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
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”
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 ~28 languages
    - x as many impls as are interesting
  - each entry contains:
    - source code
    - performance information
    - “code size”

Chapel added to site in September 2016
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
    - ~28 languages
    - as many impls as are interesting
  - each entry contains:
    - source code
    - performance information
    - “code size”
Site summary: relative performance (sorted by geometric mean)

How many times slower?

benchmarks game

09 Sep 2016 u64q

better
Site summary: relative performance (sorted by geometric mean)
Can sort results by execution time, code size, memory or CPU use:

<table>
<thead>
<tr>
<th></th>
<th>source</th>
<th>secs</th>
<th>mem</th>
<th>gz</th>
<th>cpu</th>
<th>cpu load</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>C gcc #5</td>
<td>0.60</td>
<td>820</td>
<td>2863</td>
<td>2.37</td>
<td>100% 100% 98% 100%</td>
</tr>
<tr>
<td>1.2</td>
<td>C++ g++ #5</td>
<td>0.70</td>
<td>3,356</td>
<td>1994</td>
<td>2.65</td>
<td>100% 100% 91% 92%</td>
</tr>
<tr>
<td>1.7</td>
<td>Lisp SBCL #3</td>
<td>1.01</td>
<td>55,604</td>
<td>2,907</td>
<td>3.93</td>
<td>97% 96% 99% 99%</td>
</tr>
<tr>
<td>2.3</td>
<td>Chapel #2</td>
<td>1.39</td>
<td>76,564</td>
<td>1,210</td>
<td>5.43</td>
<td>99% 99% 98% 99%</td>
</tr>
<tr>
<td>3.3</td>
<td>Rust #2</td>
<td>2.01</td>
<td>56,936</td>
<td>2,882</td>
<td>7.81</td>
<td>97% 98% 98% 98%</td>
</tr>
<tr>
<td>5.6</td>
<td>C++ g++ #2</td>
<td>3.40</td>
<td>1,880</td>
<td>2016</td>
<td>11.88</td>
<td>100% 51% 100% 100%</td>
</tr>
<tr>
<td>6.8</td>
<td>Chapel</td>
<td>4.09</td>
<td>66,584</td>
<td>1,199</td>
<td>16.25</td>
<td>100% 100% 100% 100%</td>
</tr>
<tr>
<td>8.0</td>
<td>Java #4</td>
<td>4.82</td>
<td>37,132</td>
<td>1,607</td>
<td>16.73</td>
<td>98% 98% 54% 99%</td>
</tr>
<tr>
<td>8.5</td>
<td>Haskell GHC</td>
<td>5.15</td>
<td>8,596</td>
<td>989</td>
<td>9.26</td>
<td>79% 100% 2% 2%</td>
</tr>
<tr>
<td>10</td>
<td>Java</td>
<td>6.13</td>
<td>53,760</td>
<td>1,770</td>
<td>8.78</td>
<td>42% 45% 41% 16%</td>
</tr>
<tr>
<td>10</td>
<td>Haskell GHC #4</td>
<td>6.34</td>
<td>6,908</td>
<td>989</td>
<td>12.67</td>
<td>99% 100% 2% 1%</td>
</tr>
<tr>
<td>11</td>
<td>C# .NET Core</td>
<td>6.59</td>
<td>86,076</td>
<td>1,400</td>
<td>22.96</td>
<td>99% 82% 78% 91%</td>
</tr>
<tr>
<td>11</td>
<td>Go</td>
<td>6.90</td>
<td>832</td>
<td>1,167</td>
<td>24.19</td>
<td>100% 96% 56% 100%</td>
</tr>
<tr>
<td>13</td>
<td>Go #2</td>
<td>7.59</td>
<td>1,384</td>
<td>1,408</td>
<td>27.65</td>
<td>91% 99% 99% 78%</td>
</tr>
<tr>
<td>13</td>
<td>Java #3</td>
<td>7.94</td>
<td>53,232</td>
<td>1,267</td>
<td>26.86</td>
<td>54% 96% 98% 94%</td>
</tr>
</tbody>
</table>

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

![Graph showing Binary Trees Shootout Benchmark (n=20)](image)

- **Time (seconds)**
- **factory type method**
- **initializer**
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

![Submitted Chameneos Redux Shootout Benchmark (n=6,000,000)](image)
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

![Submitted Fasta Shootout Benchmark](chart.png)
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

---

**Submitted Mandelbrot Shootout Benchmark**

- **Time (seconds)**
  - 12
  - 10
  - 8
  - 6
  - 4
  - 2
  - 0

- **02 Apr**
- **09 Apr**
- **16 Apr**

- original version
- new version

---

*Copyright 2017 Cray Inc.*
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
Can also compare languages pair-wise (for performance only):

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

- C
- C++
- Fortran
- Go
- Rust
- Swift
- Java
- Scala
- Python
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):

```chapel
proc main() {
    printColorEquations();
    const group1 = [i | 1..popSize1] new Chameneos(i, ((i-1)%3)Color);
    const group2 = [i | 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);

doforall i in population do // create a task per chameneos
    c.haveMeetings(place, population);
    delete place;
}
```

```c
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* cpu_idx;
    int physical_id;
    int core_id;
    int cpu_id;
    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 = "physical id";
    size_t physical_id_str_len = strlen(physical_id);
    char const* core_id = "core id";
    size_t core_id_str_len = strlen(core_id);
    char const* cpu_id = "cpu cores";
    size_t cpu_id_str_len = strlen(cpu_id);
    char const* apic_id = "apic id";
    size_t apic_id_str_len = strlen(apic_id);

    CPU_ZERO(&active_cpus);
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
    cpu_count = 0;
    for (i = 0; i < CPU_SETSIZE; i += 1)
    {
        if (CPU_ISSET(i, &active_cpus))
        {
            cpu_count += 1;
        }
    }
    if (cpu_count == 1)
    {
        is_smp[0] = 1;
        return;
    }
    is_smp[0] = 0;
    CPU_ZERO(affinity1);
}
```

