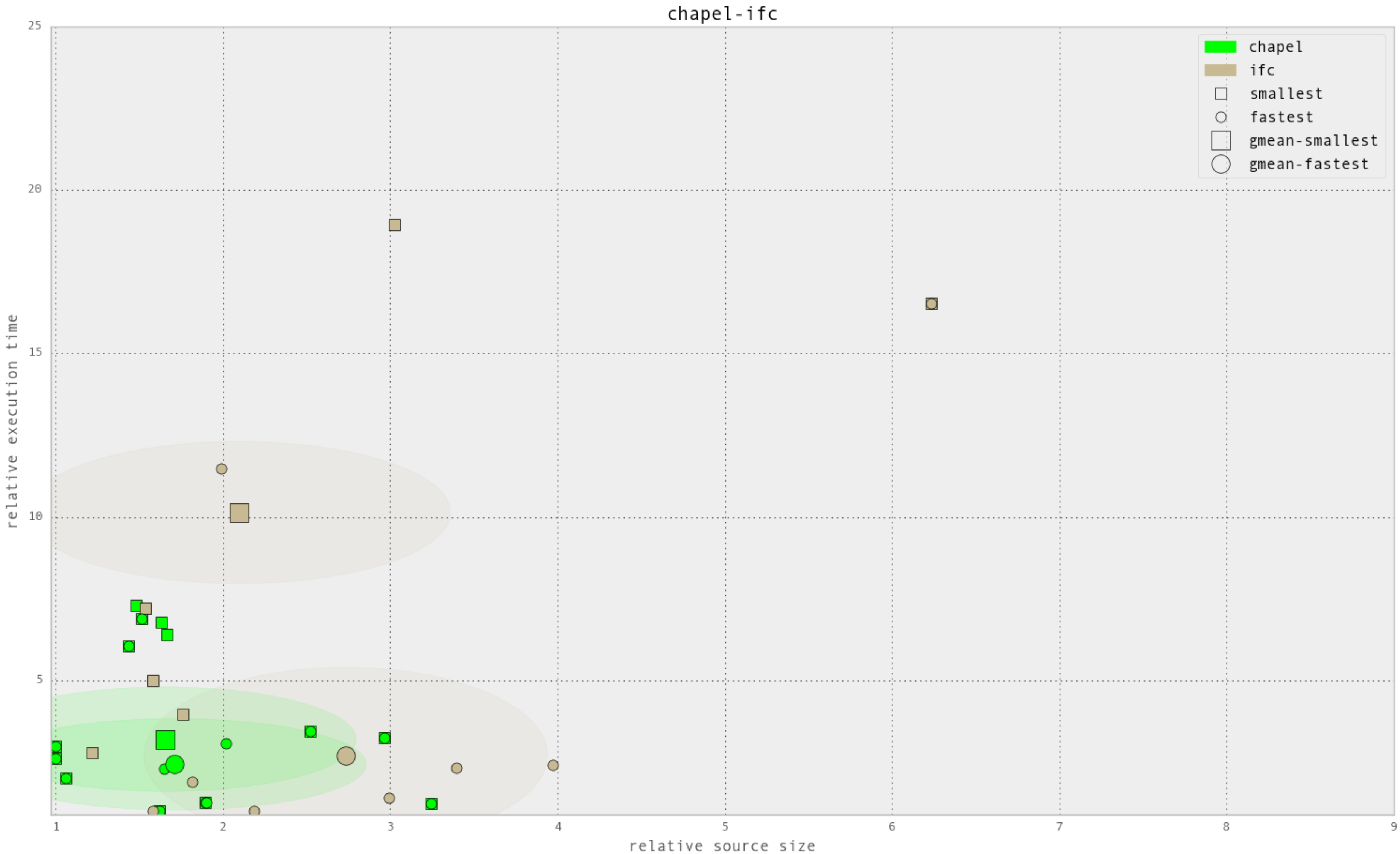


chapel-gpp
chapel
gpp
smallest
fastest
gmean-smallest
gmean-fastest
relative execution time
relative source size
25
20
15
10
5
1
2
3
4
5
6
7
8
9

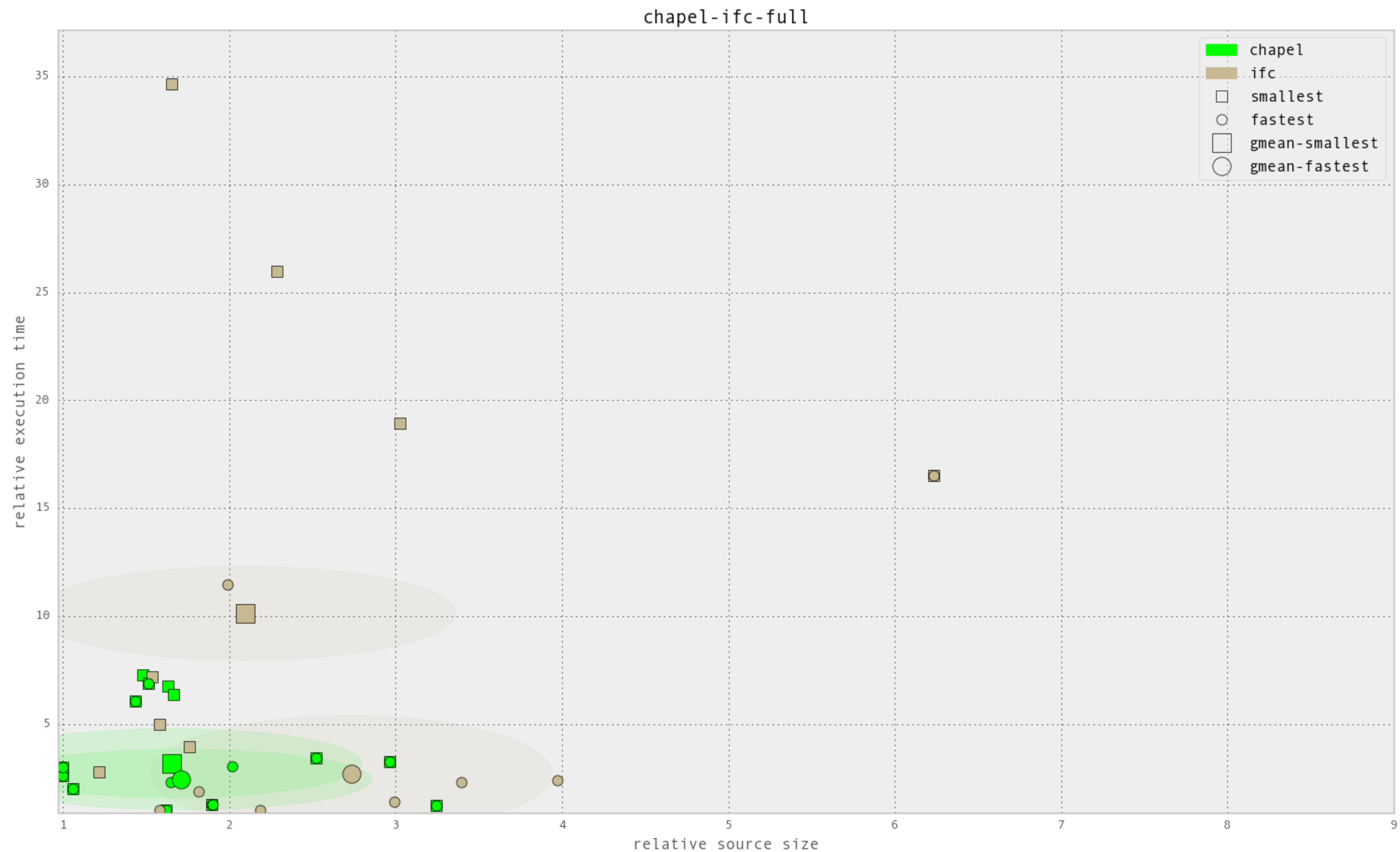
Chapel vs. C++ (zoomed out)



