Creating an Open-Source Language: From Research Prototype to Production

Brad Chamberlain

UW CSE 403

March 6, 2019

bradc@cray.com

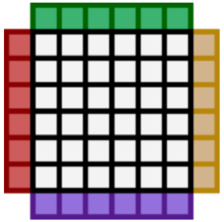
chapel-lang.org

@ChapelLanguage

Who am I?

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



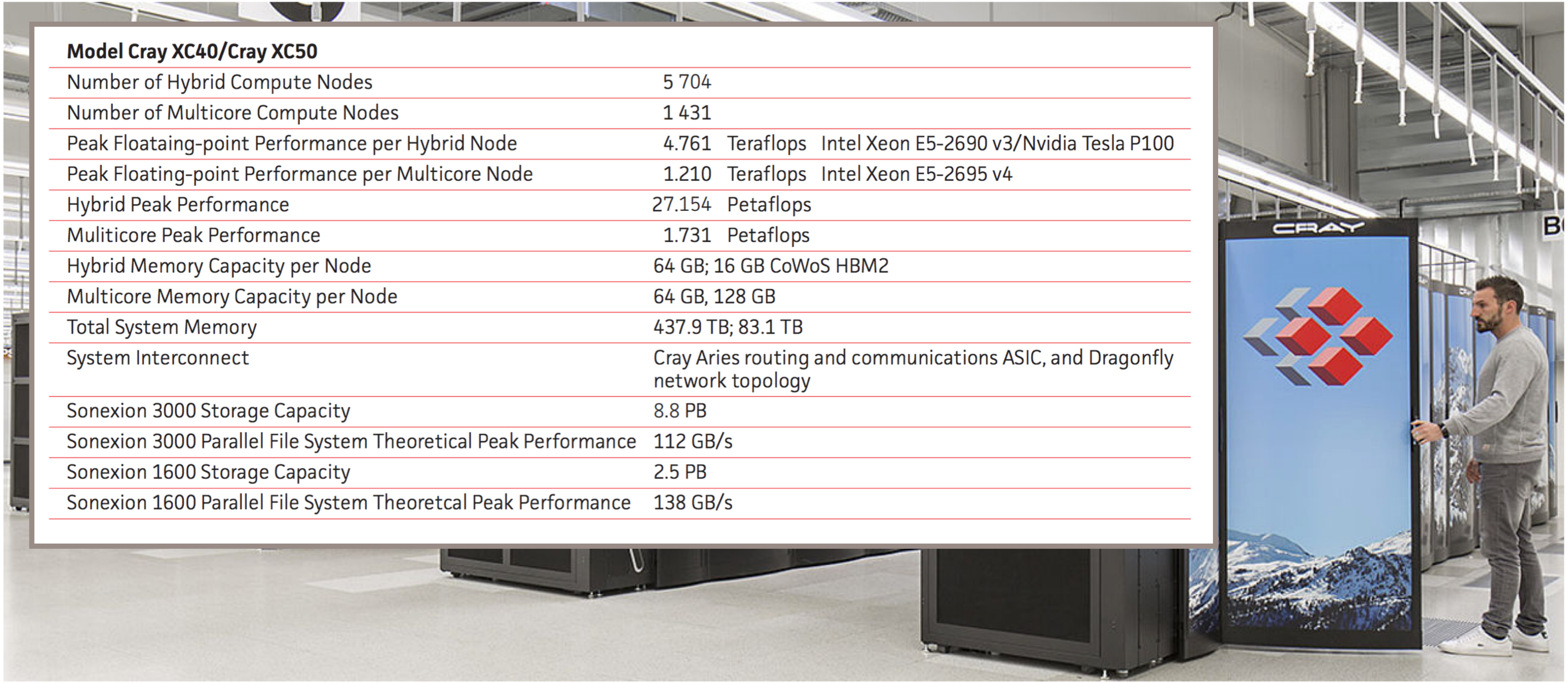
Industry:

- Currently a Principal Engineer at Cray Inc.
- Technical lead / founding member of the Chapel project
- Also spent a year at a startup: QuickSilver Technology

Piz Daint: One of Today's Most Powerful Supercomputers

https://www.cscs.ch/computers/piz-daint/

Piz Daint: One of Today's Most Powerful Supercomputers



Model Cray XC40/Cray XC50			
Number of Hybrid Compute Nodes	5 704		
Number of Multicore Compute Nodes	1 431		
Peak Floataing-point Performance per Hybrid Node	4.761	Teraflops	Intel Xeon E5-2690 v3/Nvidia Tesla P100
Peak Floating-point Performance per Multicore Node	1.210	Teraflops	Intel Xeon E5-2695 v4
Hybrid Peak Performance	27.154	Petaflops	
Mulitcore Peak Performance	1.731	Petaflops	
Hybrid Memory Capacity per Node	64 GB; 16 GB CoWoS HBM2		
Multicore Memory Capacity per Node	64 GB, 128 GB		
Total System Memory	437.9 TB; 83.1 TB		
System Interconnect	Cray Aries routing and communications ASIC, and Dragonfly network topology		
Sonexion 3000 Storage Capacity	8.8 PB		
Sonexion 3000 Parallel File System Theoretical Peak Performance	112 GB/s		
Sonexion 1600 Storage Capacity	2.5 PB		
Sonexion 1600 Parallel File System Theoretcal Peak Performance	138 GB/s		

https://www.cscs.ch/computers/piz-daint/

Outline

✓ Who's Brad? Cray?

➢ What's Chapel?

