Striving for Productivity and Performance Portability

Brad Chamberlain, Chapel Team, Cray Inc.
Performance Portability in Extreme Scale Computing: Metrics Challenges, Solutions (Dagstuhl 17431)
October 26, 2017
If the HPC Community were to create a truly productive language…
…would we ever know?

Brad Chamberlain, Chapel Team, Cray Inc.
Performance Portability in Extreme Scale Computing: Metrics Challenges, Solutions (Dagstuhl 17431)
October 26, 2017
If the HPC Community were to create a truly productive language... ...how would we ever know?

Brad Chamberlain, Chapel Team, Cray Inc.
Performance Portability in Extreme Scale Computing: Metrics Challenges, Solutions (Dagstuhl 17431)
October 26, 2017
Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.
My Background

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

Industry R&D:
- Currently a Principal Engineer at Cray Inc.
- Technical lead / founding member of the Chapel project
This talk focuses a lot on languages, due to my biases
- That said, many points likely apply to other HPC software…

I tend to use the term “language” in a very loose sense
- “A way of communicating your intent to the computer”
- Think “programming model” or “programming notation” if you prefer
Do we have productive HPC languages?

Scenario 1: Brad leaves his HPC bubble to ask a colleague:
- "I need to write a desktop program. You’re young and hip, what productive language should I use?"
- what does it say that I didn’t simply use my HPC language for this…?
- likely responses: Python, Swift, Go, R, Matlab, …

Scenario 2: Colleague asks Brad:
- "I need to write some general-purpose distributed memory code. You’ve worked in HPC for decades, what should I use?"
- my response: … [awkward embarrassment for our community]
HPC Language Characterizations

Current languages
- pragmatic
- bottom-up design
  - driven by system capabilities
  - mechanism-oriented
- alphabet soup
- NASCAR
- punk rock
- worthy of respect

Productive languages
- idealistic
- top-down design
  - driven by end-user needs
  - intent-oriented
- something more coherent
- tricked-out BMW i8
- classical symphony
- worth striving for
Productivity: Played Out?

● There’s some sense that productivity isn’t “hot” anymore
  ● “Haven’t we [solved | given up on] that by now?”

● Arguably analogous to “peace”
  ● not particularly “new” or “hip” as a concept
  ● reasonable reasons for skepticism about our ability to achieve it
    ● for productivity, these are more social than technical, in my opinion
    ● yet, clearly something to desire / strive for over the alternative

● Personally, I prefer not to throw in the towel (in either case)
Productivity and HPCS

(a brief history, from my perspective)
DARPA HPCS: High *Productivity* Computing Systems

- **Goal:** improve productivity by a factor of 10x
- **Timeframe:** Summer 2002 – Fall 2012
- **Three phases, five competitors:** Cray, HP, IBM, SGI, Sun
- Cray developed a new system architecture, network, software stack
- (this became the very successful Cray® XC30™ Supercomputer Series)
- …and a new programming language: Chapel
Productivity, as defined by HPCS

Productivity =

- performance
- **programmability** (readability, writability, maintainability, modifiability, tunability, …)
- portability
- robustness

A reasonable starting point… but how to measure 10x?

- particularly since most of these are hard to measure individually?

In phase 2, an independent team was created to define this
My “Zany Metrics” (an early brainstorming exercise)

“Zany” Metrics

- Abstractness of Code
  - how much code must change if we...
    ◆ change number of processors, shape of processor set?
    ◆ change problem size?
    ◆ make processors not divide problem size evenly?
    ◆ make processor dimensions, problem size non-2^k?
    ◆ switch dense arrays to sparse?
    ◆ change an array’s rank?

- Portability of Code
  - how much code must change...
    ◆ to run on another vendor’s machine?
    ◆ to get performance satisfactory to that vendor?

