Welcome to CHIUW 2019!

the ACM SIGPLAN
6th Annual Chapel Implementers and Users Workshop

Benjamin Robbins and Brad Chamberlain

CHIUW 2019

June 22–23, 2019

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

What is Chapel?

Chapel: A modern 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

Why Consider New Languages at all?

Syntax	• High level, elegant syntax • Improve programmer productivity
Semantics	• Static analysis can help with correctness • We need a compiler (front-end)
Performance	• If optimizations are needed to get performance • We need a compiler (back-end)
Algorithms	• Language defines what is easy and hard • Influences algorithmic thinking

[Source: Kathy Yelick, CHIUW 2018 keynote: *Why Languages Matter More Than Ever*]

A Year in the Life of Chapel

- Two major releases per year (March & September)
  - **~a month later:** detailed release notes
  - **latest release:** Chapel 1.19, released March 21st 2019
- CHIUW: Chapel Implementers and Users Workshop (May/June)
  - typically co-located with IPDPS or PLDI
- SC (November)
  - annual **CHUG (Chapel Users Group) happy hour**
  - some years: talks, tutorials, panels, BoFs, posters, …
- Talks, tutorials, research visits, social media, … (year-round)

A Year in the Life of Chapel

- Two major releases per year (March & September)
  - **~a month later:** detailed release notes
  - **latest release:** Chapel 1.19, released March 21st 2019
- CHIUW: Chapel Implementers and Users Workshop (May/June)
  - typically co-located with IPDPS or PLDI
- SC (November)
  - annual **CHUG (Chapel Users Group) happy hour**
  - some years: talks, tutorials, panels, BoFs, posters, …
- Talks, tutorials, research visits, social media, … (year-round)

CHIUW 2019 in a Nutshell

The ACM SIGPLAN 6th Annual Chapel Implementers and Users Workshop

- Keynote
- Submitted Talks and Research Papers
- State of the Project Talk
- Lightning Talks Session
- Socializing over Breaks and Meals
- Coding Day (tomorrow)

CHIUW 2019: Organizing Committee

General Chair:

- **Benjamin Robbins,** *Cray Inc.*

Steering Committee:

- **Michael Ferguson**, *Cray Inc.*
- **Nikhil Padmanabhan**, *Yale University*

Program Committee:

- **Brad Chamberlain** (chair), *Cray Inc.*
- **Maryam Dehnavi** (co-chair), *University of Toronto*
- **Rafael Asenjo**, *University of Malaga*
- **Michael Ferguson**, *Cray Inc.*
- **Oscar Hernandez**, *ORNL*
- **Hang Liu**, *UMass Lowell*
- **Nikhil Padmanabhan**, *Yale University*
- **Tyler Simon**, *UMBC*
- **Didem Unat**, *Koç University*
- **Ana Lucia Varbanescu**, *University of Amsterdam*
- **Rich Vuduc**, *Georgia Tech*
- **David Wonnacott**, *Haverford College*

CHIUW 2019: Keynote

Programming Abstractions for Orchestration of HPC Scientific Computing

Anshu Dubey, *Argonne National Laboratory / University of Chicago*

Abstract: Application developers are confronted with three axes of increasing complexity going forward; increasing heterogeneity in computing platforms at all levels, increasing heterogeneity in solvers and data management, and moving existing code bases to future programming models. While the first two will dictate which future programming models may deliver the needed performance, the third will determine their adoption. However, it is clear that the infrastructure backbone of large scale Multiphysics software has to orchestrate data and task movement between devices. The lifecycle of scientific software is several times that of platforms, therefore, any orchestration mechanism must have flexibility and configurability to remain usable on future platforms. In this presentation I will outline a model of an orchestration framework and the demands that it will place on programming models and languages.

