

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



CRAY®



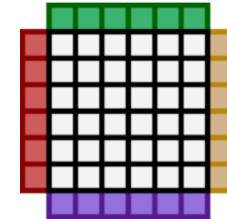
Who am I?

CRAY

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:

CRAY

- 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

CRAY



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

Piz Daint: One of Today's Most Powerful Supercomputers

CRAY

Model Cray XC40/Cray XC50

Number of Hybrid Compute Nodes	5 704
Number of Multicore Compute Nodes	1 431
Peak Floating-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
Multicore Peak Performance	1.731 Petaflops
Hybrid Memory Capacity per Node	64 GB; 16 GB CoWoS HBM2
Multicore Memory Capacity per Node	64 GB, 128 GB
Total System Memory	437.9 TB; 83.1 TB
System Interconnect	Cray Aries routing and communications ASIC, and Dragonfly network topology
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 Theoretical 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?

CRAY

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

CRAY

Chapel aims to be as...

...**programmable** as Python

...**fast** as Fortran

...**scalable** as MPI, SHMEM, or UPC

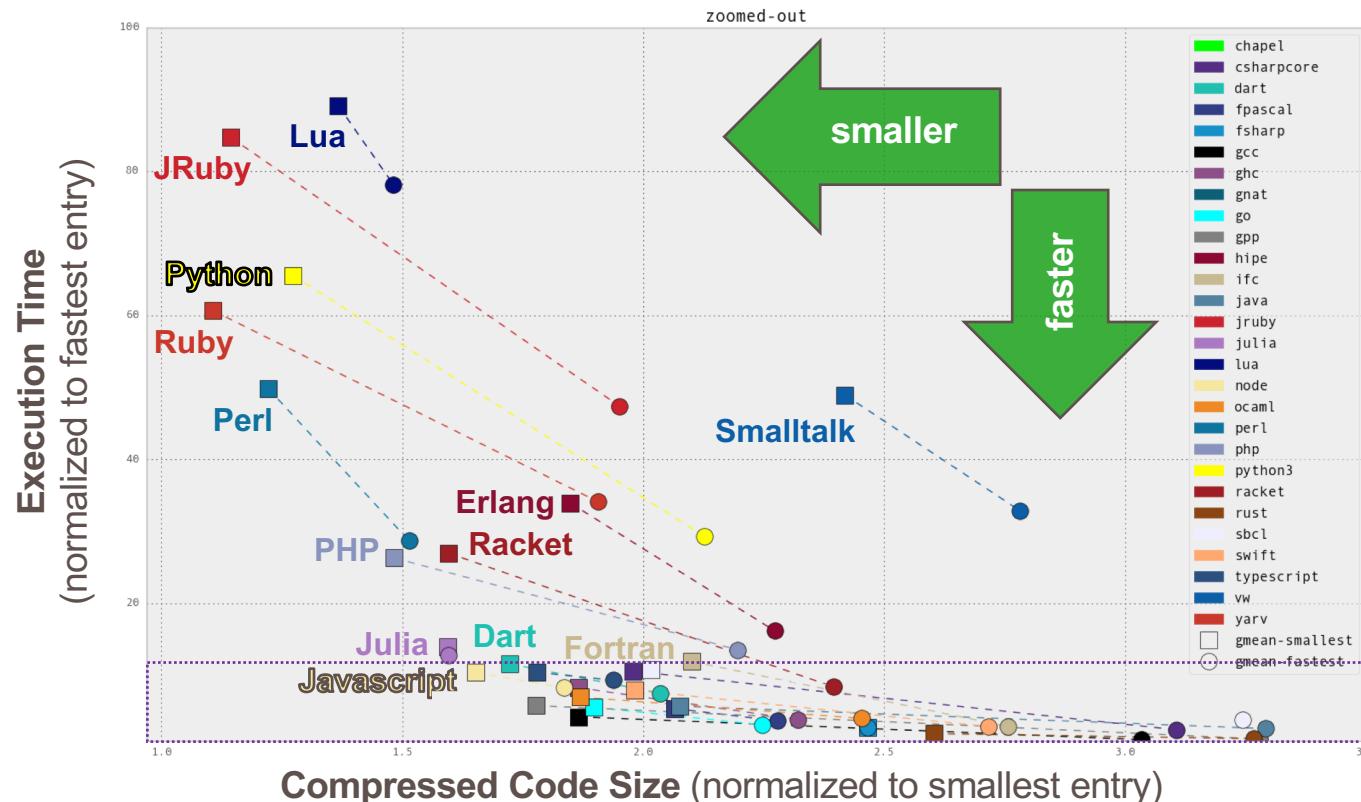
...**portable** as C

...**flexible** as C++

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

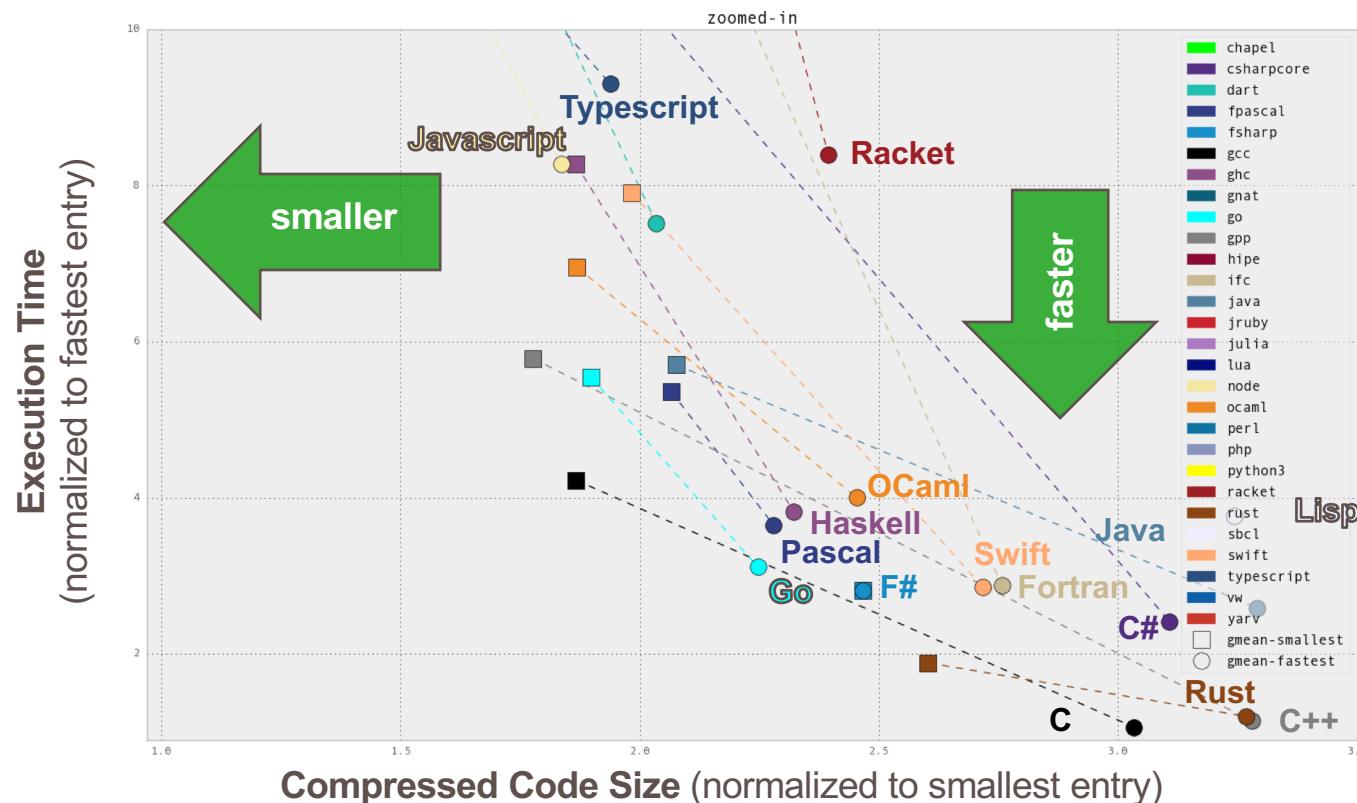
CLBG Cross-Language Summary (Dec 18, 2018)