(source: HPCS Phase II Metrics Kickoff: 2003-8-5)
"Language Bingo"

<table>
<thead>
<tr>
<th>DARPA</th>
<th>Language Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MPI</td>
</tr>
<tr>
<td>Performs Well</td>
<td>O</td>
</tr>
<tr>
<td>Performance Model</td>
<td>O</td>
</tr>
<tr>
<td>Global View</td>
<td>X</td>
</tr>
<tr>
<td>Post-scalar</td>
<td>~/X</td>
</tr>
<tr>
<td>Abstractions</td>
<td>X</td>
</tr>
<tr>
<td>Succinct</td>
<td>X</td>
</tr>
<tr>
<td>General</td>
<td>O</td>
</tr>
</tbody>
</table>

-- = no comment   O = good     ~ = so-so   X = poor   ? = unproven

(source: an early HPCS productivity meeting)
Sterling’s Model of Productivity

Productivity Factors

(Sterling Model) Version 2.1

Productivity ($\Psi$)

- Performance
  - Peak Performance ($S_p$, $C_m$)
  - Efficiency ($E$)

- Application Construction ($C_s$)
  - Programmability
  - Portability

- Availability ($A$)
  - Maintainability
  - Reliability
  - Accessibility

(source: HPCS Phase II Metrics Kickoff: 2003-8-5)
Sterling’s Model of Productivity

**General Model of Productivity**

- \( R_i = i^{th} \) result product
- \( T_i = \) time to compute result \( R_i \)
- \( T_L = \) total lifetime of machine
- \( T_V = \) total overhead time of machine
- \( T_Q = \) quiescent time of machine
- \( T_R = \) working time of machine
- \( N_R = \) total number of result products during \( T_L \)
- \( C_L = \) all costs associated with machine during \( T_L \)
- \( C_{LS} = \) application software costs during \( T_L \)
- \( C_{LO} = \) costs of ownership during \( T_L \)
- \( C_M = \) cost of procurement and initial installation
- \( C_{Si} = \) cost of application software for result \( R_i \)
- \( \Psi = \) productivity

\[
T_R = \sum_{i=1}^{N_R} T_i \\
T_L = T_R + T_V + T_Q \\
R_L = \sum_{i=1}^{N_R} R_i \\
C_L = C_{LS} + C_M + C_{LO} \\
C_{LS} = \sum_{i=1}^{N_R} C_{Si} \\
\Psi = \frac{R_L}{C_L \times T_L}
\]

(source: HPCS Phase II Metrics Kickoff: 2003-8-5)

Coincidence?!? I think not…
Various Teams’ Models of Productivity

IJHPCA special issue on HPC productivity

http://journals.sagepub.com/toc/hpcc/18/4
The Application Kernel Matrix

(source: Marina del Mar SW Productivity Workshop, 2005?)
HPCS Workflows

Workflow Coverage (2)

- Workflows comprise many steps; many overlapping
- Item in red represents areas with highest HPC specific interest

(source: Cray government review)
Timed Markov Models

Our Workflow Analysis Approach

Timed Markov Models

Formulate

Program

Compile

Debug

Test

Optimize

Run

Workflow Coverage (4)

(source: Cray government review)
User Studies: Quantitative Evaluation

Workflow 2 – Productivity Improvement Summary

- Debug Parallel Code
  - Programmers created less bugs when programming in Chapel
  - Global view of data simplifies programming
- Program Parallel
  - Fewer changes needed when going from serial to parallel
  - Multi-resolution of parallelism makes it easier to go from serial to parallel
- Designing Parallel Algorithm
  - MPI programs frequently require restructuring of the serial C code, finding right MPI lib routine

Overall Relative Productivity Improvement 2.3 to 2.6 X

(source: Cray government review)
“The biggest feature from a broad perspective for me was domains. Especially for scientific codes, it is invaluable to be able to define the couple problem domains you're working with. It makes it trivial to change the size or layout or distribution if you decide you need to, it helps guarantee that all of your different arrays match up. **A 3D rectangular grid is infinitely more clear in Chapel with domains than in C, where you have to figure out how they laid it out (is it one giant array? what is the major dimension? x? z? y?).**"

“I loved not having to think as hard about offsets and counts for the parallel version of the code **in Chapel, as opposed to the MPI version**, where I almost always had to chase down two or three indexing errors.”