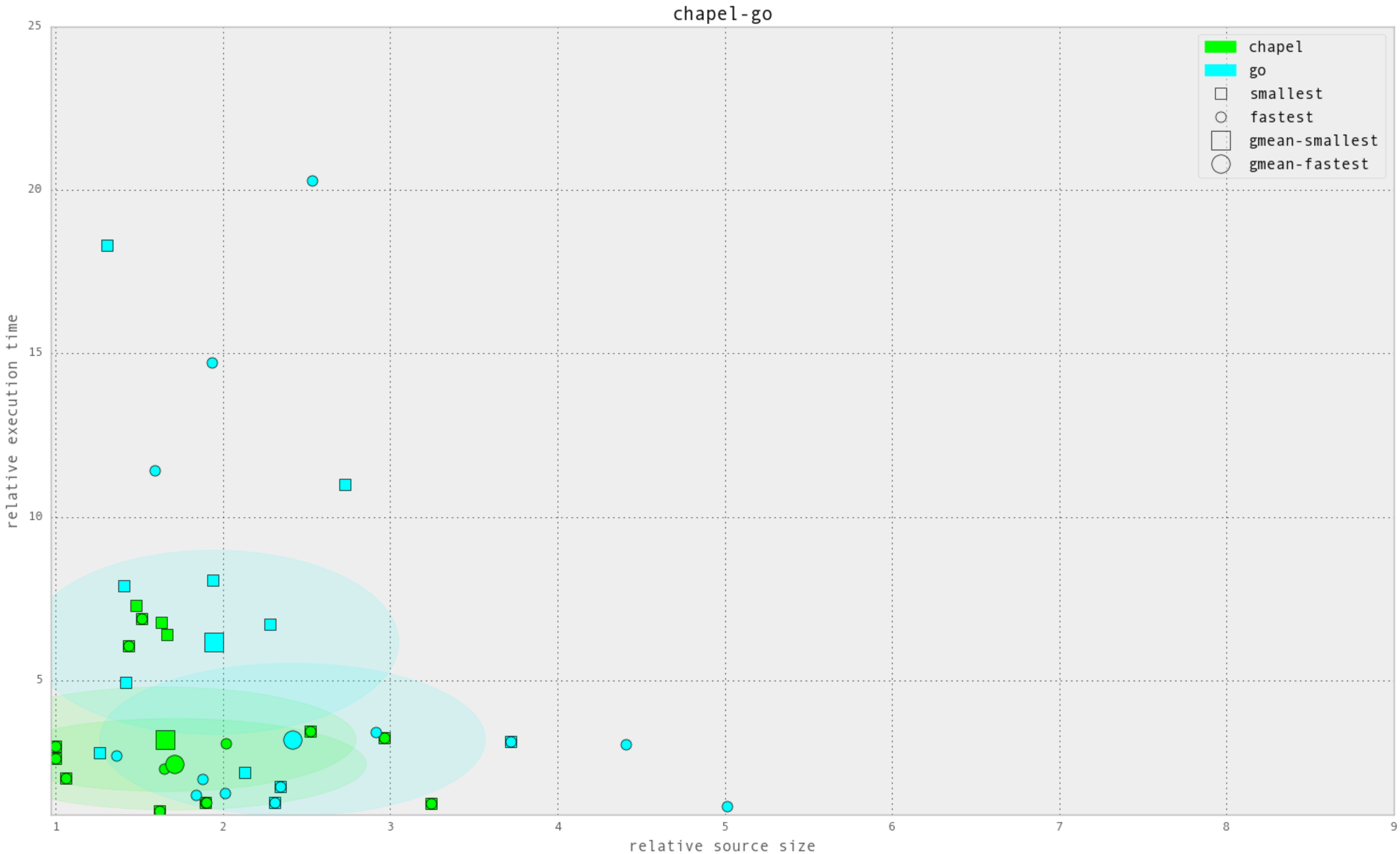
Chapel vs. Fortran



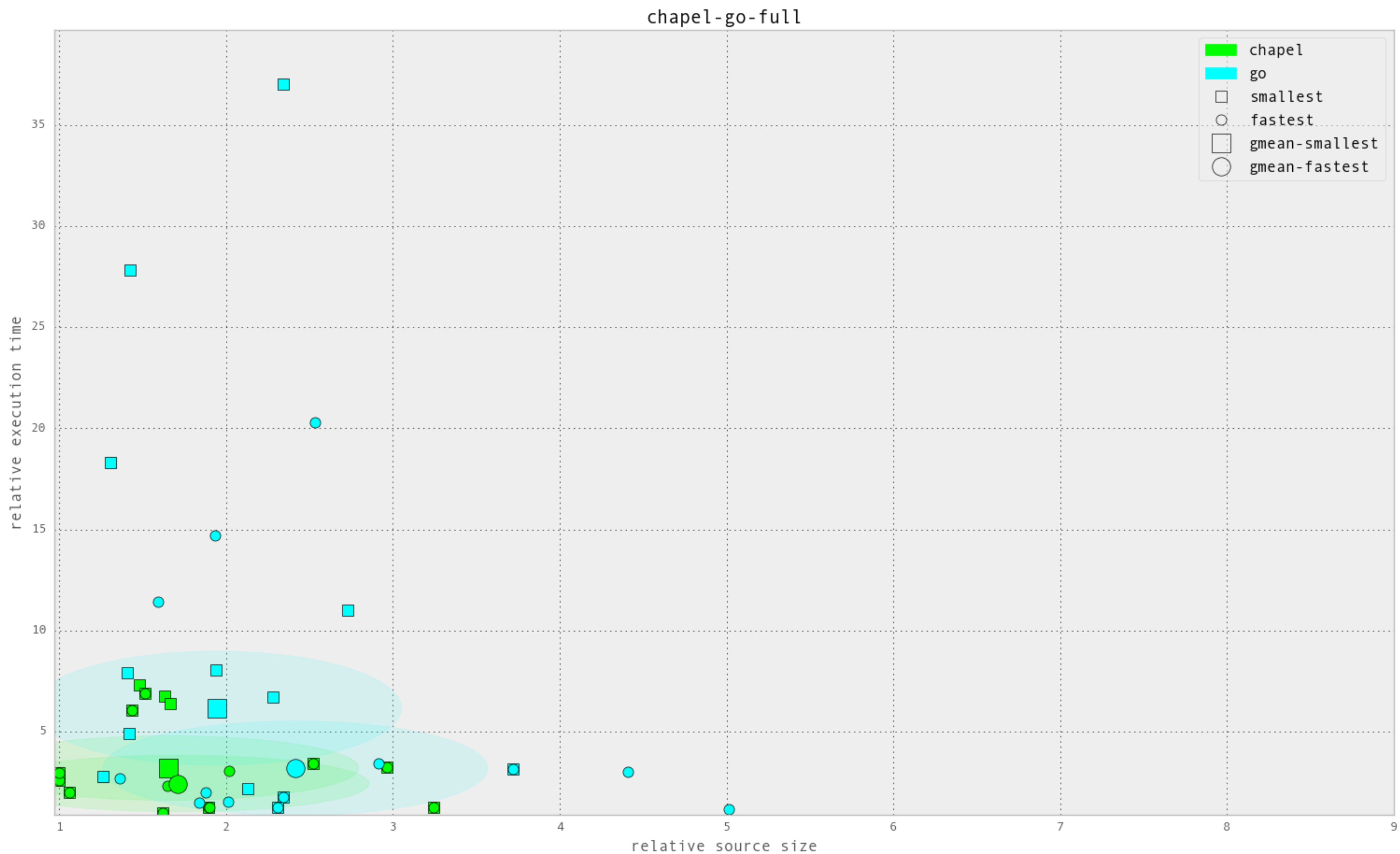
Chapel vs. Fortran (zoomed out)



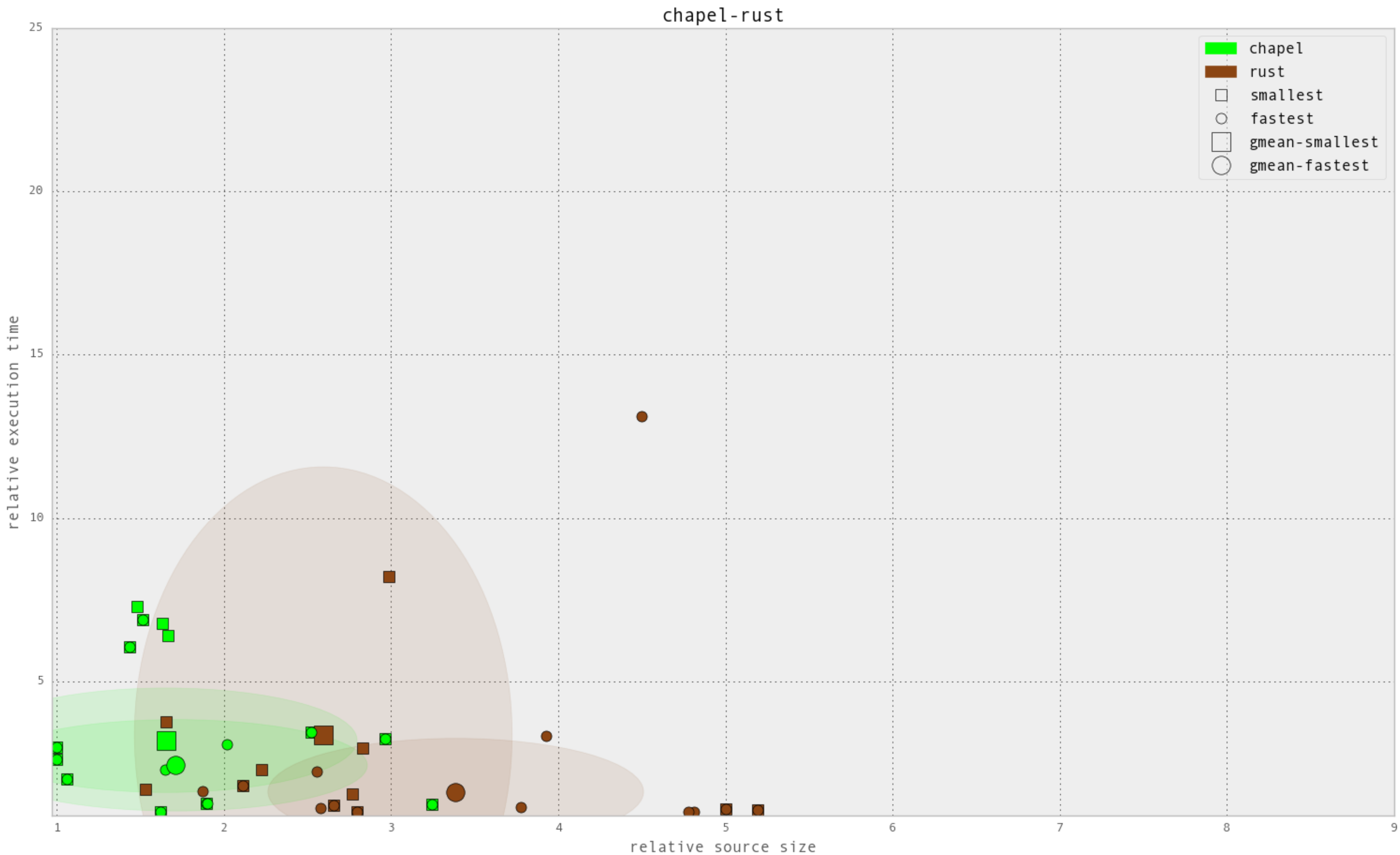
Chapel vs. Go


chapel-go
chapel
go
smallest
fastest
gmean-smallest
gmean-fastest
relative execution time
relative source size
25
20
15
10
5
1
2
3
4
5
6
7
8
9