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):

```chapel
proc main() {
    printColorEquations();
    const group1 = [i in 1..popSize1] new CHпечатам
    const group2 = [i in 1..popSize2] new CHпечатам

    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;
}
```

```c
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2) {
    live_cpus;
    ...
    idx;
    vma_id;
    re_id;
    t_cores;
    c_id;
    ...
}
```

```chapel
proc holdMeetings(population, numMeetings) {
    const place = new MeetingPlace(numMeetings);
    coforall c in population do
        c.haveMeetings(place, population);
    delete place;
}
```

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

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..popSize} new Chameneos(i, ((i-1)%3);Color);
  const char const* core_id_str = core_id_str_len = size_t cpu_cores_str = 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;
}
```

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2) {
  cpu_set_t active_cpus;
  FILE* f;
  char char const* buf [2048];
  char const* cpu_idx;
  int int cpu_id;
  int core_id;
  int cpu_cores;
  int apic_id;
  size_t cpu_count;
  size_t
  char const* processor_str = "processor";
  size_t processor_str_len = strlen(processor_str);
  char const* physical_id;
  size_t physical_id_str = "physical id";
  size_t physical_id_str_len = strlen(physical_id_str);
  char const* core_id;
  size_t core_id_str_len = strlen(core_id_str);
  char const* cpu_cores;
  size_t cpu_cores_str = "cpu cores";
  size_t cpu_cores_str_len = strlen(cpu_cores_str);`

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

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

typedef unsigned int ui;

ui extract_digit(ui nth) {
    // juggling 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, acc, d);
    mpz_mul_ui(acc, acc, 10);
    mpz_mul_ui(num, num, 10);
}

void next_term(ui k) {
    ui k2 = k * 20 + 10;
    mpz_addmul_ui(acc, acc, k2);
    mpz_mul_ui(den, den, k);
    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; i++) {
        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("\n", 1);
        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

excerpt from 423 gz 1st-place Chapel entry

```
use BigInteger;
config const n = 50; // Compute n digits of pi, 50 by default
proc main() {
    param digitsPerLine = 10;
    // Compute n digits, printing them in groups of digitsPerLine
    for (d, i) in genDigits(n) {
        writeln(d);
        if i % digitsPerLine == 0 then
            writeln\\n; i;
    }
    // Pad out any trailing digits for the final line
    if n % digitsPerLine then
        writeln(\\n * (digitsPerLine - n % digitsPerLine), \"\\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 448 gz 4th-place C gcc entry

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

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

const int k2 = 2 * k + 1;
accum.addmul(numer, 2);
accum *= k2;
denom *= k2;
numerator *= k;

int main(int argc, char **argv) {
    int d, k, i;
    int n = atoi(argv[1]);
    mpf_initj(jmp1);
    mpf_initf(tmp1);
    mpf_init_set_ui(acc, 0);
    mpf_init_set_ui(den, 1);
    mpf_init_set_ui(num, 1);

    for (i = k = 0; i < n; i++) {
        nextTerm(k);
        if (mpf_cmp(numer, acc) > 0)
            continue;
        d = extract_digit(3);
        if (d == extract_digit(4))
            continue;
        putchar('0' + d);
        if (++i % 10 == 0)
            printf(\\n); i);
        eliminate_digit(d);
    }
    return 0;
}
```
Comparing Chapel vs. C pidigits

```chapel
use BigInteger;
config const n = 50;  // Compute n digits of pi, 50 by default
proc main() {
    var digitsPerLine = 10;
    // Generate n digits, printing them in groups of digitsPerLine
    for (d, i) in genDigits() {
        write(d);
        if (i % digitsPerLine == 0 then
            writeln("\n", i);
    }
    // Pad out any trailing digits for the final line
    writeln("\n", (digitsPerLine - n % digitsPerLine), \n", n");
}
iter genDigits(numDigits) {
    var numer, denom, bigint = 1, accum, tmpl, tmp2, bigist;
    if (i = 0.0 && i = 0.0) {
        numer *= 10;
    }
    mpz_addmul_ui(acc, num, 2U);
    mpz_mul_ui(acc, acc, k2);
    mpz_mul_ui(den, den, k2);
    mpz_mul_ui(num, num, k);
    return tmpl: int;
}
}```

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

typedef unsigned int ui;

ui extract_digit(ui nth) {
    // juggling 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_div_q(tmp1, tmp2, den);
    return mpz_get_ui(tmp1);
}

void eliminate_digit(ui d) {
    mpz_submul_ui(acc, acc, 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) {
    if (argc != 3) {
        return 0;
    }
    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 = 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("\n", i);
        eliminate_digit(d);
    }
    return 0;
}
```
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?
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 vs. C (zoomed out)
Chapel vs. C++
Chapel vs. C++ (zoomed out)
Chapel vs. Fortran (zoomed out)
Chapel vs. Go
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.