Hewlett Packard Enterprise

STATE OF THE CHAPEL PROJECT

Brad Chamberlain, HPE

CHIUW 2022, June 10, 2022

WHAT IS CHAPEL?

Chapel: A modern parallel programming language

- portable & scalable
- open-source & collaborative

Goals:

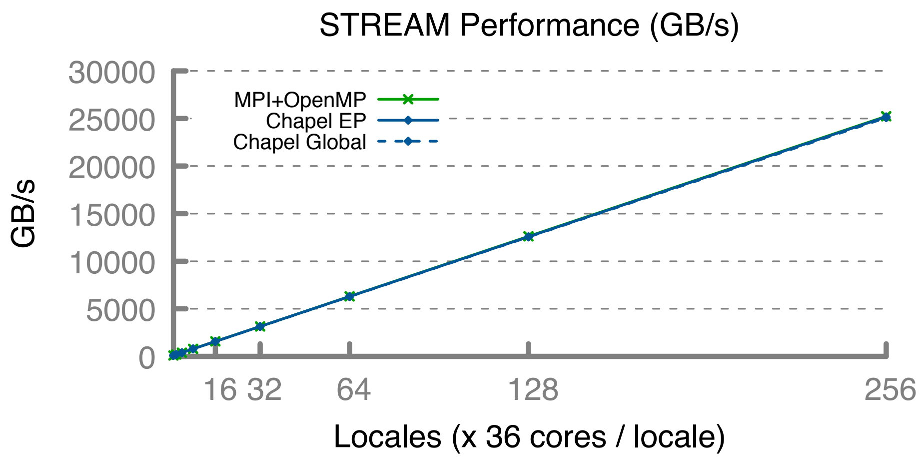
- Support general parallel programming
- Make parallel programming at scale far more productive
  - Python-like support for rapid prototyping, clear code
  - yet with the performance, scaling, GPU support of Fortran/C/C++, MPI, OpenMP, CUDA, ...

FOR HPC BENCHMARKS, CHAPEL TENDS TO BE CONCISE, CLEAR, AND COMPETITIVE

STREAM TRIAD: C + MPI + OPENMP

```c
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );

  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
      fprintf( outFile, "Failed to allocate memory
      fclose( outFile );
    }
    return 1;
  }

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 1.0;
  }
  scalar = 3.0;

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);

  return 0;
}
```

```chapel
use BlockDist;

config const m = 1000,
             alpha = 3.0;

const Dom = {1..m} dmapped …;
var A, B, C: [Dom] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```

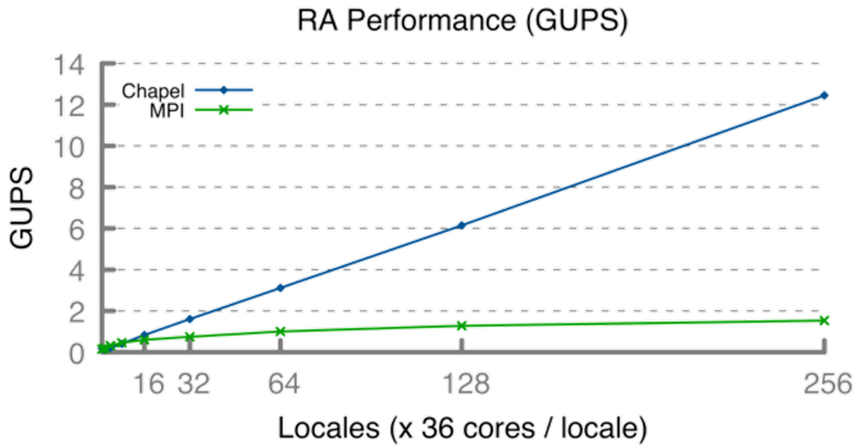


HPCC RA: MPI KERNEL

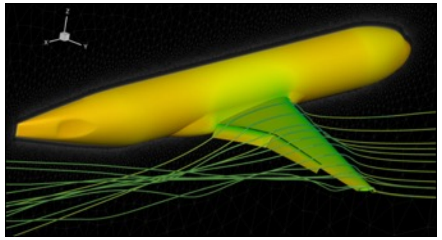
```c
/* Perform updates to main table.  The scalar equivalent is:
 *
 *   for (i=0; i<NUPDATE; i++) {
 *     Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *     Table[Ran & (TABSIZE-1)] ^= Ran;
 *   }
 */

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,
                 UPDATE_TAG, MPI_COMM_WORLD,
       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.dtype
         bufferBase = 0;
         for (j=0; j < recvUpdates; j ++) {
           inmsg = LocalRecvBuffer[bufferBase
           LocalOffset = (inmsg & (tparams.Ta
                         tparams.GlobalStartM
           HPCC_Table[LocalOffset] ^= inmsg;
         }
       } else if (status.MPI_TAG == FINISHED
         /* 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;