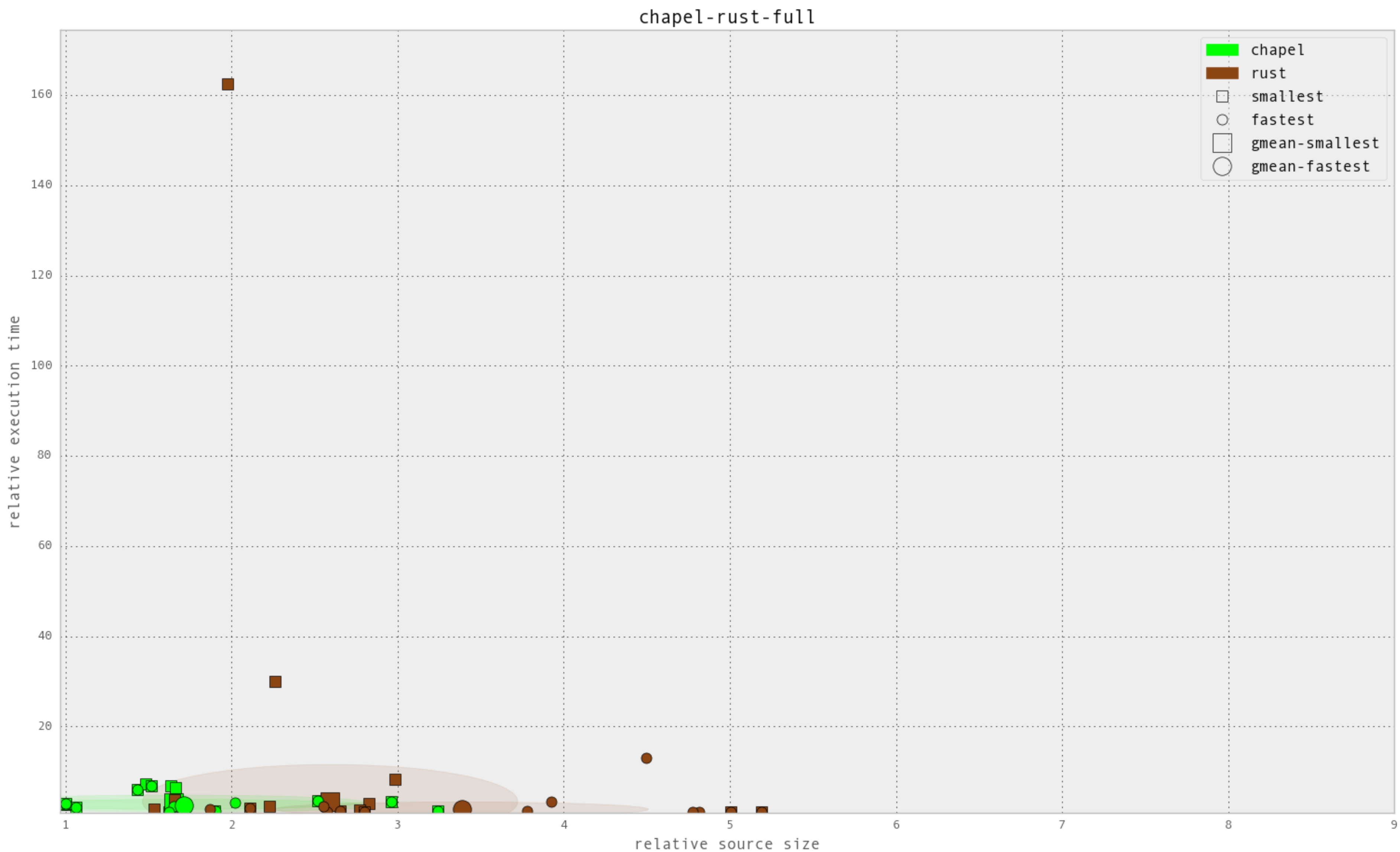
Chapel vs. Go (zoomed out)



