State of the Chapel Project

Brad Chamberlain, Chapel Team, Cray Inc.

CHIUW 2018
May 25, 2018
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
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 Community Partners

(and several others…)

https://chapel-lang.org/collaborations.html
A Year in the Life of Chapel

- **Two major releases per year** (was: Apr & Oct; now: Mar & Sept)
  - ~a month later: detailed release notes
  - latest release: Chapel 1.17, released April 5th 2018

- **CHIUW**: Chapel Implementers and Users Workshop (May–June)

- **SC** (November)
  - talks, tutorials, panels, BoFs, posters, exhibits, …
  - annual CHUG (Chapel Users Group) happy hour

- **Talks, tutorials, research visits, social media, …** (year-round)
A Year in the Life of Chapel

- Two major releases per year (was: Apr & Oct; now: Mar & Sept)
  - ~a month later: detailed release notes
  - latest release: Chapel 1.17, released April 5th 2018

- CHIUW: Chapel Implementers and Users Workshop (May–June)

- SC (November)
  - talks, tutorials, panels, BoFs, posters, exhibits, …
  - annual CHUG (Chapel Users Group) happy hour

- Talks, tutorials, research visits, social media, … (year-round)
Welcome to CHIUW!

8:30: Chapel 101 (optional)
9:00: Welcome, State of the Project
9:30: Break
10:00: Talks: Applications of Chapel
11:00: Quick Break
11:10: Talks: Chapel Design and Evolution
12:10: Lunch
1:40: Keynote Talk: “Why Languages Matter”, Kathy Yelick
2:40: Talks: Chapel Performance
3:00: Break
3:30: Talks: Tools for Chapel
4:30: Lightning Talks and Flash Discussions
5:30: Wrap-up / Head to Dinner
CHIUW 2018: Organizing Committee

General Chairs:
- Michael Ferguson, Cray Inc.
- Nikhil Padmanabhan, Yale University

Program Committee:
- Brad Chamberlain (chair), Cray Inc.
- Aparna Chandramowlishwaran (co-chair), UC Irvine
- Mike Chu, AMD
- Anshu Dubey, Argonne National Laboratory
- Jonathan Dursi, The Hospital for Sick Children, Toronto
- Hal Finkel, Argonne National Laboratory
- Marta Garcia Gasulla, Barcelona Supercomputing Center
- Clemens Grelck, University of Amsterdam
- Jeff Hammond, Intel
- Bryce Lelbach, Nvidia
- Michelle Strout, University of Arizona
- Kenjiro Taura, University of Tokyo
- David Wonnacott, Haverford College
Kathy Yelick, “Why Languages Matter”

Abstract: In the next few years, exascale computing systems will become available to the scientific community. These systems will require new levels of parallelization, new models of memory and storage, and a variety of node architectures for processors and accelerators. In the decade that follows, we can expect more of these changes, as well as increasing levels of hardware specialization. These systems will provide simulation and analysis capabilities at unprecedented scales, and when combined with advanced physical models, mathematical and statistical methods, and computer science and abstractions, they will lead to scientific breakthroughs. Yet the full power of these systems will only be realized if there is sufficient high-level programming support that will abstract details of the machines and give programmers a natural interface for writing new science applications.
Kathy Yelick, “Why Languages Matter”

Abstract: In the next few years, exascale computing systems will become available to the scientific community. These systems will require new levels of parallelization, new models of memory and storage, and a variety of node architectures for processors and accelerators. In the decade that follows, we can expect more of these changes, as well as increasing levels of hardware specialization. These systems will provide simulation and analysis capabilities at unprecedented scales, and when combined with advanced physical models, mathematical and statistical methods, and computer science and abstractions, they will lead to scientific breakthroughs. Yet the full power of these systems will only be realized if there is sufficient high-level programming support that will abstract details of the machines and give programmers a natural interface for writing new science applications.
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?)
Parallel Sparse Tensor Decomposition in Chapel
Thomas Rolinger (University of Maryland), Tyler Simon, and Christopher Krieger (Laboratory for Physical Sciences)