       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);
```

72

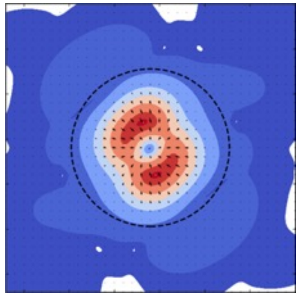
```chapel
…
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask].xor(r);
…
```



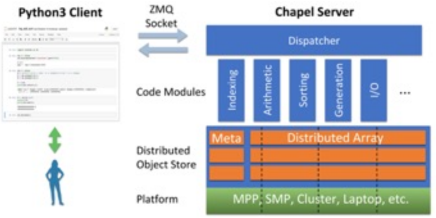
FLAGSHIP CHAPEL APPLICATIONS



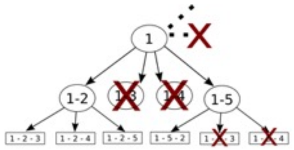
CHAMPS: 3D Unstructured CFD
Éric Laurendeau, Simon Bourgault-Côté, Matthieu Parenteau, et al.
École Polytechnique Montréal



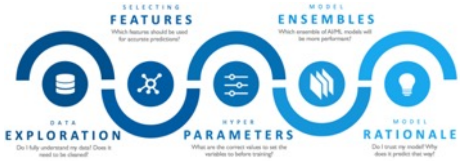
ChplUltra: Simulating Ultralight Dark Matter
Nikhil Padmanabhan, J. Luna Zagorac, *et al.*
Yale University / University of Auckland



Arkouda: NumPy at Massive Scale
Mike Merrill, Bill Reus, et al.
US DoD



ChOp: Chapel-based Optimization
Tiago Carneiro, Nouredine Melab, *et al.*
INRIA Lille, France



CrayAI: Distributed Machine Learning
Hewlett Packard Enterprise

?

Your application here?

(images provided by their respective teams and used with permission)

THE CHAPEL TEAM

HPE’s Chapel team currently consists of 16 full-time employees, 3 summer interns, and our director

- We also have 1 more full-time engineer joining this month and a few open positions

Chapel Development Team at HPE

see: https://chapel-lang.org/contributors.html
and https://chapel-lang.org/jobs.html

CHAPEL RELEASES

Three releases since CHIUW 2021:

Chapel 1.25.0: September 23, 2021

Chapel 1.25.1: December 9, 2021

Chapel 1.26.0: March 31, 2022

Up next:

Chapel 1.27.0: June 30, 2022 (anticipated)

(We expect to release on a quarterly schedule going forward)

STATE OF THE CHAPEL PROJECT IN 2022

- In a word: fantastic!

- For more detail, let’s look at ten highlighted areas/efforts since CHIUW 2022

NEW FACES

NEW FACES AT CHIUW 2022

- **Large-Scale and User-Friendly Exact Diagonalization in Chapel** Talk @ 8:50 PT
  - Tom Westerhout, Mikhail I. Katsnelson (*Radboud University*)
- **Extending Chapel to Support Fabric Attached Memory** Talk @ 10:05 PT
  - Amitha C, Clarete Crasta and Sharad Singhal (*Hewlett Packard Enterprise*)
- **Integrating Chapel programs and MPI-Based Libraries for High-performance Graph Analysis** Talk @ 10:25 PT
  - Trevor McCrary (*Georgia Institute of Technology*), Karen Devine, and Andrew Younge (*Sandia National Laboratories*)
- **An Introduction to GASNet-EX for Chapel Users** Talk @ 1:10 PT
  - Dan Bonachea and Paul H. Hargrove (*Lawrence Berkeley National Lab*)
- **ChapelPerf: A Performance Suite for Chapel** Talk @ 1:50 PT
  - Ricardo Jesus and Michèle Weiland (*EPCC, The University of Edinburgh*)
