What is Chapel?

**Chapel**: A productive parallel programming language

- portable & scalable
- open-source & collaborative

**Goals:**

- Support general parallel programming
  - “any parallel algorithm on any parallel hardware”
- Make parallel programming at scale far more productive
What does “Productivity” mean to you?

**Recent Graduates:**
“something similar to what I used in school: Python, Matlab, Java, …”

**Seasoned HPC Programmers:**
“that sugary stuff that I don’t need because I was born to suffer”
want full control to ensure performance

**Computational Scientists:**
“something that lets me express my parallel computations without having to wrestle with architecture-specific details”

**Chapel Team:**
“something that lets computational scientists express what they want, without taking away the control that HPC programmers want, implemented in a language as attractive as recent graduates want.”
Chapel and Productivity

Chapel aims to be as...

...programmable as Python
...fast as Fortran
...scalable as MPI, SHMEM, or UPC
...portable as C
...flexible as C++
...fun as [your favorite programming language]
The Challenge

Q: So why don’t we already have such a language?
A: Technical challenges?
  ● while they exist, we don’t think this is the main issue...
A: Due to a lack of...
  ...long-term efforts
  ...resources
  ...co-design between developers and users
  ...community will
  ...patience

Chapel is our attempt to reverse this trend
Website supporting cross-language comparisons

- 10 toy benchmark programs x ~27 languages x many implementations
- exercise key computational idioms
- specific approach prescribed
Chapel’s approach to the CLBG:
- striving for elegance over heroism
- ideally: “Want to learn how program xyz works? Read the Chapel version.”
CLBG: Fast-faster-fastest graph (Sep 2016)

Relative performance, sorted by geometric mean
CLBG: Fast-faster-fastest graph (June 2018)

Relative performance, sorted by geometric mean

How many times slower?