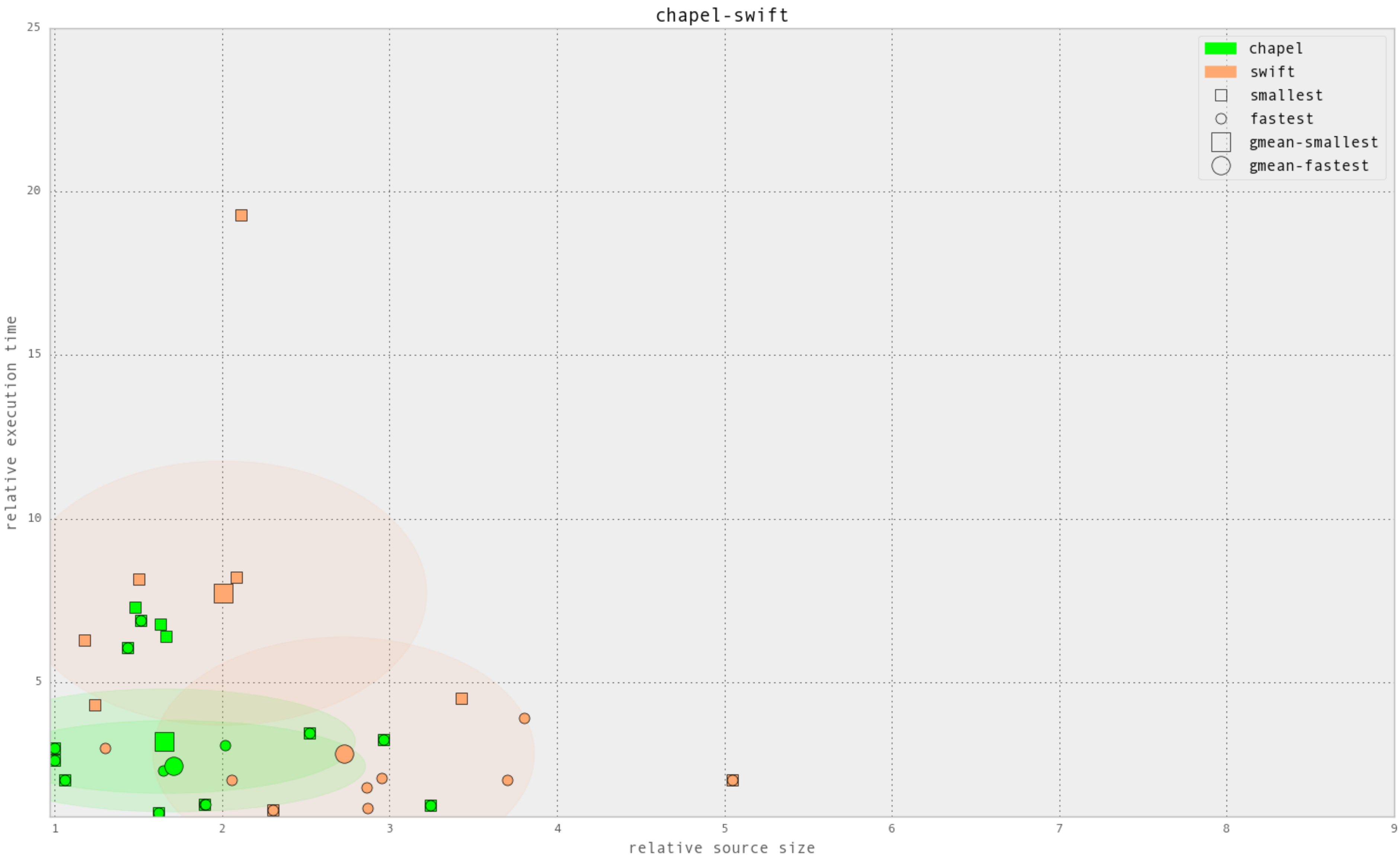
Chapel vs. Rust



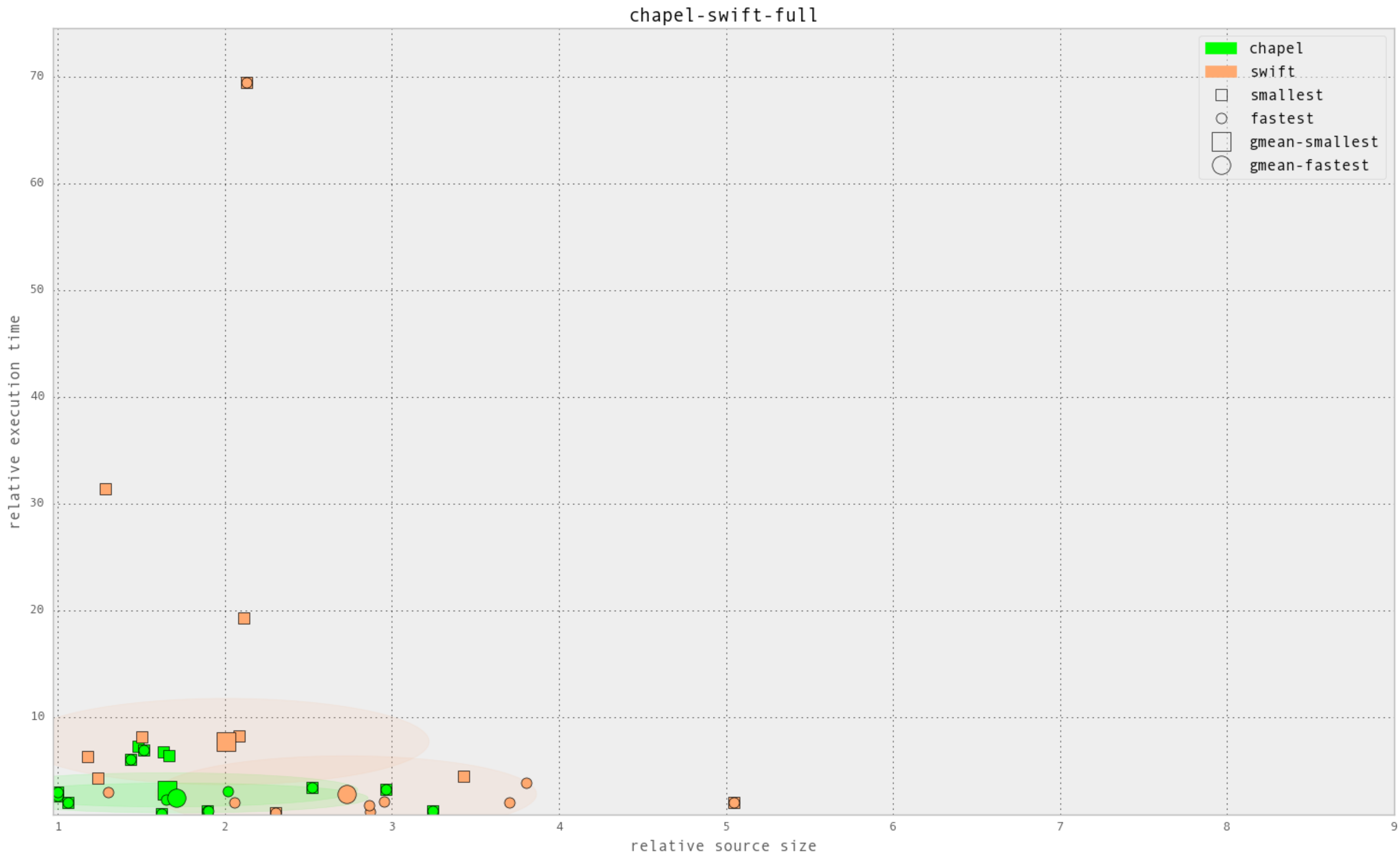
Chapel vs. Rust (zoomed out)