Iterator-Based Optimization of Imperfectly-Nested Loops
Daniel Feshbach, Mary Glaser (Haverford College), Michelle Strout (University of Arizona), and David Wonnacott (Haverford College)

Investigating Data Layout Transformations in Chapel
Apan Qasem (Texas State University), Ashwin Ajji, and Mike Chu (AMD)

RCUArray: An RCU-like Parallel-Safe Distributed Resizable Array
Louis Jenkins (Bloomsburg University)

Purity: An Integrated, Fine-Grain, Data-Centric Communication Profiler for the Chapel Language
Richard Johnson and Jeffrey Hollingsworth (University of Maryland)
Transitioning from Constructors to Initializers in Chapel
Lydia Duncan and Michael Noakes (Cray Inc.)

Adding Lifetime Checking to Chapel
Michael Ferguson (Cray Inc.)

Tales from the Trenches: Whipping Chapel Performance Into Shape
Elliot Ronaghan, Ben Harshbarger, and Greg Titus (Cray Inc.)

ChplBlamer: A Data-centric and Code-centric Combined Profiler for Multi-locale Chapel Programs
Hui Zhang and Jeffrey Hollingsworth (University of Maryland)

Mason, Chapel’s Package Manager
Ben Albrecht (Cray Inc.), Sam Partee (Haverford College), Ben Harshbarger, and Preston Sahabu (Cray Inc.)
CHIUW 2018: Lightning Talks & Flash Discussions

● Continuing last year’s successful session
● Last session of the day!
● Goal: high-energy hot topics for low attention spans!
● Format: Short talks, Q&A, war stories, …whatever!
● Sign up for a slot!
CHIUW 2018: Lightning Talks & Flash Discussions

- Continuing last year’s successful session
- Last session of the day!
- Goal: high-energy hot topics for low attention spans!
- Format: Short talks, Q&A, war stories, …whatever!
- Sign up for a slot!
Typically, we’ve held a code camp on day 2 of CHIUW
- work on questions, challenges, coding in small teams
- takes advantage of being in one place

This year’s advance response was a bit tepid

So, taking a more *ad hoc* approach
- Plan is to work in pairs / small groups in common areas
- If have a topic you’re interested in partnering on, let us know
- If there’s lots of last-minute interest, we’ll see about a room

8:30: Chapel 101 (optional)
9:00: Welcome, State of the Project
9:30: Break
10:00: Talks: Applications of Chapel
11:00: Quick Break
11:10: Talks: Chapel Design and Evolution
12:10: Lunch
1:40: Keynote Talk: “Why Languages Matter”, Kathy Yelick
2:40: Talks: Chapel Performance
3:00: Break
3:30: Talks: Tools for Chapel
4:30: Lightning Talks and Flash Discussions
5:30: Wrap-up / Head to Dinner
A Brief History of Chapel
A Brief History of Chapel

Chapel’s Infancy: DARPA HPCS (2003–2012)

- ~6–7 Chapel developers at Cray
- Research focus:
  - distinguish locality from parallelism
  - seamlessly mix data- and task-parallelism
  - support user-defined distributed arrays, parallel iterators
- Captured post-HPCS project status in CUG 2013 paper:

  *The State of the Chapel Union*

  Chamberlain, Choi, Dumler, Hildebrandt, Iten, Litvinov, Titus
A Brief History of Chapel

Chapel’s Infancy: DARPA HPCS (2003–2012)

- ~6–7 Chapel developers at Cray
- Research focus:
  - distinguish locality from parallelism
  - seamlessly mix data- and task-parallelism
  - support user-defined distributed arrays, parallel iterators
- Captured post-HPCS project status in CUG 2013 paper:
  "The State of the Chapel Union"
  Chamberlain, Choi, Dumler, Hildebrandt, Iten, Litvinov, Titus