“Lastly, **I'm a huge huge fan of the type inference used in Chapel.** I like that I don't have to specify types everywhere--they can just be inferred from how I'm using them, but if I mess something up, the compiler catches it.”
Summary: Many Useful Concepts/Techniques…

BUT…
Which is more productive?
Which is more productive? (tipping my hand)

If I published a study showing that one was 3.6x more productive than the other…
…would you switch?
…would you believe it?
Which is more productive?
Productivity: Your Mileage May Vary

- Productivity is a highly personal, social phenomenon
  - “I’ll know it when I see it”

- To that end, our evaluations should be social, not analytic
  - Support personal weighing of tradeoffs
  - Allocate time and spaces for evaluating potential solutions
Poll: How many are familiar with the Computer Language Benchmarks Game?
**Website supporting cross-language comparisons**

- 13 toy benchmark programs x ~28 languages x several entries
  - exercise key computational idioms
  - specific approach prescribed

---

<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>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>JRuby</td>
<td>Rust</td>
<td>Smalltalk</td>
<td></td>
</tr>
<tr>
<td>Swift</td>
<td>TypeScript</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*For researchers* fast-faster-fastest stories
Like the Application Kernel Matrix in several respects...

The Application Kernel Matrix

(source: Marina del Mar SW Productivity Workshop, 2005?)
Website supporting cross-language comparisons

- 13 toy benchmark programs x ~28 languages x many implementations
- exercise key computational idioms
- specific approach prescribed

Take results with a grain of salt
- your mileage may vary

That said, it is one of the only such games in town…
Chapel’s approach to the CLBG:
- striving for elegance over heroism
- ideally: “Want to learn how program xyz works? Read the Chapel version.”
Relative performance, sorted by geometric mean

<table>
<thead>
<tr>
<th>language</th>
<th>program time / fastest program time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>C++</td>
<td>5</td>
</tr>
<tr>
<td>Ada 2005 GNAT</td>
<td>10</td>
</tr>
<tr>
<td>Rust</td>
<td>30</td>
</tr>
<tr>
<td>Java</td>
<td>100</td>
</tr>
<tr>
<td>Fortran</td>
<td>300</td>
</tr>
<tr>
<td>Intel</td>
<td></td>
</tr>
<tr>
<td>Go</td>
<td></td>
</tr>
<tr>
<td>Scala</td>
<td></td>
</tr>
<tr>
<td>.NET Core</td>
<td></td>
</tr>
<tr>
<td>Haskell</td>
<td></td>
</tr>
<tr>
<td>GHC</td>
<td></td>
</tr>
<tr>
<td>OCaml</td>
<td></td>
</tr>
<tr>
<td>Clujure</td>
<td></td>
</tr>
<tr>
<td>Chapel</td>
<td></td>
</tr>
<tr>
<td>SBCL</td>
<td></td>
</tr>
<tr>
<td>OCaml</td>
<td></td>
</tr>
<tr>
<td>C#</td>
<td></td>
</tr>
<tr>
<td>Mono</td>
<td></td>
</tr>
<tr>
<td>LLVM</td>
<td></td>
</tr>
<tr>
<td>F#</td>
<td></td>
</tr>
<tr>
<td>Monolium</td>
<td></td>
</tr>
<tr>
<td>Pascal</td>
<td></td>
</tr>
<tr>
<td>Free Pascal</td>
<td></td>
</tr>
<tr>
<td>Swift</td>
<td></td>
</tr>
</tbody>
</table>

How many times slower?

benchmarks game 09 Sep 2016 u64q

Copyright 2017 Cray Inc.
Relative performance, sorted by geometric mean
Relative performance, sorted by geometric mean

<table>
<thead>
<tr>
<th>Program</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>faster</td>
</tr>
<tr>
<td>Rust</td>
<td>faster</td>
</tr>
<tr>
<td>C++</td>
<td>faster</td>
</tr>
<tr>
<td>g++</td>
<td>fastest</td>
</tr>
<tr>
<td>Ada 2005</td>
<td>GNAT</td>
</tr>
<tr>
<td>Fortran Intel</td>
<td>Chapel</td>
</tr>
<tr>
<td>Pascal Free</td>
<td>OCaml</td>
</tr>
</tbody>
</table>

