

Chapel 101

Michael Ferguson, Chapel Team, Cray Inc.

CHI UW 2018

May 25, 2018



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.



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



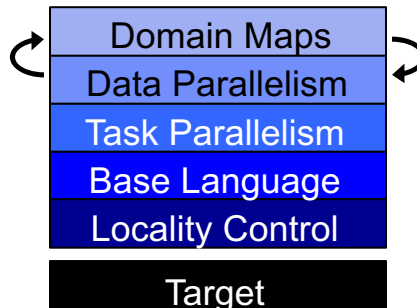
Chapel aims to be as...

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

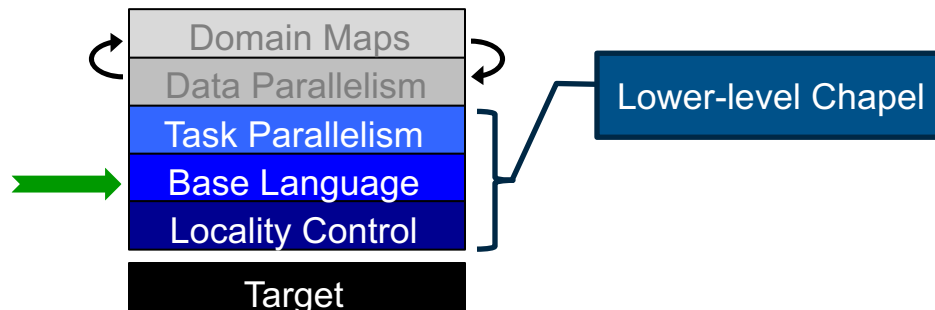


Chapel language feature areas

Chapel language concepts



Base Language



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



Configuration declarations
(support command-line overrides)
`./fib --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);
```

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



Base Language Features, by example

Iterators

```
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

Static type inference for:

- arguments
- return types
- variables

```
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

Zippered iteration

```
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



Range types and operators

```
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

Tuples

```
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

```
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  
...
```

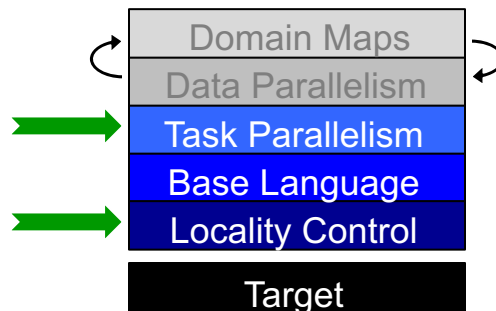


Other Base Language Features

- 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...



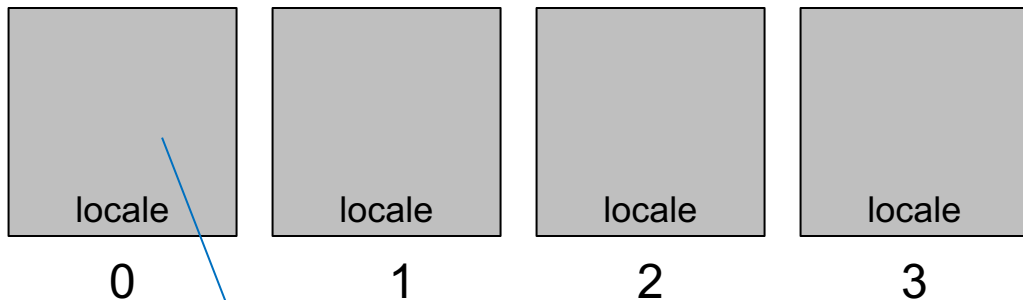
Task Parallelism and Locality Control





- Unit of the target system useful for reasoning about locality
 - Each locale can run tasks and store variables
 - Has processors and memory (or can defer to something that does)
 - For most HPC systems, locale == compute node

Locales:



User's main() executes on locale #0

Task Parallelism and Locality, by example

taskParallel.chpl

```
const numTasks = here.numPUs();  
coforall tid in 1..numTasks do  
  writef("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

Abstraction of
System Resources

taskParallel.chpl

```
const numTasks = here.numPUs();  
coforall tid in 1..numTasks do  
  writef("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

High-Level
Task Parallelism

taskParallel.chpl

```
const numTasks = here.numPUs();  
coforall tid in 1..numTasks do  
  writef("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

taskParallel.chpl

```
const numTasks = here.numPUs();  
coforall tid in 1..numTasks do  
  writef("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

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("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

Abstraction of
System Resources

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("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

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("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



Not seen here:

Data-centric task coordination
via atomic and full/empty vars

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("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

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("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
```

Parallelism and Locality: Distinct in Chapel



- This is a **parallel**, but local program:

```
coforall i in 1..msgs do
  writeln("Hello from task ", i);
```

- This is a **distributed**, but serial program:

```
writeln("Hello from locale 0!");
on Locales[1] do writeln("Hello from locale 1!");
on Locales[2] do writeln("Hello from locale 2!");
```

- This is a **distributed parallel** program:

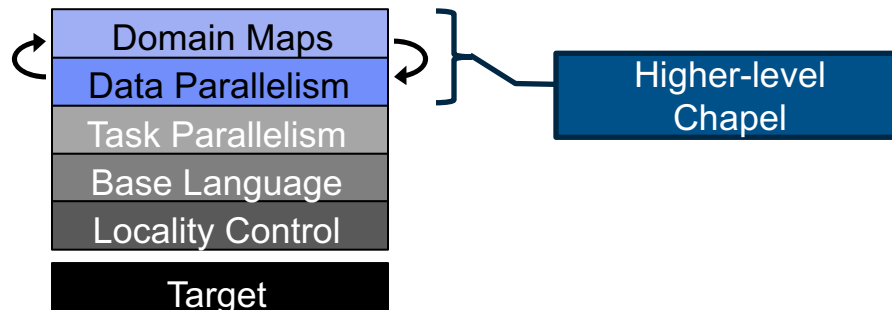
```
coforall i in 1..msgs do
  on Locales[i%numLocales] do
    writeln("Hello from task ", i,
           " running on locale ", here.id);
```



Data Parallelism in Chapel



Chapel language concepts



Data Parallelism, by example

dataParallel.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



Domains (Index Sets)

dataParallel.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



Arrays

dataParallel.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

dataParallel.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

This is a shared memory program

Nothing has referred to remote locales, explicitly or implicitly

dataParallel.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
```



Distributed Data Parallelism, by example



Domain Maps
(Map Data Parallelism to the System)

dataParallel.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

dataParallel.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
```


The Chapel Team at Cray (May 2018)



13 full-time employees + ~2 summer interns



CHI UW 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Community Partners



Lawrence Berkeley
National Laboratory



Sandia National Laboratories



Yale

(and several others...)

<https://chapel-lang.org/collaborations.html>




Chapel Resources





<https://chapel-lang.org>

- downloads
- documentation
- resources
- presentations
- papers



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](#) or [book chapter](#)
- watch an [overview talk](#) or browse its [slides](#)
- [download](#) the release
- browse [sample programs](#)
- view [other resources](#) to learn how to trivially write distributed programs like this:

```
use CyclicDist;           // use the Cyclic distribution library
config const n = 100;      // use --n=<val> when executing to override this default

forall i in {1..n} dmapped Cyclic(startIdx=1) do
  writeln("Hello from iteration ", i, " of ", n, " running on node ", here.id);
```

What's Hot?

- **Chapel 1.17** is now available—[download](#) a copy or browse its [release notes](#)
- The [advance program](#) for **CHIUW 2018** is now available—hope to see you there!
- Chapel is proud to be a [Rails Girls Summer of Code 2018 organization](#)
- Watch talks from [ACCU 2017](#), [CHIUW 2017](#), and [ATPESC 2016](#) on [YouTube](#)
- [Browse slides](#) from **SIAM PP18**, **NWCPP**, **SeaLang**, **SC17**, and other recent talks
- Also see: [What's New?](#)

Home
What Is Chapel?

What's New?
Upcoming Events
Job Opportunities

How Can I Learn Chapel?
Contributing to Chapel

Documentation

Download Chapel
Try It Now
Release Notes

User Resources
Educator Resources
Developer Resources









Social Media / Blog Posts
Press

Presentations
Tutorials
Publications and Papers

CHIUW
CHUG

Contributors / Credits
Research / Collaborations

chapel-lang.org
chapel_info@cray.com





Chapel Social Media (no account required)



<http://twitter.com/ChapelLanguage>

<http://facebook.com/ChapelLanguage>

<https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/>



CHI'UW 2018

COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Community



<https://stackoverflow.com/questions/tagged/chapel>

<https://github.com/chapel-lang/chapel/issues>

<https://gitter.im/chapel-lang/chapel>

chapel-announce@lists.sourceforge.net

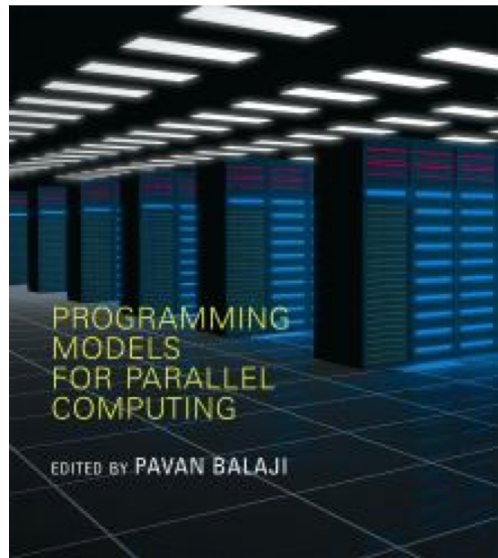
A collage of four screenshots representing the Chapel community's online presence. Top left: A Stack Overflow page for the 'chapel' tag, showing questions like 'Tuple Concatenation in Chapel' and 'Is there a way to use non-scalar values in functions with Chapel'. Top middle: A GitHub repository page for 'chapel-lang / chapel', displaying a list of open issues such as 'Implement "bounded-coforall" optimization for remote coforalls' and 'Consider using processor atomics for remote coforalls EndCount'. Middle: A Gitter chat window with a purple header that says 'Where communities thrive'. The chat shows a discussion about array syntax, with code snippets like 'var A[1..10] int; var B = A; // makes a copy of A' and 'ref C = A; // refers to A'. Bottom: A screenshot of a mailing list or forum thread discussing the 'write()' format specifier for booleans.



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>

Suggested Reading (short attention spans)

[CHIUW 2017: Surveying the Chapel Landscape](#), [Cray Blog](#), July 2017.

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

[Chapel: Productive Parallel Programming](#), [Cray Blog](#), May 2013.

- *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*

[\[Ten\] Myths About Scalable Programming Languages](#), [IEEE TCSC Blog](#)

([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*

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



Legal Disclaimer



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

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

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

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

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

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

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





CRAY
THE SUPERCOMPUTER COMPANY