- **From C and Python to Chapel as My Main Programming Language** Talk @ 3:00 PT
  - Nelson Dias (*Federal University of Parana, Brazil*)

COMPILER IMPROVEMENTS

‘DYNO’ COMPILER REWORK

Background:

- The Chapel compiler…

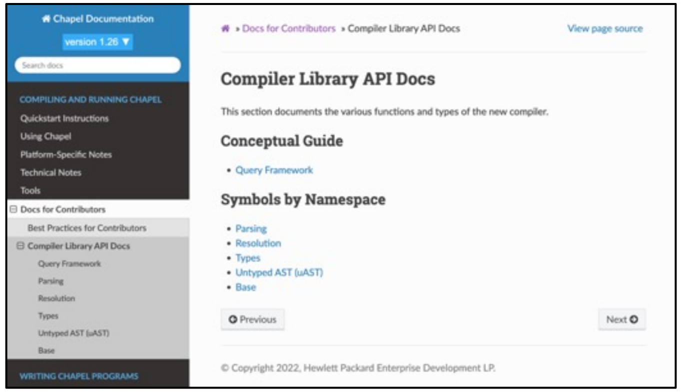
  …is slow (seconds to minutes)

  …can be hard to understand when there are errors

  …isn’t terribly well-architected: inflexible, challenging to get started with
- Largely reflects its origins as a scrappy research project, by a small team, moving fast

This Effort:

- This year, kicked off an effort to massively rearchitect it and address these lacks:
  - better user experience
  - easier to start contributing to
  - faster / more flexible: separate compilation, dynamic evaluation of code, …



Status:

- the ‘dyno’ parser will be the default in Chapel 1.27.0
- rewrites and restructuring of later passes also underway
- code structure documented online: https://chapel-lang.org/docs/developer/compiler-internals/index.html

LLVM BY DEFAULT

Background:

- Traditionally, Chapel has generated C code as its “portable assembly”
  - LLVM-based back-end was also available as an option

In Chapel 1.25:

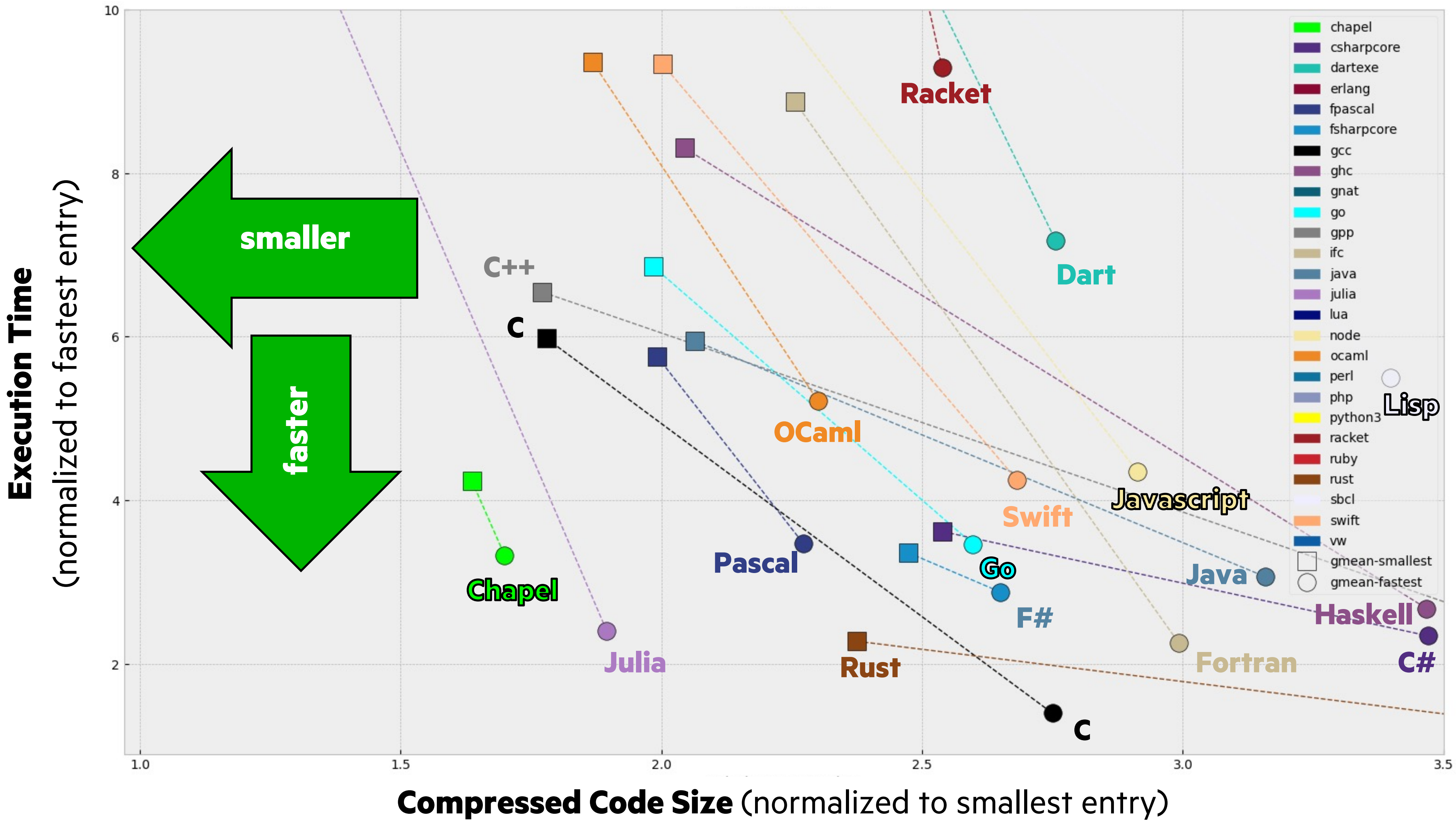
- Finally made good on a long-term intention to switch to the LLVM back-end by default (version 11)
  - C-based compilation is still available as an option
- **Motivation:**
  - reduces burden of trying to support all versions of all C compilers
  - communicates Chapel semantics more directly to back-end than C permits
  - leverages community investment in, and familiarity with, LLVM
  - modestly reduces compilation times, on average
  - provides an attractive path for targeting GPUs

Since then:

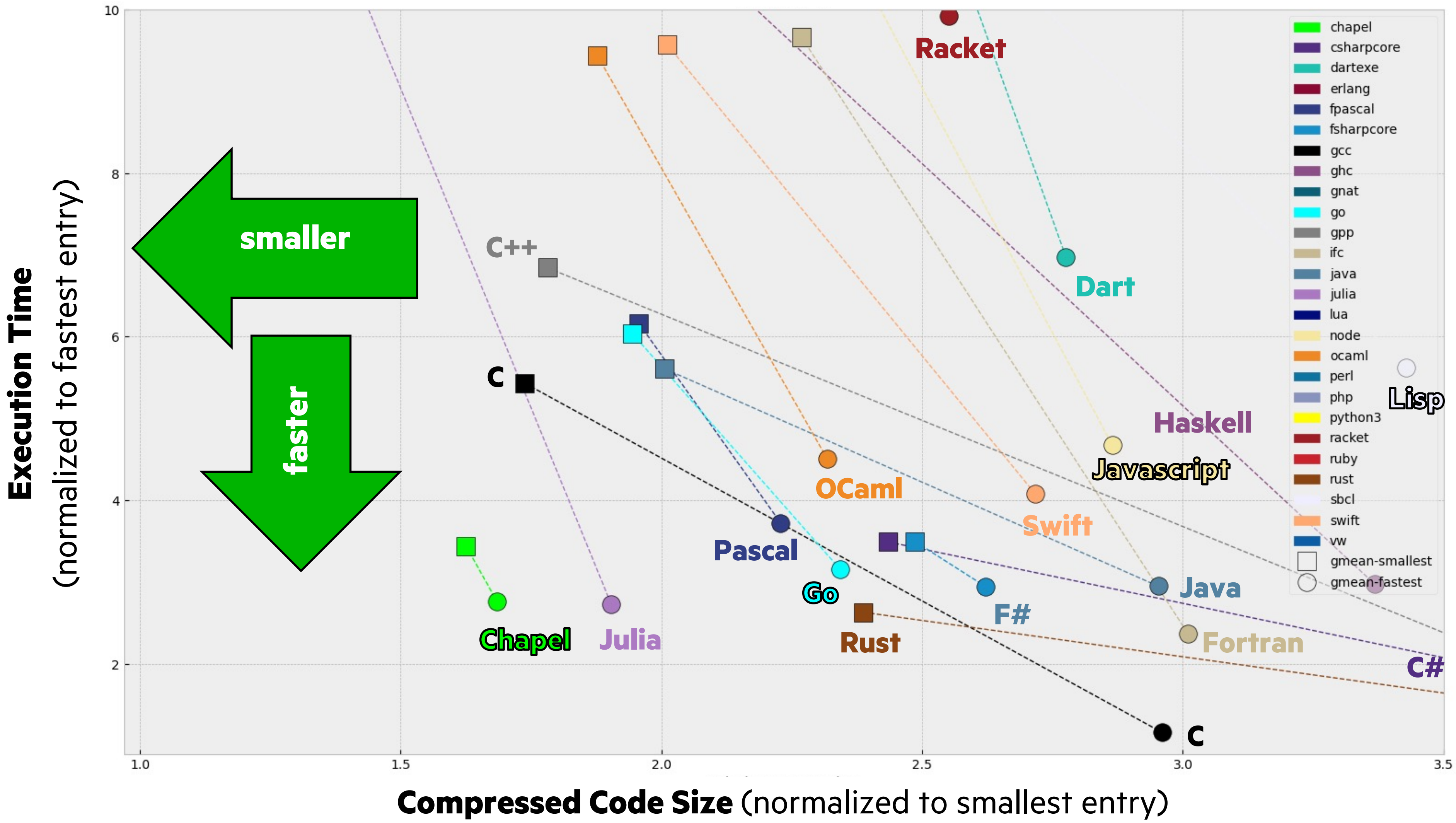
- Chapel 1.26.0: added support for LLVM 12 and 13
- Chapel 1.27.0: will add support for LLVM 14

COMPUTER LANGUAGE BENCHMARKS GAME STANDINGS

CLBG: ALL-LANGUAGE SUMMARY, CHIUW 2021 (ZOOMED-IN)



CLBG: ALL-LANGUAGE SUMMARY, CHIUW 2022 (ZOOMED-IN)



LANGUAGE / LIBRARY HIGHLIGHTS

NEW LANGUAGE FEATURES

‘manage’ statements: support Python-like context management

resizing arrays of non-nilable classes: implemented using context managers

- challenges relate to elements not having a sensible default value

‘foreach’ loops: express parallel loops that should be implemented by the current task

- help indicate opportunities for vectorization or GPU execution when a ‘forall’ loop’s tasks would be overkill