program time / fastest program time

benchmarks game

01 Jun 2018 u64q
Can sort results by various metrics: execution time, code size, memory use, CPU use:

<table>
<thead>
<tr>
<th>pidigits description</th>
<th>program source code, command-line and measurements</th>
<th>x</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></td>
<td></td>
<td>1.0</td>
<td>Chapel #2</td>
<td>1.62</td>
<td>6,484</td>
<td>423</td>
<td>1.63</td>
<td>99% 1% 1% 2%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0</td>
<td>Chapel</td>
<td>1.62</td>
<td>6,488</td>
<td>501</td>
<td>1.63</td>
<td>99% 1% 1% 1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.1</td>
<td>Free Pascal #3</td>
<td>1.73</td>
<td>2,428</td>
<td>530</td>
<td>1.72</td>
<td>0% 2% 100% 1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.1</td>
<td>Rust #3</td>
<td>1.74</td>
<td>4,488</td>
<td>1366</td>
<td>1.74</td>
<td>1% 100% 1% 0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.1</td>
<td>Rust</td>
<td>1.74</td>
<td>4,616</td>
<td>1420</td>
<td>1.74</td>
<td>1% 100% 1% 0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.1</td>
<td>Rust #2</td>
<td>1.74</td>
<td>4,636</td>
<td>1306</td>
<td>1.74</td>
<td>1% 100% 0% 0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.1</td>
<td>C gcc</td>
<td>1.75</td>
<td>2,728</td>
<td>452</td>
<td>1.74</td>
<td>1% 2% 0% 100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.1</td>
<td>Ada 2012 GNAT #2</td>
<td>1.75</td>
<td>4,312</td>
<td>1068</td>
<td>1.75</td>
<td>1% 100% 0% 0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.1</td>
<td>Swift #2</td>
<td>1.76</td>
<td>8,492</td>
<td>601</td>
<td>1.76</td>
<td>1% 100% 1% 0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.1</td>
<td>Lisp SBCL #4</td>
<td>1.79</td>
<td>20,196</td>
<td>940</td>
<td>1.79</td>
<td>1% 2% 1% 100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.2</td>
<td>C++ g++ #4</td>
<td>1.89</td>
<td>4,284</td>
<td>513</td>
<td>1.88</td>
<td>5% 0% 1% 100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.3</td>
<td>Go #3</td>
<td>2.04</td>
<td>8,976</td>
<td>603</td>
<td>2.04</td>
<td>1% 0% 0% 100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.3</td>
<td>PHP #5</td>
<td>2.12</td>
<td>10,664</td>
<td>399</td>
<td>2.11</td>
<td>100% 0% 1% 1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.3</td>
<td>PHP #4</td>
<td>2.12</td>
<td>10,512</td>
<td>389</td>
<td>2.12</td>
<td>100% 0% 0% 2%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>gz == code size metric strip comments and extra whitespace, then gzip</th>
<th>program source code, command-line and measurements</th>
<th>x</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></td>
<td></td>
<td>1.0</td>
<td>Perl #4</td>
<td>3.50</td>
<td>7,348</td>
<td>261</td>
<td>3.50</td>
<td>100% 1% 1% 1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.5</td>
<td>Python 3 #2</td>
<td>3.51</td>
<td>10,500</td>
<td>386</td>
<td>3.50</td>
<td>1% 1% 0% 100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.5</td>
<td>PHP #4</td>
<td>2.12</td>
<td>10,512</td>
<td>389</td>
<td>2.12</td>
<td>100% 0% 0% 2%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.5</td>
<td>Perl #2</td>
<td>3.83</td>
<td>7,320</td>
<td>389</td>
<td>3.83</td>
<td>2% 1% 100% 1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.5</td>
<td>PHP #5</td>
<td>2.12</td>
<td>10,664</td>
<td>399</td>
<td>2.11</td>
<td>100% 0% 1% 1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.6</td>
<td>Chapel #2</td>
<td>1.62</td>
<td>6,484</td>
<td>423</td>
<td>1.63</td>
<td>100% 1% 1% 1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.7</td>
<td>C gcc</td>
<td>1.75</td>
<td>2,728</td>
<td>452</td>
<td>1.74</td>
<td>1% 2% 0% 100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.7</td>
<td>Racket</td>
<td>27.58</td>
<td>124,156</td>
<td>453</td>
<td>27.56</td>
<td>1% 1% 0% 100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.8</td>
<td>OCaml #5</td>
<td>6.72</td>
<td>19,836</td>
<td>458</td>
<td>6.71</td>
<td>1% 1% 0% 100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.8</td>
<td>Perl</td>
<td>15.45</td>
<td>10,876</td>
<td>463</td>
<td>15.44</td>
<td>100% 0% 1% 1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.9</td>
<td>Ruby #5</td>
<td>3.29</td>
<td>277,496</td>
<td>485</td>
<td>6.58</td>
<td>8% 63% 32% 100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.9</td>
<td>Lisp SBCL #3</td>
<td>11.99</td>
<td>325,776</td>
<td>493</td>
<td>11.96</td>
<td>1% 1% 100% 0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.9</td>
<td>Chapel</td>
<td>1.62</td>
<td>6,488</td>
<td>501</td>
<td>1.63</td>
<td>99% 1% 1% 1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.9</td>
<td>PHP #3</td>
<td>2.14</td>
<td>10,672</td>
<td>504</td>
<td>2.14</td>
<td>1% 0% 0% 100%</td>
</tr>
</tbody>
</table>
Can also compare languages pair-wise:

- but only sorted by execution speed...

### Chapel versus C++ g++ fastest programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Exec Time</th>
<th>Mem</th>
<th>Gz</th>
<th>Cpu</th>
<th>Cpu Load</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chapel</strong></td>
<td>2.20</td>
<td>1,497,876</td>
<td>707</td>
<td>5.10</td>
<td>96% 42% 58% 38%</td>
</tr>
<tr>
<td><strong>C++ g++</strong></td>
<td>2.95</td>
<td>980,472</td>
<td>2280</td>
<td>4.56</td>
<td>50% 41% 16% 50%</td>
</tr>
</tbody>
</table>

### Chapel versus Python 3 fastest programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Exec Time</th>
<th>Mem</th>
<th>Gz</th>
<th>Cpu</th>
<th>Cpu Load</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chapel</strong></td>
<td>5.09</td>
<td>36,328</td>
<td>620</td>
<td>20.09</td>
<td>99% 99% 99% 99%</td>
</tr>
<tr>
<td><strong>Python 3</strong></td>
<td>279.68</td>
<td>49,344</td>
<td>688</td>
<td>1,117.29</td>
<td>100% 100% 100% 100%</td>
</tr>
</tbody>
</table>