- Software Engineering & Chapel
- Parting Thoughts
- Chapel Resources

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

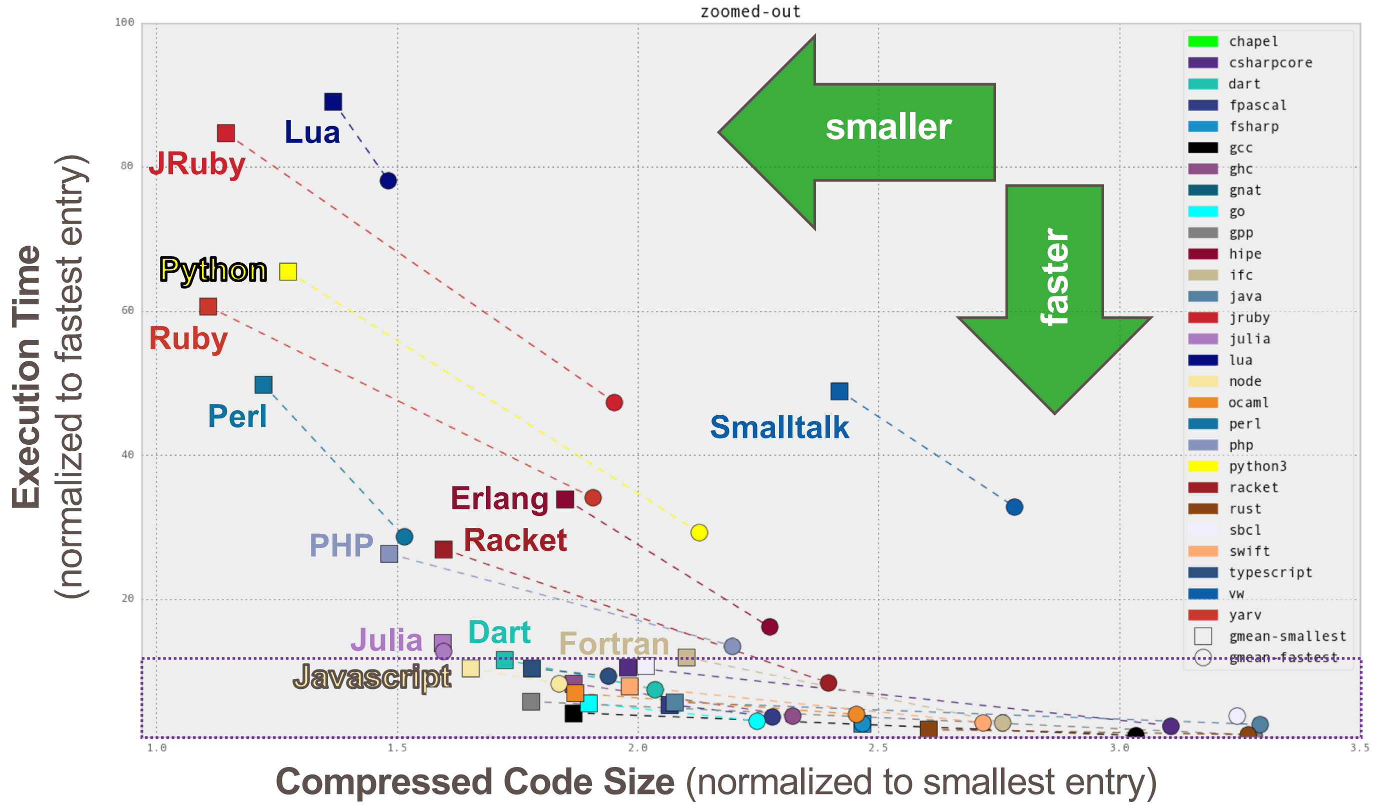
…**scalable** as MPI, SHMEM, or UPC

…**portable** as C

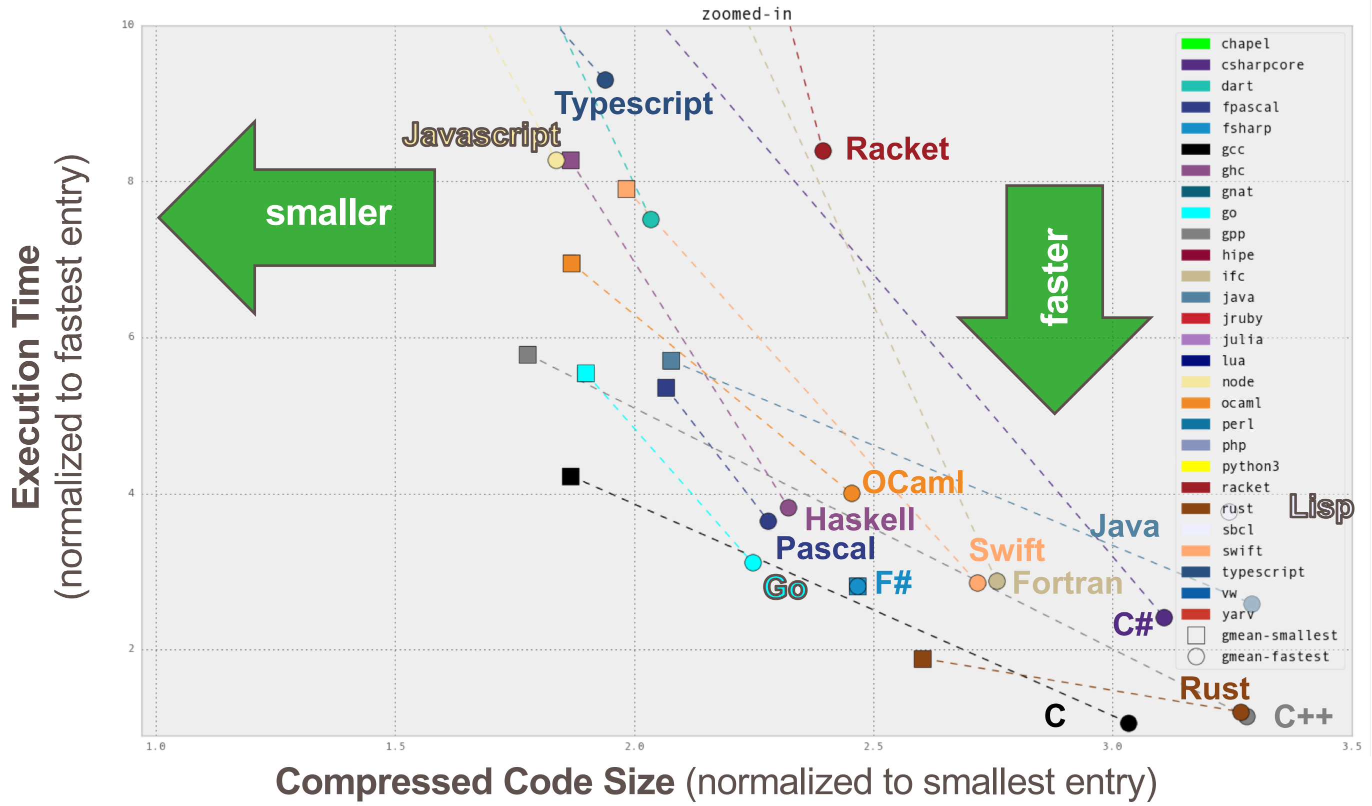
…**flexible** as C++

…**fun** as [your favorite programming language]

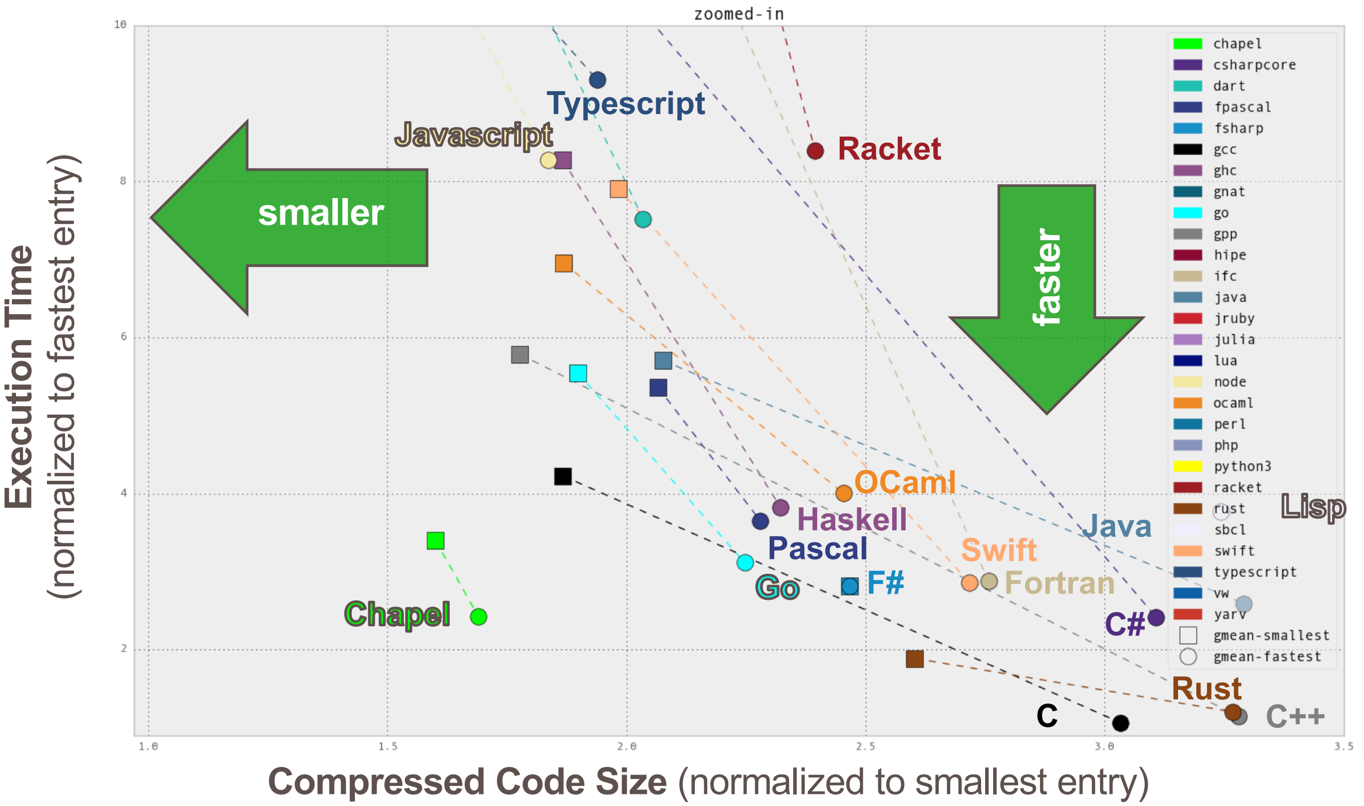
CLBG Cross-Language Summary (Dec 18, 2018)



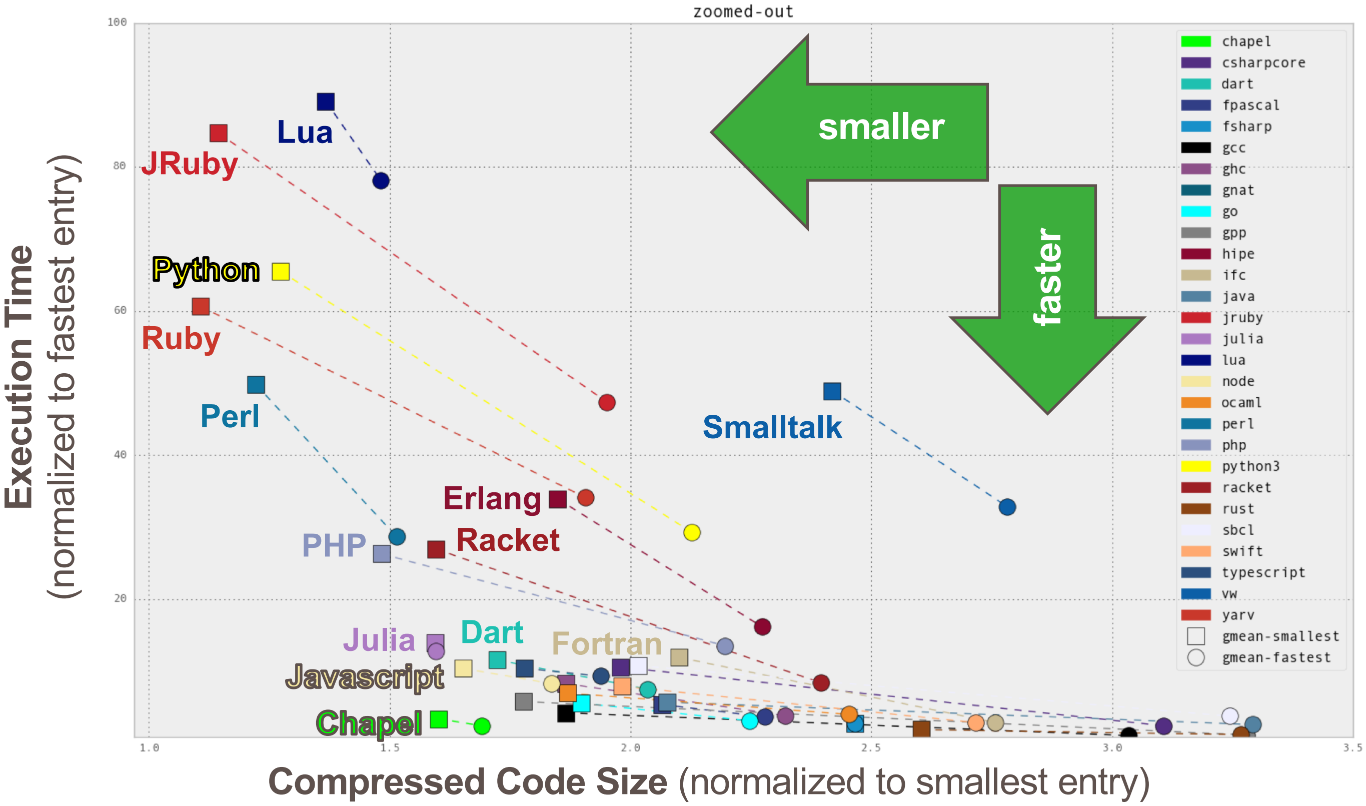
CLBG Cross-Language Summary (Dec 18, 2018, zoomed)