CHIUW 2019: Community Talks and Papers

GPUIterator: Bridging the Gap between Chapel and GPU Platforms

- Akihiro Hayashi (*Rice*), Sri Raj Paul, Vivek Sarkar (*Georgia Tech*)

Implementing Stencil Problems in Chapel: An Experience Report

- Per Fuchs, Pieter Hijma (*Vrije Universiteit Amsterdam*), Clemens Grelck (*University of Amsterdam*)

Arkouda: Interactive Data Exploration Backed by Chapel

- Michael Merrill, William Reus, Timothy Neumann (*DOD*)

Chapel Graph Library (CGL)

- Louis Jenkins (*University of Rochester*), Marcin Zalewski (*PNNL*)

CHIUW 2019: Cray Talks and Papers

Calling Chapel Code: Interoperability Improvements

- Lydia Duncan, David Iten (*Cray*)

Towards Radix Sorting in the Chapel Standard Library

- Michael Ferguson (*Cray*)

Chapel Unblocked: Recent Communication Optimizations in Chapel

- Elliot Ronaghan, Ben Harshbarger, Gregory Titus, Michael Ferguson (*Cray*)

Chapel in Cray HPO

- Benjamin Albrecht, Alex Heye, Benjamin Robbins (*Cray*)

CHIUW 2019: Lightning Talks & Flash Discussions

- Continuing our recent tradition
- Last session of the day!
- **Goal:** high-energy topics for tired attention spans!
- **Format:** Short talks, Q&A, war stories, discussions, …whatever!
- Sign up for a slot!

CHIUW 2019: Lightning Talks & Flash Discussions

- Continuing our recent tradition
- Last session of the day!
- **Goal:** high-energy topics for tired attention spans!
- **Format:** Short talks, Q&A, war stories, discussions, …whatever!
- Sign up for a slot!

CHIUW 2019: Coding Day tomorrow

location: Hyatt Regency, Suite 318

time: 9:00

goal: work on challenges in small teams while we're in one place

proposed activities:

- **Python Interop demo:** Lydia
- **Arkouda**: Mike
- **Review GPUIterator code:** Akihiro, Michael, …
- **NUMA improvements / benchmark study:** Pieter, Elliot, …
- **Optimize graph benchmarks:** Louis, Marcin, Brad(?), …
- **Arkouda improvements:** Mike, Lydia(?), Brad(?), Michael(?), …
- **Random numbers / FFTs:** Nikhil, BenA(?), …
- **Others?** (Kick off Discourse page? Kick-start Chapel blog?)

CHIUW 2019: Agenda (chapel-lang.org/CHIUW2019.html)

9:00:	Chapel 101 (optional)
9:30:	Welcome, State of the Project
10:00:	Talks: Chapel Implementation Improvements
11:00:	Break
11:20:	Talks: Chapel Performance and Optimization
12:35:	Lunch
2:00:	Keynote Talk, Anshu Dubey
3:00:	Talks: Applications of Chapel
3:30:	Break
4:00:	Talks: Applications of Chapel, continued
4:50:	Lightning Talks and Flash Discussions
5:30:	Wrap-up / Head to Dinner

State of the Chapel Project

Brad Chamberlain

CHIUW 2019

June 22, 2019


bradc@cray.com


chapel-lang.org


@ChapelLanguage

A Brief History of Chapel

A Brief History of Chapel: HPCS

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

- ~6–7 Chapel developers at Cray
- **Research focus:** Can we create a language that…

  …distinguishes locality from parallelism?

  …seamlessly mixes data- and task-parallelism?

  …supports user-defined distributed arrays and 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: HPCS

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

- ~6–7 Chapel developers at Cray
- **Research foc**
  - …distinguis
  - …seamless
  - …supports u
- Captured post
  - *The State of*
    - Chamberlai

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: post-HPCS

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: post-HPCS

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

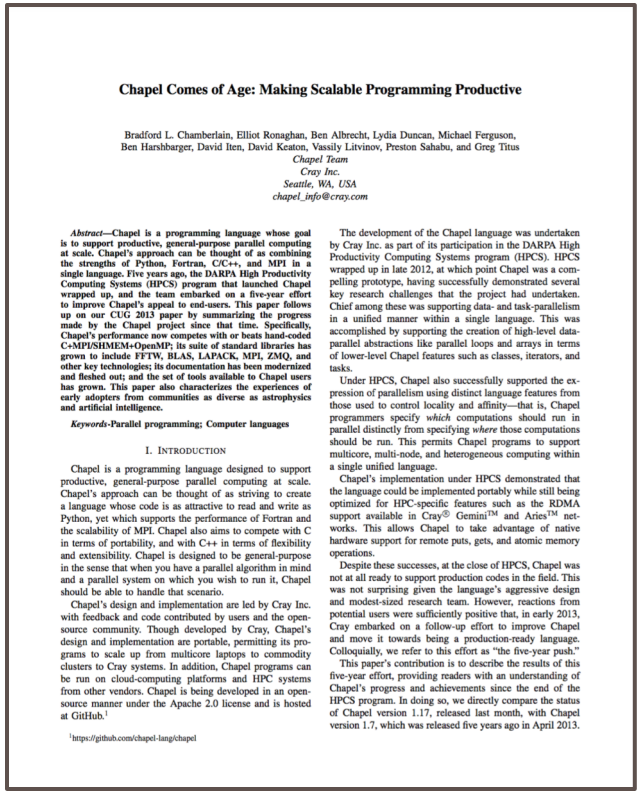
Chapel's Adolescence: "the five-year push" (2013–2018)