### Benchmarks Game

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Exec Time</th>
<th>Mem</th>
<th>Gz</th>
<th>Cpu</th>
<th>Cpu Load</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fannkuch-redux</strong></td>
<td>12.07</td>
<td>4,556</td>
<td>728</td>
<td>48.05</td>
<td>100% 100% 100% 100%</td>
</tr>
<tr>
<td><strong>C++ g++</strong></td>
<td>10.62</td>
<td>2,040</td>
<td>980</td>
<td>41.91</td>
<td>100% 95% 100% 100%</td>
</tr>
</tbody>
</table>
CLBG: Chapel Entries (September 21, 2018)
CLBG: Chapel Entries (September 21, 2018)

Execution Time (normalized to fastest entry)

Compressed Code Size (normalized to smallest entry)

(fastest)

(labels indicate fastest entries)

- chapel
- smallest
- fastest
- gmean-smallest
- gmean-fastest
CLBG Cross-Language Summary
(September 21, 2018 standings)

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)

smaller

faster
CLBG Cross-Language Summary
(September 21, 2018 standings, zoomed in)

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)

smaller

t faster

Copyright 2018 Cray Inc.
CLBG Cross-Language Summary
(September 21, 2018 standings, zoomed in)

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)

smaller

faster

Haskell
OCaml
Pascal
Typescript
JS
Racket
Scala
C#
Swift
Java
Fortran
C++
C
Rust
Racket
Scala
faster
smaller
CLBG Cross-Language Summary
(September 21, 2018 standings)
CLBG: Qualitative Code Comparisons

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

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

    char const* processor_str = "processor";
    char const* physical_id_str = "physical id";
    char const* core_id_str = "core id";
    char const* cpu_cores_str = "cpu cores";
    char const* cpu_affinity_str = "cpu affinity";

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

excerpt from 1210 gz Chapel entry

```
proc main() {
    printColorEquations();
    const group1 = [i in 1..popSize] 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 2863 gz C gcc entry
CLBG: Qualitative Code Comparisons

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

```
proc main()
  printColorEquations();
  const group1 = [i in 1..popSize1] new Chameneos[i, c in 1..numColors1];
  const group2 = [i in 1..popSize2] new Chameneos[i, c in 1..numColors2];
  print(group1);
  print(group2);
end main;

proc holdMeetings(population, numMeetings)
  const place = new MeetingPlace(numMeetings);
  coforall c in population do // create a meeting place
    c.haveMeetings(place, population);
  delete place;
end holdMeetings;
```

excerpt from 1210 gz Chapel entry

```
proc holdMeetings(population, numMeetings)
  const place = new MeetingPlace(numMeetings);
  coforall c in population do // create a meeting place
    c.haveMeetings(place, population);
  delete place;
end holdMeetings;
```

excerpt from 2863 gz C gcc entry
CLBG: Qualitative Code Comparisons

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

```c
char const* core_id_str = "core id";
size_t core_id_str_len = strlen(core_id_str);
char const* cpu_cores_str = "cpu cores";
size_t cpu_cores_str_len = strlen(cpu_cores_str);

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

**excerpt from 1210 gz Chapel entry**

```c
void get_affinity(int* ia_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
    cpu_set_t active_cpus;
    FILE* f;
    char buf [2048];
    char const* pos;
    int cpu_id;
    int physical_id;
    int core_id;
    int cpu_cores;
    int affinity;
    size_t cpu_count;
    size_t i;
    char const* processor_str = "processor";
    char const* physical_id_str = "physical id";
    size_t physical_id_str_len = strlen(physical_id_str);
    size_t processor_str_len = strlen(processor_str);
    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)
    {
        ia_smp[0] = 1;
        CPU_ZERO(affinity1);
    }
```

**excerpt from 2863 gz C gcc entry**
The Chapel Team at Cray (May 2018)

~12 full-time employees + ~2 summer interns
Chapel Community Partners

(and several others…)

https://chapel-lang.org/collaborations.html
Outline

✓ What is Chapel?