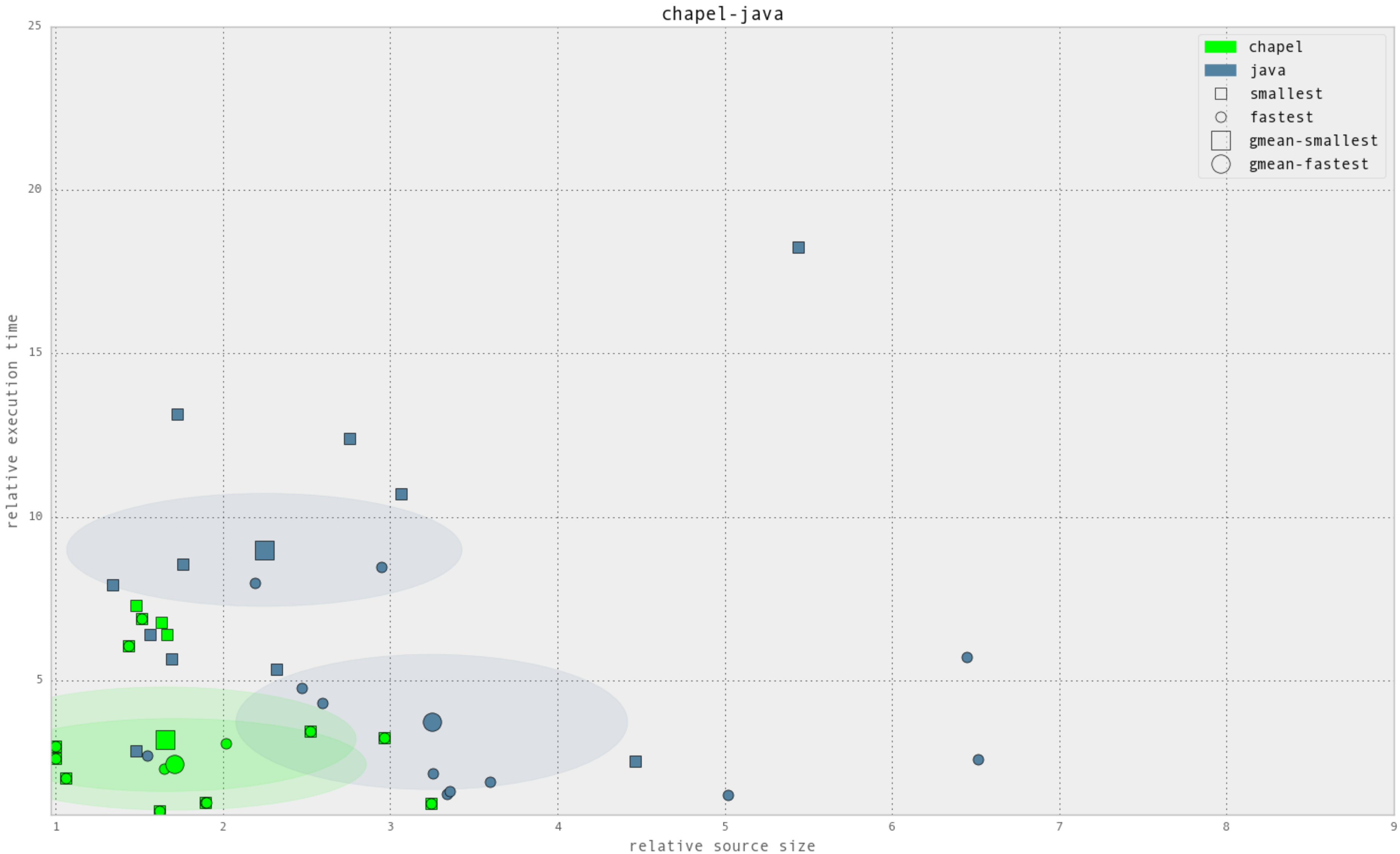
Chapel vs. Swift



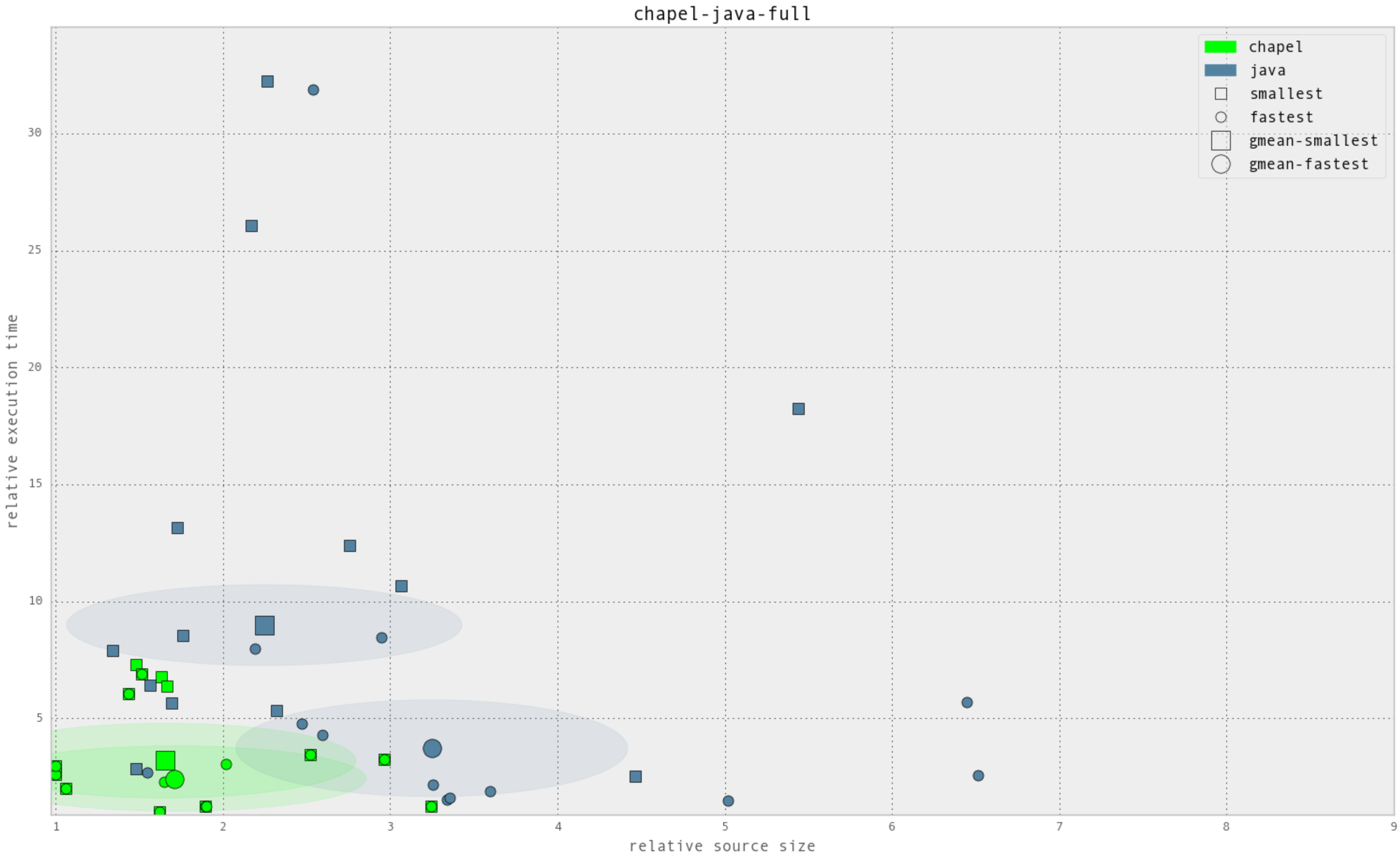
Chapel vs. Swift (zoomed out)