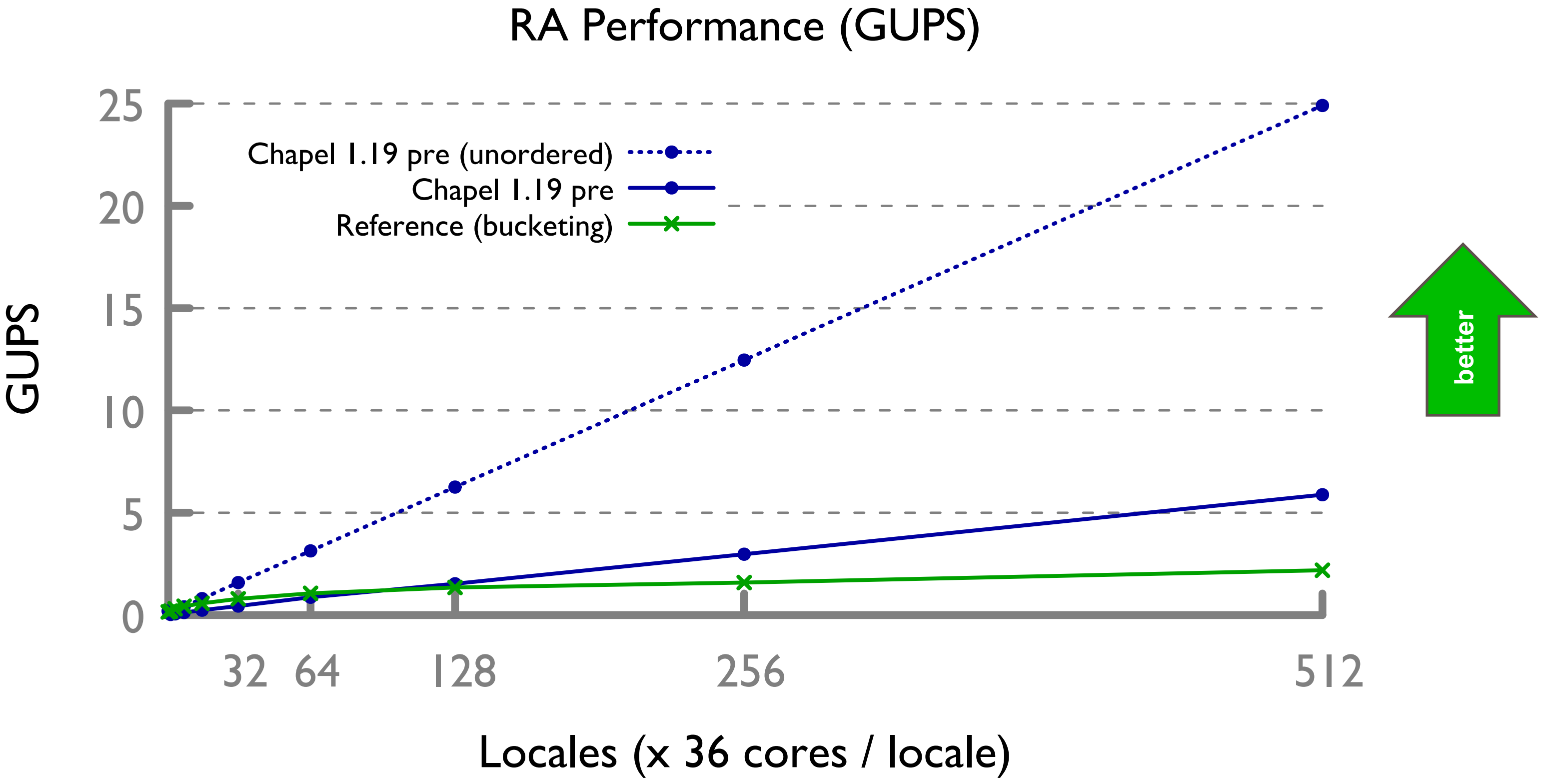
CLBG Cross-Language Summary (Dec 18, 2018, zoomed)



CLBG Cross-Language Summary (Dec 18, 2018)



HPCC RA: buffering vs. network atomics



HPCC RA: MPI kernel

```c
/* Perform updates to main table.  The scalar equivalent is:
 *
 *   for (i=0; i<NUPDATE; i++) {
 *     Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *     Table[Ran & (TABSIZE-1)] ^= Ran;
 *   }
 */

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
 while (i < SendCnt) {
   /* receive messages */
   do {
     MPI_Test(&inreq, &have_done, &status);
     if (have_done) {
       if (status.MPI_TAG == UPDATE_TAG) {
         MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
         bufferBase = 0;
         for (j=0; j < recvUpdates; j ++) {
           inmsg = LocalRecvBuffer[bufferBase+j];
           LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                         tparams.GlobalStartMyProc;
           HPCC_Table[LocalOffset] ^= inmsg;
         }
       } else if (status.MPI_TAG == FINISHED_TAG) {
         NumberReceiving--;
       } else
         MPI_Abort( MPI_COMM_WORLD, -1 );
       MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                 MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
     }
   } while (have_done && NumberReceiving > 0);
   if (pendingUpdates < maxPendingUpdates) {
     Ran = (Ran << 1) ^ ((s64Int) Ran < ZERO64B ? POLY : ZERO64B);
     GlobalOffset = Ran & (tparams.TableSize-1);
     if ( GlobalOffset < tparams.Top)
       WhichPe = ( GlobalOffset / (tparams.MinLocalTableSize + 1) );
     else
       WhichPe = ( (GlobalOffset - tparams.Remainder) /
                  tparams.MinLocalTableSize );
     if (WhichPe == tparams.MyProc) {
       LocalOffset = (Ran & (tparams.TableSize - 1)) -
                     tparams.GlobalStartMyProc;
       HPCC_Table[LocalOffset] ^= Ran;
     }  else {
       HPCC_InsertUpdate(Ran, WhichPe, Buckets);
       pendingUpdates++;
     }
     i++;
   }
   else {
     MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
     if (have_done) {
       outreq = MPI_REQUEST_NULL;
       pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                            &peUpdates);
       MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
                 UPDATE_TAG, MPI_COMM_WORLD, &outreq);
       pendingUpdates -= peUpdates;
     }
   }
 }
/* send remaining updates in buckets */
while (pendingUpdates > 0) {
  /* receive messages */
  do {
    MPI_Test(&inreq, &have_done, &status);
    if (have_done) {
      if (status.MPI_TAG == UPDATE_TAG) {
        MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
        bufferBase = 0;
        for (j=0; j < recvUpdates; j ++) {
          inmsg = LocalRecvBuffer[bufferBase+j];
          LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                        tparams.GlobalStartMyProc;
          HPCC_Table[LocalOffset] ^= inmsg;
        }
      } else if (status.MPI_TAG == FINISHED_TAG) {
        /* we got a done message.  Thanks for playing... */
        NumberReceiving--;
      } else {
        MPI_Abort( MPI_COMM_WORLD, -1 );
      }
      MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    }
  } while (have_done && NumberReceiving > 0);

  MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
  if (have_done) {
    outreq = MPI_REQUEST_NULL;
    pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                         &peUpdates);
    MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
              UPDATE_TAG, MPI_COMM_WORLD, &outreq);
    pendingUpdates -= peUpdates;
  }
}
/* send our done messages */
for (proc_count = 0 ; proc_count < tparams.NumProcs ; ++proc_count) {
  if (proc_count == tparams.MyProc) { tparams.finish_req[tparams.MyProc] =
                                      MPI_REQUEST_NULL; continue; }
  /* send garbage - who cares, no one will look at it */
  MPI_Isend(&Ran, 0, tparams.dtype64, proc_count, FINISHED_TAG,
            MPI_COMM_WORLD, tparams.finish_req + proc_count);
}
/* Finish everyone else up... */
while (NumberReceiving > 0) {
  MPI_Wait(&inreq, &status);
  if (status.MPI_TAG == UPDATE_TAG) {
    MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
    bufferBase = 0;
    for (j=0; j < recvUpdates; j ++) {
      inmsg = LocalRecvBuffer[bufferBase+j];
      LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                    tparams.GlobalStartMyProc;
      HPCC_Table[LocalOffset] ^= inmsg;
    }
  } else if (status.MPI_TAG == FINISHED_TAG) {
    /* we got a done message.  Thanks for playing... */
    NumberReceiving--;
  } else {
    MPI_Abort( MPI_COMM_WORLD, -1 );
  }
  MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
            MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
}

MPI_Waitall( tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);
```

HPCC RA: MPI kernel comment vs. Chapel

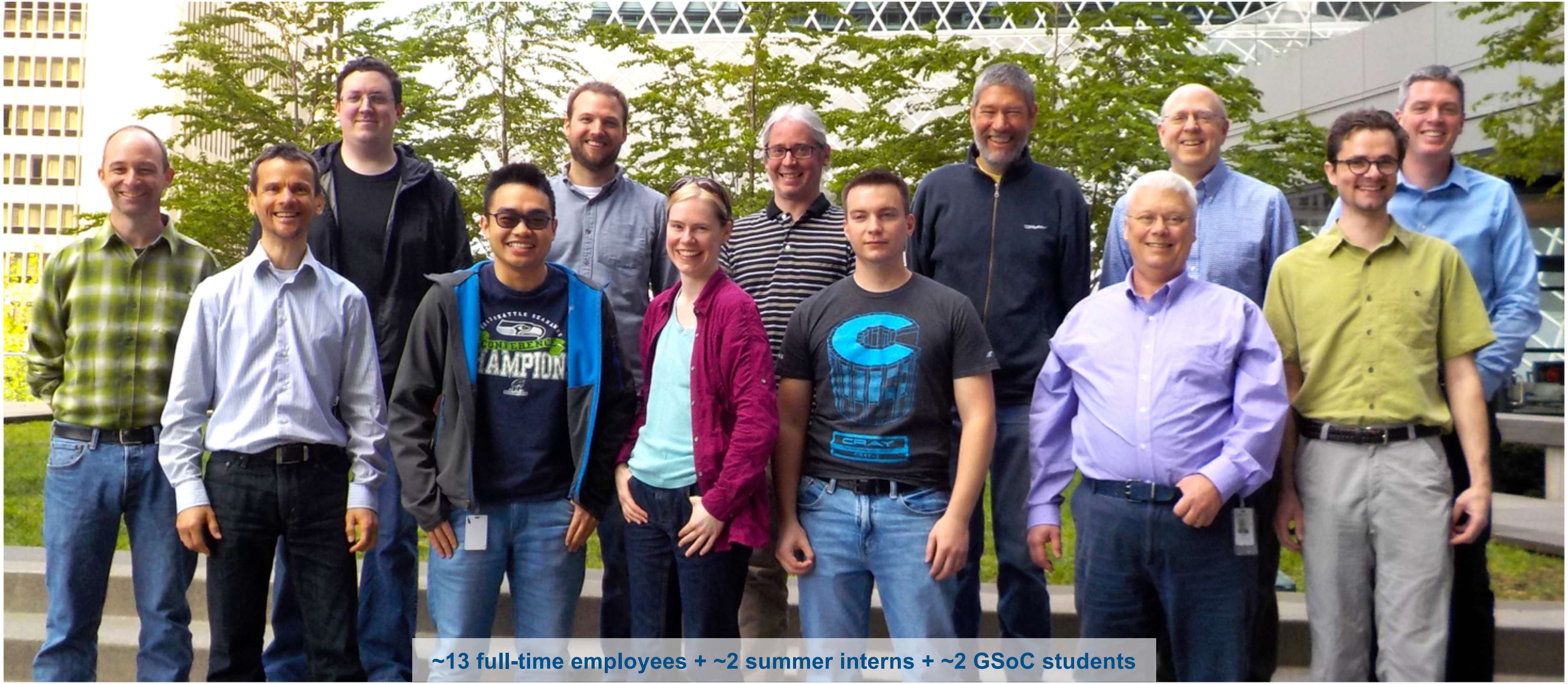
Chapel Kernel

```
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask].xor(r);
```