Post-HPCS barriers to using Chapel in practice:
- Performance and Scalability
- Immature Language Features
- Insufficient Libraries
- Memory Leaks
- Lack of Tools
- Lack of Documentation
- Fear of Being the Only User

Yet user interest in Chapel’s potential was high...
A Brief History of Chapel

Chapel’s Infancy: DARPA HPCS (2003–2012)
Chapel’s Adolescence: “the five-year push” (2013–2018)

- ~13–14 Chapel developers at Cray
- Development focus
  - address weak points in HPCS prototype
Chapel Comes of Age: Making Scalable Programming Productive

Bradford L. Chamberlain, Elliot Ronaghan, Ben Albrecht, Lydia Duncan, Michael Eissmann, Ben Harshbarger, David Iten, David Keaton, Vassily Livinov, Preston Sahaba, and Cray Team
Cray Inc.
Seattle, WA, USA
chapel_info@cray.com

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

Keywords—Parallel programming; Computer languages

I. INTRODUCTION

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

The development of the Chapel language was led by Cray as part of its participation in the DARPA High Productivity Computing Systems program. Chapel wrapped up in late 2012, at which time the language was still in an experimental prototype phase, with key research challenges remaining. Since then, Chapel has been well received in the high-performance computing (HPC) community, with key milestones including the release of the first full version of Chapel in early 2015 and the growth of active communities of users and developers. Chapel has also evolved significantly, with improvements in its language, libraries, tools, and documentation.

Under HPCS, Chapel also succeeded in its goal to provide a high-level, productive, and scalable parallelism model. Chapel’s innovations include a novel light-weight parallelism model, called workflow, that is designed to be easy to use and to support productive parallelism at scale.

Chapel’s implementation under Linux uses the OpenSHMEM parallel programming interface for shared memory. Chapel also supports MPI and a dual-computer communications model called HPCX. These parallelization models allow Chapel to take advantage of the most recent parallel architectures, including multi-core, multi-node, and heterogeneous systems.

In summary, Chapel has proven to be a productive, scalable, parallel computing model that is well suited to the needs of the HPC community.
Chapel Comes of Age: Making Scalable Programming Productive

Bradford L. Chamberlain, Elliot Ronaghan, Ben Albrecht, Lydia Duncan, Michael Ferguson, Ben Harshbarger, David Iten, David Keaton, Vasily Liviniov, Preston Sahaba, and Greg Titus
Chapel Team
Cray Inc.
Seattle, WA, USA

Abstract—Chapel is a programming system designed to support productive, parallel code at scale. Chapel’s approach is to combine the strengths of Python, single language. Five years after its initial release, Chapel has grown to include FFTW, other key technologies, and a robust ecosystem of tools and libraries. In this paper, we present a survey of current and prospective Chapel users to gather feedback and improve Chapel’s utility for their computations.

Throughout Chapel’s development, we have worked closely with users and prospective users to get their feedback, and to improve Chapel’s utility for their computations. In preparing this paper, we sent a short survey to a number of current and prospective Chapel users so that we could convey their perspectives on Chapel in their own words. This section summarizes a few of the responses that we received. We start with two current users of Chapel from the fields of Astrophysics and Artificial Intelligence (AI).

VII. USER PERSPECTIVES

Throughout Chapel’s development, we have worked closely with users and prospective users to get their feedback, and to improve Chapel’s utility for their computations. In preparing this paper, we sent a short survey to a number of current and prospective Chapel users so that we could convey their perspectives on Chapel in their own words. This section summarizes a few of the responses that we received. We start with two current users of Chapel from the fields of Astrophysics and Artificial Intelligence (AI).

Nikhil Padmanabhan is an Associate Professor of Physics and Astronomy at Yale University, and a self-described "Astrophysicist without a degree in Astrophysics.

Genomics Researcher

DOE Scientist

Time-to-Science Astrophysicist

Commercial AI Scientist
Notably, user responses all resonated with this goal:

Chapel aims to be as...