How many times slower?

benchmarks game 22 Sep 2017 u64q
Can sort results by execution time, code size, memory or 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% 100% 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% 1% 0%</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% 0% 100% 0%</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% 1%</td>
</tr>
<tr>
<td>Rust</td>
<td>1.74</td>
<td>7,892</td>
<td>1420</td>
<td>1.74</td>
<td>1% 100% 2% 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% 1% 1% 0%</td>
</tr>
<tr>
<td>Lisp SBCL #4</td>
<td>1.79</td>
<td>3,868</td>
<td>508</td>
<td>1.89</td>
<td>1% 1% 2% 1%</td>
</tr>
<tr>
<td>C++ g++ #4</td>
<td>1.89</td>
<td>3,248</td>
<td>479</td>
<td>1.93</td>
<td>1% 1% 1% 99%</td>
</tr>
<tr>
<td>Lua #5</td>
<td>1.94</td>
<td>2,02</td>
<td>10,744</td>
<td>603</td>
<td>2.02</td>
</tr>
<tr>
<td>Go #3</td>
<td>2.02</td>
<td>10,744</td>
<td>603</td>
<td>2.02</td>
<td>2% 0% 5% 96%</td>
</tr>
<tr>
<td>PHP #5</td>
<td>2.15</td>
<td>9,884</td>
<td>394</td>
<td>2.15</td>
<td>1% 0% 100% 0%</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% 0% 2%</td>
</tr>
<tr>
<td>Racket #2</td>
<td>2.17</td>
<td>27,660</td>
<td>1122</td>
<td>2.17</td>
<td>1% 0% 1% 0%</td>
</tr>
</tbody>
</table>

 gz == code size metric
strip comments and extra whitespace, then gzip
Can also compare languages pair-wise:

- but only sorted by execution speed...

<table>
<thead>
<tr>
<th></th>
<th>k-nucleotide</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>secs</td>
<td>mem</td>
<td>gz</td>
<td>cpu</td>
<td>cpu load</td>
</tr>
<tr>
<td>Chapel</td>
<td>16.69</td>
<td>350,432</td>
<td>1063</td>
<td>62.96</td>
<td>100% 92% 93% 93%</td>
</tr>
<tr>
<td>Fortran Intel</td>
<td>87.62</td>
<td>203,604</td>
<td>2238</td>
<td>87.57</td>
<td>1% 0% 100% 0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>fasta</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>secs</td>
<td>mem</td>
<td>gz</td>
<td>cpu</td>
<td>cpu load</td>
</tr>
<tr>
<td>Chapel</td>
<td>1.71</td>
<td>52,184</td>
<td>1392</td>
<td>5.90</td>
<td>99% 82% 83% 82%</td>
</tr>
<tr>
<td>Fortran Intel</td>
<td>2.53</td>
<td>8</td>
<td>1327</td>
<td>2.53</td>
<td>0% 1% 0% 100%</td>
</tr>
</tbody>
</table>
Scatter plots of CLBG code size x speed

Compressed Code Size (normalized to smallest entry) vs. Execution Time (normalized to fastest entry)
CLBG Language Cross-Language Summary
(Oct 2017 standings)

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)

smaller

faster
CLBG Language Cross-Language Summary
(Oct 2017 standings, zoomed in)
CLBG Language Cross-Language Summary
(Oct 2017 standings, zoomed in)

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)

smaller

taller

Typescript
Javascript
Scala
Haskell
F#
OCaml
Swift
Go
Pascal
C#
Swift
Java
Ruby
Scala
Go
C++
C
Chapel
Fortran
Rust

Copyright 2017 Cray Inc.
Can also browse program source code (*but this requires actual thought!):
CLBG: Qualitative Comparisons

