Cray: The Supercomputer Company

Piz Daint specifications

Model Cray XC40/Cray XC50

- Number of Hybrid Compute Nodes: 5,320
- Number of Multicore Compute Nodes: 1,431
- Peak Floating-point Performance per Hybrid Node: 4,761 Terasflops Intel Xenon E5-2690 v3/Nvidia Tesla P100
- Peak Floating-point Performance per Multicore Node: 1,210 Terasflops Intel Xenon E5-2695 v4
- Hybrid Peak Performance: 25,326 Petaflops
- Multicore Peak Performance: 1,731 Petaflops
- Hybrid Memory Capacity per Node: 64 GB, 16 GB CoWoS HBM2
- Multicore Memory Capacity per Node: 64 GB, 128 GB
- Total System Memory: 437.9 TB; 83.1 TB
- System Interconnect: Cray Aries routing and communications ASIC, and Dragonfly network topology
- Sonexon 3000 Storage Capacity: 6.2 PB
- Sonexon 3000 Parallel File System Theoretical Peak Performance: 112 GB/s
- Sonexon 1600 Storage Capacity: 2.5 PB
- Sonexon 1600 Parallel File System Theoretical Peak Performance: 138 GB/s

https://www.cray.com

https://www.cscs.ch/computers/piz-daint/
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]
Computer Language Benchmarks Game (CLBG)

Website supporting cross-language comparisons

- 10 toy benchmark programs x ~27 languages x many implementations
- exercise key computational idioms
- specific approach prescribed

The Computer Language Benchmarks Game

Which programs are faster?

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

<table>
<thead>
<tr>
<th>Ada</th>
<th>C</th>
<th>Chapel</th>
<th>C#</th>
<th>C++</th>
<th>Dart</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Erlang</td>
<td>F#</td>
<td>Fortran</td>
<td>Go</td>
<td>Hack</td>
<td></td>
</tr>
<tr>
<td>Haskell</td>
<td>Java</td>
<td>JavaScript</td>
<td>Lisp</td>
<td>Lua</td>
<td></td>
</tr>
<tr>
<td>OCaml</td>
<td>Pascal</td>
<td>Perl</td>
<td>PHP</td>
<td>Python</td>
<td></td>
</tr>
<tr>
<td>Racket</td>
<td>Ruby</td>
<td>Rust</td>
<td>Smalltalk</td>
<td>Swift</td>
<td></td>
</tr>
<tr>
<td>TypeScript</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Which are fast? Trust, and verify

{ for researchers }
Can sort results by various metrics: execution time, code size, memory use, CPU use:

<table>
<thead>
<tr>
<th>Source</th>
<th>Secs</th>
<th>Mem</th>
<th>Gz</th>
<th>Cpu</th>
<th>Cpu Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapel #2</td>
<td>1.62</td>
<td>34,024</td>
<td>423</td>
<td>1.64</td>
<td>99% 3% 1% 4%</td>
</tr>
<tr>
<td>Chapel</td>
<td>1.62</td>
<td>33,652</td>
<td>501</td>
<td>1.64</td>
<td>100% 0% 1% 1%</td>
</tr>
<tr>
<td>Pascal Free Pascal #3</td>
<td>1.73</td>
<td>2,284</td>
<td>482</td>
<td>1.72</td>
<td>1% 1% 1% 1%</td>
</tr>
<tr>
<td>C gcc</td>
<td>1.73</td>
<td>2,116</td>
<td>448</td>
<td>1.73</td>
<td>1% 99% 0% 1%</td>
</tr>
<tr>
<td>Ada 2005 GNAT #2</td>
<td>1.74</td>
<td>3,776</td>
<td>1065</td>
<td>1.73</td>
<td>1% 100% 1%</td>
</tr>
<tr>
<td>Rust #2</td>
<td>1.74</td>
<td>7,876</td>
<td>1306</td>
<td>1.74</td>
<td>1% 100% 1%</td>
</tr>
<tr>
<td>Rust</td>
<td>1.74</td>
<td>7,892</td>
<td>1420</td>
<td>1.74</td>
<td>1% 2% 1% 1%</td>
</tr>
<tr>
<td>Swift #2</td>
<td>1.75</td>
<td>8,532</td>
<td>601</td>
<td>1.75</td>
<td>100% 0% 2% 1%</td>
</tr>
<tr>
<td>Lisp SBCL #4</td>
<td>1.79</td>
<td>25,164</td>
<td>940</td>
<td>1.79</td>
<td>3% 2% 1% 100%</td>
</tr>
<tr>
<td>C++ g++ #4</td>
<td>1.80</td>
<td>3,868</td>
<td>508</td>
<td>1.89</td>
<td>100% 1% 2% 1%</td>
</tr>
<tr>
<td>Lua #3</td>
<td>1.94</td>
<td>3,248</td>
<td>479</td>
<td>1.93</td>
<td>1% 1% 1% 1%</td>
</tr>
<tr>
<td>Go #3</td>
<td>2.02</td>
<td>10,744</td>
<td>603</td>
<td>2.02</td>
<td>1% 5% 96% 0%</td>
</tr>
<tr>
<td>PHP #5</td>
<td>2.15</td>
<td>9,884</td>
<td>394</td>
<td>2.15</td>
<td>1% 1% 1% 1%</td>
</tr>
<tr>
<td>PHP #4</td>
<td>2.16</td>
<td>9,856</td>
<td>384</td>
<td>2.16</td>
<td>100% 0% 1% 0%</td>
</tr>
<tr>
<td>Racket #2</td>
<td>2.17</td>
<td>27,660</td>
<td>1122</td>
<td>2.17</td>
<td>1% 1% 0% 0%</td>
</tr>
</tbody>
</table>