...**programmable** as Python
...**fast** as Fortran
...**scalable** as MPI, SHMEM, or UPC
...**portable** as C
...**flexible** as C++
...**fun** as [your favorite language]
A Brief History of Chapel

Chapel’s Infancy: DARPA HPCS (2003–2012)

Chapel’s Adolescence: “the five-year push” (2013–2018)

- ~13–14 Chapel developers at Cray
- Development focus
  - address weak points in HPCS prototype
A Brief History of Chapel: What’s Next?

Chapel’s Infancy: DARPA HPCS (2003–2012)
Chapel’s Adolescence: “the five-year push” (2013–2018)
Chapel’s College Years: “three! more! years!” (2018-2021)

- Continue development focus:
  - Stabilize/Harden Language Core: “no backwards breaking changes”
  - Interoperability / Usability: Python, Jupyter, C++, …
  - Portability: Libfabric/OFI, GPUs, Cloud computing
  - Data Structures: Sparse, DataFrames, Distributed Associative Arrays
  - Chapel AI, Increased Adoption
Chapel: Highlights of the Past Year (or Five)
Chapel Language and Libraries
Language: Highlights Since CHIUW 2017

● **User-defined Initializers:** ready for use
  ● constructor replacement; fix for OOP problems
  ● see Lydia’s talk this morning

● **Error Handling:** ready for use

● ‘defer’ Statement: registers cleanup actions

● **Uninterpreted Strings:** can contain linefeeds, escapes

● **Delete-Free Programming**
  ● improving ‘Owned’ / ‘Shared’ and migrating into the language
  ● see Michael’s talk this morning
Libraries: New Since CHIUW 2017

- **Crypto**: new module based on OpenSSL
  - developed by Sarthak Munshi, GSoC 2017
- **DistributedBag / DistributedDeque**: distributed collections
  - developed by Louis Jenkins, GSoC 2017, speaking this morning
- **DistributedIters**: distributed load-balancing iterators
- **TOML**: initial support for reading TOML files
Libraries: Improved Since CHIUW 2017

- **LinearAlgebra**: various ongoing improvements
- **MPI**: improved support for mixing with various configurations
  - co-developed by Nikhil Padmanabhan
- **ZMQ**: improved interoperability with Python via ZMQ
  - developed by Nick Park
- **Path**: added missing routines
  - developed by Sarthak Munshi, Surya Priy, Unnati Parekh, Prithvi Patel, and Varsha Verma
- **Math**: added Bessel functions
  - developed by Nimit Bhardwaj
Libraries: Post-HPCS

**After HPCS:** ~25 library modules

- documented via source comments, if at all:
Libraries: Now

Now: ~60 library modules

- web-documented, many user-contributed
Libraries: Now

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

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

**Parallelism:** Futures, Barrier, DynamicIter

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

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

**Others:** BigInteger, BitOps, Crypto, Curl, DateTime, Random, Reflection, Regexp, Search, Sort, Spawn, …
Arrays, Domain Maps: New Since CHIUW 2017

- **Sparse:**
  - Added support for CSC layouts
  - Reduced communication for Block-Sparse Arrays

- **Replicated:** Improved behavior

- **Rank Change / Reindex:** Reduced communication
Performance, Generated Code, and Memory Leaks
Performance: Improvements since Chapel 1.16

STREAM Performance

PRK Stencil Performance

Reduction Efficiency

ISx Time

Copyright 2018 Cray Inc.
Performance: Improvements since Chapel 1.16

 STREAM Performance

 PRK Stencil Performance

 Reduction Efficiency

 ISx Time

(For much more on performance, see Elliot’s talk this afternoon)
Memory Leaks: Since CHIUW 2017

Memory leaks in testing reduced ~100x from 1.15 to 1.17:
Memory Leaks: Post-HPCS vs. Now