- ~13–14 Chap
- Development
  - address w

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

CUG 2018 Paper: Summary of Five-year Push



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

A Brief History of Chapel: Now

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**

A Brief History of Chapel: Now

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 2.0

- Users don’t like when their code breaks due to language evolution
- ***Chapel 2.0:*** a future release in which we commit to avoiding breaking changes
- Short list of key remaining feature areas to focus on:
  - initializers (replacement for constructors)
  - memory managed classes / lifetime checking
  - nilable vs. non-nilable class types
  - UTF-8 strings
  - move collection APIs from arrays/domains to standard types in the library
  - constrained generics (?)

Chapel 2.0 Release Candidate

- This fall's release is intended to be a candidate for Chapel 2.0
  - If no issues found, release Chapel 2.0 next spring
  - Otherwise, try again in subsequent release

Post-Chapel 2.0

- Thereafter...
  - …probably naïve to believe we won't make backwards-breaking changes
  - …but hopefully far fewer than in recent releases
  - …expect to follow semantic versioning (treating 2.0 as Chapel's 1.0)
- Would like to establish a means for having users weigh in on proposed changes
  - Would be a shame to freeze features that all current users believe broken
- Concept of continuing to support previous versions when requested via flags
  - e.g., `chpl --std=chpl-2.0`

Highlights Since CHIUW 2018

Object-Oriented Improvements

- initializers are now the default; constructors have been deprecated
- classes now support four memory management types:
  - **shared:** object deallocated when no references remain
  - **owned:** object deallocated when it no longer has an owner
  - **unmanaged:** object won't be automatically deallocated
  - **borrowed:** refers to an object without affecting its memory management
- compile-time lifetime checking and nil-checking now helps avoid common errors
- 'override' keyword to avoid mistakes in methods

Other Language Improvements

- Shape preservation for loop- and promoted expressions

```
const D = {1..n, 1..n};
var A, B: [D] real = …;
var C = A + B;                    // C's domain is now D
```

- Task-private variables for `forall` loops
- Compile-time floating point operations
- Enum improvements
- Underscores in numeric literals and strings
- UTF-8 strings (ongoing)