MPI Comment

```
/* Perform updates to main table.  The scalar equivalent is:
 *
 *     for (i=0; i<NUPDATE; i++) {
 *       Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *       Table[Ran & (TABSIZE-1)] ^= Ran;
 *     }
 */
```

The Chapel Team at Cray (May 2018)



~13 full-time employees + ~2 summer interns + ~2 GSoC students

Software Engineering & Chapel

Disclaimers

- Anything I say may not translate at all to any other job / project you may take on
- I'm reporting on my group's practices and not necessarily those of Cray broadly
- These slides are not particularly pretty…

My year at QuickSilver (between UW and Cray)

- Worked for an Extreme Programming (XP) software group
  - Nowadays, more likely to be Agile software development, Scrum, Kanban, …
- My takeaways:
  - frequent planning (sprints)
  - customer involvement / focus
  - daily standups
  - test-first development
  - pair programming always
  - brags
  - sustainable pace

My year at QuickSilver (between UW and Cray)

- Worked for an Extreme Programming (XP) software group
  - Nowadays, more likely to be Agile software development, Scrum, Kanban
- My takeaways:
  - frequent planning (sprints)
  - customer involvement / focus
  - daily standups
  - test-first development
  - pair programming always
  - brags
  - sustainable pace

A Brief History of Chapel

2003-2006: Initial Concept / Splashing Around (2-4 devs)

- blank slate development

2006-2012: Developing a Research Prototype (6-7 devs)

- research focus

2013-2018: Transition from Research to Production-Grade (~12 devs)

- increased focus on users, adoption, stability

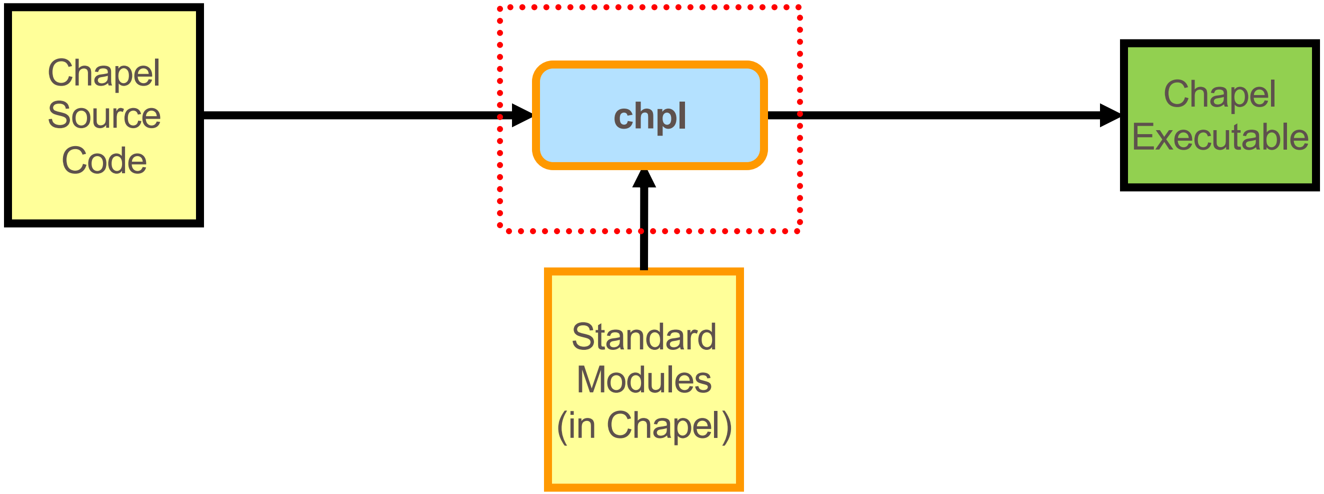
2019-2021: Striving for Adoption (12-15(?) devs)

- lock down production use cases and users

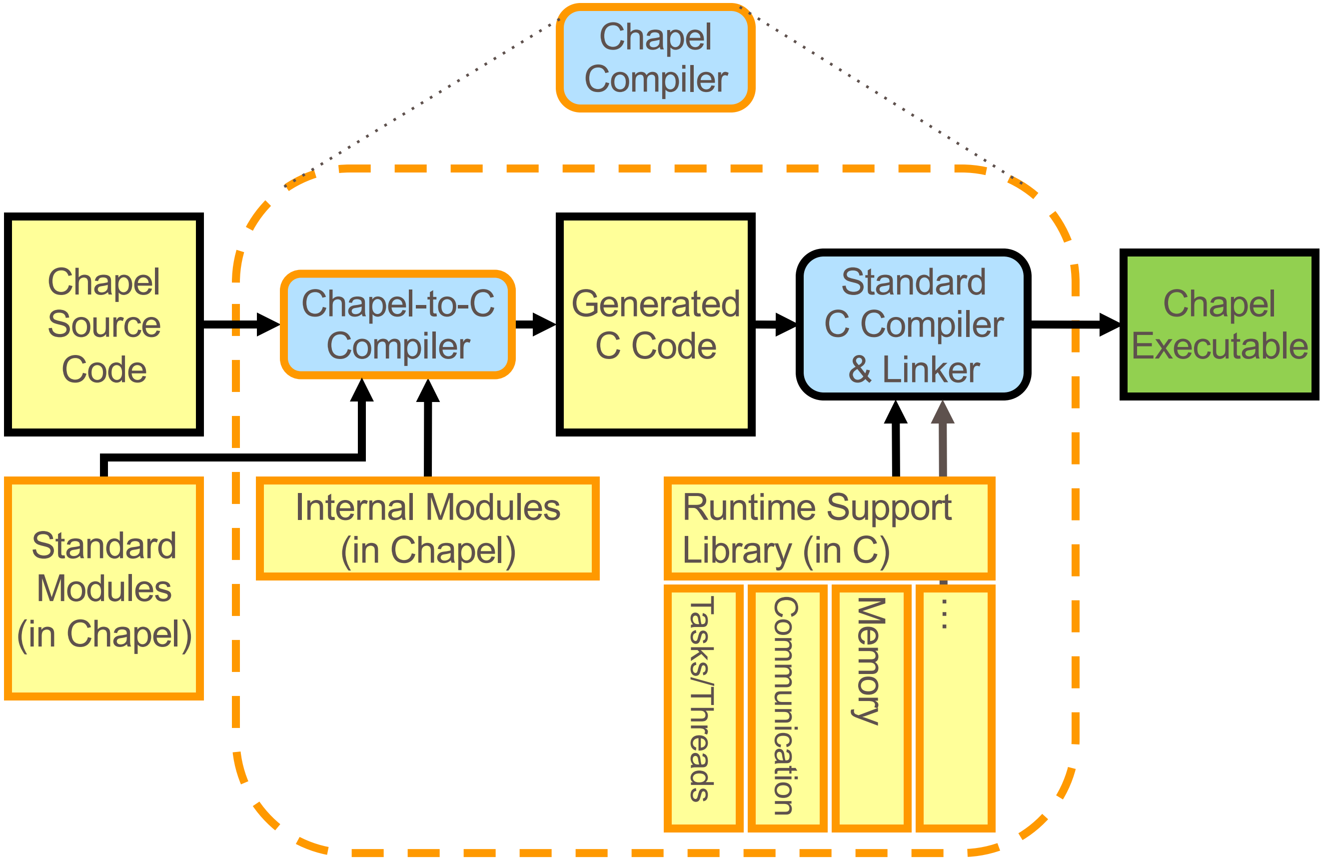
What did we set out to create?

- Language definition
- Compiler
- Runtime (access to system-level capabilities: memory, network, threads, …)
- Standard Libraries
- Tools (minimal)