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

excerpt from 1210 gz Chapel entry

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

cobegin {
    holdMeetings(group1, n);
    holdMeetings(group2, n);
}
p($group1);
print($group2);
for c in group1 do delete c;
for c in group2 do delete c;

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

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

```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* pos;
    int cpu_idx;
    int physical_id;
    int core_id;
    int cpu_num;
    int apic_id;
    size_t i;

    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;
    }
}
```

```
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);
```

excerpt from 1210 gz Chapel entry

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

    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
**Benchmark Suite Scorecard (EMBRACE 2017)**

<table>
<thead>
<tr>
<th>HPC user interest</th>
<th>clear paper &amp; pencil</th>
<th>fast reference code</th>
<th>clear reference code</th>
<th>productivity framework</th>
<th>open competition</th>
<th>prescribed approach</th>
<th>historical continuum</th>
<th>apples-to-apples</th>
<th>self-serve comparisons</th>
<th>self-run entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPB</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>DOE PROXY APPS</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>CLBG</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>PRK</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Benchmark Suite Scorecard (EMBRACE 2017)

<table>
<thead>
<tr>
<th></th>
<th>HPC user interest</th>
<th>clear paper &amp; pencil</th>
<th>clear reference code</th>
<th>forum for comparison</th>
<th>productivity framework</th>
<th>prescribed approach</th>
<th>open competitive</th>
<th>historical continuum</th>
<th>apples-to-apples</th>
<th>self-serve comparisons</th>
<th>self-run entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>~</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HPCC</td>
<td>~</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>DOE PROXY APPS</td>
<td>✓</td>
<td>~</td>
<td>✓</td>
<td>~</td>
<td>x</td>
<td>x</td>
<td>~</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>CLBG</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PRK</td>
<td>~</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>~</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Copyright 2017 Cray Inc.
1. Establish an ongoing, online language bake-off

- The “HPC Computer Language Benchmarks Game”
- Support personal comparisons between technologies
- **Challenges:**
  - Finding someone to be the enthusiastic host and benevolent dictator
  - What systems to run on? and whose?
  - What benchmarks? **Intel PRK suite?**
  - How to manage entries? What rules? How reviewed?
  - Website design and implementation
Scalable Parallel Programming Concerns

Q: What should parallel programmers focus on?
A: **Serial Code:** Software engineering and performance
   **Parallelism:** What should execute simultaneously?
   **Locality:** Where should those tasks execute?
   **Mapping:** How to map the program to the system?

For portable performance?

**Separation of Concerns:** (drink!)
Why Consider New Languages at all?

● Do we need a language? And a compiler?
  ● If higher-level syntax is needed for productivity
    ● We need a language
  ● If static analysis is needed to help with correctness
    ● We need a compiler (front-end)
  ● If static optimizations are needed to get performance
    ● We need a compiler (back-end)

(Source: HPCS productivity workshop panel, ~2004?)
Poll: Familiar with Chapel?
Poll: Familiar with Chapel? Have opinions about Chapel?
Poll: Familiar with Chapel?
Have opinions about Chapel?
Have downloaded / tried Chapel?
Poll: Familiar with Chapel?
Have opinions about Chapel?
Have downloaded / tried Chapel?
Within the past 12–18 months?
What is Chapel?

**Chapel:** A productive parallel programming language

- portable
- open-source
- a collaborative effort

**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 strives to be…

…as programmable as Python
…as fast as Fortran
…as scalable as MPI (or SHMEM or UPC)
…as portable as C
…as flexible as C++
…as fun as [your favorite programming language]
The Chapel Team at Cray (May 2017)

14 full-time employees + 2 summer interns
Chapel Community R&D Efforts

(and several others…)

http://chapel.cray.com/collaborations.html
The Challenge

Q: So why don’t we already have such a language already?
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
  …community will
  …co-design between developers and users
  …patience