Library Improvements

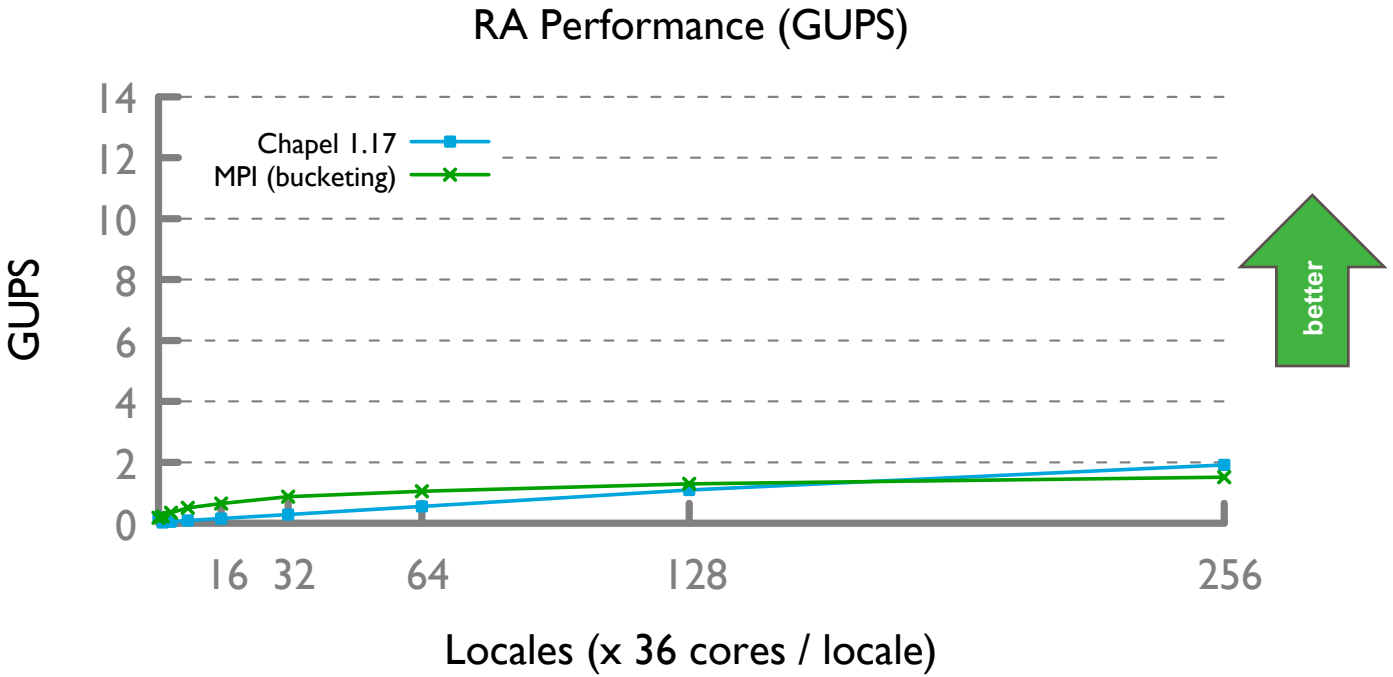
- I/O: HDF5 and NetCDF support
- Distributed associative domains and arrays
- LinearAlgebra improvements
- Parallel radix sort
  - See Michael Ferguson's talk this afternoon