CRAY



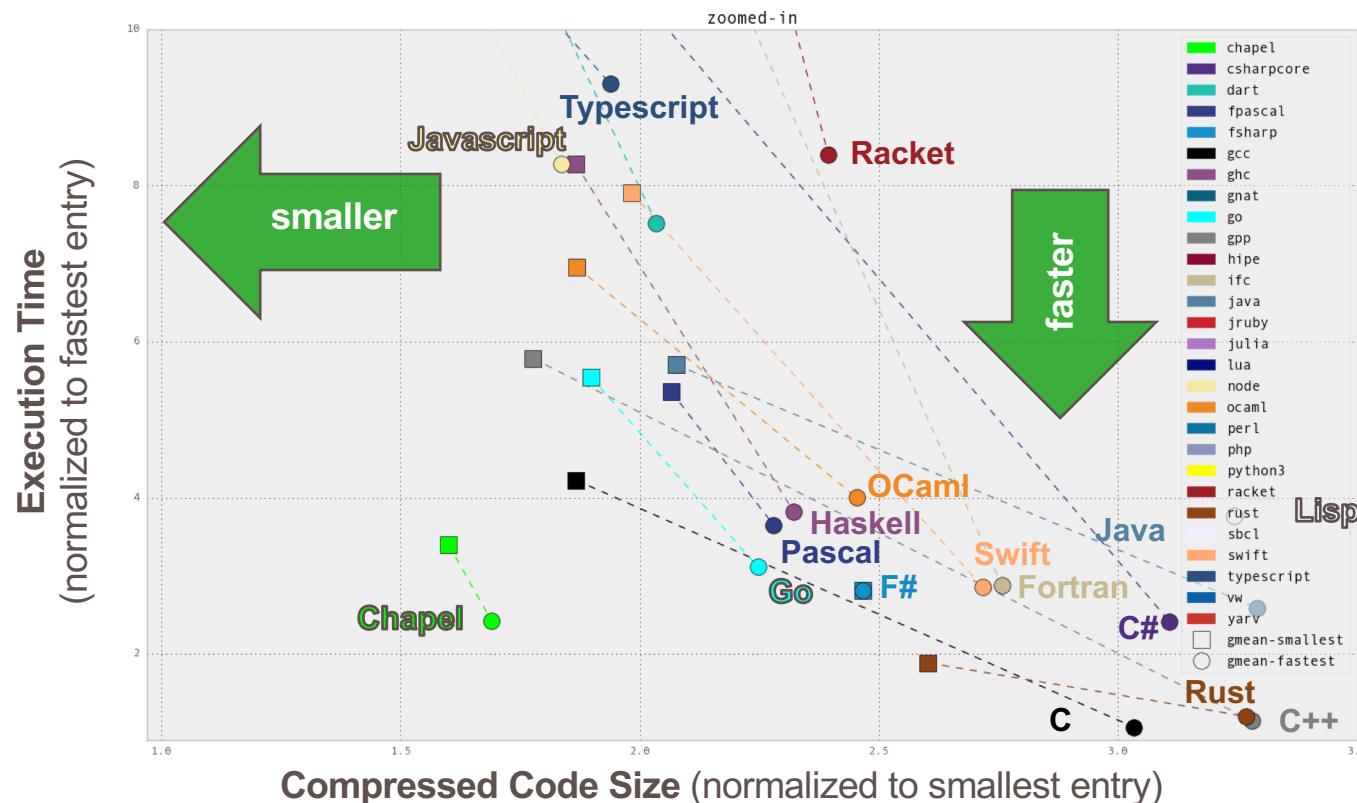
CLBG Cross-Language Summary (Dec 18, 2018, zoomed)

CRAY



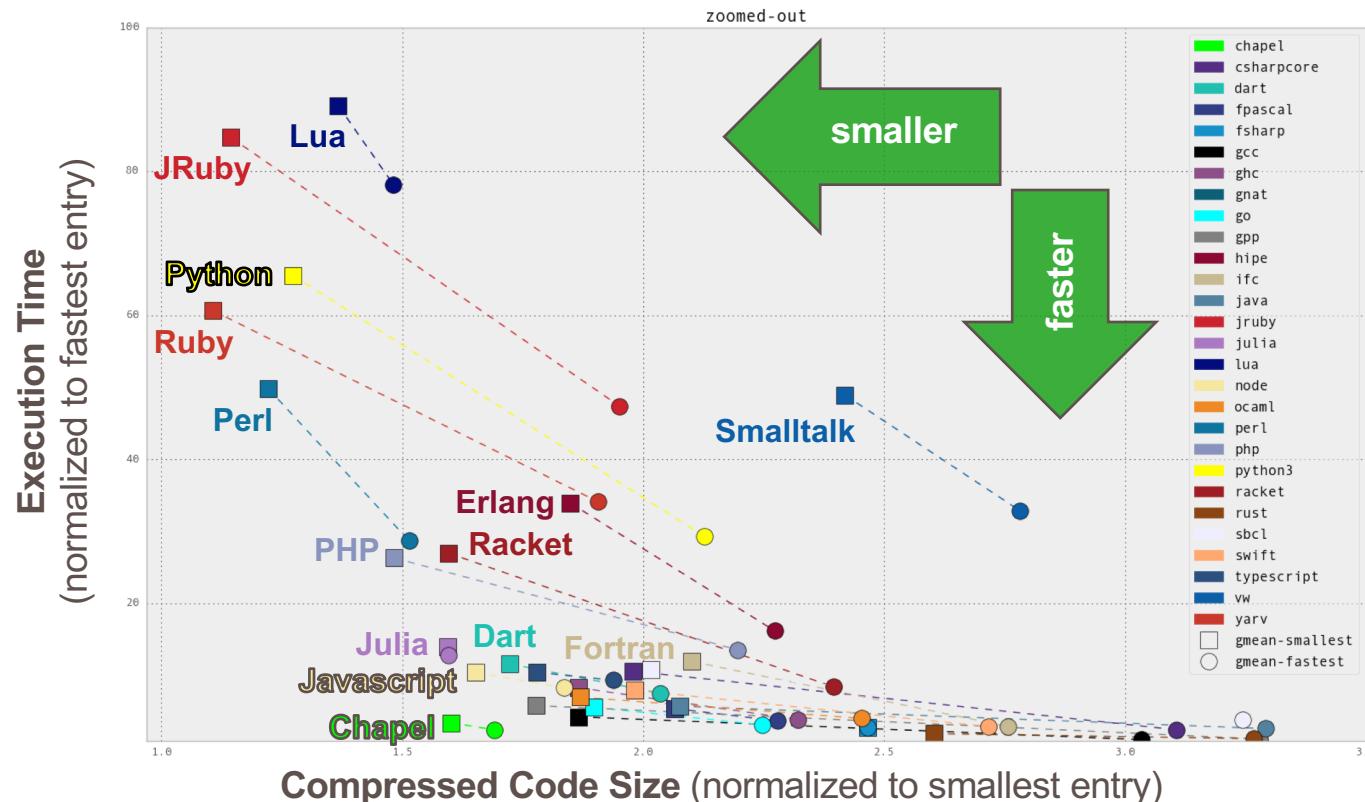
CLBG Cross-Language Summary (Dec 18, 2018, zoomed)