gz == code size metric strip comments and extra whitespace, then gzip
Plotting normalized CLBG code size x speed

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)

- chapel
- smallest
- fastest
- gmean-smallest
- gmean-fastest
CLBG Cross-Language Summary
(June 1, 2018 standings)

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)

smaller

faster
CLBG Cross-Language Summary
(June 1, 2018 standings, zoomed in)
CLBG Cross-Language Summary
(June 1, 2018 standings, zoomed in)

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)

smaller

faster

Copyright 2018 Cray Inc.
CLBG Cross-Language Summary
(June 1, 2018 standings)
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, ((i-1)%3));Color;
    const group2 = { i in 1..popSize2 } new Chameneos(i, colors1[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
                println(c1, " + ", c2, " -> ", getNewColor(c1, c2));
    }

    // 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
            place = new MeetingPlace(c, population);
        delete place;
    }
```

```
void get_affinity(int* is_smpr, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
    cpu_set_t
    FTS*
    char
    char const*
    int
    int
    int
    int
    int
    size_t
    size_t
    char const*
    size_t
    char const*
    size_t
    char const*
    size_t
    char const*
    size_t
    char const*
    size_t
    char const*

    processor_str = "processor";
    physical_id_str_len = strlen(processor_str);
    processor_str_len = strlen(processor_str);
    physical_id_str_len = strlen(physical_id_str);
    core_id_str = "core_id";
    core_id_str_len = strlen(core_id_str);
    cpu_id_str = "cpu_id";
    cpu_id_str_len = strlen(cpu_id_str);
    cpu_count = 0;
    cpu_set_zero(affinity1);
    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)
    {
        is_smpr[0] = 0;
        return;
    }
    is_smpr[0] = 1;
    CPU_ZERO(affinity1);
```

excerpt from 1210 gz Chapel entry
excerpt from 2863 gz C gcc entry
Can also browse program source code *(but this requires actual thought!):*
CLBG: Qualitative Code Comparisons

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

excerpt from 1210 gz Chapel entry

```
char const* core_id_str = "core id";
size_t core_id_str_len = strlen(core_id_str);
char const* cpu_cores_str = "cpu core";
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 2863 gz C gcc 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* cpu_id;
int physical_id;
int id;
int cpu_cores;
int apic_id;
int cpu_count;
size_t size_t

char const* processor_str = "processor";
char const* physical_id_str = "physical id";
size_t size_t

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;
    }
```

excerpt from 1210 gz Chapel entry

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

13 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
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

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

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

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

Configuration declarations (support command-line overrides)
```bash
./fib --n=1000000
```
Iterators

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

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