Chapel is our attempt to reverse this trend
Chapel language feature areas

<table>
<thead>
<tr>
<th>Chapel language concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Maps</td>
</tr>
<tr>
<td>Data Parallelism</td>
</tr>
<tr>
<td>Task Parallelism</td>
</tr>
<tr>
<td>Base Language</td>
</tr>
<tr>
<td>Locality Control</td>
</tr>
<tr>
<td>Target</td>
</tr>
</tbody>
</table>
Base Language

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

Lower-level Chapel

Target
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);
```

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
</table>

Copyright 2017 Cray Inc.
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```

```
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```

```
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```

```
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```

```
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```

```
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```

```
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```

```
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```

```
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```

```
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```

```
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```

```
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```

```
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```

```
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 (to avoid command-line argument parsing)
```
./a.out --n=1000000
```
Base Language Features, by example

```
config const n = 10;

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

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

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

Static type inference for:
- arguments
- return types
- variables
Iterative Fibonacci function:

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

Configuration:

```perl
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

```
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
...
```
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);
```

tuples

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

```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);
```

```
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
...
```
Task Parallelism and Locality Control

Diagram:
- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target
Task Parallelism and Locality, by example

```
taskParallel.chpl

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

```
prompt> chpl taskParallel.chpl -o taskParallel
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

Abstraction of System Resources

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

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

```
prompt> chpl taskParallel.chpl -o taskParallel
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

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

prompt> chpl taskParallel.chpl -o taskParallel
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.maxTaskPar;
    coforall tid in 1..numPUs() do
      printf("Hello from task %n of %n " +
        "running on %s\n",
      tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl -o taskParallel
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

High-Level Task Parallelism

```
taskParallel.chpl

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

```
prompt> chpl taskParallel.chpl -o taskParallel
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.maxTaskPar;
    coforall tid in 1..numPUs() do
      printf("Hello from task \%n of \%n " +
        "running on \%s\n", tid, numTasks, here.name);
  }
```

Not seen here:
Data-centric task coordination via atomic and full/empty vars

Prompt: `chpl taskParallel.chpl -o taskParallel`
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.maxTaskPar;
    coforall tid in 1..numPUs() do
      printf("Hello from task %d of %d running on %s
", tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl -o taskParallel
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
```
Higher-Level Features

Chapel language concepts

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

Higher-level 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 -o dataParallel
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 -o dataParallel
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

### dataParallel.chpl

```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);
```

```bash
prompt> chpl dataParallel.chpl -o dataParallel
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

```
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 -o dataParallel
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

```
use CyclicDist;
config const n = 1000;
var D = {1..n, 1..n}
dxmapped 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 -o dataParallel
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;
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 -o dataParallel
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

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 -o dataParallel
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

Chapel’s prescriptive approach:

```chpl
forall (i, j) in D do...
```

⇒ invoke and inline D’s default parallel iterator
  • defined by D’s type / domain map

```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 -o dataParallel
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
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);

∀ invoke and inline D’s default parallel iterator
  • defined by D’s type / domain map

Example code:
dataParallel.chpl

Chapel’s prescriptive approach:
forall (i,j) in D do...

cyclic domain map
on each target locale...
  • create task per core
  • block local indices across tasks

default domain map

prompt> chpl dataParallel.chpl -o dataParallel
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
Chapel’s prescriptive approach:

```chapel
forall (i,j) in D do...
```

What if I don’t like D’s iteration strategy?

- Write and call your own parallel iterator:
  ```chapel
  forall (i,j) in myParIter(D) do...
  ```
- Or, use a different domain map:
  ```chapel
  var D = {1..n, 1..n} dmapped Block(...);
  ```
- Or, write and use your own domain map:
  ```chapel
  var D = {1..n, 1..n} dmapped MyDomMap(...);
  ```

Domain Maps specify...