CRAY



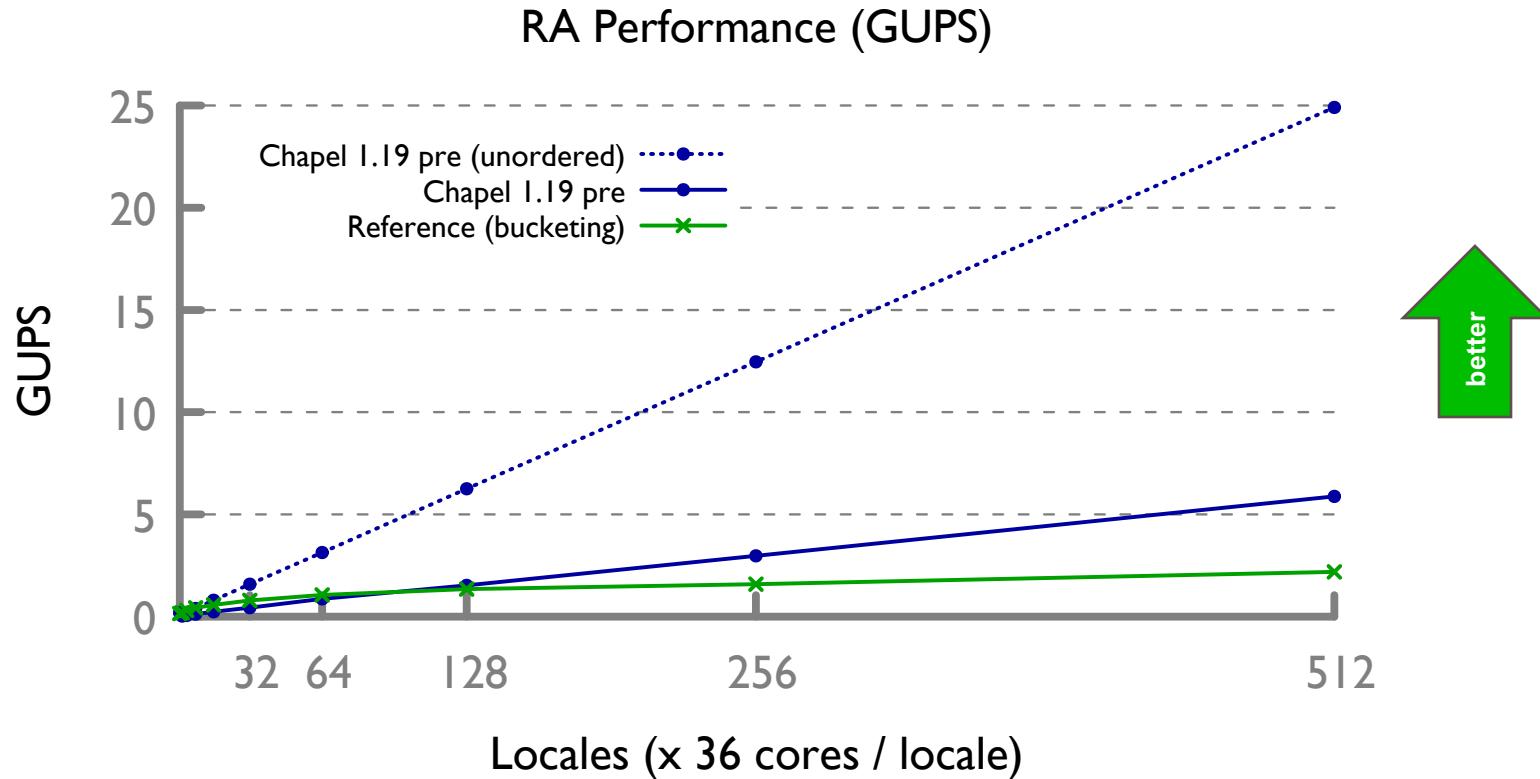
CLBG Cross-Language Summary (Dec 18, 2018)

CRAY



HPCC RA: buffering vs. network atomics

CRAY



HPCC RA: MPI kernel

CRAY

```

/* 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_NUL;
        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_NUL;
    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_NUL; 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

CRAY

```

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

MPI_Irecv(&localRecvBuffer, localBufferSize, tparams.tdata,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD);
while (i < sendCnt) {
    /* receive messages */
    do {
        MPI_Test(&inreq, &have_done, &status);
        if (have_done) {
            if (status.MPI_TAG == UPDTE_TAG) {
                MPI_Get_count(&status, tparams.dtype64,
                             bufferBase = 0;

```

Chapel Kernel

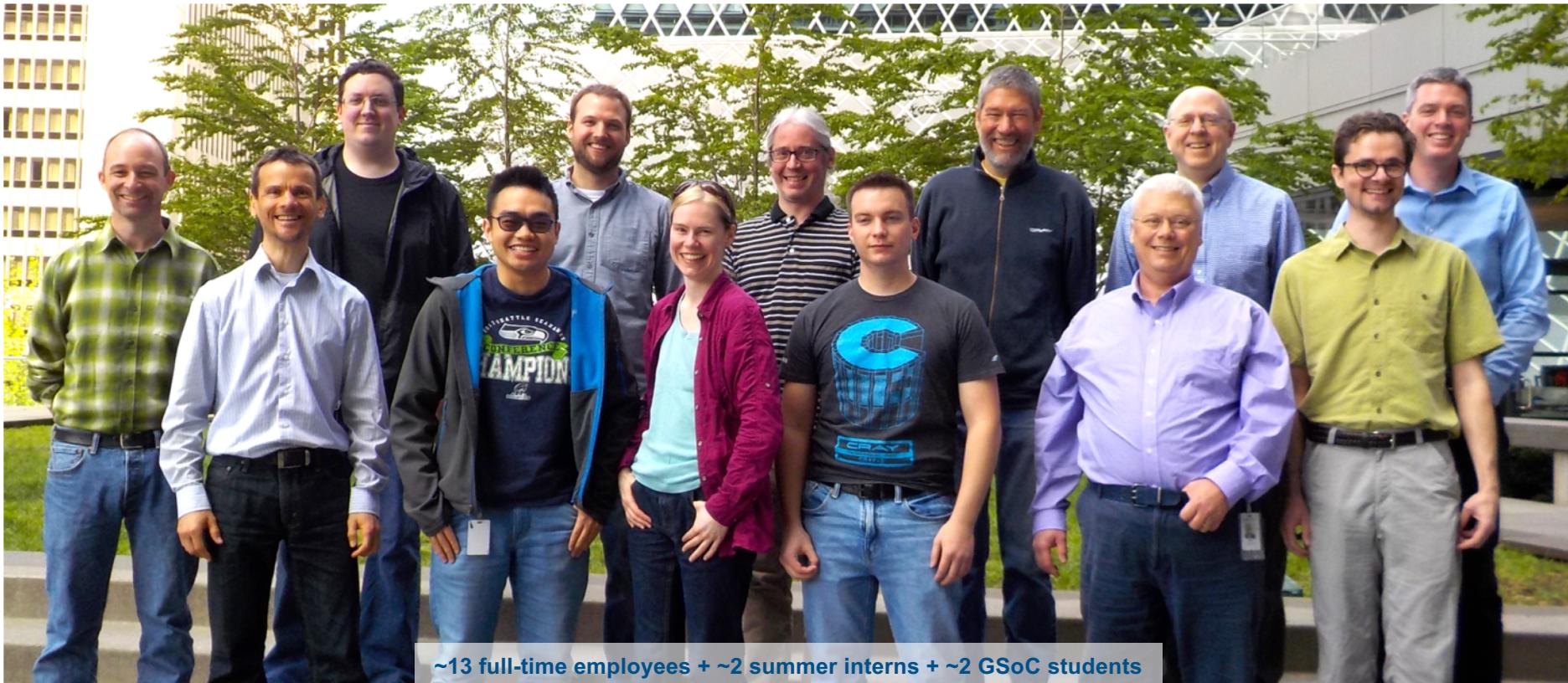
```
forall (_ , r) in zip(Updates, RASTream()) do
    T[r & indexMask].xor(r);
```

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 & (TABSIZ-1)] ^= Ran;  
*      }  
*/
```