```CLUSI-style
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;
  }
}

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

```plaintext
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

Static Type Inference for:
- arguments
- return types
- variables

Explicit types also supported

Static Type Inference for:
- variables
- arguments
- return types
iter fib(n) {
  var current = 0,
      next = 1;

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

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

0
1
1
2
3
5
8...

Base Language Features, by example
Base Language Features, by example

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

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

config const n = 10;

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

Zippered iteration:

```
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
// Base Language Features, by example

// Configuration
config const n = 10;

// Iterative Fibonacci generator
iter fib(n) {
  var current = 0,
  next = 1;

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

// Example usage
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
...
```
iter fib(n) {
    var current = 0,
    next = 1;

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

config const n = 10;

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

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

Tuples
Base Language Features, by example

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

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

```c
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
  writeln("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

```
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
```
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);
```

High-Level Task Parallelism

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);
```

```bash
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

```chpl
taskParallel.chpl

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

```bash
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

```
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);
  }
```

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

```chpl
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      printff("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
Data Parallelism, by example

```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

```
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

```
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

```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);
```

Data-Parallel Forall Loops

```
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
```
Domain Maps (Map Data Parallelism to the System)

Distributed Data Parallelism, by example

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

```
prompt> chpl dataParallel.chpl
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);
```

```
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
```
Chapel Has Several Domain / Array Types

- **dense**
- **strided**
- **sparse**

**associative**