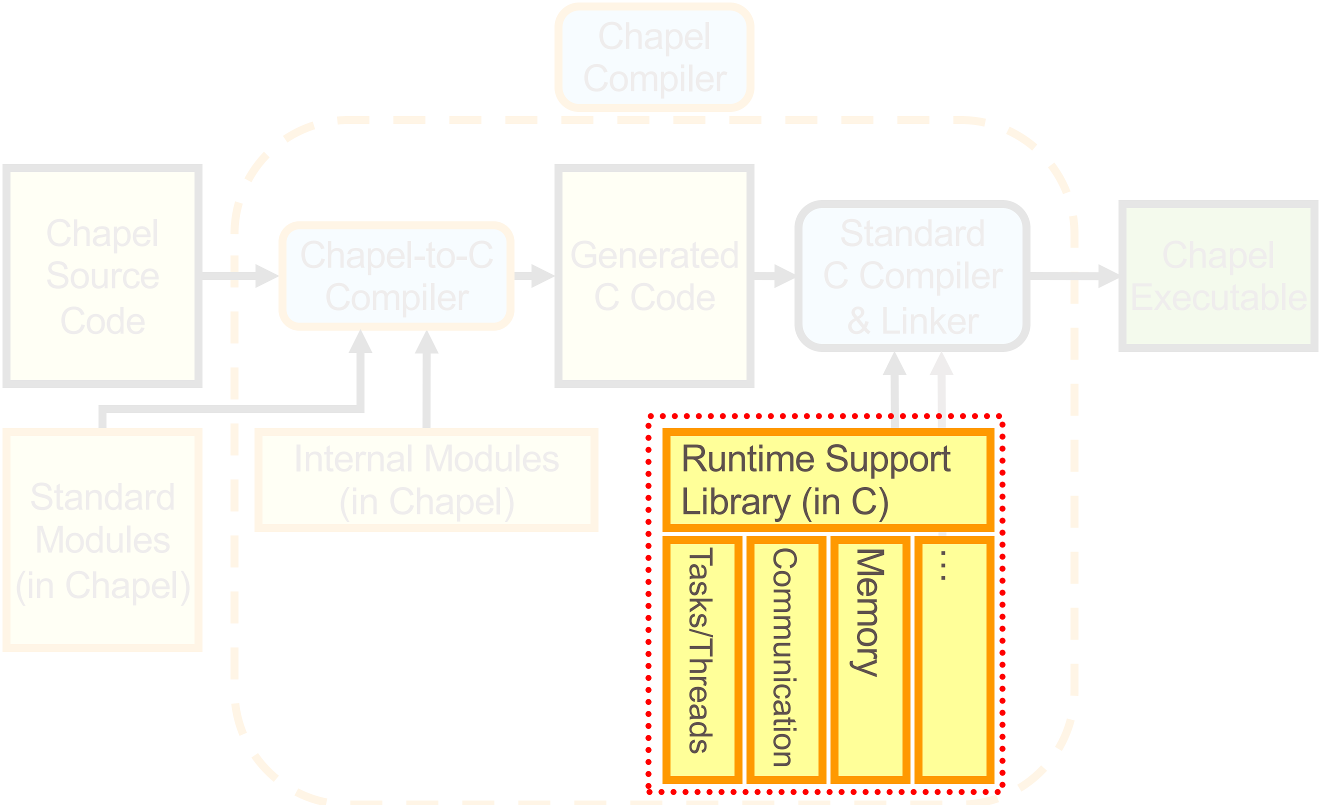
Compiling Chapel



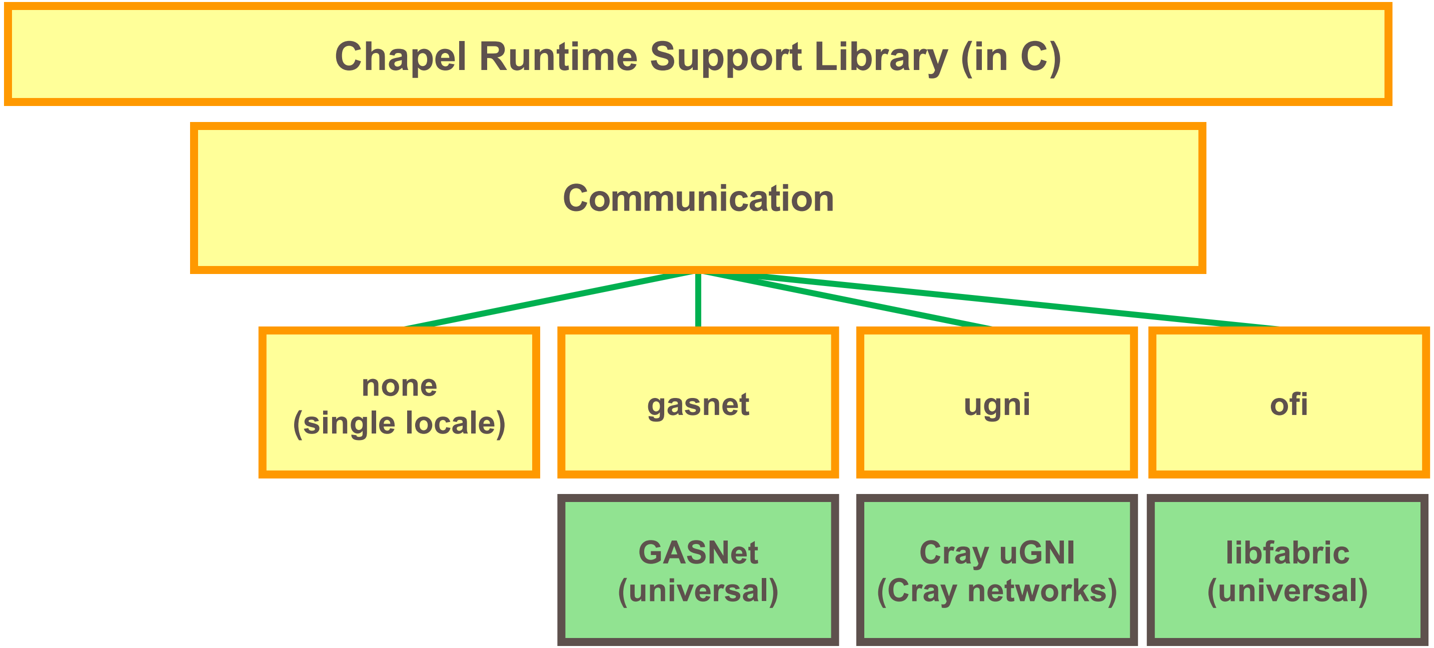
Chapel Compiler Architecture



Chapel Compilation Architecture



Runtime Communication Layer: Communication



Do you want to impose a style on developers?

- Yes?
  - How strict?

```
if (flag) {
  foo();
} else {
  bar();
}
```

```
if (flag)
{
    foo();
}
else
{
  bar();
}
```

```
if (flag)
  foo();
else
  bar();
```

Do you want to impose a style on developers?

- Yes?
  - How strict?

```
if (isConst(expr)) { … }
if (isConst(expr) == true) { … }
if ((isConst(expr) == true) == true) { … }  // ???
```

Do you want to impose a style on developers?

- Yes?
  - How strict?
- No?
  - “Let each developer’s style serve as their handwriting”
  - Can use code review to squash bad habits, develop common sense of taste

What about code reviews?

- Can you afford the time to do them?
- Can you afford not to?
- What is the intention / what do you expect to gain from it?

What level of code documentation will you require?

- Comment every file / routine / variable / code block?
- Write self-documenting code?
- Document in commit / merge comments?

Do you want to develop open- or closed-source?

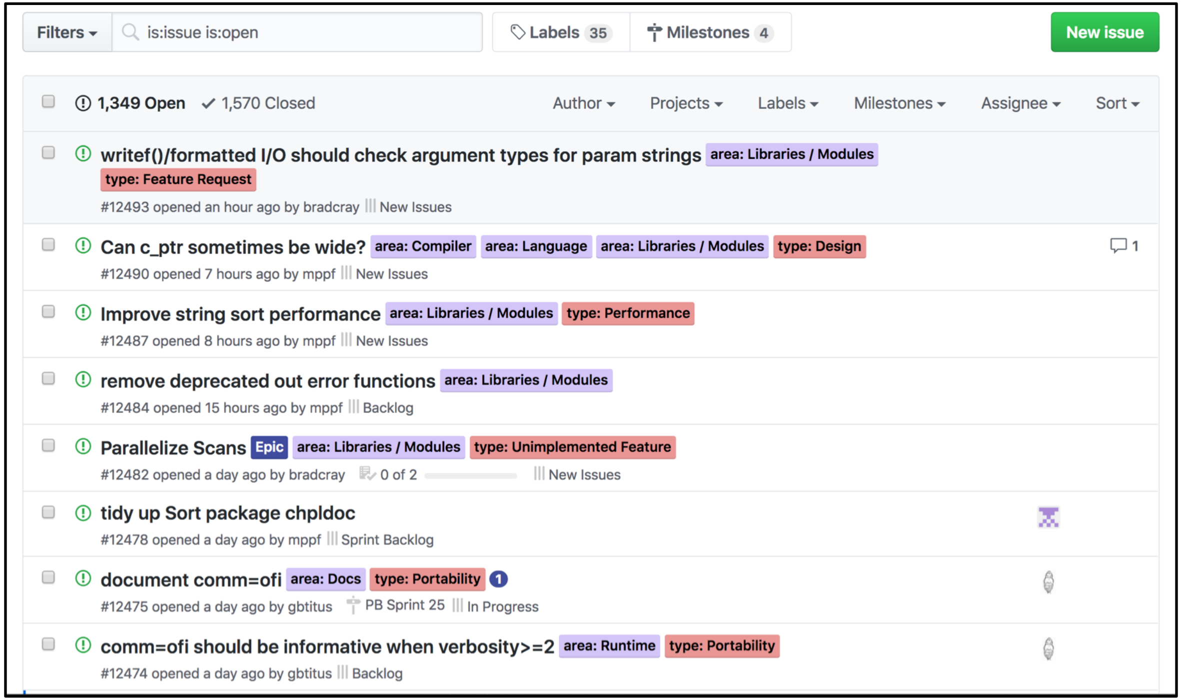
- (even if ultimate goal is to open-source…)
- Potential advantages to open-source:
  - leverage open-source community, developers, and code (?)
  - get immediate and continual feedback
  - “seems like the right thing to do”
- Potential disadvantages:
  - means living with your warts showing
  - no such thing as a free lunch

Source Control Management?

- “Yes” is the only reasonable answer
- Historically, Chapel has used:
  - CVS
  - SVN
  - Git
- Git familiarity is perhaps the most valuable SW dev skill after programming
  - as well as GitHub or GitLab (hosting sites that support git repositories)

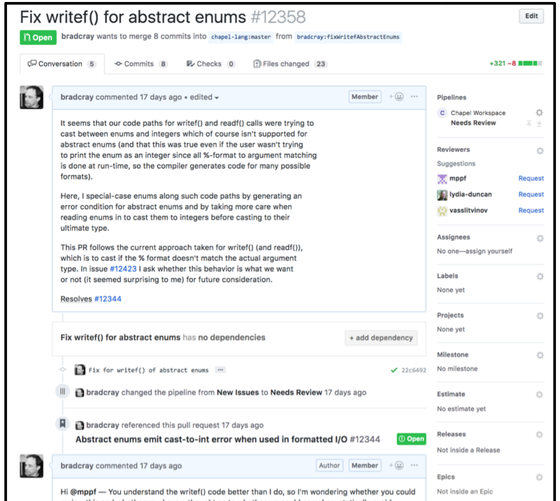
Chapel's use of GitHub

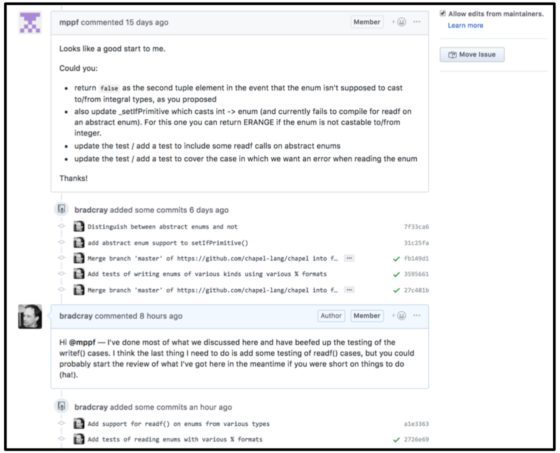
- use GitHub issues to track bugs, feature requests, stories, tasks, epics



Chapel’s use of GitHub

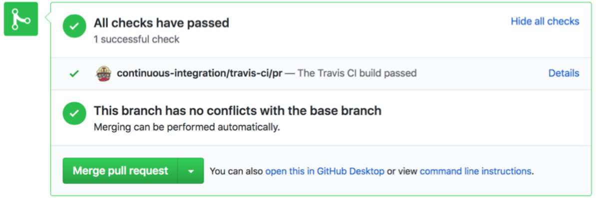
- use GitHub issues to track bugs, feature requests, tasks, stories
- submit proposed changes as pull requests (PRs)
  - must be reviewed by core developer (someone who takes turns doing triage)
    - use GitHub comments/reviews to give feedback
    - if reviewer / reviewee can’t agree, escalate to the group for more opinions