The Chapel Team at Cray (May 2018)

CRAY



Software Engineering & Chapel



Disclaimers

CRAY

- 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

CRAY

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

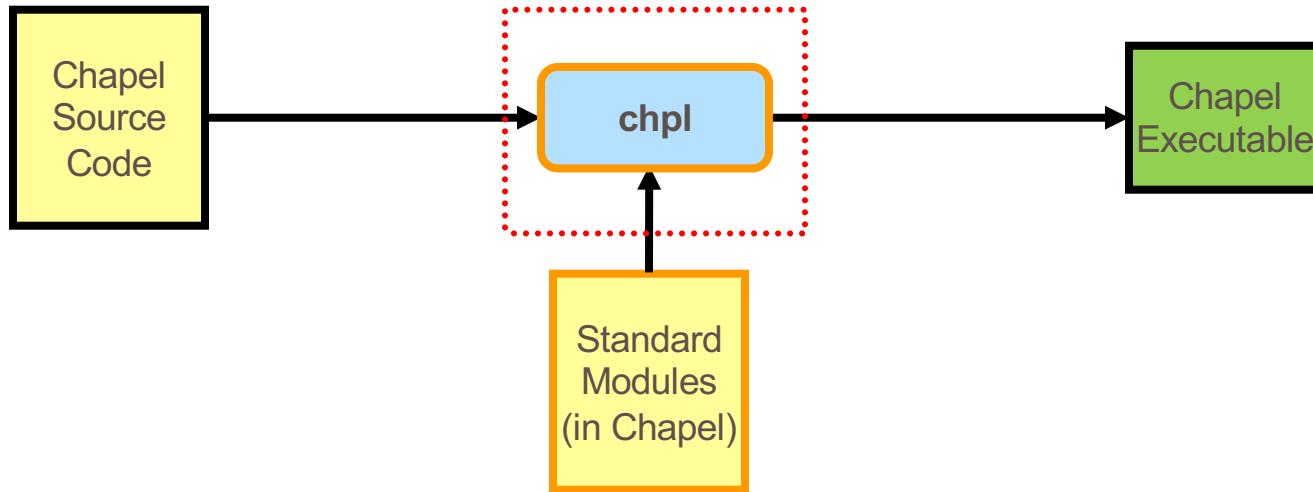
What did we set out to create?