Performance Improvements

- Significant improvements in a number of areas

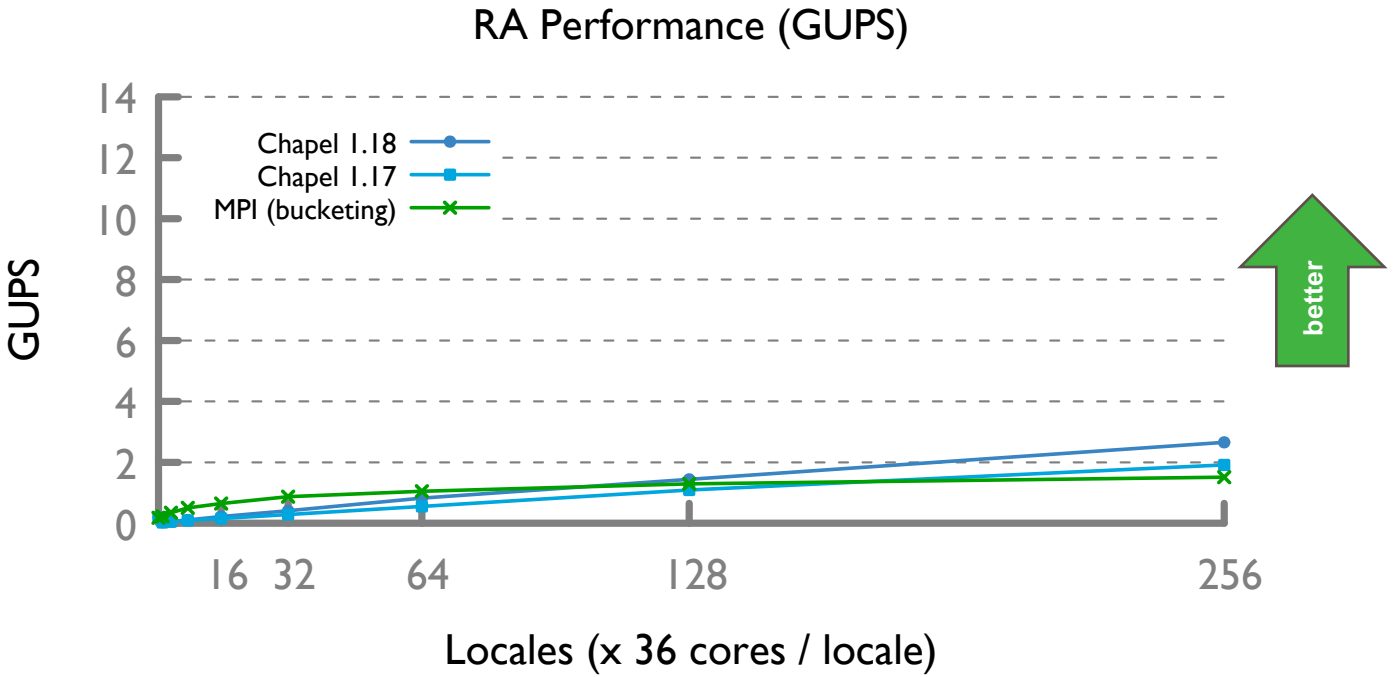
Performance Improvements

- Significant improvements in a number of areas, for example:



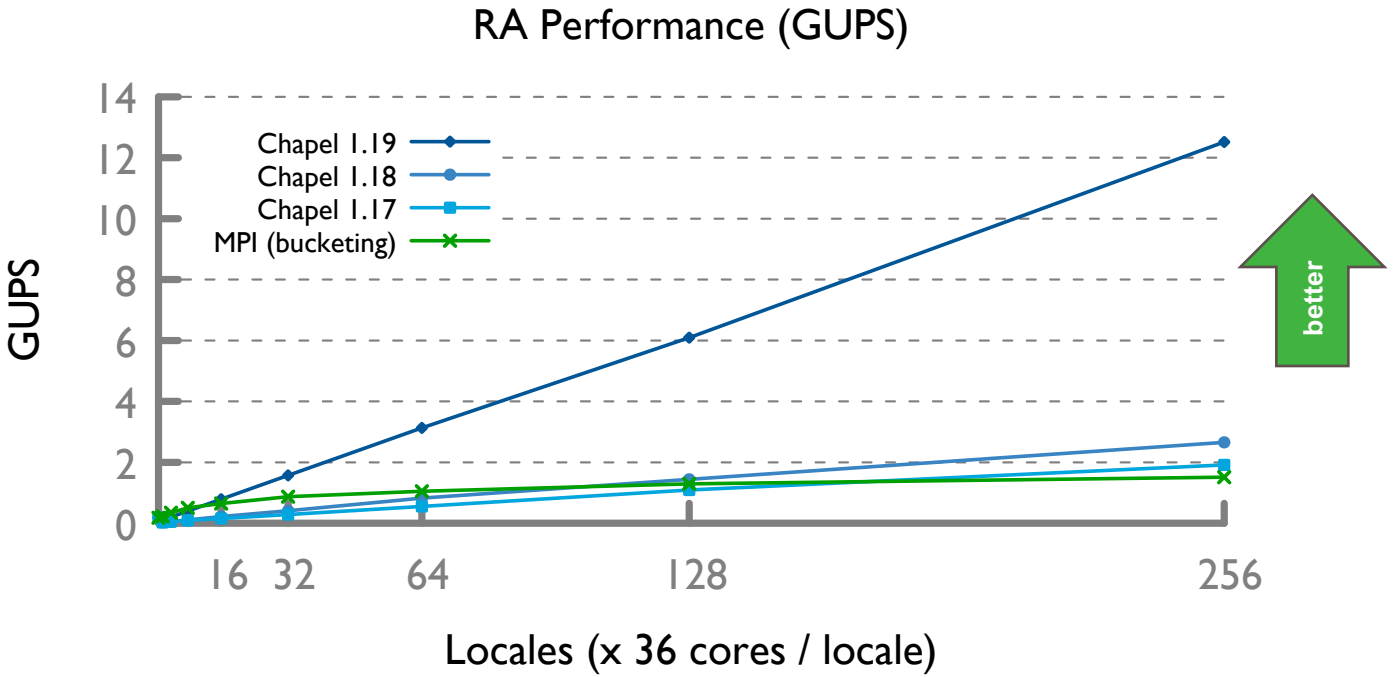
Performance Improvements

- Significant improvements in a number of areas, for example:



Performance Improvements

- Significant improvements in a number of areas, for example:



HPCC RA: MPI vs. Chapel

Chapel Kernel

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

```
/* 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.dty
          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 == 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);
                                                  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);
                                                                  e,
                                                                  pe,
 }
 /* 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);
```

Performance Improvements

- Significant improvements in a number of areas
  - see Elliot Ronaghan's talk this afternoon
  - also, see Pieter Hijma's talk for a user perspective on performance

Interoperability and Portability Improvements

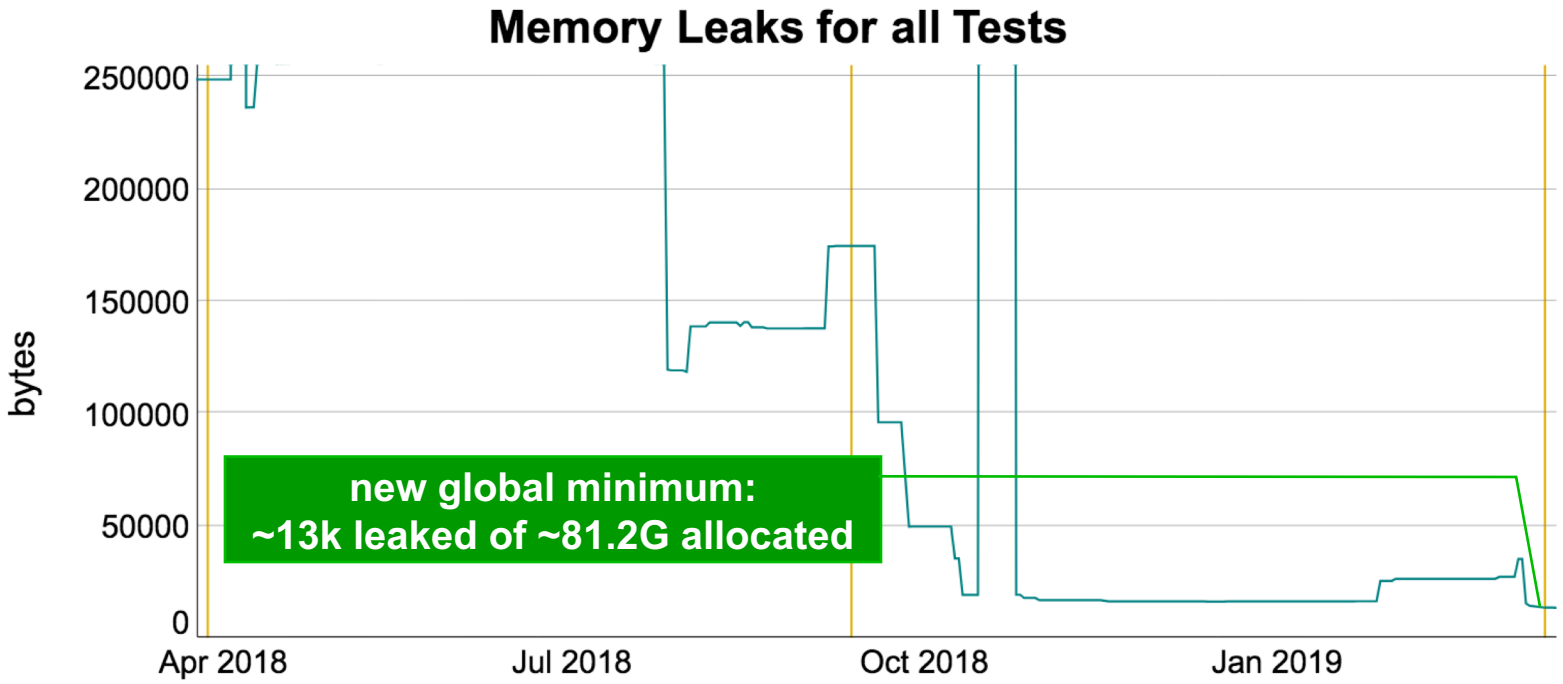
Interoperability:

- New ‘c_array’ type
- Nice improvements to Python, Fortran, and C interoperability
  - see Lydia Duncan’s talk this morning