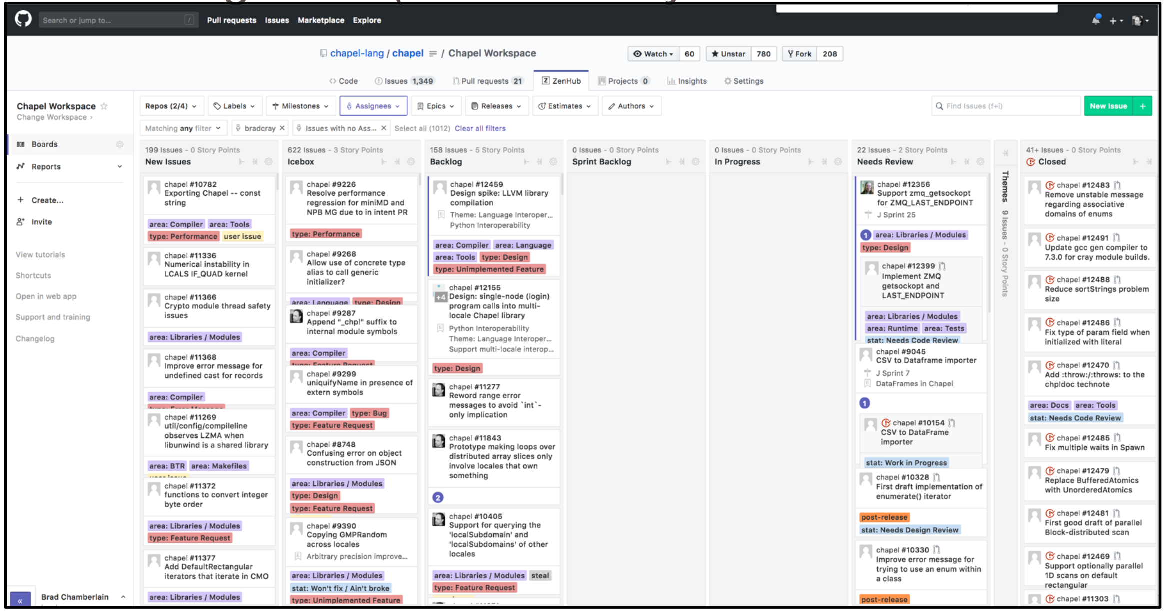
Chapel's use of GitHub

- use GitHub issues to track bugs, feature requests, tasks, stories
- submit proposed changes as pull requests (PRs)
  - must be reviewed by core developer (someone who takes turns doing triage)
    - use GitHub comments/reviews to give feedback
    - if reviewer / reviewee can't agree, escalate to the group for more opinions
  - must also pass testing
    - at minimum, a complete run of linux testing
    - optionally, other configurations as considered valuable
    - Travis testing run automatically on each PR
  - key smoke tests also run post-merge



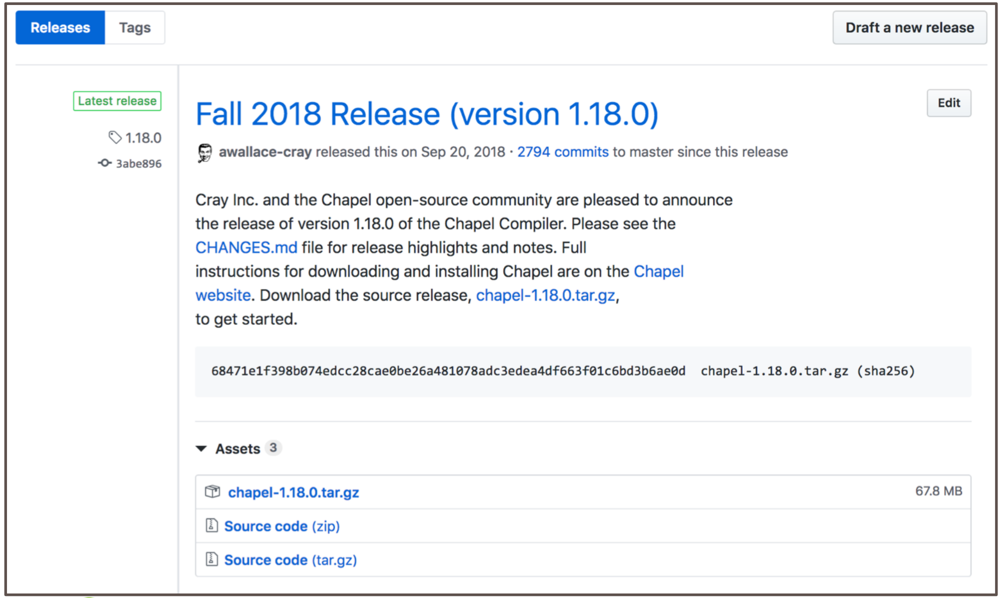
Chapel’s use of GitHub

- use GitHub issues to track bugs, feature requests, tasks, stories
- submit proposed changes as pull requests (PRs)
- use ZenHub for tracking tasks (a Kanban-style board for tasks / stories)



Chapel’s use of GitHub

- use GitHub issues to track bugs, feature requests, tasks, stories
- submit proposed changes as pull requests (PRs)
- use ZenHub for tracking tasks (a Kanban-style board for tasks / stories)
- releases hosted on GitHub as well



Testing: Our key to sanity

- homegrown system
- crawls directory structure looking for things to test
  - simplest form:
    - `hello.chpl` # source file
    - `hello.good` # expected output of compilation + execution steps
  - extended form:
    - additional files to specify:
      - command-line options for compiler and executable
      - actions to take before compiling, running, diffing, …
      - etc.

What do we run? (Correctness testing)

- 9500+ tests

  x back-end compilers (gnu, clang, llvm, icc, cce, pgi)

  x platforms (Linux, Mac OS X, Crays, Cygwin, …)

  x processor types (x86, arm, knl, …)

  x machine models (flat, numa)

  x tasking options (fifo, qthreads)

  x options for communication (local, gasnet, ugni, libfabric, …)

  x build options (quickstart, preferred, valgrind, …)

  x compiler options (normal, --fast, --baseline, --verify, …)

  x …

What do we run? (Correctness testing)

- 9500+ tests …or 300+ (release examples only) or ~6 ("hellos")

  x back-end compilers (gnu, clang, llvm, icc, cce, pgi)

  x platforms (Linux, Mac OS X, Crays, Cygwin, …)

  x processor types (x86, arm, knl, …)

  x machine models (flat, numa)

  x tasking options (fifo, qthreads)

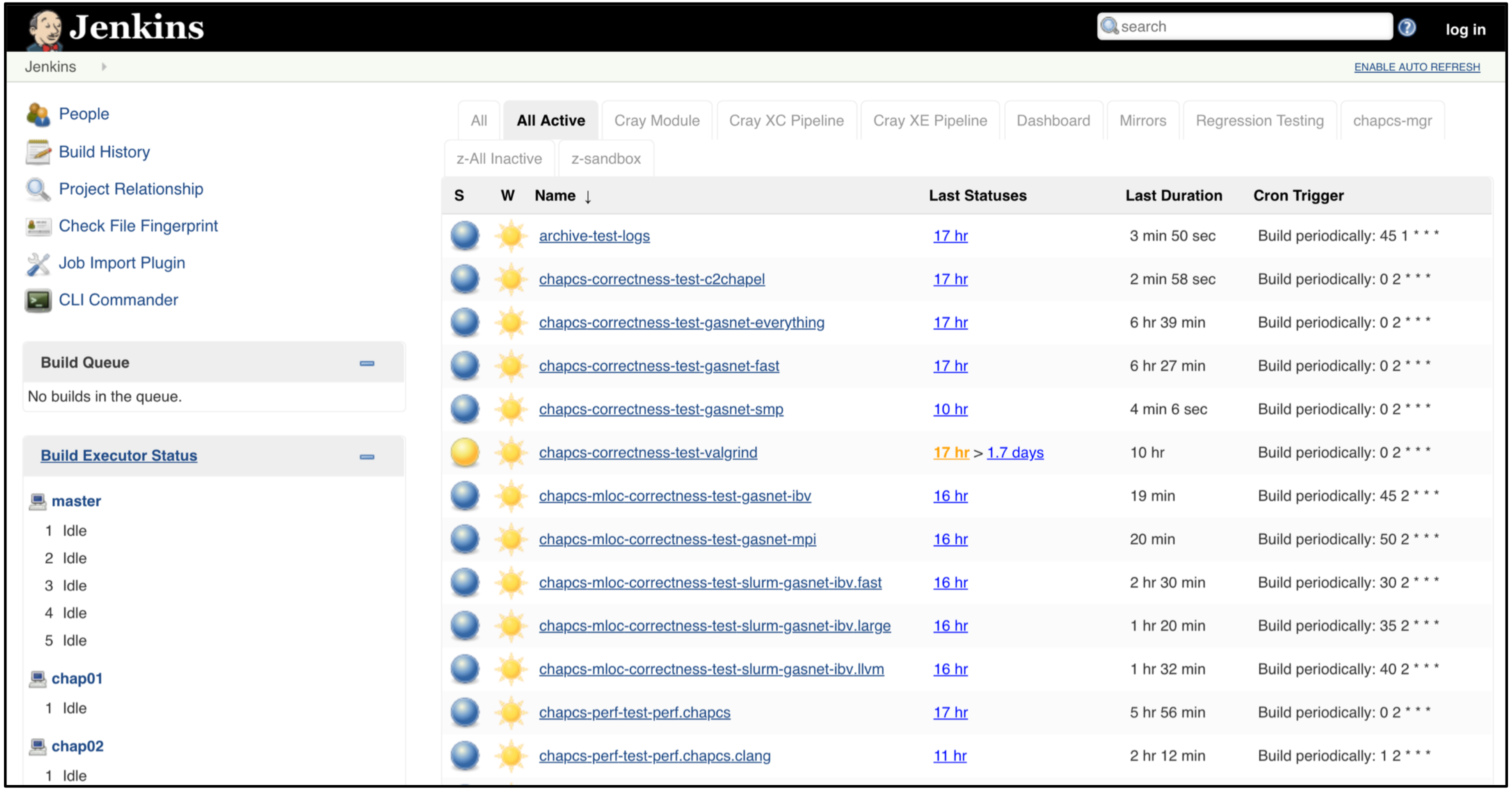
  x options for communication (local, gasnet, ugni, libfabric, …)

  x build options (quickstart, preferred, valgrind, …)

  x compiler options (normal, --fast, --baseline, --verify, …)

  x …

Testing managed through Jenkins



Jenkins

search

log in

Jenkins

ENABLE AUTO REFRESH

- People
- Build History
- Project Relationship
- Check File Fingerprint
- Job Import Plugin
- CLI Commander

Build Queue

No builds in the queue.