```
foreach i in 1..n do    // assert that this loop is order-independent
  a[i] = b[p[i]];
```

operators: prototyped in 1.24, now ready for use

- addressed an otherwise vague namespace issue

```
operator R.+(x: R, y: R) { … }
```

NEW LIBRARY PACKAGE MODULES

- **Socket:** supports TCP-/UDP-based socket communications
- **Channels:** supports Go-style channels for message queues between tasks
- **CopyAggregation:** makes available the aggregator abstractions used by Arkouda and Bale IndexGather
- **ArgumentParser:** in support of richer command-line options than ‘config’ supports
- **ConcurrentMap:** adds an efficient concurrent map

GOOGLE SUMMER OF CODE 2021 PROJECTS

Divye Nayyar

Go-Style Channels

MENTORS

Michael Ferguson

Aniket Mathur

Lakshya Singh

Socket Library

MENTORS

Michael Ferguson

Krishna Dey

Ankush Bhardwaj

Prasanth Duvvuri

Matrix Exponentials

MENTORS

Lydia Duncan

Garvit Dewan

Engin Kayraklioglu

Nikhil Padmanabhan

CHAPEL 2.0

Background:

- For the past several years, we have been working toward a forthcoming Chapel 2.0 release
- Intent: stop making backward-breaking changes to core language and library features

Status:

- Major language-related changes have largely wound down
- Primary remaining effort is on stabilizing the standard libraries

CHAPEL 2.0: MODULE STABILIZATION

Module	1.24	1.25	1.26
Builtins	Review Started	✓ Stable	✓ Stable
ChplConfig*			Review Started
Heap			
List		Review Started	Review Started
Map		Review Started	Progress
Set			Review Started
FileSystem			
IO		Review Started	Progress
Path	Review Started	✓ Stable	✓ Stable
Reflection		Review Started	Progress
Types			Review Started
BigInteger		Review Started	Progress
Math			Review Started
Random			Review Started
Barriers		Review Started	Progress
CTypes*		Review Started	Progress
Subprocess*		Review Started	Progress
Sys		Review Started	Review Started
SysBasic		Review Started	Progress
SysError			
DateTime			Review Started
Regex*	Review Started	Progress	Progress
Time	Review Started	Progress	Progress
Version			Review Started
String / Bytes		Review Started	Progress
Ranges	Review Started	Progress	Progress
Domains	Review Started	Progress	Progress
Arrays		Review Started	Progress
Shared / Owned	Review Started	Review Started	Review Started
Errors			
Memory.MoveInitialization			
Locales			

✓ Stable   Progress   Review Started

USER PUBLICATION HIGHLIGHTS

USER PUBLICATION HIGHLIGHTS

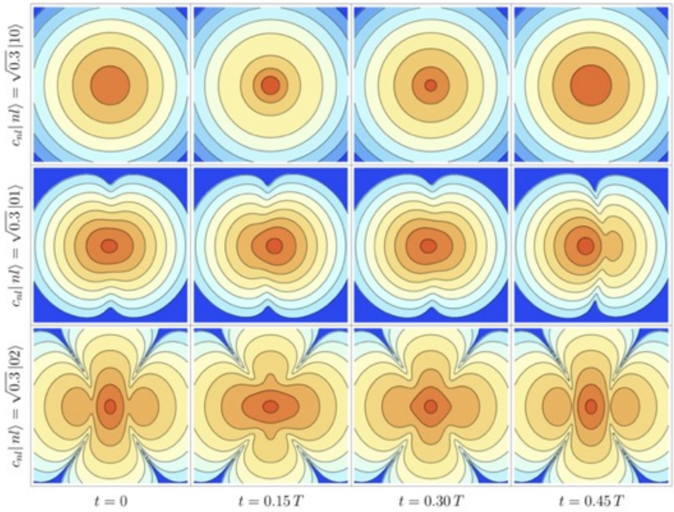
ChOp: Tiago Carneiro, Guillaume Helbecque, Jan Gmys, Loizos Koutsantonis, Nouredine Melab, Emmanuel Kieffer, Pascal Bouvry (*U. Luxembourg, Inria Lille*):

- **A performance-oriented comparative study of the Chapel high-productivity language to conventional programming environments**
  - *13th International Workshop on Programming Models and Applications for Multicores and Manycores* (PMAM 2022), Seoul, South Korea, April 2, 2022.
- **A Local Search for Automatic Parameterization of Distributed Tree Search Algorithms**
  - *12th IEEE Workshop Parallel / Distributed Combinatorics and Optimization (PDCO 2022)*, June 3, 2022.

ChplUltra: J. Luna Zagorac, Isabel Sands, Nikhil Padmanabhan, and Richard Easther (*Yale University, University of Auckland*)

- **Schrödinger-Poisson Solitons: Perturbation Theory**
  - ArXiv, September 4, 2021 / April 15, 2022