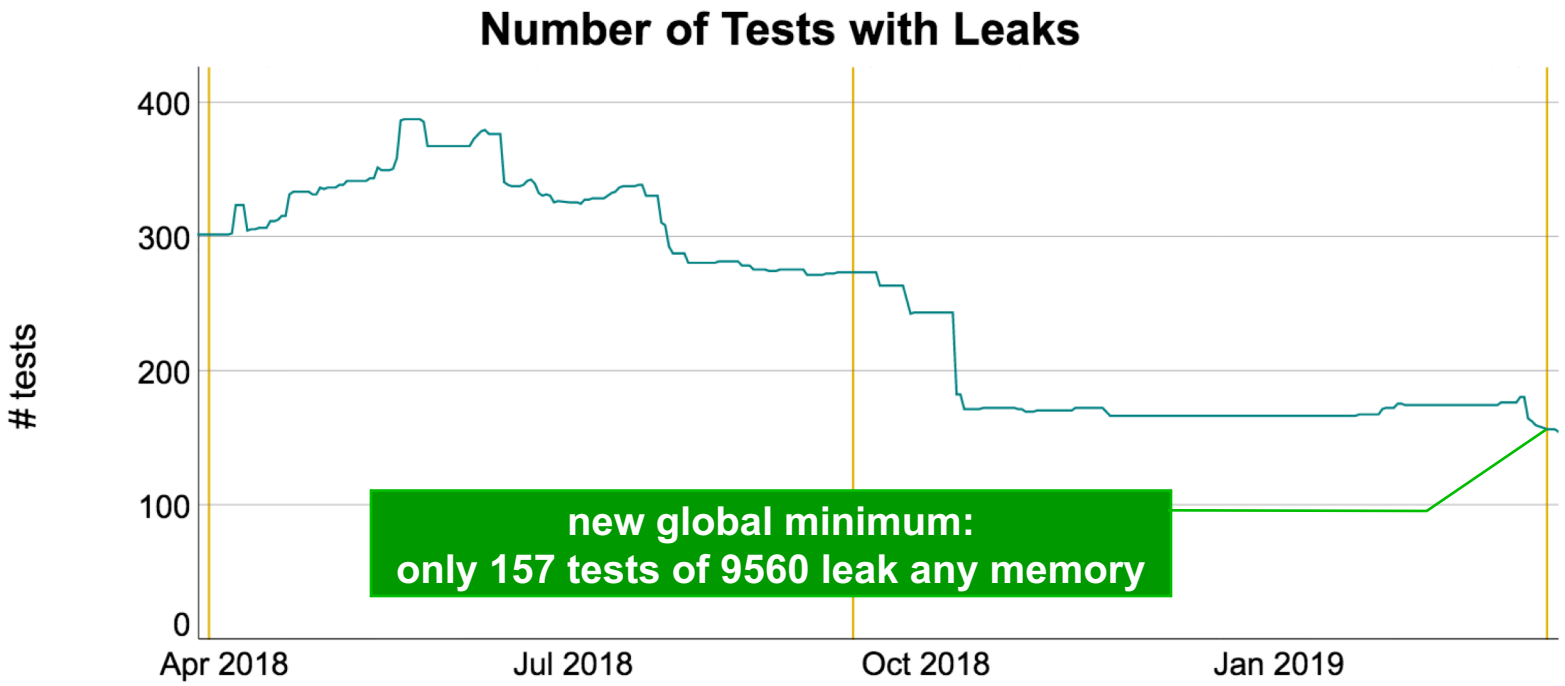
Portability:

- New libfabric-based ‘ofi’ runtime option for communication
- Improved support for ARM-based systems
- Initial support for Shasta systems