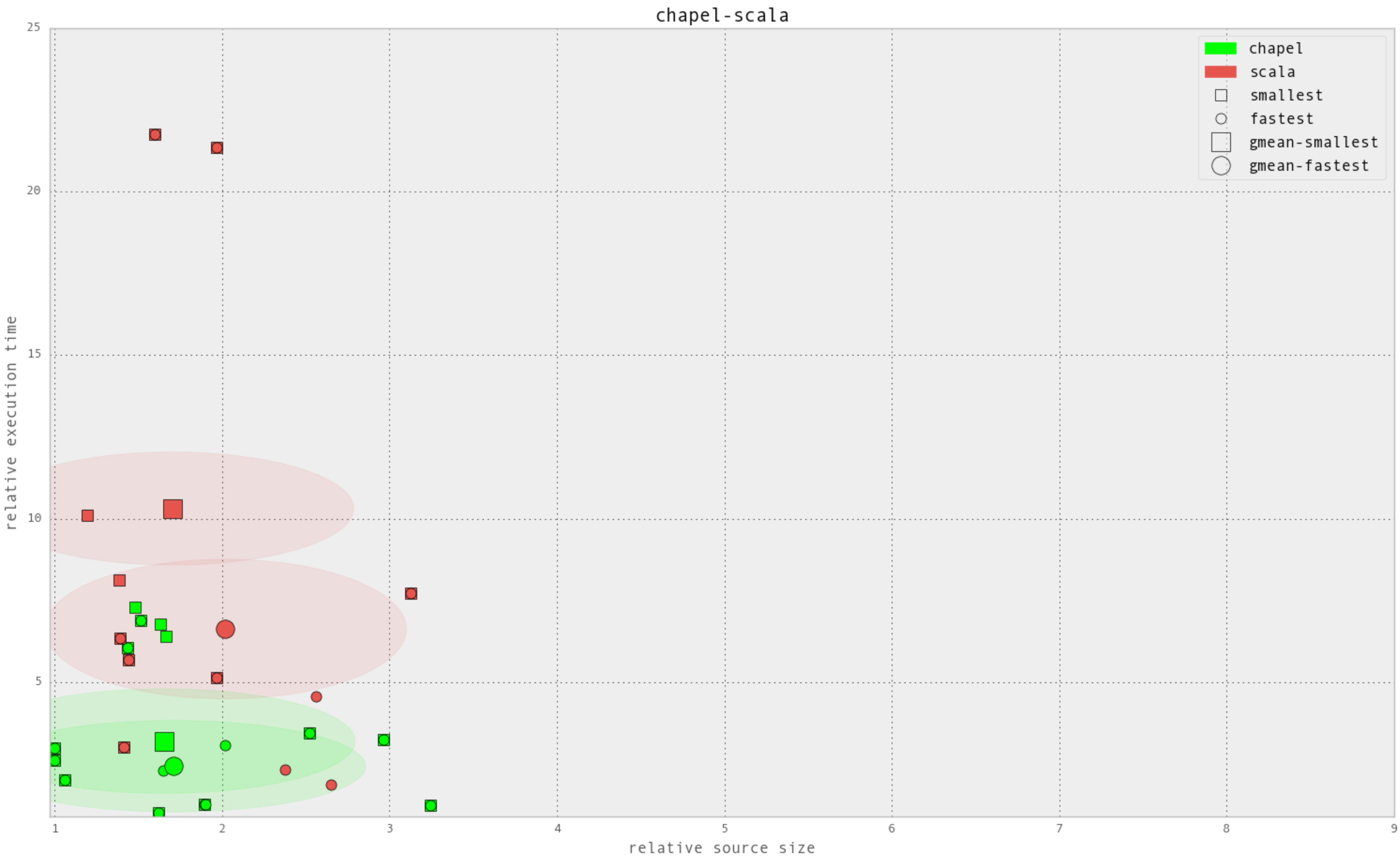
Chapel vs. Java



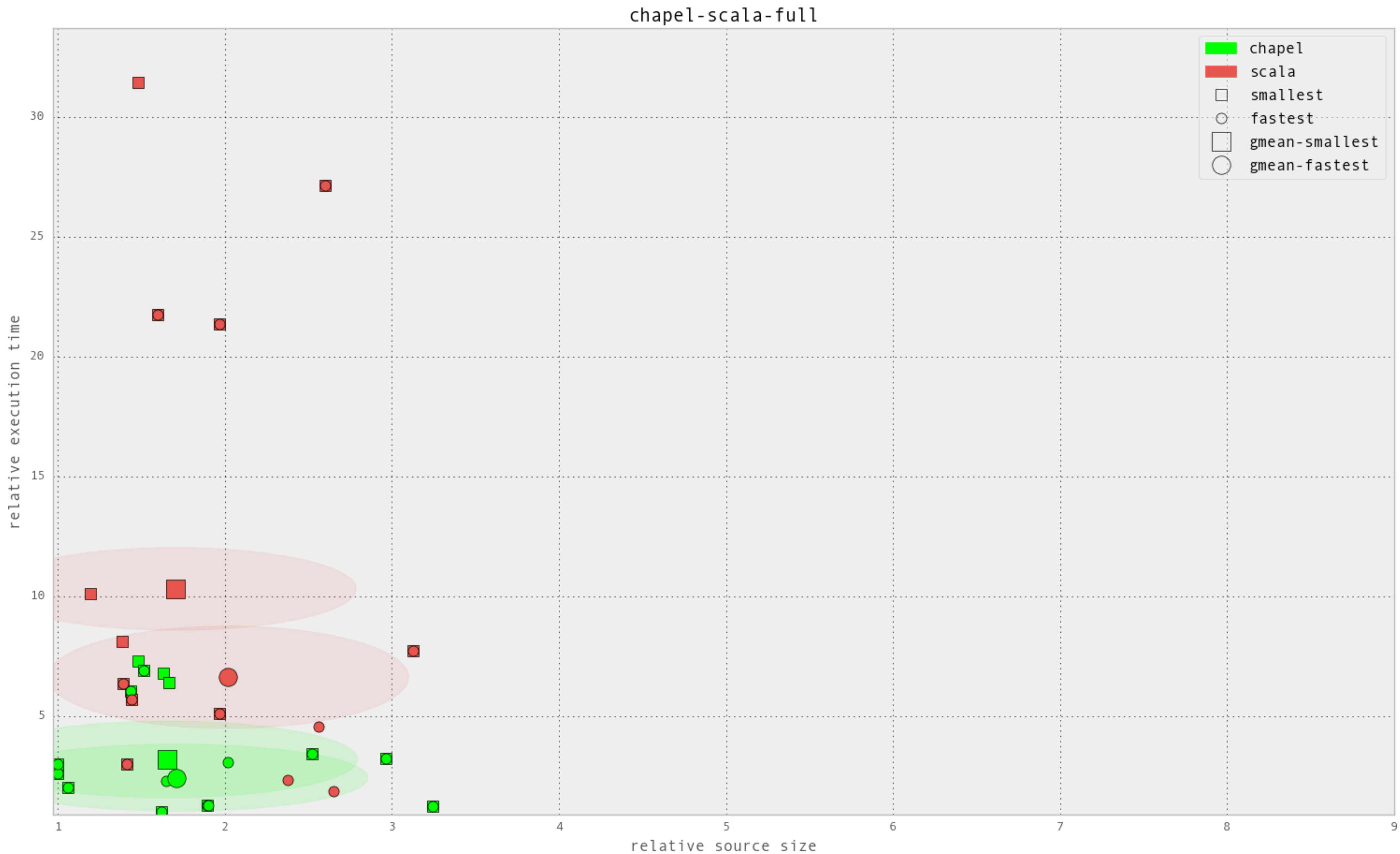
Chapel vs. Java (zoomed out)