➢ Overview of Chapel Features
  ● Chapel Results and Resources
  ● Wrap-up
Chapel language feature areas

Chapel language concepts

- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target Machine
Base Language

Domain Maps
Data Parallelism
Task Parallelism
Base Language
Locality Control
Target Machine

Lower-level Chapel
Base Language Features, by example

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

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

config const n = 10;

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

0
1
1
2
3
5
8
...
Base Language Features, by example

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

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

```
config const n = 10;

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

Configuration declarations (support command-line overrides)

`./fib --n=1000000`

```
0
1
1
2
3
5
8
...
```
Iterators

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

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

```CLU
config const n = 10;

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

0
1
1
2
3
5
8
...
Base Language Features, by example

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

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

config const n = 10;
for f in fib(n) do writeln(f);
```

Static type inference for:
- arguments
- return types
- variables
Base Language Features, by example

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

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

```
config const n: int = 10;

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

Explicit types also supported
Base Language Features, by example

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

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

config const n = 10;

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

0
1
1
2
3
5
8
...
Base Language Features, by example

```javascript
iter fib(n) {
    var current = 0,
        next = 1;
    for i in 1..n {
        yield current;
        current += next;
        current <=> next;
    }
}
```

```javascript
config const n = 10;
for (i, f) in zip(0..#n, fib(n)) do
    writeln("fib #", i, " is ", f);
```

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

Zippered iteration
Base Language Features, by example

**Iter**
```
iter fib(n) {
    var current = 0,
        next = 1;
    for i in 1..n {
        yield current;
        current += next;
        current <=> next;
    }
}
```

**Config**
```
config const n = 10;
```

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

<table>
<thead>
<tr>
<th>fib</th>
<th>#0 is 0</th>
<th>#1 is 1</th>
<th>#2 is 1</th>
<th>#3 is 2</th>
<th>#4 is 3</th>
<th>#5 is 5</th>
<th>#6 is 8</th>
</tr>
</thead>
</table>
...
Base Language Features, by example

```python
iter fib(n) {
    var current = 0,
        next = 1;
    for i in 1..n {
        yield current;
        current += next;
        current <=> next;
    }
}
```

```python
config const n = 10;
for (i, f) in zip(0..#n, fib(n)) do
    writeln("fib ", i, " is " , f);
```

```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
...```
Base Language Features, by example

```javascript
iter fib(n) {
    var current = 0,
        next = 1;
    for i in 1..n {
        yield current;
        current += next;
        current <=> next;
    }
}
```

```javascript
config const n = 10;
for (i, f) in zip(0..#n, fib(n)) do
    writeln("fib #", i, " is ", f);
```

fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
...
Other Key Base Language Features

- Object-oriented features
- Generic programming / polymorphism
- Procedure overloading / filtering
- Default args, arg intents, keyword-based arg passing
- Argument type queries / pattern-matching
- Compile-time meta-programming
- Modules (namespaces)
- Error-handling
- and more…
Task Parallelism and Locality Control

- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target Machine
Locales, briefly

- Locales can run tasks and store variables
  - Think “compute node”

Locales:

- User’s main() executes on locale #0
Task Parallelism and Locality, by example

```chpl
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
    printf("Hello from task %n of %n "+
        "running on %s\n", tid, numTasks, here.name);
```

```shell
prompt> chpl taskParallel.chpl
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```
Task Parallelism and Locality, by example

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

Abstraction of System Resources

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  printf("Hello from task %n of %n " +
         "running on %s\n",
      tid, numTasks, here.name);

prompt> chpl taskParallel.chpl
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
This is a shared memory program
Nothing has referred to remote locales, explicitly or implicitly

```chpl
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  printf("Hello from task %n of %n +
         "running on %s\n",
         tid, numTasks, here.name);
```

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```
Task Parallelism and Locality, by example

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

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```
Task Parallelism and Locality, by example

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

Abstraction of System Resources

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```
Task Parallelism and Locality, by example

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

Control of Locality/Affinity

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```
Task Parallelism and Locality, by example

taskParallel.chpl

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

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```
Other Key Task Parallel Features

- **atomic / sync variables**: for sharing data & coordination
- **begin / cobegin statements**: other ways of creating tasks
Data Parallelism in Chapel

Chapel language concepts

- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target Machine

Higher-level Chapel
Data Parallelism, by example

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

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

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```
Data Parallelism, by example

Domains (Index Sets)

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

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

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```
Arrays

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

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

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```
Data Parallelism, by example

Data-Parallel Forall Loops

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

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

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```
Data Parallelism, by example

This is a shared memory program
Nothing has referred to remote locales, explicitly or implicitly

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

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

prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
Distributed Data Parallelism, by example

Domain Maps
(Map Data Parallelism to the System)

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

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5 --numLocales=4
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```
Distributed Data Parallelism, by example

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

### Example Output

```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5 --numLocales=4
  1.1 1.3 1.5 1.7 1.9
  2.1 2.3 2.5 2.7 2.9
  3.1 3.3 3.5 3.7 3.9
  4.1 4.3 4.5 4.7 4.9
  5.1 5.3 5.5 5.7 5.9
```
Other Key Data Parallel Features