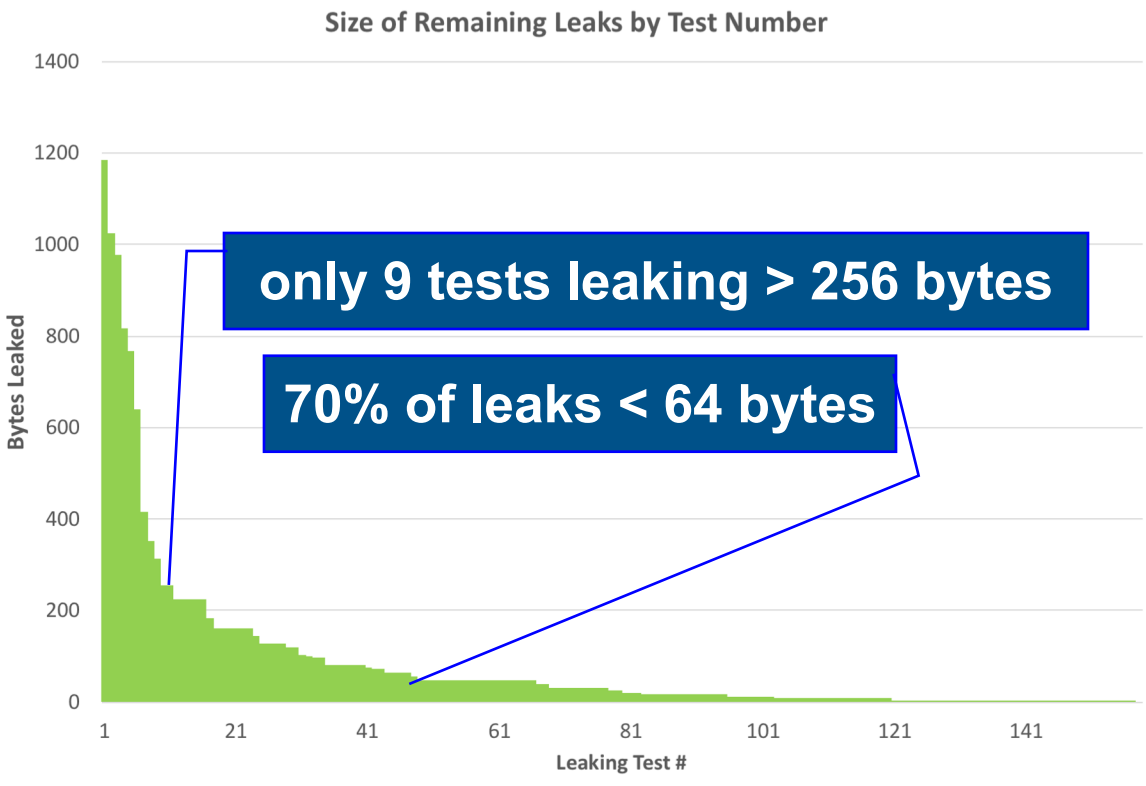
Memory Leak Improvements



Memory Leak Improvements



Memory Leaks as of 1.19 (zoomed in)



Additional Improvements

- LLVM back-end improvements
- ‘mason’ improvements including external Spack dependencies
- Bug fixes
- Error message improvements
- Documentation improvements
- Feature improvements

A Brief History of Chapel: What Now?

CRAY

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**

A Brief History of Chapel: What Now?

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++, … **+ Fortran**
  - **Portability:** Libfabric/OFI, GPUs, Cloud computing **+ Cray Shasta**
  - **Data Structures:** Sparse, DataFrames, Distributed Associative Arrays
  - **Chapel AI, Increased Adoption**

see Akihiro Hayashi's talk, up next

Notable New Use Cases / Users

Arkouda: Chapel implementations of NumPy operations

- see Mike Merrill's talk this afternoon

CHGL: Chapel Graph Library

- see Louis Jenkin's talk this afternoon

Cray AI

- see Ben Albrecht's talk this afternoon

CFD codes in Chapel

- Matthieu Parenteau and Simon Bourgault-Côté (*École Polytechnique de Montréal*)

Distributed tree search algorithms

- Tiago Carneiro and Noredine Melab (*INRIA Lille / Universitat de Lille*)
- studies published in ICCS and HPCS

What's Next?

Near-term Priorities

- Focus on Chapel 2.0 release candidate
- Shasta readiness
- Address performance / scalability gaps
- Interoperability improvements
- User support
- LLVM back-end used by default

Medium-term Priorities

- GPU support
- Compilation speed
- Performance on non-Cray systems
- …?

Parallel Applications Workshop, Alternatives To MPI+X

Sunday, November 17th, 2019

Held in conjunction with SC19

SC19 Denver, CO | hpc is now.

Submission deadline July 31

Like CHIUW, accepts papers and talks

In cooperation with:

TCHPC

Welcome to CHIUW 2019!

the ACM SIGPLAN
6th Annual Chapel Implementers and Users Workshop

June 22–23, 2019

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

CRAY

CRAY CHAPEL

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-/