CRAY

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

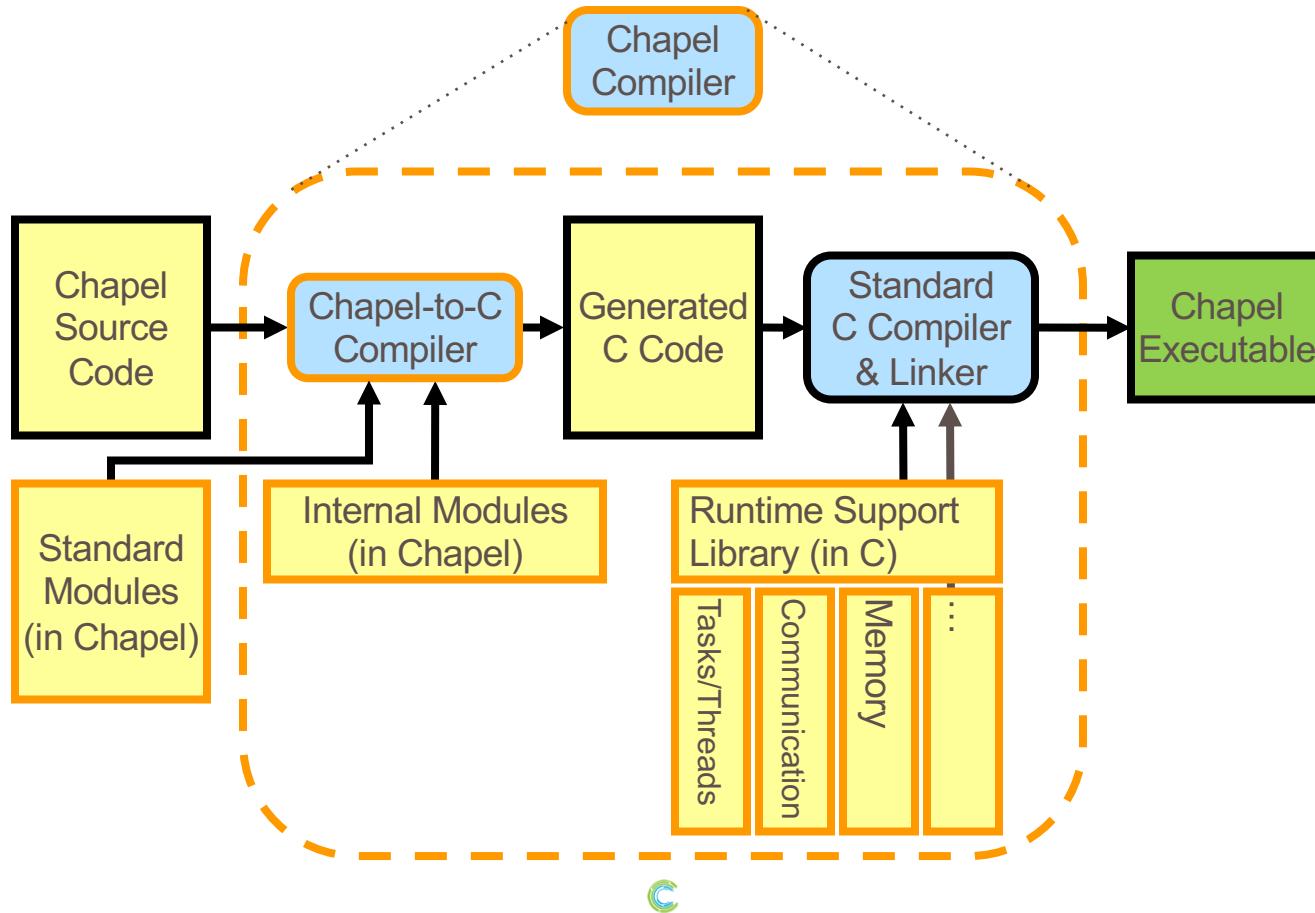
Compiling Chapel

CRAY



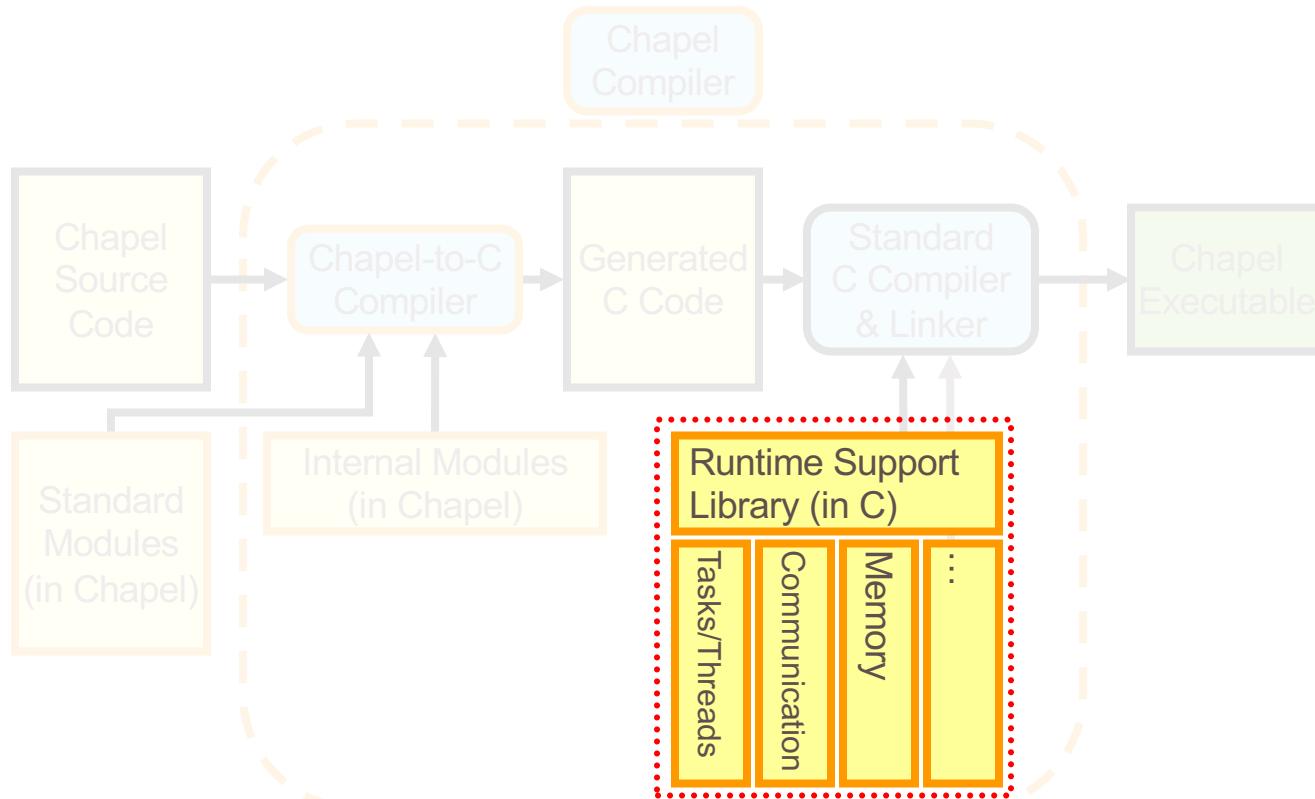
Chapel Compiler Architecture

CRAY



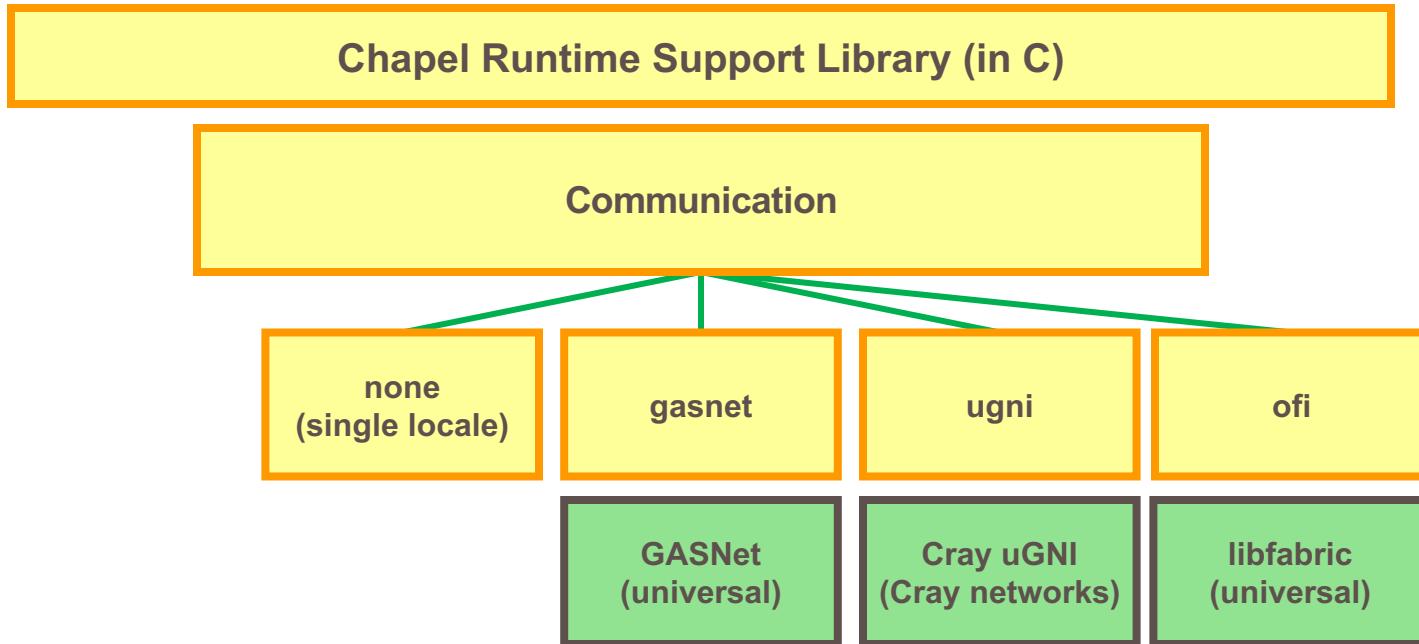
Chapel Compilation Architecture

CRAY



Runtime Communication Layer: Communication

CRAY



Do you want to impose a style on developers?

CRAY

- Yes?

- How strict?

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

Do you want to impose a style on developers?

CRAY

- Yes?
 - How strict?

```
if (isConst(expr)) { ... }
```

```
if (isConst(expr) == true) { ... }
```

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

Do you want to impose a style on developers?

CRAY

- 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?

CRAY

- 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?

CRAY

- (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?

CRAY

- “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

CRAY

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

The screenshot shows a GitHub search results page with the query "is:issue is:open". The results list several open issues for the Chapel project:

- #12493**: writef() / formatted I/O should check argument types for param strings (area: Libraries / Modules, type: Feature Request)
- #12490**: Can c_ptr sometimes be wide? (area: Compiler, area: Language, area: Libraries / Modules, type: Design)
- #12487**: Improve string sort performance (area: Libraries / Modules, type: Performance)
- #12484**: remove deprecated out error functions (area: Libraries / Modules)
- #12482**: Parallelize Scans (Epic, area: Libraries / Modules, type: Unimplemented Feature)
- #12478**: tidy up Sort package chpldoc
- #12475**: document comm=ofi (area: Docs, type: Portability)
- #12474**: comm=ofi should be informative when verbosity>=2 (area: Runtime, type: Portability)

Each issue card includes the issue number, title, labels, and a brief description. The interface shows standard GitHub navigation and filtering options at the top.

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

The image contains two side-by-side screenshots of the GitHub web interface.

Screenshot 1: Pull Request #12358

This screenshot shows a pull request titled "Fix writef() for abstract enums". The pull request has 8 commits from user "bradgray". The pipeline status shows "Needs Review". The pull request details include:

- Puller:** bradgray commented 17 days ago + edited
- Reviewers:** mpff, yelias-duncan, vassilivinov
- Assignees:** None
- Labels:** None
- Milestone:** No milestone
- Estimate:** No estimate yet
- Releases:** Not inside a Release
- Epic:** Not inside an Epic

The pull request body contains code snippets and explanatory text about casting between enums and integers.

Screenshot 2: GitHub Comments Thread

This screenshot shows a thread of comments on the pull request:

- mpff:** commented 15 days ago

Looks like a good start to me.

Could you:

 - return FALSE as the second tuple element in the event that the enum isn't supposed to cast to some integer types, as you proposed
 - also update `sehlfPrimitive` which casts int -> enum (and currently fails to compile for `readf` on an abstract enum). For this one you can return ERANGE if the enum is not castable to/from integer.
 - update the test / add a test to include some `readf` calls on abstract enums
 - update the test / add a test to cover the case in which we want an error when reading the enum type.