- **Flavors of Domains/Arrays**: sparse, associative, …
- **Slicing**: refer to subarrays using ranges / domains
- **Promotion**: call scalar arrays with array arguments
- **Reductions**: collapse arrays to scalars or subarrays
- **Scans**: compute parallel prefix operations
Chapel Results and Resources
HPC Patterns: Chapel vs. Reference

LCALS Serial Kernels: Chapel vs. Reference

LPCC RA: Chapel vs. Reference

HPCC STREAM Triad: Chapel vs. Reference

ISx: Chapel vs. Reference

PRK Stencil: Chapel vs. Reference

Nightly performance tickers online at: https://chapel-lang.org/perf-nightly.html
HPC Patterns: Chapel vs. Reference

**LCALS Serial Kernels: Chapel vs. Reference**
- Local loop kernels

**HPCC RA: Chapel vs. Reference**
- Global Random Updates

**HPCC STREAM Triad: Chapel vs. Reference**
- STREAM Triad
- ISx

**PRK Stencil: Chapel vs. Reference**
- PRK Stencil

**ISx: Chapel vs. Reference**
- Bucket-Exchange Pattern

Nightly performance tickers online at: https://chapel-lang.org/perf-nightly.html
HPC Patterns: Chapel vs. Reference

- **LCALS Serial Kernels**: Chapel vs. Reference
  - Local loop kernels
- **STREAM Triad**: Chapel vs. Reference
  - Embarrassing/Pleasing Parallelism
- **HPCC RA**: Chapel vs. Reference
  - Global Random Updates
  - PRK Stencil Performance
- **ISx**: Chapel vs. Reference
  - Bucket-Exchange Pattern
- **PRK Stencil**: Chapel vs. Reference
  - Stencil Boundary Exchanges

Nightly performance tickers online at: [https://chapel-lang.org/perf-nightly.html](https://chapel-lang.org/perf-nightly.html)
HPCC RA: Chapel vs. Reference

RA Performance (GUPS)

GUPS

Reference (bucketing)
Reference (no bucketing)
Chapel 1.18 (pre-release)

Locales (x 36 cores / locale)
/* Perform updates to main table. The scalar equivalent is: */

/* */

/* mpi_recv */

HPCC Random Access Kernel: MPI

while (i < SendCnt) {
    /* receive messages */
    do {
        MPI_Test(&inreq, have_done, status);  
    } while (have_done == 0);
    if (have_done) {
        if (status.MPI_TAG == UPDATE_TAG) {
            /* send remaining updates in buckets */
            while (pendingUpdates > 0) {
                /* receive messages */
                do {
                    MPI_Test(&inreq, have_done, status);  
                } while (have_done == 0);
                if (have_done) {
                    if (status.MPI_TAG == UPDATE_TAG) {
                        /* we got a done message. Thanks for playing... */
                        NumberReceiving--;  
                    } else {
                        MPI_Abort( MPI_COMM_WORLD, -1 );
                    }
                } else if (status.MPI_TAG == FINISHED_TAG) {
                    NumberReceiving--;  
                } else {
                    MPI_Abort( MPI_COMM_WORLD, -1 );
                }
            }
        } else {
            MPI_InsertUpdate(Ran, WhichPu, Buckets);  
            pendingUpdates++;  
        }  
        if (have_done) {
            MPI_Test(&inreq, have_done, status);  
        } else {
            MPI_Test(&inreq, have_done, MPI_STATUS_IGNORE);  
        }  
        if (have_done) {
            update = MPI_REQUEST_NULL; continue;  
        }  
    } /* send our done messages */
    for (proc_count = 0; proc_count < tparams.NumProcs; proc_count++) {
        if (proc_count == tparams.MyProc) {  
            tparams.finish_req = MPI_REQUEST_NULL; continue;  
        } /* send garbage - who cares, no one will look at it */
        MPI_Isend(&Ran, 0, tparams.dtype64, proc_count, FINISHED_TAG,  
                  MPI_COMM_WORLD, tparams.finish_req);  
    } /* Finish everyone else up... */
    while (NumberReceiving > 0) {
        MPI_Waitall(tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);
        MPI_Test(&outreq, haveDone, MPI_STATUS_IGNORE);  
        if (haveDone) {
            outreq = MPI_REQUEST_NULL;  
        } else {
            MPI_Isend(&localSendBuffer, localBufferSize, tparams.dtype64,  
                      UPDATE_TAG, MPI_COMM_WORLD, outreq);  
            pendingUpdates -= pendingUpdates;
        }  
    }  
}