```
“steve”
“lee”
“sung”
“david”
“jacob”
“albert”
“brad”
```

**unstructured**
Chapel Has Several Domain / Array Types

- dense
- strided
- sparse

- associative
- unstructured
Chapel Has Several Domain / Array Types

- dense
- strided
- sparse

- associative
- unstructured
Other Key Data Parallel Features

● **Promotion:** call scalar arrays with array arguments
  
  ```
  var B = sin(A); // results in parallel evaluation
  ```

● **Slicing:** refer to subarrays using range / domain arguments
  
  ```
  ...A[lo..hi, ..]...
  ...
  ...OceanTemp[Coastal]...
  ```

● **Reductions:** collapse arrays to scalars or subarrays
  
  ```
  const hottest = max reduce OceanTemp;
  ```
Chapel Results and Resources
HPC Patterns: Chapel vs. Reference

**LCALS Serial Kernels: Chapel vs. Reference**

**HCMP STREAM Triad: Chapel vs. Reference**

**ISx: Chapel vs. Reference**

**PRK Stencil: Chapel vs. Reference**

**HPCC RA Stencil**

**PRK Stencil: Chapel vs. Reference**

**HPCC RA: 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 BA: Chapel vs. Reference**

Global Random Updates

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

Embarrassing/Pleasing Parallelism

**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
HPC Patterns: Chapel vs. Reference

**LCALS Serial Kernels:** Chapel vs. Reference

Local loop kernels

**LCALS**

**HPCC RA**

**STREAM Triad**

**ISx**

**PRK Stencil**

**Embarrassing/Pleasing Parallelism**

**Bucket-Exchange Pattern**

**Stencil Boundary Exchanges**

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

RA Performance (GUPS)

Locales (x 36 cores / locale)

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

Copyright 2018 Cray Inc.
HPCC Random Access Kernel: MPI

/* Perform updates to main table. The scalar equivalent is:
  *  for (int i=UPDATE; i < UPDATE + maxPendingUpdates; i++) {
  *    Ran = (Ran << 1) ^ ((s64Int) Ran < ZERO64B) ? POLY : ZERO64B;
  *    GlobalOffset = (GlobalOffset + (tparams.TableSize - 1)) -
  *      tparams.GlobalStartMyProc;
  *    HPCC_Table[LocalOffset] ^= Ran;
  */

MPI_Irecv(RecvBuffer, localBufferSize, tparams.dtype64, 
MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, recvUpdates);
while (i < SendCnt) {
  /* receive messages */
  for (i = 0; i < receiveUpdates; j++) {
    imag = LocalRecvBuffer[bufferBase + LocalOffset] =
      LocalOffset = (imag & (tparams.TableSize - 1)) -
      tparams.GlobalStartMyProc;
    HPCC_Table[LocalOffset] ^= imag;
  }
} else if (status.MPI_TAG == FINISHED_TAG) {
  NumberReceiving -= iStatus;
} else
  MPI_Abort( MPI_COMM_WORLD, -1 );
MPI_Irecv(LocalRecvBuffer, localBufferSize, tparams.dtype64, 
MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, recvUpdates);
while (have-done & NumberReceiving > 0) {
  if (pendingUpdates < maxPendingUpdates) {
    Ran = (Ran + COUNT) & ZERO64B ? POLY : ZERO64B;
    GlobalOffset = (GlobalOffset + proc_count); -
    tparams.MinLocalTableSize + 1); 
  } else
    WhichPe = (GlobalOffset - tparams.Remainder) / 
    tparams.MinLocalTableSize;
  if (WhichPe == tparams.MyProc) {
    LocalOffset = (Ran & (tparams.TableSize - 1)) -
      tparams.GlobalStartMyProc;
    HPCC_Table[LocalOffset] ^= Ran;
  } else {
    HPCC_InsertUpdate(Ran, WhichPe, Buckets);
    pendingUpdates++;
  }
  i++;}
} else (
  MPI_Test(Recvreq, have-done, MPI_STATUS_IGNORE);
  if (have-done) {
    bufferBase = 0;
    for (j = 0; j < receiveUpdates; j++) {
      imag = LocalRecvBuffer[bufferBase + LocalOffset] =
        LocalOffset = (imag & (tparams.TableSize - 1)) -
        tparams.GlobalStartMyProc;
      HPCC_Table[LocalOffset] ^= imag;
    }
  } else if (status.MPI_TAG == FINISHED_TAG) {
    NumberReceiving -= iStatus;
  } else
    MPI_Abort( MPI_COMM_WORLD, -1 );
MPI_Irecv(LocalRecvBuffer, localBufferSize, tparams.dtype64, 
MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, recvUpdates);
while (have-done & NumberReceiving > 0)
  if (status.MPI_TAG == UPDATE_TAG) {
    MPI_Get_count(outreq, tparams.dtype64, 
    processors, tparams.numProcs);
    bufferBase = 0;
    for (j = 0; j < receiveUpdates; j++) {
      imag = LocalRecvBuffer[bufferBase + LocalOffset] =
        LocalOffset = (imag & (tparams.TableSize - 1)) -
        tparams.GlobalStartMyProc;
      HPCC_Table[LocalOffset] ^= imag;
    }
  } else if (status.MPI_TAG == FINISHED_TAG) {
    /* we got a done message. Thanks for playing... */
    NumberReceiving -= iStatus;
  } else
    MPI_Abort( MPI_COMM_WORLD, -1 );
MPI_Irecv(LocalRecvBuffer, localBufferSize, tparams.dtype64, 
MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, recvUpdates);
while (have-done & NumberReceiving > 0)
/* 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;
}

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

MPI Comment

/* 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 Kernel

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

**Distributed Computing:** DistributedIters, 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, ...
- **chpldoc**: documentation tool
- **mason**: package manager
- **c2chapel**: interoperability aid
- **bash tab completion**: command-line help
- **chplvis**: performance visualizer / debugger
https://chapel-lang.org

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

https://chapel-lang.org/docs: ~200 pages, including primer examples
Chapel for Python Programmers

Developed by Simon Lund

Chapel for Python Programmers

Subtitle: How I Learned to Stop Worrying and Love the Curlybracket.

So, what is Chapel and why should you care? We all know that Python is the best thing since sliced bread. Python comes with batteries included and there is nothing that can't be expressed with Python in a short, concise, elegant, and easily readable manner. But, if you find yourself using any of these packages - Bohrium, Cython, distarray, mpi4py, threading, multiprocessing, NumPy, Numba, and/or NumExpr - you might have done so because you felt that Python's batteries needed a recharge.

You might also have started venturing deeper into the world of curlybrackets. Implementing low-level methods in C/C++ and binding them to Python. In the process you might have felt that you gained performance but lost your

https://chapel-for-python-programmers.readthedocs.io/
Chapel Social Media (no account required)

http://twitter.com/ChapelLanguage
http://facebook.com/ChapelLanguage
https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/
Chapel Community

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

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

Other Chapel papers/publications available at [https://chapel-lang.org/papers.html](https://chapel-lang.org/papers.html)
Summary

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.