Build Executor Status

master

1 Idle
2 Idle
3 Idle
4 Idle
5 Idle

chap01

1 Idle

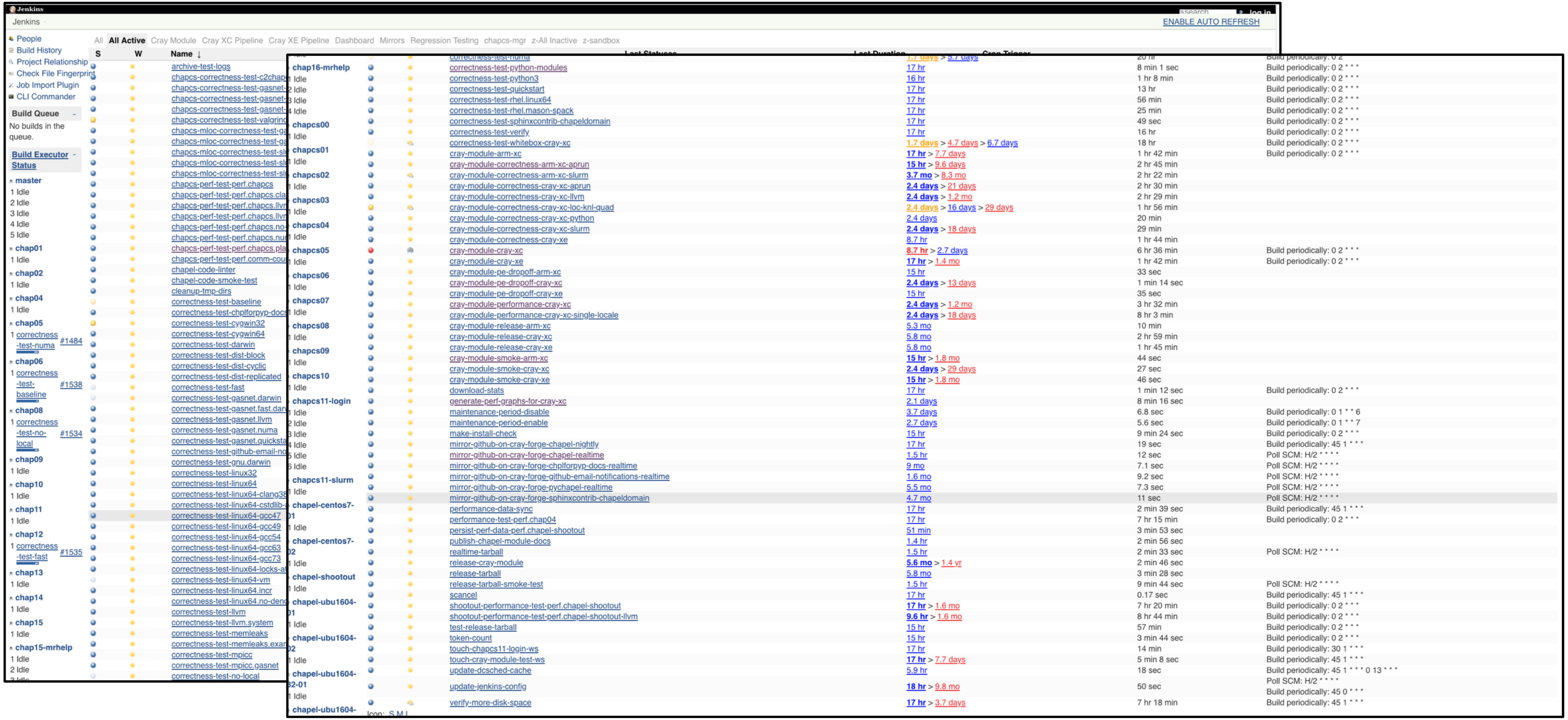
chap02

1 Idle

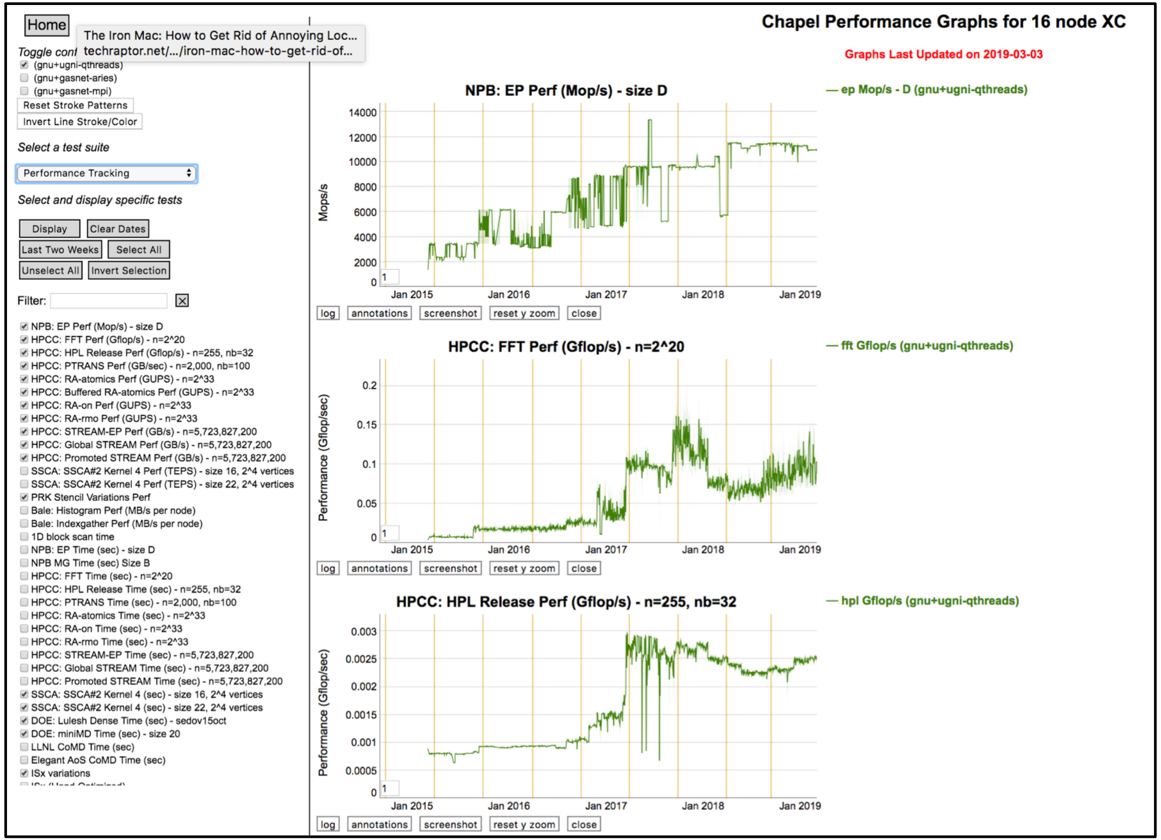
All | **All Active** | Cray Module | Cray XC Pipeline | Cray XE Pipeline | Dashboard | Mirrors | Regression Testing | chapcs-mgr | z-All Inactive | z-sandbox

S	W	Name ↓	Last Statuses	Last Duration	Cron Trigger
		archive-test-logs	17 hr	3 min 50 sec	Build periodically: 45 1 * * *
		chapcs-correctness-test-c2chapel	17 hr	2 min 58 sec	Build periodically: 0 2 * * *
		chapcs-correctness-test-gasnet-everything	17 hr	6 hr 39 min	Build periodically: 0 2 * * *
		chapcs-correctness-test-gasnet-fast	17 hr	6 hr 27 min	Build periodically: 0 2 * * *
		chapcs-correctness-test-gasnet-smp	10 hr	4 min 6 sec	Build periodically: 0 2 * * *
		chapcs-correctness-test-valgrind	17 hr > 1.7 days	10 hr	Build periodically: 0 2 * * *
		chapcs-mloc-correctness-test-gasnet-ibv	16 hr	19 min	Build periodically: 45 2 * * *
		chapcs-mloc-correctness-test-gasnet-mpi	16 hr	20 min	Build periodically: 50 2 * * *
		chapcs-mloc-correctness-test-slurm-gasnet-ibv.fast	16 hr	2 hr 30 min	Build periodically: 30 2 * * *
		chapcs-mloc-correctness-test-slurm-gasnet-ibv.large	16 hr	1 hr 20 min	Build periodically: 35 2 * * *
		chapcs-mloc-correctness-test-slurm-gasnet-ibv.llvm	16 hr	1 hr 32 min	Build periodically: 40 2 * * *
		chapcs-perf-test-perf.chapcs	17 hr	5 hr 56 min	Build periodically: 0 2 * * *
		chapcs-perf-test-perf.chapcs.clang	11 hr	2 hr 12 min	Build periodically: 1 2 * * *

Testing managed through Jenkins



Also Performance Tests



Current gaps in our testing

- no unit testing
  - all tests are end-to-end runs of the compiler
  - (happily, compilers are more amenable to this than some types of software)
- no testing of examples in code-based documentation
  - language specification is (mostly) tested
  - wishlist: something that would test code in my Powerpoint slides…
- no "fuzzing" / random testing
  - an important way to simulate novice users?

Parting Thoughts

Technical Choices

- Find / create ways to eat your own dogfood
- Create tools to help yourself
  - Particularly with repetitive tasks, recurring pain points
- Use existing techniques and technologies when available and appropriate
  - “Why waste an hour in the library when you can spend a month in the lab?”