- **A Light in the Dark: UltraLight Dark matter Phenomenology in Simulations**
  - Ph.D. thesis, defended by Dr. J. Luna Zagorac on April 1, 2022

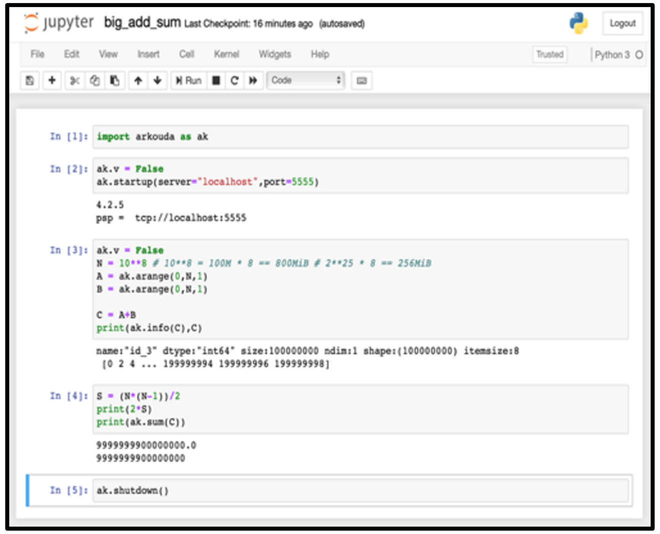
Talk by Luna @ 9:30



ARKOUDA HIGHLIGHTS

ARKOUDA'S HIGH-LEVEL APPROACH

Arkouda Client
(written in Python)



Arkouda Server
(written in Chapel)

User writes Python code in Jupyter, making NumPy/Pandas calls

KEY ARKOUDA FEATURES

- **Massive-Scale Data**
  - TB-sized arrays
- **Interactive Rates**
  - seconds to minutes per op
- **Extensible**
- **Fast and Scalable**
- **Open-Source:**
  - github.com/Bears-R-Us/arkouda/

benchmark	NumPy 0.75 GB	Arkouda (serial) 0.75 GB 1 core, 1 node	Arkouda (parallel) 0.75 GB 36 cores x 1 node	Arkouda (distributed) 384 GB 36 cores x 512 nodes
argsort	0.03 GiB/s --	0.05 GiB/s **1.66x**	0.50 GiB/s **16.7x**	55.12 GiB/s **1837.3x**
coargsort	0.03 GiB/s --	0.07 GiB/s **2.3x**	0.50 GiB/s **16.7x**	29.54 GiB/s **984.7x**
gather	1.15 GiB/s --	0.45 GiB/s 0.4x	13.45 GiB/s **11.7x**	539.52 GiB/s **469.1x**
reduce	9.90 GiB/s --	11.66 GiB/s **1.2x**	118.57 GiB/s **12.0x**	43683.00 GiB/s **4412.4x**
scan	2.78 GiB/s --	2.12 GiB/s 0.8x	8.90 GiB/s **3.2x**	741.14 GiB/s **266.6x**
scatter	1.17 GiB/s --	1.12 GiB/s 1.0x	13.77 GiB/s **11.8x**	914.67 GiB/s **781.8x**
stream	3.94 GiB/s --	2.92 GiB/s 0.7x	24.58 GiB/s **6.2x**	6266.22 GiB/s **1590.4x**

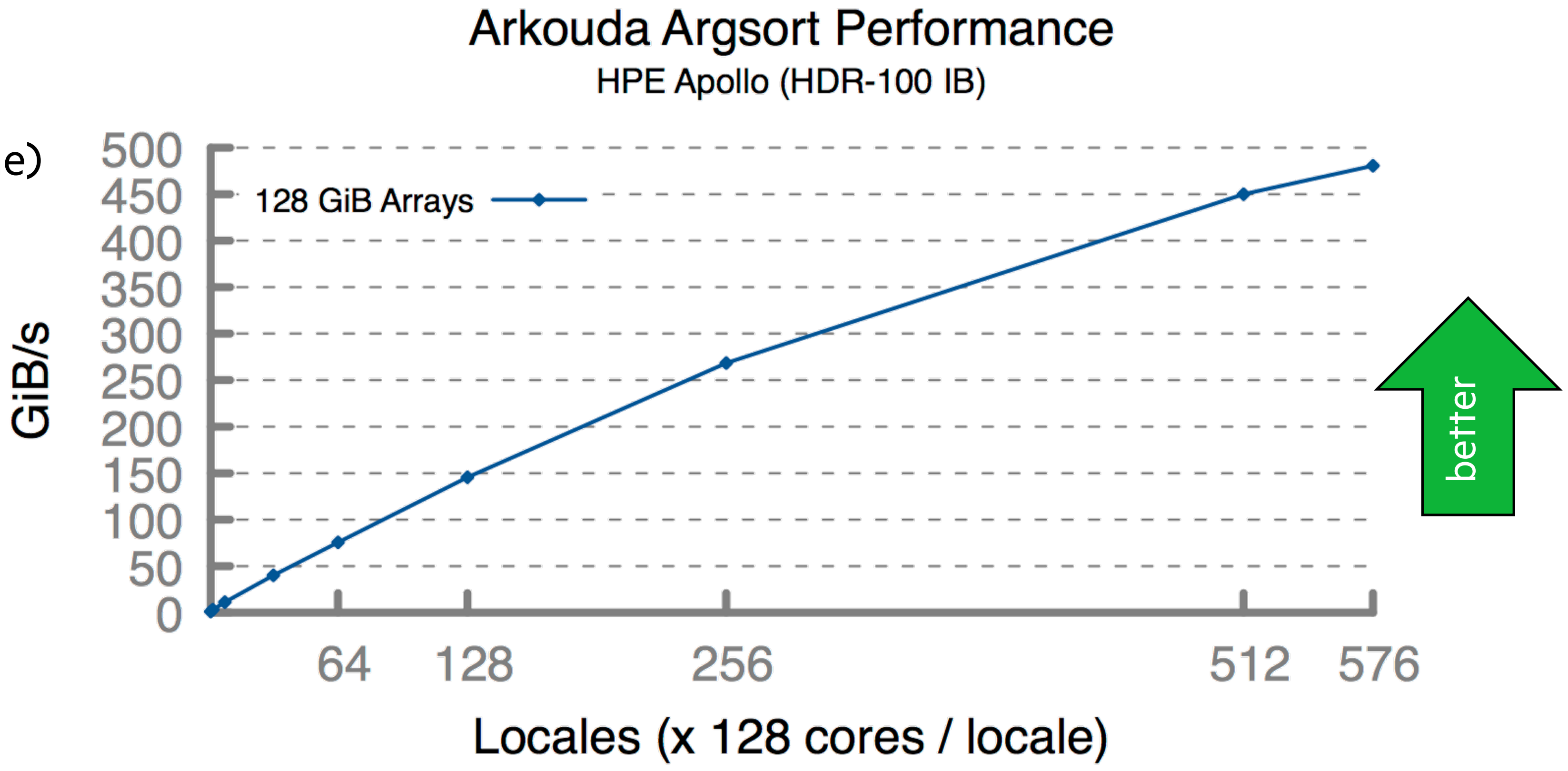
ARKOUDA HIGHLIGHTS SINCE CHIUW 2021

- **Dataframe support:** including support for additional types
- **Modular builds:** select feature set at Arkouda build-time
- **NJIT repository w/ graph capabilities** **Talk by Zhihui Du@ 2:40 PT**
- **Large-scale string processing improvements**
- **Parquet file I/O** **Talk by Ben McDonald @ 2:20 PT**
- **And much more:** See https://github.com/Bears-R-Us/arkouda/releases for details

- **Also, performance improvements…**

ARKOUDA ARGSORT AT MASSIVE SCALE

- Ran on a large Apollo system, summer 2022
  - 73,728 cores of AMD Rome
  - 72 TiB of 8-byte values
  - 480 GiB/s (2.5 minutes elapsed time)
  - ~100 lines of Chapel code



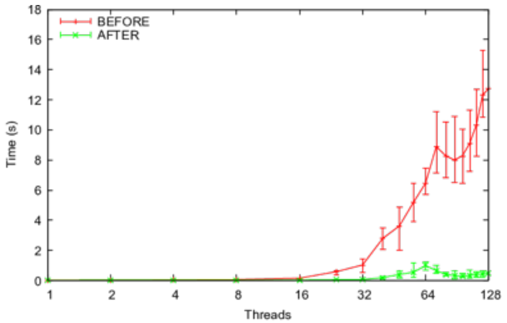
Close to world-record performance—quite likely a record for performance/SLOC

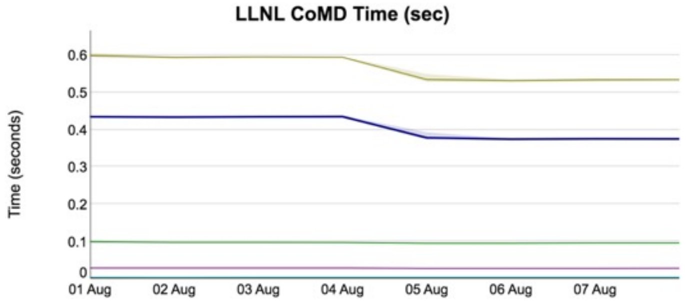
PERFORMANCE IMPROVEMENTS

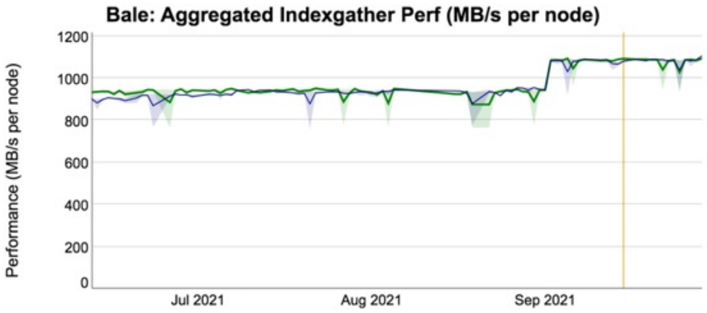
MANY, MANY PERFORMANCE IMPROVEMENTS

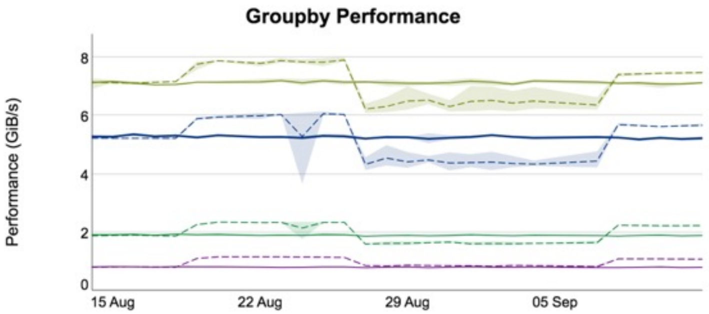
Primarily motivated by…

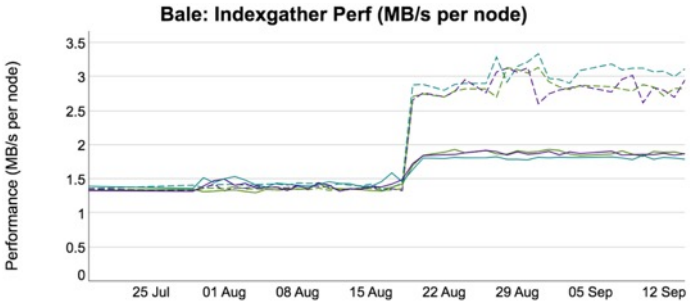
- …targeting new platforms
  - InfiniBand-based systems
  - high core-count chips like AMD Rome
  - large-memory nodes
- …user codes

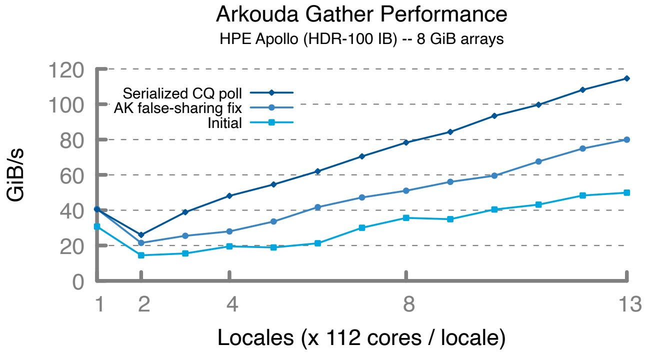


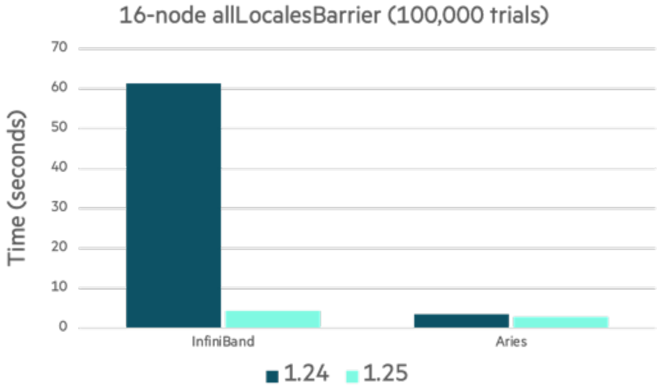


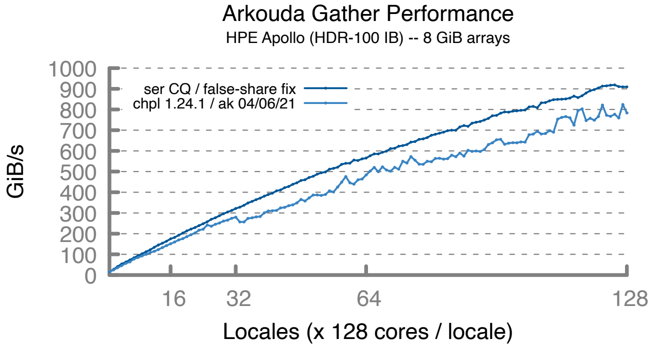












CHAPEL ON GPUS

CHAPEL ON GPUS

Background:

- GPUs have become a key feature in many HPC systems
- We have long described Chapel's goal as being "any parallel algorithm on any parallel hardware"
- Yet, historically, Chapel releases have only supported GPUs via interoperability
  - i.e., call GPU code written in CUDA, OpenCL, OpenMP, … as an extern routine

Talk by Akihiro Hayashi @ 12:35 PT

What's New?

- Lots of progress since CHIUW 2021…

CHAPEL FOR GPUS: CHAPEL 1.24 / CHIUW 2021

Targeting GPUs with Chapel was possible for the first time, but very low-level:

```
pragma "codegen for GPU"
export proc add_nums(A: c_ptr(real(64))){
  A[0] = A[0]+5;
}

var funcPtr = createFunction();
var A = [1, 2, 3, 4, 5];
__primitive("gpu kernel launch", funcPtr,
            <grid and block size>,…,
            c_ptrTo(A), …);
writeln(A);
```

```
extern {
  #define FATBIN_FILE "chpl__gpu.fatbin"
  double createFunction(){
    fatbinBuffer = <read FATBIN_FILE into buffer>
    cuModuleLoadData(&cudaModule, fatbinBuffer);
    cuModuleGetFunction(&function, cudaModule,
                        "add_nums");}

}
```

CHAPEL FOR GPUS: CHAPEL 1.25

Raised the level of abstraction significantly, yet with significant restrictions:

- only relatively simple computations
- single GPU only
- single locale only

```
on here.getChild(1) {
  var A = [1, 2, 3, 4, 5];
  forall a in A do
    a += 5;
}
```

CHAPEL FOR GPUS: CHAPEL 1.26

Improved generality: computational styles, multiple GPUs, CPU+GPU parallelism

```
cobegin {
  A[0..<cpuSize] += 1;    // do part of the work on the CPUs

  // simultaneously, do the rest of the work on the GPUs in parallel
  coforall subloc in 1..numGPUs do on here.getChild(subloc) {
    const myShare = cpuSize+gpuSize*(subloc-1)..#gpuSize;