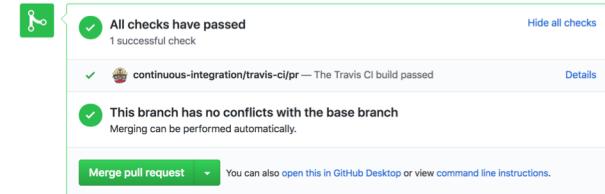
Thanks!
- bradgray:** added some commits 6 days ago
 - Distinguish between abstract enums and not
 - Add abstract enum support to `sehlfPrimitive()`
 - Merge branch 'master' of https://github.com/chapel-lang/chapel into f...
 - Add tests of writing enums of various kinds using various % forests
 - Merge branch 'master' of https://github.com/chapel-lang/chapel into f...
- bradgray:** commented 8 hours ago

Hi @mpff — I've done most of what we discussed here and have beefed up the testing of the `writef()` cases. I think the last thing I need to do is add some testing of `readf()` cases, but you could probably start the review of what I've got here in the meantime if you were short on things to do (hat).
- bradgray:** added some commits an hour ago
 - Add support for `readf()` on enums from various types
 - Add tests of reading enums with various % formats

Chapel's use of GitHub

CRAY

- 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)

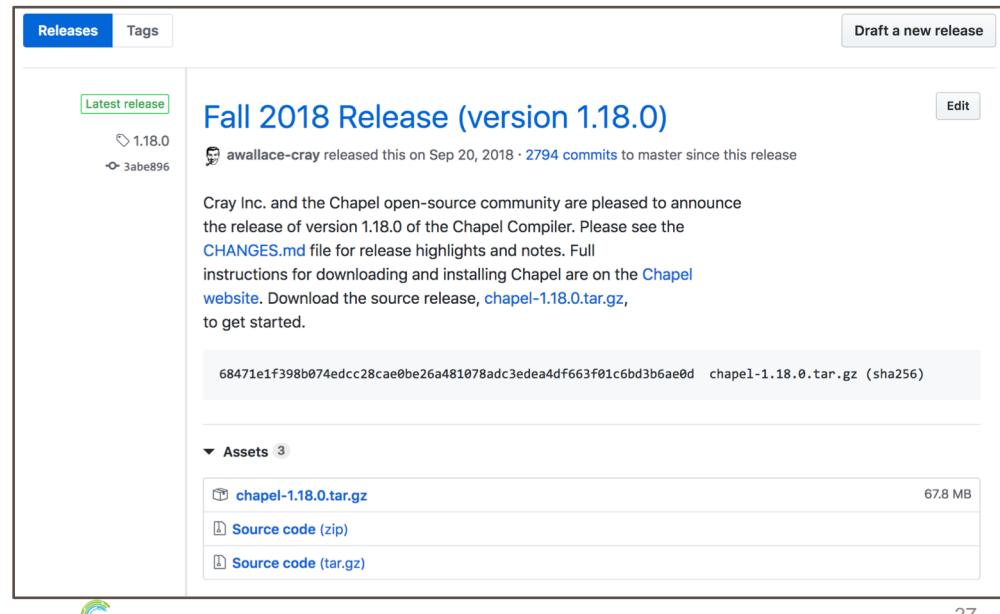
The screenshot shows the GitHub workspace for the 'chapel-lang/chapel' repository. The interface is organized into several boards:

- Boards:** A sidebar on the left lists various boards: Boards, Reports, + Create..., Invite, View tutorials, Shortcuts, Open in web app, Support and training, and Changelog.
- Reports:** A section showing 199 issues - 3 Story Points.
- New Issues:** A section showing 622 issues - 3 Story Points.
- Icelbox:** A board showing 168 issues - 5 Story Points. It includes cards for:
 - chapel #9226 Resolve performance regression for minIMD and NPD MG due to in intent PR (area: Compiler, type: Performance, user issue)
 - chapel #9268 Allow use of concrete types to call generic initializers! (area: Libraries / Modules, type: Design, feature request)
 - chapel #9299 Improve error message for undefined cast for records (area: Compiler, type: Bug)
 - chapel #9300 uniquifyName in presence of extern symbols (area: Compiler, type: Bug)
 - chapel #9327 Confusing error on object construction from JSON (area: Libraries / Modules, type: Feature Request)
 - chapel #9390 Copying GMPRandom across locales (area: Libraries / Modules, type: Feature Request)
 - chapel #9377 Add DefaultRectangular iterators that iterate in CMO (area: Libraries / Modules, type: Feature Request)
- Backlog:** A board showing 0 issues - 0 Story Points.
- Sprint Backlog:** A board showing 0 issues - 0 Story Points.
- In Progress:** A board showing 22 issues - 2 Story Points.
- Needs Review:** A board showing 41 issues - 0 Story Points.
- Closed:** A board showing 41 issues - 0 Story Points.

Each card displays the issue number, title, assignee, labels (e.g., area: Compiler, type: Bug), and a brief description. The interface also includes a search bar, filters, and navigation buttons at the top.

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

CRAY

- 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)

CRAY

- 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)

CRAY

- 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

CRAY

Jenkins

All Active Cray Module Cray XC Pipeline Cray XE Pipeline Dashboard Mirrors Regression Testing chapcs-mgr

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-lbv	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-lbv.fast	16 hr	2 hr 30 min	Build periodically: 30 2 * * *
		chapcs-mloc-correctness-test-slurm-gasnet-lbv.large	16 hr	1 hr 20 min	Build periodically: 35 2 * * *
		chapcs-mloc-correctness-test-slurm-gasnet-lbv.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 * * *

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

Build Queue No builds in the queue.

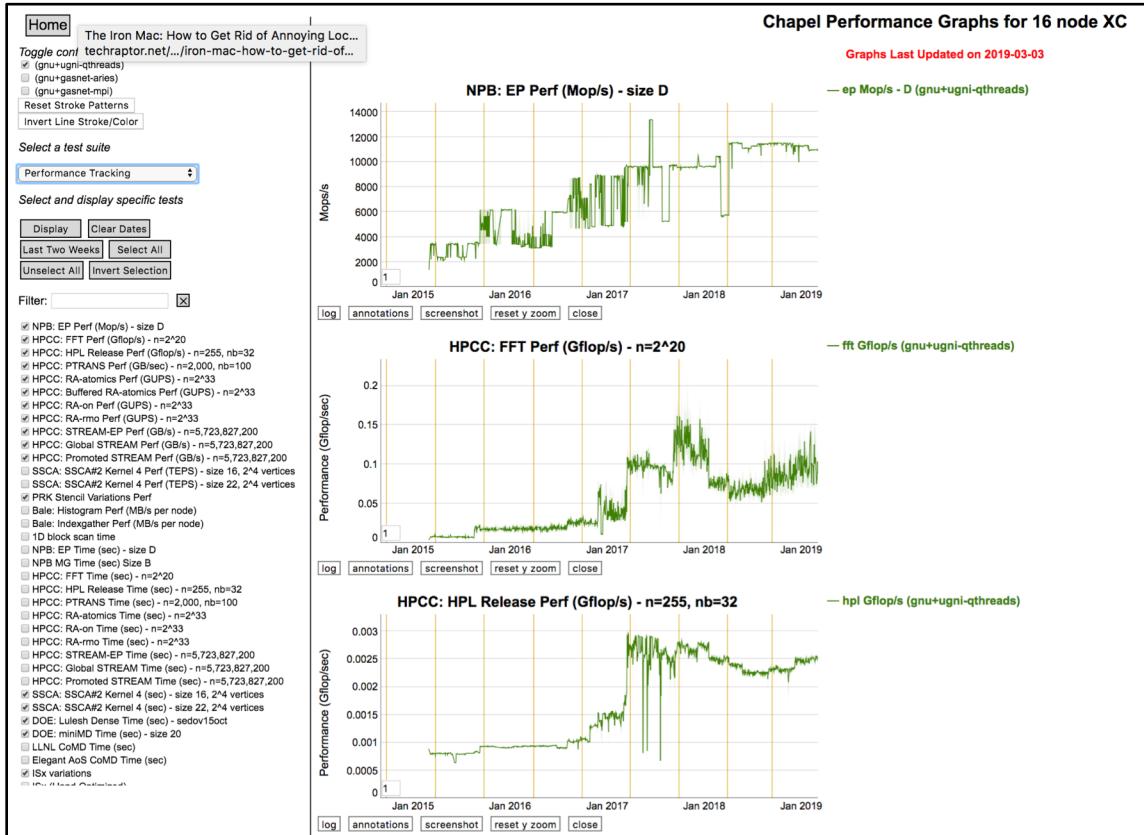
Build Executor Status master chap01 chap02