Copyright 2018 Cray Inc.
/* Perform updates to main table. The scalar equivalent is: */

forall (_, r) in zip(Updates, RAStream()) do
    T[r & indexMask] ^= r;

/* Perform updates to main table. The scalar equivalent is: */

for (i=0; i<NUPDATE; i++) {
    Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
    Table[Ran & (TABSIZE-1)] ^= Ran;
}
Chapel Libraries

~60 library modules

- web-documented, many user-contributed
Chapel Libraries

**Math:** FFTW, BLAS, LAPACK, LinearAlgebra, Math

**Inter-Process Communication:** MPI, ZMQ (ZeroMQ)

**Parallelism:** Futures, Barrier, DynamicIter

**Distributed Computing:** DistributedIter, DistributedBag, DistributedDeque, Block, Cyclic, Block-Cyclic, …

**File Systems:** FileSystem, Path, HDFS

**Others:** BigInteger, BitOps, Crypto, Curl, DateTime, Random, Reflection, Regexp, Search, Sort, Spawn, …
Chapel Tools

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

https://chapel-lang.org

- downloads
- presentations
- papers
- resources
- documentation
Chapel Online Documentation

https://chapel-lang.org/docs: ~200 pages, including primer examples
Chapel Social Media (no account required)

http://twitter.com/ChapelLanguage

http://facebook.com/ChapelLanguage

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

read-only mailing list: chapel-announce@lists.sourceforge.net (~15 mails / year)

https://stackoverflow.com/questions/tagged/chapel
https://github.com/chapel-lang/chapel/issues
https://gitter.im/chapel-lang/chapel

https://cray.com
Suggested Reading (short attention spans)

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

- a short-and-sweet introduction to Chapel

**Six Ways to Say “Hello” in Chapel** (parts 1, 2, 3), Cray Blog, Sep-Oct 2015.
- a series of articles illustrating the basics of parallelism and locality in Chapel

**Why Chapel?** (parts 1, 2, 3), Cray Blog, Jun-Oct 2014.
- a series of articles answering common questions about why we are pursuing Chapel in spite of the inherent challenges

(index available on chapel-lang.org “blog posts” page), Apr-Nov 2012.
- a series of technical opinion pieces designed to argue against standard reasons given for not developing high-level parallel languages
Suggested Reading (healthy attention spans)

Chapel chapter from *Programming Models for Parallel Computing*
- a detailed overview of Chapel’s history, motivating themes, features
- published by MIT Press, November 2015
- edited by Pavan Balaji (Argonne)
- chapter is also available [online](https://chapel-lang.org/papers.html)

Other Chapel papers/publications available at [https://chapel-lang.org/papers.html](https://chapel-lang.org/papers.html)
Abstract—Chapel is a programming language whose goal is to support productive, general-purpose parallel computing at scale. Chapel’s approach can be thought of as combining the strengths of Python, Fortran, C/C++, and MPI in a single language. Five years ago, the DARPA High Productivity Computing Systems (HPCS) program that launched Chapel wrapped up, and the team embarked on a five-year effort to improve Chapel’s appeal to end-users. This paper follows up on our CUG 2013 paper by summarizing the progress made by the Chapel project since that time. Specifically, Chapel’s performance now competes or beats hand-coded C++ and OpenMP; its suite of standard libraries has grown to include FFTW, BLAS, LAPACK, MPI, ZMQ, and other key technologies; its documentation has been modernized and fleshed out; and the set of tools available to Chapel users has grown. This paper also characterizes the experiences of early adopters from communities as diverse as astrophysics and artificial intelligence.