    var AonThisGPU = A[myShare];    // copy a chunk of work to the unified memory
    AonThisGPU += 1;
    A[myShare] = AonThisGPU;        // copy the results back
  }
}
```

CHAPEL FOR GPUS: WHAT'S NEXT?

Coming up in Chapel 1.27:

- Targeting GPUs using multiple locales
- Improved representation of GPU sublocales
- Support for more general computations

Talk by Engin Kayraklioglu @ 12:35 PT

Thereafter:

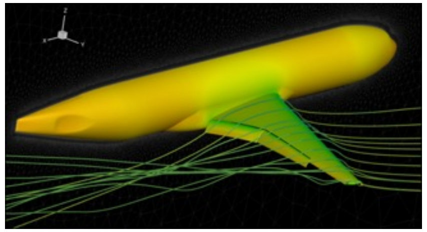
- Benchmarking, performance analysis, and optimization
- Portability across vendors (Nvidia-only today)
- Increasingly general computations

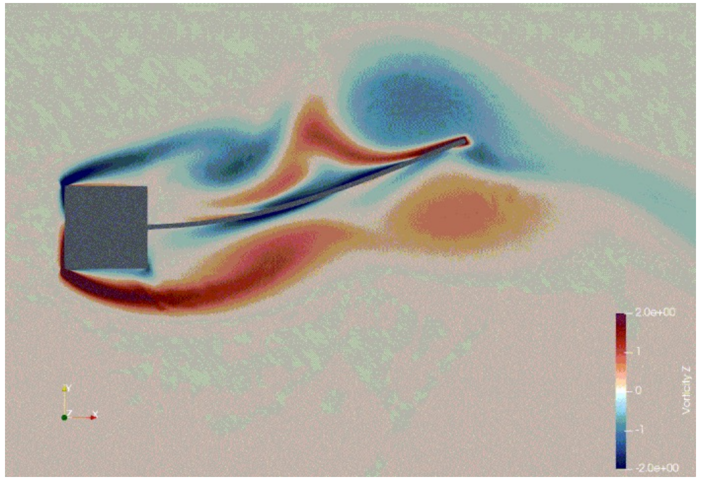

CHAMPS HIGHLIGHTS

CHAMPS SUMMARY

What is it?

- 3D unstructured CFD framework for airplane simulation
- ~100k lines of Chapel written from scratch in ~3 years

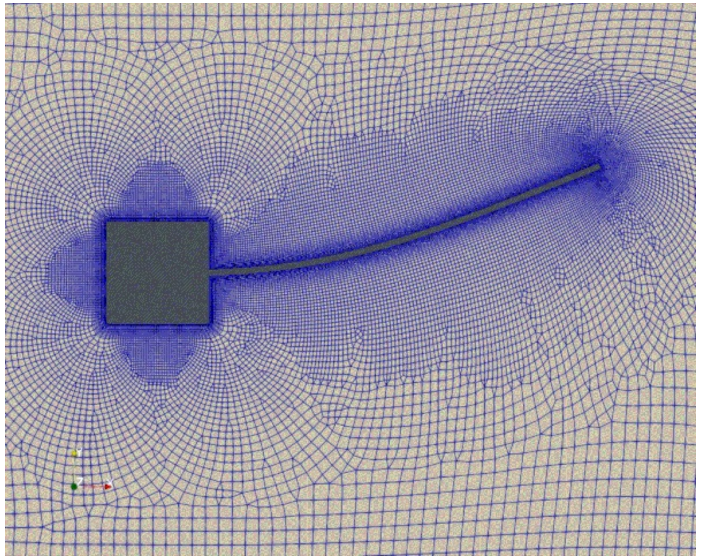




Who wrote it?

- Professor Éric Laurendeau’s students + postdocs at Polytechnique Montreal

POLYTECHNIQUE
MONTRÉAL



Why Chapel?

- performance and scalability competitive with MPI + C++
- students found it far more productive to use

(images provided by the CHAMPS team and used with permission)

CHAMPS: EXCERPT FROM ÉRIC'S CHIUW 2021 KEYNOTE

HPC Lessons From 30 Years of Practice in CFD Towards Aircraft Design and Analysis (June 4, 2021)

"To show you what Chapel did in our lab… [our previous framework] ended up 120k lines. And my students said, 'We can't handle it anymore. It's too complex, we lost track of everything.' And today, they went ***from 120k lines to 48k lines, so 3x less****.*

But the code is not 2D, it's 3D. And it's not structured, it's unstructured, which is way more complex. And it's multi-physics… ***So, I've got industrial-type code in 48k lines.****"*

"[Chapel] promotes the programming efficiency … ***We ask students at the master's degree to do stuff that would take 2 years and they do it in 3 months.*** *So, if you want to take a summer internship and you say, 'program a new turbulence model,' well they manage. And before, it was impossible to do."*

POLYTECHNIQUE MONTRÉAL

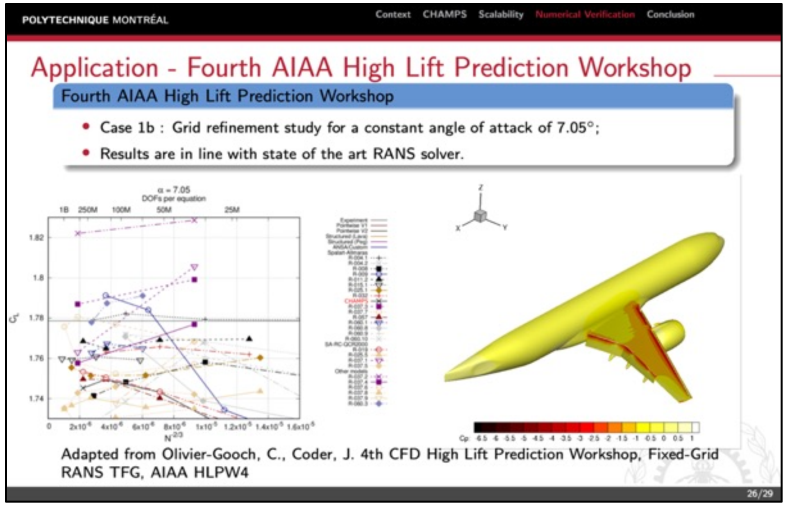
"So, for me, this is like the proof of the benefit of Chapel, ***plus the smiles I have on my students everyday in the lab because they love Chapel as well.*** *So that's the key, that's the takeaway."*

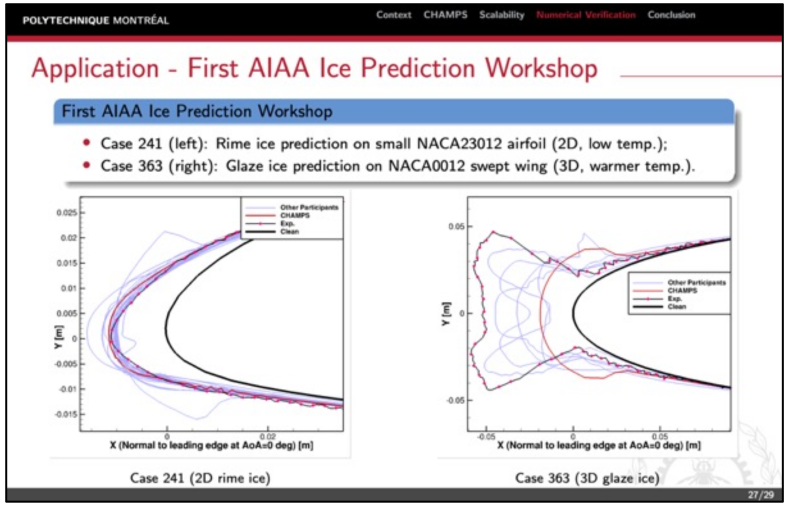
- Talk available online: https://youtu.be/wD-a_KyB8aI?t=1904 (hyperlink jumps to the section quoted here)

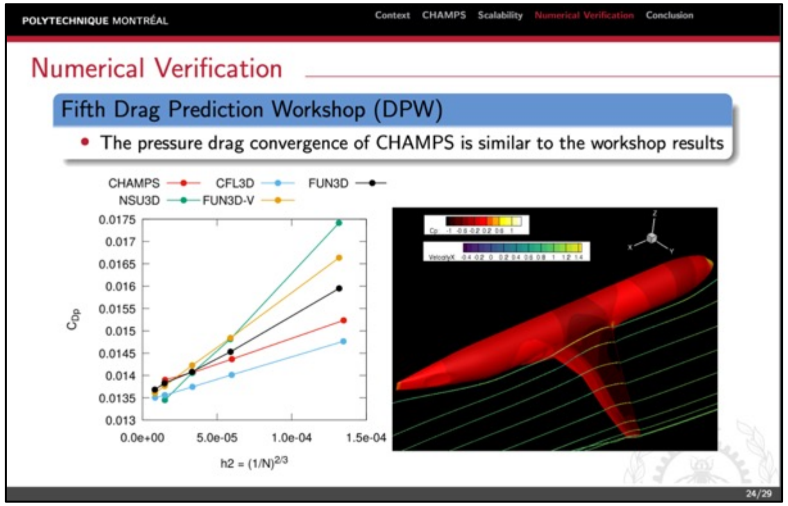
CHAMPS IN THE COMMUNITY SINCE CHIUW 2021

Community Activities:

- While on sabbatical, Éric has presented CHAMPS and Chapel at Université de Strasbourg and ONERA