Healthy Attitudes for the Tech World

- Be comfortable with uncertainty, imperfection, changes—they're bound to occur
  - Don't be afraid to rewrite code
  - Don't be overly protective of code that you've written
- Don't be a bean counter (at least about unimportant beans)
  - true problems have a tendency to make themselves known
- Find ways to make your process fun for yourself / your team

Who you are matters a ton

- Don’t be a jerk
    - If you are a jerk, fake it until you’re not
- It’s truly a small world
- Being capable is so much more important than seniority, expertise, …

One more

- Don’t expect that you’ll remember everything forever (you won’t)

  => Create notes, documentation, comments for yourself as much as anyone

Tips from my team

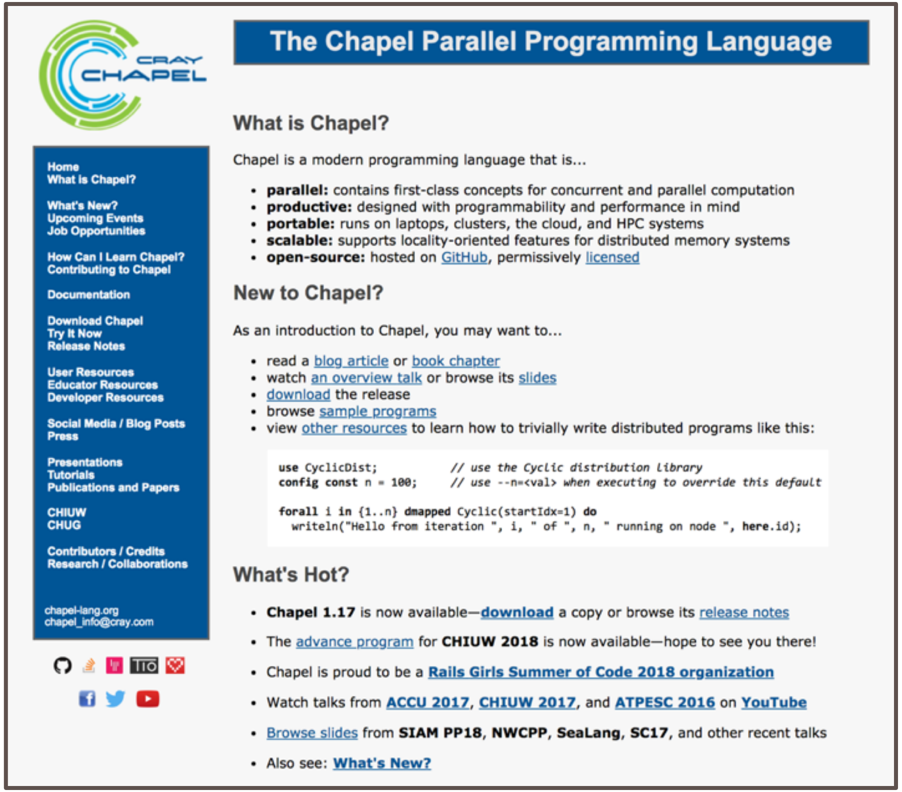
- Learn how to break problems into smaller subcomponents
  - easier to estimate level of effort required
  - easier to determine edge cases, avoid backtracking

Chapel Resources

Chapel Central

https://chapel-lang.org

- downloads
- presentations
- papers
- resources
- documentation



The Chapel Parallel Programming Language

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

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?**

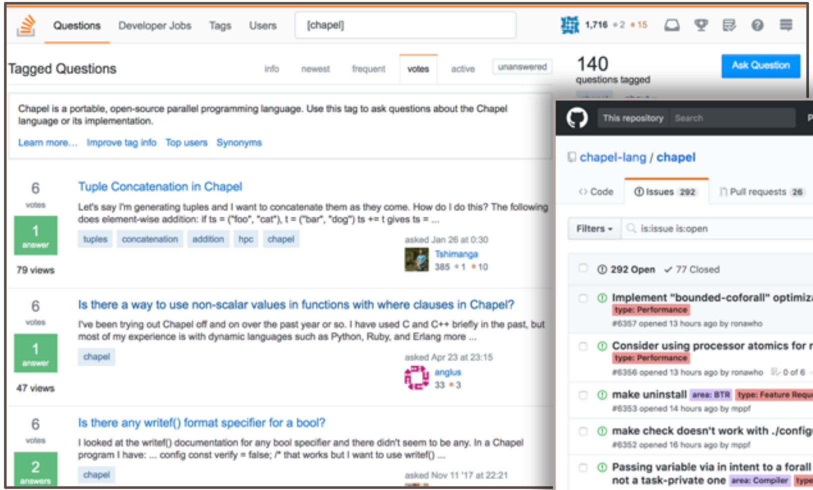
Chapel Social Media (no account required)

http://twitter.com/ChapelLanguage

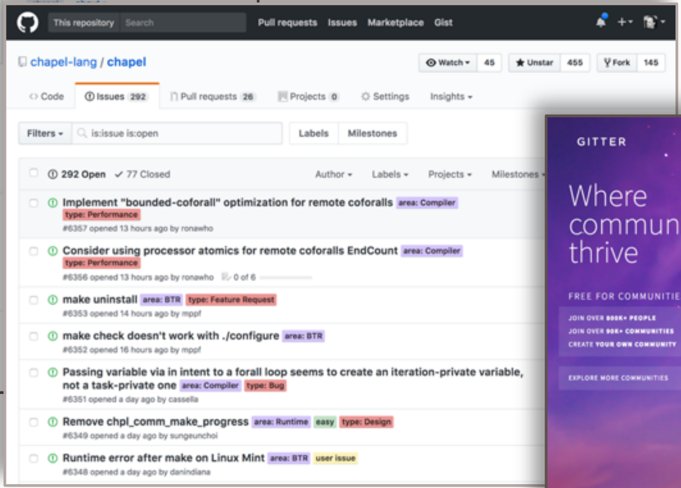
http://facebook.com/ChapelLanguage

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

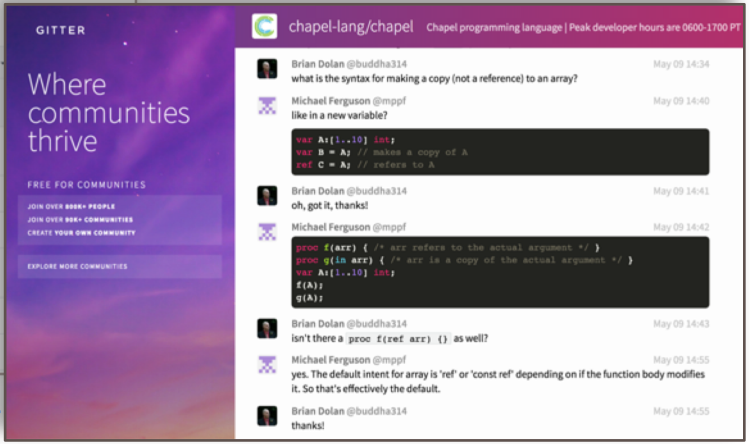
Chapel Community



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



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



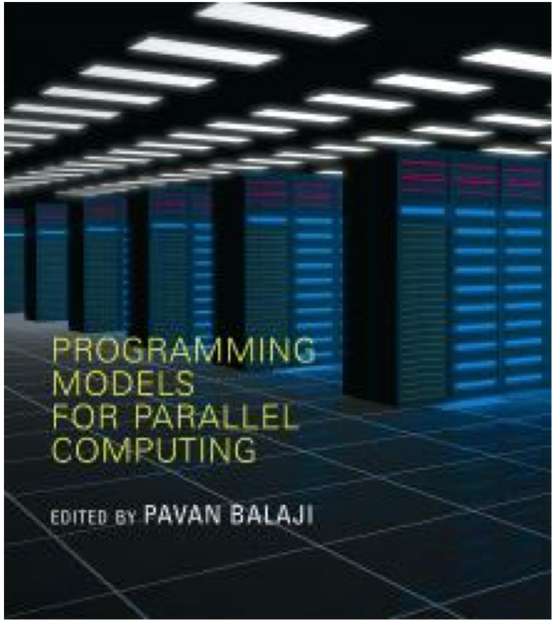
https://gitter.im/chapel-lang/chapel

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

Suggested Reading: Chapel history and overview

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



Suggested Reading: Recent Progress (CUG 2018)

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, Vassily Litvinov, Preston Sahabu, and Greg Titus
Chapel 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 in terms of portability, and with C++ in terms of flexibility and extensibility. Chapel is designed to be general-purpose in the sense that when you have a parallel algorithm in mind and a parallel system on which you wish to run it, Chapel should be able to handle that scenario.

Chapel's design and implementation are led by Cray Inc. with feedback and code contributed by users and the open-source community. Though developed by Cray, Chapel's design and implementation are portable, permitting its programs to scale up from multicore laptops to commodity clusters to Cray systems. In addition, Chapel programs can be run on cloud-computing platforms and HPC systems from other vendors. Chapel is being developed in an open-source manner under the Apache 2.0 license and is hosted at GitHub.[1]

[1] https://github.com/chapel-lang/chapel

The development of the Chapel language was undertaken by Cray Inc. as part of its participation in the DARPA High Productivity Computing Systems program (HPCS). HPCS wrapped up in late 2012, at which point Chapel was a compelling prototype, having successfully demonstrated several key research challenges that the project had undertaken. Chief among these was supporting data- and task-parallelism in a unified manner within a single language. This was accomplished by supporting the creation of high-level data-parallel abstractions like parallel loops and arrays in terms of lower-level Chapel features such as classes, iterators, and tasks.

Under HPCS, Chapel also successfully supported the expression of parallelism using distinct language features from those used to control locality and affinity—that is, Chapel programmers specify *which* computations should run in parallel distinctly from specifying *where* those computations should be run. This permits Chapel programs to support multicore, multi-node, and heterogeneous computing within a single unified language.

Chapel's implementation under HPCS demonstrated that the language could be implemented portably while still being optimized for HPC-specific features such as the RDMA support available in Cray® Gemini™ and Aries™ networks. This allows Chapel to take advantage of native hardware support for remote puts, gets, and atomic memory operations.

Despite these successes, at the close of HPCS, Chapel was not at all ready to support production codes in the field. This was not surprising given the language's aggressive design and modest-sized research team. However, reactions from potential users were sufficiently positive that, in early 2013, Cray embarked on a follow-up effort to improve Chapel and move it towards being a production-ready language. Colloquially, we refer to this effort as "the five-year push."

This paper's contribution is to describe the results of this five-year effort, providing readers with an understanding of Chapel's progress and achievements since the end of the HPCS program. In doing so, we directly compare the status of Chapel version 1.17, released last month, with Chapel version 1.7, which was released five years ago in April 2013.

paper and slides available at chapel-lang.org

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.

THANK YOU

QUESTIONS?

bradc@cray.com
@ChapelLanguage
chapel-lang.org

cray.com
@cray_inc
linkedin.com/company/cray-inc-