Keywords—Parallel programming; Computer languages

I. INTRODUCTION

Chapel is a programming language designed to support productive, general-purpose parallel computing at scale. Chapel’s approach can be thought of as striving to create a language whose code is as attractive to read and write as Python, yet which supports the performance of Fortran and the scalability of MPI. Chapel also aims to compete with C++

Paper and slides available at chapel-lang.org
Chapel User Profiles

Current Users:
- Time-to-science Cosmologist
- Commercial AI Scientist

Potential Users:
- Genomic Researcher
- DOE Scientist
User Profile: Time-to-Science Cosmologist

Name: Nikhil Padmanabhan
Title: Associate Professor of Physics and Astronomy, Yale University

Computations: Surveys of galaxies to constrain cosmological models, n-body simulations of gravity

Why Chapel? “My interests in Chapel developed from a desire to have a lower barrier to writing parallel codes. In particular, I often find myself writing prototype codes (often serial), but then need to scale these codes to run on large numbers of simulations/datasets. Chapel allows me to smoothly transition from serial to parallel codes with a minimal number of changes.

“Another important issue for me is "my time to solution" (some measure of productivity vs performance). Raw performance is rarely the only consideration.”
Why Chapel? “I have used Fortran, R, Java and Python extensively. If I had to give up Chapel, I would probably move to C++. I prefer Chapel due to the extreme legibility and performance. We have abandoned Python on large problems for performance reasons.

“We’ve now developed thousands of lines of Chapel code and a half dozen open source libraries for things like database connectivity, numerical libraries, graph processing, and even a REST framework. We’ve done this because AI is about to face an HPC crisis, and the folks at Chapel understand the intersection of usability and scalability.”
Potential User Profile: Genomic Researcher

Name: Jonathan Dursi  
Title: Senior Research Associate, The Hospital for Sick Children, Toronto  
Computations: Human genomics, bioinformatics, and medical informatics

Why Chapel? “My interest in Chapel lies in its potential for bioinformatics tools that are currently either written in elaborately crafted, threaded but single node, C++ code, or in Python. Either has advantages and disadvantages (performance vs rapid development cycles), but neither has a clear path to cross-node computation, for performance as well as larger memory and memory bandwidth. Chapel has the potential to have some of the best of both worlds in terms of C++ and Python, as well as having a path to distributed memory.”
Potential User Profile: DOE Scientist

Name: Anshu Dubey  
Title: Computer Scientist, Argonne National Laboratory  
Computations: Design and development of Multiphysics software that can serve multiple science domains; solvers for PDEs and ODEs

Why Chapel? “In Multiphysics applications separation of concerns and use of high level abstractions is critical for sustainable software. Chapel combines language features that would enable this for clean implementation.

“HPC Scientific software is made more complex than it needs to be because the only language designed for scientific work, Fortran, is losing ground for various reasons. Its object oriented features are clunky and make it nearly as unsuitable as other languages for scientific work. Chapel appears to be parallel and modern Fortran done better, therefore has the potential to become a more suitable language.”
Chapel and Productivity

Chapel aims to be as...

...programmable as Python
...fast as Fortran
...scalable as MPI, SHMEM, or UPC
...portable as C
...flexible as C++
...fun as [your favorite language]
Where to..

Submit bug reports:
GitHub issues for chapel-lang/chapel: public bug forum
chapel_bugs@cray.com: for reporting non-public bugs

Ask User-Oriented Questions:
StackOverflow: when appropriate / other users might care
Gitter (chapel-lang/chapel): community chat with archives
chapel-users@lists.sourceforge.net: user discussions

Discuss Chapel development
chapel-developers@lists.sourceforge.net: developer discussions
GitHub issues for chapel-lang/chapel: for feature requests, design discussions

Discuss Chapel’s use in education
chapel-education@lists.sourceforge.net: educator discussions

Directly contact Chapel team at Cray: chapel_info@cray.com
Wrap-up
The Chapel language offers a unique combination of productivity, performance, and parallelism.

We’re interested in finding and working with the next generation of Chapel users.
Chapel’s Home in the Landscape of New Scientific Computing Languages (and what it can learn from the neighbours)

Jonathan Dursi, The Hospital for Sick Children, Toronto
“My opinion as an outsider...is that Chapel is important, Chapel is mature, and Chapel is just getting started.

“If the scientific community is going to have frameworks...that are actually designed for our problems, they’re going to come from a project like Chapel.

“And the thing about Chapel is that the set of all things that are ‘projects like Chapel’ is ‘Chapel.’”

—Jonathan Dursi

Chapel’s Home in the New Landscape of Scientific Frameworks
(and what it can learn from the neighbours)
CHIUW 2017 keynote

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 publicly 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, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. 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 on this website are the property of their respective owners.
Chapel Comes of Age: Productive Parallelism at Scale
CUG 2018
Brad Chamberlain, Chapel Team, Cray Inc.