Total Memory Leaked in Nightly Testing

<table>
<thead>
<tr>
<th>Chapel 1.7</th>
<th>Chapel 1.17</th>
</tr>
</thead>
<tbody>
<tr>
<td>2137.0 MB</td>
<td>0.237 MB</td>
</tr>
</tbody>
</table>

Better
Memory Leaks: Post-HPCS vs. Now

Total Number of Nightly Tests

- Chapel 1.7: 4410
- Chapel 1.17: 8478
Memory Leaks: Post-HPCS vs. Now

Fraction of Tests Leaking Memory

<table>
<thead>
<tr>
<th>Number of Tests</th>
<th>Chapel 1.7</th>
<th>Chapel 1.17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fraction of Tests Leaking Memory</td>
<td>4410</td>
<td>8478</td>
</tr>
<tr>
<td>Number of Tests</td>
<td>3128</td>
<td>302</td>
</tr>
</tbody>
</table>

Copyright 2018 Cray Inc.
Memory Leaks: Remaining Leaks

~1/3 of memory leaked by one test (SSCA#2)

~2/3 of leaking tests leak < 256 bytes

~1/3 of leaking tests leak < 64 bytes
Portability: Highlights Since CHIUW 2017

- **ARM**: Chapel support for Cray XC50 with ARM processors
- **FreeBSD, PowerPC**: Improved portability
- **OmniPath**: Added support
- **gcc**: Improved portability to new versions
Chapel Ecosystem
Tools: Highlights Since CHIUW 2017

● **mason**: package manager
  ● see Ben Albrecht’s talk this afternoon

● **c2chapel**: convert C header files to ‘extern’ declarations

● **bash tab completion**: command-line help for ‘chpl’ args

● **chpl**:
  ● now names executable after main file rather than ‘a.out’
  ● now offers suggestions for unfamiliar flags
  ● improved support for LLVM back-end

● **configure + make install**: added familiar ways to build
After HPCS:

- a PDF language specification
- a Quick Reference sheet
- a number of READMEs
- ~22 primer examples
Now: 200+ modern, hyperlinked, web-based documentation pages
Website: Highlights Since CHIUW 2017

- Added color-coded documentation version menu

Chapel on StackOverflow

- StackOverflow ‘chapel’ questions are on the rise

143 questions tagged (up ~116 since CHIUW 2017)
Try It Online (TIO): now supports Chapel

https://tio.run/

Chapel
- Compiler flags
- Header
- Code
  - 56 chars, 56 bytes (UTF-8)
  - coforall i in 1..10 do
    - writeln("Hello from task ", i);

- Footer
- Input
- Arguments
- Output
  - Hello from task 2
  - Hello from task 4
  - Hello from task 6
  - Hello from task 8
  - Hello from task 10
  - Hello from task 1
  - Hello from task 3
  - Hello from task 5
  - Hello from task 7
  - Hello from task 9

Copyright 2017 Cray Inc.
Summary

Chapel has made huge strides over the past year/5 years

We’ve addressed many historical barriers to using Chapel

We’re continuing our work to support and improve Chapel

We’re looking for the next generation of Chapel users, as well as concrete use cases for AI / ML
In Memory of Burton Smith
8:30: Chapel 101 (optional)
9:00: Welcome, State of the Project
9:30: Break
10:00: Talks: Applications of Chapel
11:00: Quick Break
11:10: Talks: Chapel Design and Evolution
12:10: Lunch
1:40: Keynote Talk: “Why Languages Matter”, Kathy Yelick
2:40: Talks: Chapel Performance
3:00: Break
3:30: Talks: Tools for Chapel
4:30: Lightning Talks and Flash Discussions
5:30: Wrap-up / Head to Dinner
Chapel Resources
Chapel Central

https://chapel-lang.org

- downloads
- documentation
- resources
- presentations
- papers
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
Suggested Reading (healthy attention spans)

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

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

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

- a short-and-sweet introduction to Chapel

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

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

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