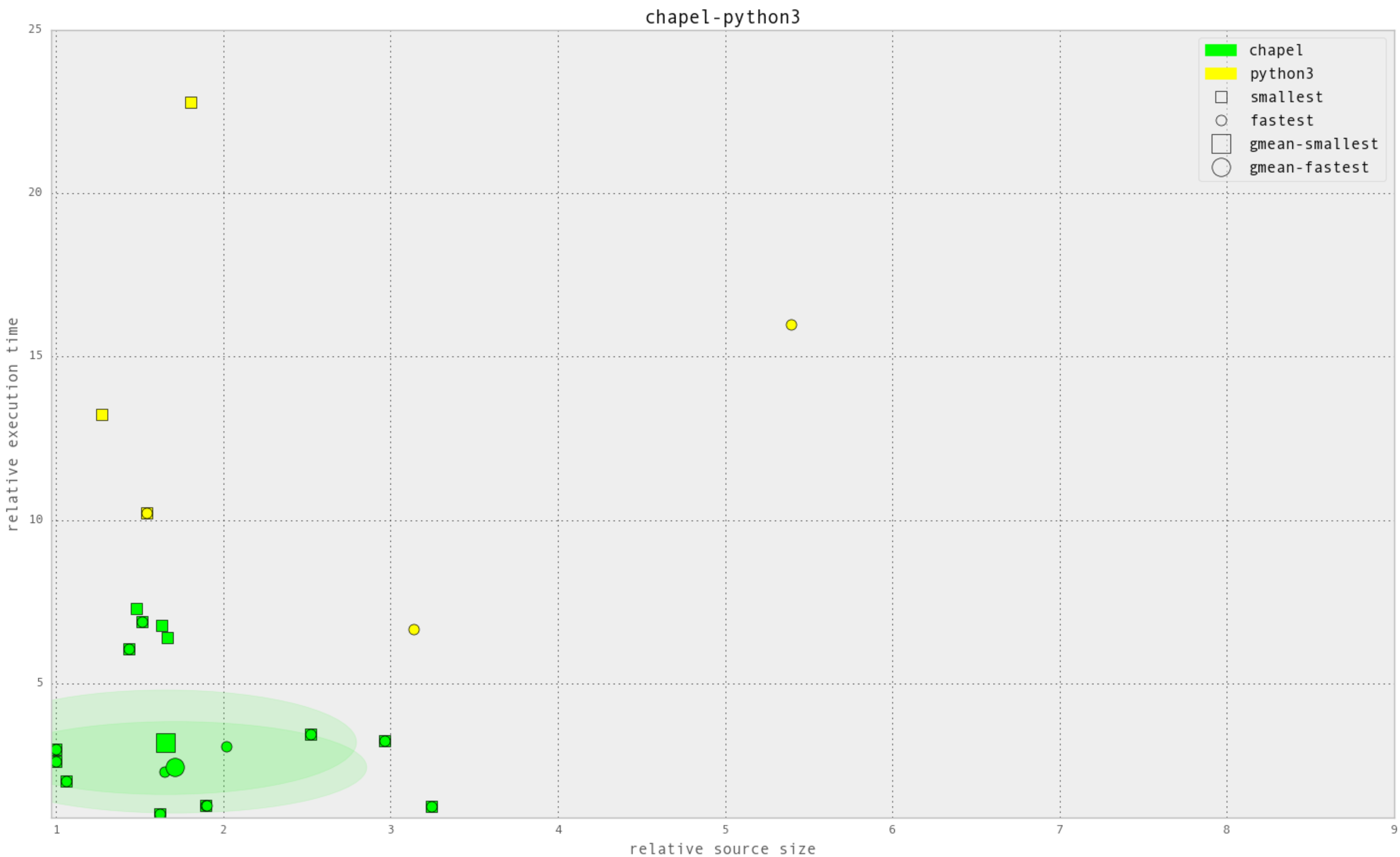
Chapel vs. Scala



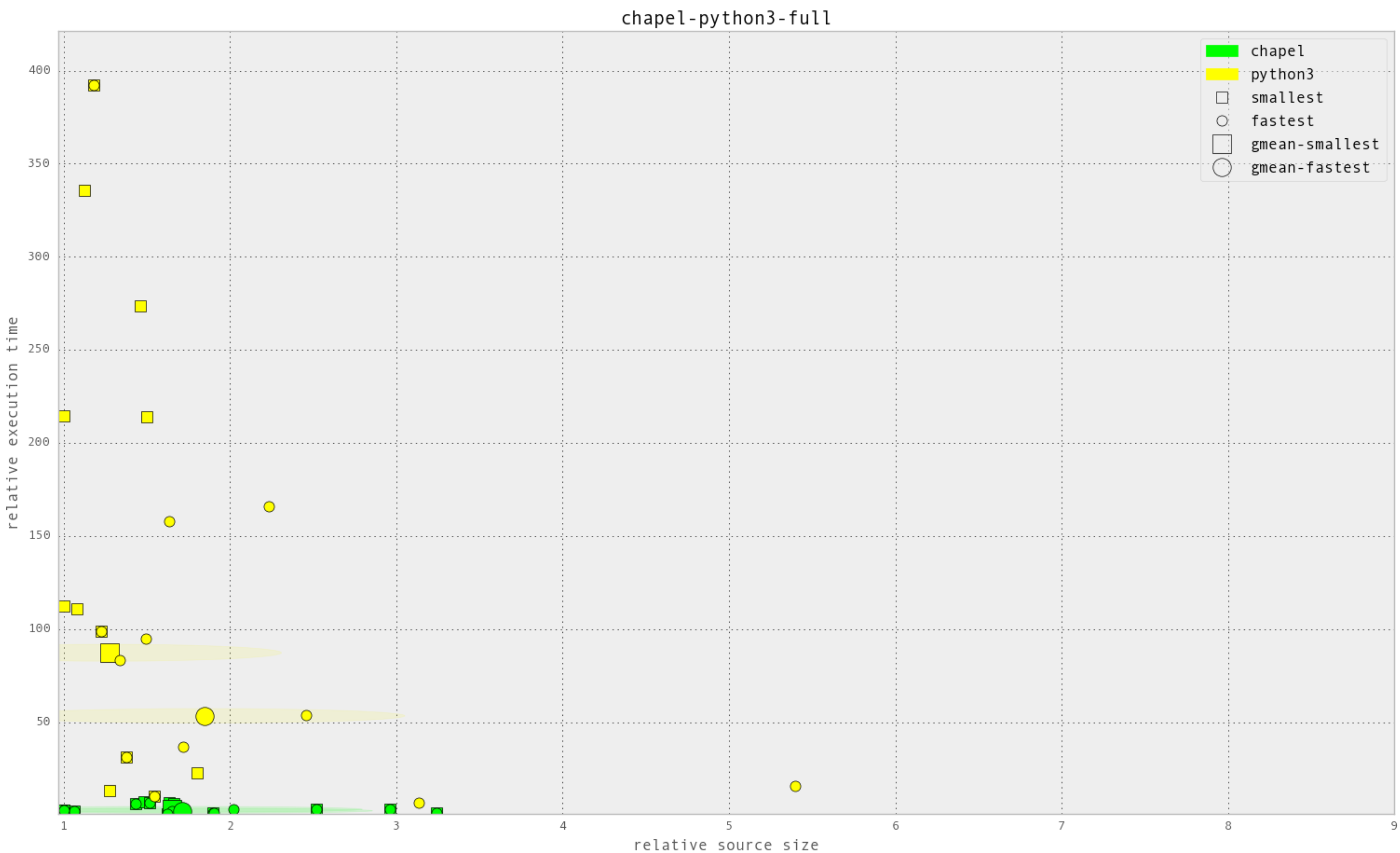
Chapel vs. Scala (zoomed out)



Chapel vs. Python



Chapel vs. Python (zoomed out)



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 CHAPEL

CRAY

THE SUPERCOMPUTER COMPANY