Testing managed through Jenkins



Jenkins				Last Statuses			Last Duration		Crash Totals		Log in	
				Last Status			Last Duration		Crash Totals		Log in	
				Last Status			Last Duration		Crash Totals		Log in	
People	All	W	Name ↓	correkness-test-regrutual	correctness-test-pyton-modules	correctness-test-pyton3	1.7 hr > 2.7 days	8.7 days	20 hr	Build periodically: 0 2	Log in	ENABLE AUTO REFRESH
Build History	All Active		archive-test-log	idle	idle	idle	17 hr	8 min 1 sec	8 min 1 sec	Build periodically: 0 2 ***		
Project Relationship			chapcs-correctness-test-c2chapel	idle	idle	idle	16 hr	1 hr 8 min	1 hr 8 min	Build periodically: 0 2 ***		
Check File Fingerprint			chapcs-correctness-test-gassian	idle	idle	idle	17 hr	13 hr	13 hr	Build periodically: 0 2 ***		
Job Import Plugin			chapcs-correctness-test-ham	idle	idle	idle	17 hr	56 min	56 min	Build periodically: 0 2 ***		
CLI Commander			chapcs-correctness-test-hamnet	idle	idle	idle	17 hr	25 min	25 min	Build periodically: 0 2 ***		
Build Queue	-		chapcs-correctness-test-valring	idle	idle	idle	17 hr	49 sec	49 sec	Build periodically: 0 2 ***		
No builds in the queue.			chapcs-mico-correctness-test-qd	idle	idle	idle	17 hr	16 sec	16 sec	Build periodically: 0 2 ***		
Build Executor -			chapcs-mico-correctness-test-qd	idle	idle	idle	17 hr	18 hr	18 hr	Build periodically: 0 2 ***		
Status			chapcs-mico-correctness-test-qd	idle	idle	idle	17 hr > 4.7 days	6.7 days	1 hr 42 min	Build periodically: 0 2 ***		
+ master			chapcs-mico-correctness-test-qd	idle	idle	idle	17 hr > 7.7 days	2 hr 45 min	2 hr 45 min	Build periodically: 0 2 ***		
1 idle			chapcs-mico-correctness-test-qd	idle	idle	idle	15 hr > 9.6 min	2 hr 22 min	2 hr 22 min	Build periodically: 0 2 ***		
2 idle			chapcs-mico-correctness-test-qd	idle	idle	idle	3.7 mo > 8.3 mo	35 sec	35 sec	Build periodically: 0 2 ***		
3 idle			chapcs-mico-correctness-test-qd	idle	idle	idle	2.4 days > 21 days	2 hr 30 min	2 hr 29 min	Build periodically: 0 2 ***		
4 idle			chapcs-mico-correctness-test-qd	idle	idle	idle	2.4 days > 1.2 mo	1 hr 56 min	1 hr 56 min	Build periodically: 0 2 ***		
5 idle			chapcs-mico-correctness-test-qd	idle	idle	idle	2.4 days > 16 days	29 min	29 min	Build periodically: 0 2 ***		
+ chap01			chapcs-mico-correctness-test-qd	idle	idle	idle	2.4 days > 18 days	20 min	20 min	Build periodically: 0 2 ***		
1 idle			chapcs-mico-correctness-test-qd	idle	idle	idle	8.7 hr	1 hr 44 min	1 hr 44 min	Build periodically: 0 2 ***		
+ chap02			chapcs-mico-correctness-test-qd	idle	idle	idle	8.7 hr > 7.7 days	6 hr 36 min	6 hr 36 min	Build periodically: 0 2 ***		
1 idle			chapcs-mico-correctness-test-qd	idle	idle	idle	17 hr > 1.4 mo	1 hr 42 min	1 hr 42 min	Build periodically: 0 2 ***		
+ chap04			chapcs-mico-correctness-test-qd	idle	idle	idle	15 hr	33 sec	33 sec	Build periodically: 0 2 ***		
1 idle			chapcs-mico-correctness-test-qd	idle	idle	idle	2.4 days > 13 days	35 sec	35 sec	Build periodically: 0 2 ***		
+ chap05			chapcs-mico-correctness-test-qd	idle	idle	idle	15 hr	1 min 14 sec	1 min 14 sec	Build periodically: 0 2 ***		
1 correctness	#1484		chapcs-mico-correctness-test-qd	idle	idle	idle	2.4 days > 1.2 mo	3 hr 32 min	3 hr 32 min	Build periodically: 0 2 ***		
+ chap06			chapcs-mico-correctness-test-qd	idle	idle	idle	2.4 days > 18 days	8 hr 3 min	8 hr 3 min	Build periodically: 0 2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	5.8 min	10 min	10 min	Build periodically: 0 2 ***		
+ chap07			chapcs-mico-correctness-test-qd	idle	idle	idle	5.8 min	2 hr 59 min	2 hr 59 min	Build periodically: 0 2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	5.8 min	1 hr 45 min	1 hr 45 min	Build periodically: 0 2 ***		
+ chap08			chapcs-mico-correctness-test-qd	idle	idle	idle	5.8 min	44 sec	44 sec	Build periodically: 0 2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	5.8 min	27 sec	27 sec	Build periodically: 0 2 ***		
+ chap09			chapcs-mico-correctness-test-qd	idle	idle	idle	5.8 min	46 sec	46 sec	Build periodically: 0 2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	17 hr	1 min 12 sec	1 min 12 sec	Build periodically: 0 2 ***		
+ chap10			chapcs-mico-correctness-test-qd	idle	idle	idle	2.1 days	8 min 16 sec	8 min 16 sec	Build periodically: 0 1 *** 6		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	3.7 days	6.8 sec	6.8 sec	Build periodically: 0 1 *** 7		
+ chap11			chapcs-mico-correctness-test-qd	idle	idle	idle	2.7 days	53 sec	53 sec	Build periodically: 0 2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	16 hr	9 min 24 sec	9 min 24 sec	Build periodically: 45 1 ***		
+ chap12			chapcs-mico-correctness-test-qd	idle	idle	idle	17 hr	19 sec	19 sec	Poll SCM: H/2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	17 hr	12 sec	12 sec	Poll SCM: H/2 ***		
+ chap13			chapcs-mico-correctness-test-qd	idle	idle	idle	17 hr	7.1 sec	7.1 sec	Poll SCM: H/2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	17 hr	9.2 sec	9.2 sec	Poll SCM: H/2 ***		
+ chap14			chapcs-mico-correctness-test-qd	idle	idle	idle	17 hr	7.3 sec	7.3 sec	Poll SCM: H/2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	4.7 mo	11 sec	11 sec	Poll SCM: H/2 ***		
+ chap15			chapcs-mico-correctness-test-qd	idle	idle	idle	17 hr	2 min 39 sec	2 min 39 sec	Build periodically: 45 1 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	17 hr	7 hr 15 min	7 hr 15 min	Build periodically: 0 2 ***		
+ chap16-mrhelp			chapcs-mico-correctness-test-qd	idle	idle	idle	51 min	3 min 57 sec	3 min 57 sec	Poll SCM: H/2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	1.5 hr	2 min 56 sec	2 min 56 sec	Poll SCM: H/2 ***		
+ chap17			chapcs-mico-correctness-test-qd	idle	idle	idle	1.5 hr	2 min 33 sec	2 min 33 sec	Poll SCM: H/2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	1.5 hr	2 min 46 sec	2 min 46 sec	Poll SCM: H/2 ***		
+ chap18			chapcs-mico-correctness-test-qd	idle	idle	idle	1.5 hr	3 min 28 sec	3 min 28 sec	Poll SCM: H/2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	1.5 hr	9 min 44 sec	9 min 44 sec	Poll SCM: H/2 ***		
+ chap19			chapcs-mico-correctness-test-qd	idle	idle	idle	1.5 hr	0.17 sec	0.17 sec	Build periodically: 45 1 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	1.5 hr	7 hr 20 min	7 hr 20 min	Build periodically: 0 2 ***		
+ chap20			chapcs-mico-correctness-test-qd	idle	idle	idle	1.5 hr	8 hr 44 min	8 hr 44 min	Build periodically: 0 2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	1.5 hr	57 min	57 min	Build periodically: 0 2 ***		
+ chap21			chapcs-mico-correctness-test-qd	idle	idle	idle	1.5 hr	3 min 44 sec	3 min 44 sec	Build periodically: 0 2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	1.5 hr	14 min	14 min	Build periodically: 30 1 ***		
+ chap22			chapcs-mico-correctness-test-qd	idle	idle	idle	1.5 hr	5 min 8 sec	5 min 8 sec	Build periodically: 45 1 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	1.5 hr	18 sec	18 sec	Build periodically: 45 1 *** 0 13 ***		
+ chap23			chapcs-mico-correctness-test-qd	idle	idle	idle	18 hr	50 sec	50 sec	Poll SCM: H/2 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	18 hr	7 hr 18 min	7 hr 18 min	Build periodically: 45 0 ***		
+ chap24			chapcs-mico-correctness-test-qd	idle	idle	idle	18 hr	7.7 days	7.7 days	Build periodically: 45 1 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	18 hr	9.9 hr	9.9 hr	Build periodically: 45 1 ***		
+ chap25			chapcs-mico-correctness-test-qd	idle	idle	idle	18 hr	9.8 mo	9.8 mo	Build periodically: 45 1 ***		
1 correctness	#1538		chapcs-mico-correctness-test-qd	idle	idle	idle	18 hr	3.7 days	3.7 days	Build periodically: 45 1 ***		
+ chap26			chapcs-mico-correctness-test-qd	idle	idle	idle	18 hr	Icon - S.M.I.	Icon - S.M.I.	Build periodically: 45 1 ***		