- mapping of indices to locales
- layout of domains / arrays in memory
- parallel iteration strategies
- core operations on arrays / domains

Distributed Data Parallelism, by example

---

```chapel
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;
```
Chapel and Performance Portability

- Avoid locking key policy decisions into the language
  - Array memory layout?
  - Sparse storage format?
  - Parallel loop policies?
  - Abstract node architecture?
Chapel and Performance Portability

- Avoid locking key policy decisions into the language
  - Array memory layout? not defined by Chapel
  - Sparse storage format? not defined by Chapel
  - Parallel loop policies? not defined by Chapel
  - Abstract node architecture? not defined by Chapel

- Instead, permit users to specify these in Chapel itself
  - support performance portability through…
    …a separation of concerns (drink!)
    …abstractions—known to the compiler, and therefore optimizable
  - goal: to make Chapel a future-proof language
Chapel Performance: Increasingly Competitive
(novel for Chapel; new within the past 12–18 months)
Crossing the Rapids

Research Prototype

Adopted in Production

Chris’s next MET Office model

Anshu’s next DOE app

[Your production app here]

What are the next stepping stones?

Who’s interested in meeting us partway?

MiniMD
ISx
CoMD
CLBG
PRK Stencil
RA
LULESH
Stream
LCALS
Codes from startups
Time-to-science academic codes

My Vision for Advancing Productive Alternatives

1. Establish an ongoing online bake-off

2. Create forums for apps-languages pair-programming
   ● e.g., host a “speed-dating” Dagstuhl Seminar
     ● $n$ productive language groups
     ● $n$ apps groups looking for alternatives
       ● where $n = 3–5$?
     ● session 0: everyone gives lightning summaries of their language / app
     ● sessions 1–n: rotate apps x language groups

meanwhile, we’re interested in doing this anytime

(so, send open-minded apps groups our way)
“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 for solving scientific problems 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

The Chapel Parallel Programming Language

What is Chapel?

Chapel is a modern programming language that is...

- **parallel**: contains first-class concepts for concurrent and parallel computation
- **productive**: designed with programmability and performance in mind
- **portable**: runs on laptops, clusters, the cloud, and HPC systems
- **scalable**: supports locality-oriented features for distributed memory systems
- **open-source**: hosted on Github, permissively licensed

New to Chapel?

As an introduction to Chapel, you may want to...

- read a [blog article](http://chapel-lang.org) or [book chapter](http://chapel-lang.org)
- watch an [overview talk](http://chapel-lang.org) or browse its [slides](http://chapel-lang.org)
- download the release
- browse [sample programs](http://chapel-lang.org)
- view other resources to learn how to trivially write distributed programs like this:

```chapel
use CyclicDist;  // use the Cyclic distribution library
config const n = 100;

forall i in [1..n] do
    if i mod n == 0 then
        s enlarges_cycle(startId) do
            writeln("Hello from iteration ", i, " of ", n, " running on node ", here.id);
    end
```

What's Hot?

- Chapel 1.16 is now available—download a copy today!
- The CHIUW 2018 [call for participation](http://chapel-lang.org) is now available!
- A recent [Cray blog post](http://chapel-lang.org) reports on highlights from CHIUW 2017.
- Chapel is now one of the supported languages on [Try It Online!](http://chapel-lang.org)
- [Browse slides](http://chapel-lang.org) from PADAL, EAGE, EMBRADE, ACCU, and other recent talks.
- See also: [What's New?](http://chapel-lang.org)
How to Stalk Chapel

http://facebook.com/ChapelLanguage
http://twitter.com/ChapelLanguage
https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/chapel-announce@lists.sourceforge.net
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 now also available [online](http://chapel-lang.org/papers.html)

Other Chapel papers/publications available at [http://chapel-lang.org/papers.html](http://chapel-lang.org/papers.html)
Suggested Reading (short attention spans)

- a run-down of recent events

- 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

- a series of technical opinion pieces designed to argue against standard reasons given for not developing high-level parallel languages
Chapel StackOverflow and GitHub Issues
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
#chapel-users (irc.freenode.net): user-oriented IRC channel
chapel-users@lists.sourceforge.net: user discussions

Discuss Chapel development
chapel-developers@lists.sourceforge.net: developer discussions
#chapel-developers (irc.freenode.net): developer-oriented IRC channel

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

Directly contact Chapel team at Cray: chapel_info@cray.com
Questions?
Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.