Also Performance Tests



Current gaps in our testing

CRAY

- 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

CRAY

- 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

CRAY

- 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

CRAY

- 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

CRAY

- 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

CRAY

- 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

CRAY

<https://chapel-lang.org>

- downloads
- presentations
- papers
- resources
- documentation



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} dmapred 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 **CHI UW 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](#), [CHI UW 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)

CRAY

[http://twitter.com/ChapelLanguage](https://twitter.com/ChapelLanguage)

[http://facebook.com/ChapelLanguage](https://facebook.com/ChapelLanguage)

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

Chapel Community

CRAY

Questions Developer Jobs Tags Users [chapel]

Tagged Questions

Chapel is a portable, open-source parallel programming language. Use this tag to ask questions about the Chapel language or its implementation.

Learn more... Improve tag info Top users Synonyms

Tuple Concatenation in Chapel
Let's say I'm generating tuples and I want to concatenate them as they come. How do I do this? The following does element-wise addition: if `ts = ("foo", "cat"), t = ("bar", "dog") ts += t` gives `ts = ...`

Is there a way to use non-scalar values in functions with where clauses in Chapel?
I've been trying out Chapel off and on over the past year or so. I have used C and C++ briefly in the past, but most of my experience is with dynamic languages such as Python, Ruby, and Erlang more ...

Is there any writeln() format specifier for a bool?
I looked at the `writeln()` documentation for any bool specifier and there didn't seem to be any. In a Chapel program I have: ... config const verify = false; /* that works but I want to use writeln() ...

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

This repository Search Pull requests Issues Marketplace Gist

chapel-lang / chapel

Code Issues 292 Pull requests 26 Projects 0 Settings Insights

Filters ▾ IsIssue:open Labels Milestones

292 Open 77 Closed

Implement "bounded-coforall" optimization for remote coforalls area: Compiler type: Performance #6357 opened 13 hours ago by ronawho

Consider using processor atomics for remote coforalls EndCount area: Compiler type: Performance #6356 opened 13 hours ago by ronawho 0 of 6

make uninstall area: BTR type: Feature Request #6353 opened 14 hours ago by mpf

make check doesn't work with ./configure area: BTR #6352 opened 16 hours ago by mpf

Passing variable via intent to a forall loop seems to create an iteration-private variable, not a task-private one area: Compiler type: Bug #6351 opened a day ago by casselle

Remove chpl_comm_make_progress area: Runtime easy type: Design #6349 opened a day ago by sunghunchoi

Runtime error after make on Linux Mint area: BTR user issue #6348 opened a day ago by denindiana

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

GITTER

chapel-lang/chapel Chapel programming language | Peak developer hours are 0600-1700 PT

Brian Dolan @buddha314 what is the syntax for making a copy (not a reference) to an array? May 09 14:34

Michael Ferguson @mpff like in a new variable? May 09 14:40

```
var A[1..10] int;
var B = A; // makes a copy of A
ref C = A; // refers to A
```

Brian Dolan @buddha314 oh, got it, thanks! May 09 14:41

Michael Ferguson @mpff May 09 14:42

```
proc f(x) { /* x refers to the actual argument */ }
proc g(in arr) { /* arr is a copy of the actual argument */ }
var A[1..10] int;
f(A);
g(A);
```

Brian Dolan @buddha314 isn't there a proc f(ref arr) {} as well? May 09 14:43

Michael Ferguson @mpff yes. The default intent for array is 'ref' or 'const ref' depending on if the function body modifies it. So that's effectively the default. May 09 14:45

Brian Dolan @buddha314 thanks! May 09 14:45

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

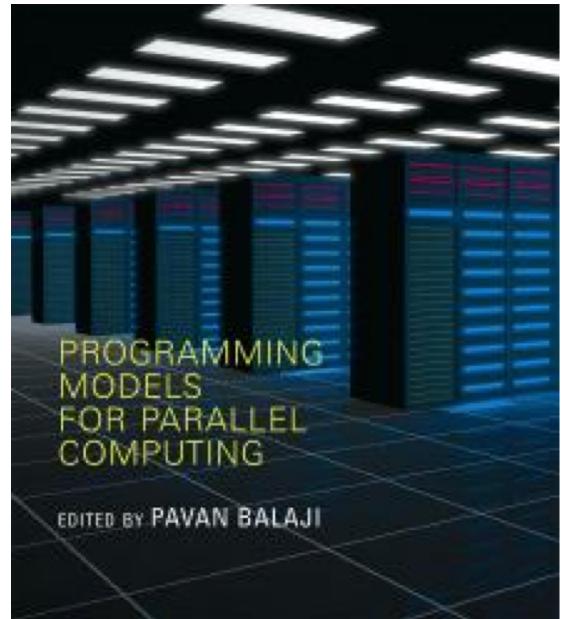
read-only mailing list: chapel-announce@lists.sourceforge.net (~15 mails / year)

Suggested Reading: Chapel history and overview

CRAY

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 Hershberger, 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. Over years of the DARPA High Productivity Computing Systems (HPCS) program that launched Chapel wrapped up, and the team embarked on a five-year effort to move the Chapel compiler to end-users. This paper follows up on our CUG 2016 paper summarizing the progress made by the Chapel project since that time. Specifically, Chapel's performance now competes with or beats hand-coded GPU/FIRETEAM, MPI, 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 contributors 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 want to specify exactly how 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>

paper and slides available at chapel-lang.org



**Chapel Comes of Age:
Productive Parallelism at Scale** 
CUG 2018
Brad Chamberlain, Chapel Team, Cray Inc.

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-