- Student presentations at CASI/IASC Aero 21 Conference and to CFD Society of Canada (CFDSC)
- Team participated in the 4th AIAA High-lift Prediction Workshop and 1st AIAA Ice Prediction Workshop
  – Generating comparable results to high-profile sites: Boeing, Lockheed Martin, NASA, JAXA, Georgia Tech, …







CHAMPS HIGHLIGHTS SINCE CHIUW 2021

Progress since CHIUW 2021

- Code has more than doubled in size since CHIUW 2021
  - ~48k lines during Éric's CHIUW 2021 keynote
  - >100k lines now
  - contributions represent the work of ~7 students / postdocs
- Released CHAMPS 2.0
  - Many new features and capabilities

Talk by Frédéric Plante @ 9:10 PT

What's Next?

- Later this month, giving 6–7 presentations at the AIAA Aviation Forum and Exposition
- Éric will continue his sabbatical tour by presenting at DLR (German Aerospace Center)
- Participating in the 7th AIAA Drag Prediction Workshop

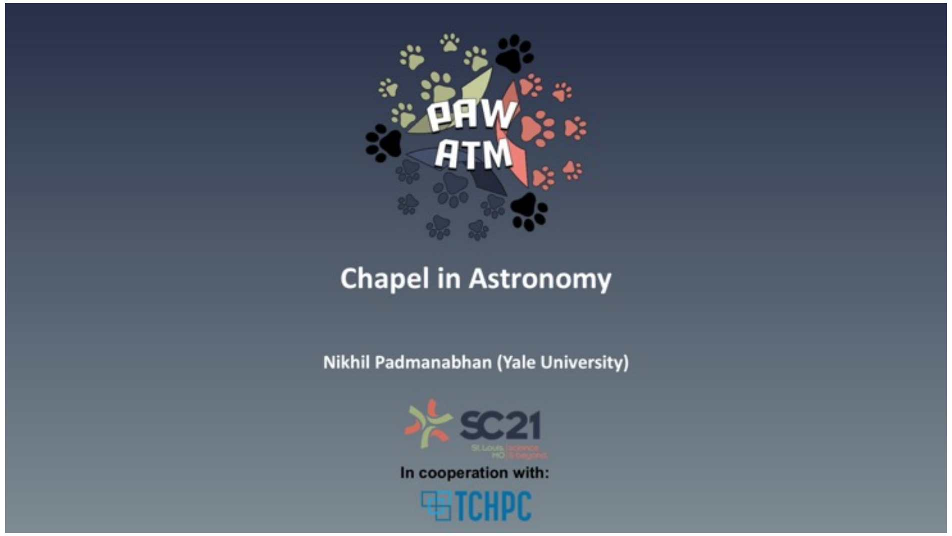
OUTREACH HIGHLIGHTS

CHAPEL AT PAW-ATM 2021 (AT SC21)

Thomas Rolinger published + presented paper

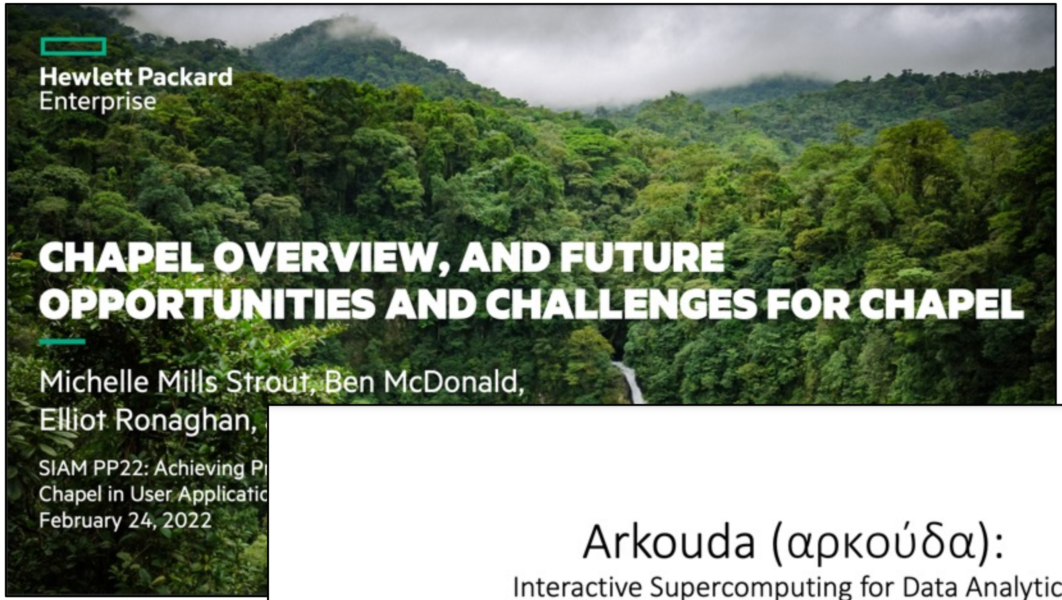
Talk by Thomas @ 1:30

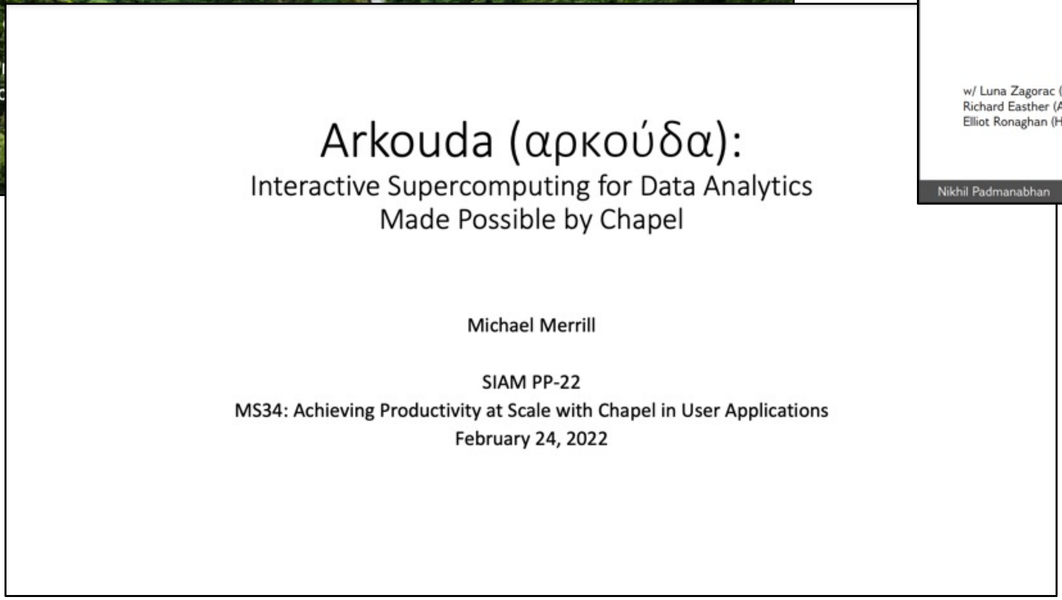
ChplUltra and CHAMPS PIs participated in PAW-ATM panel

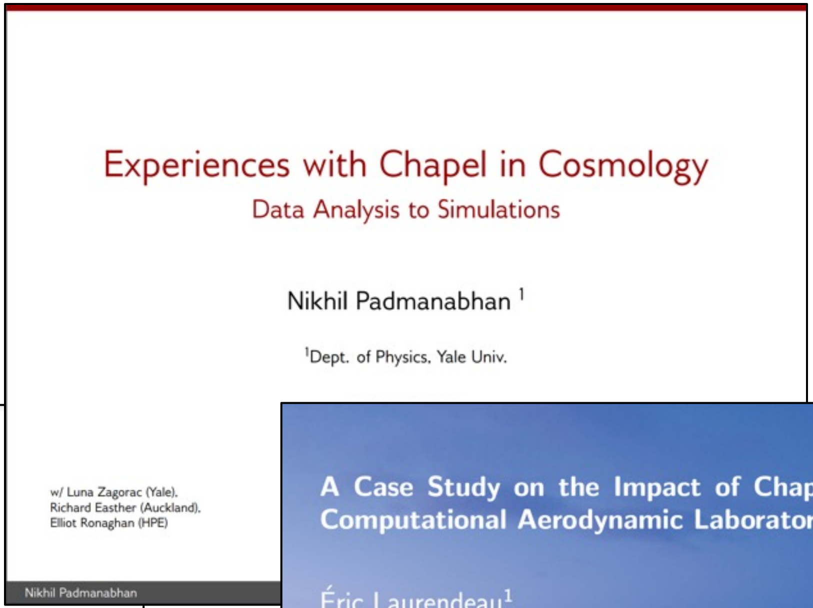


SIAM PP22 MINISYMPOSIUM

"Achieving Productivity at Scale with Chapel in User Applications", February 24, 2022







and many, many more talks as well; slides and often videos, available at https://chapel-lang.org/presentations.html

WHAT’S NEXT?

WHAT'S NEXT?

Tuning Chapel for HPE Cray EX Supercomputers / HPE Slingshot interconnect

- achieving promising initial performance, but additional tuning remains

Continuing three key efforts

- targeting GPUs
- 'dyno' compiler rework
- Chapel 2.0 stabilization

Growing the community

- supporting existing users and identifying new ones
- Advent of Code 2022 working group
- Chapel blog

The 5th Annual Parallel Applications Workshop, Alternatives To MPI+X

Held in conjunction with SC22

Deadline: July 29, 2022

Submission Styles: Papers, Talks, Pictures, Videos


TWEET OF THE YEAR

BEST CHAPEL TWEETS SINCE CHIUW 2021 (?)

Christopher D. Long (Крістофер Д. Лонг) @octonion · Apr 14 •••

The most underrated and overlooked programming language, in my opinion, is @ChapelLanguage. Primarily designed for supercomputing, it's always as fast or faster than any other language I've used, and it's very feature-rich with respect to abstract data structures.

Christopher D. Long (Крістофер Д. Лонг) @octonion · Apr 14 •••

Replying to @hipsterelectron and @ChapelLanguage

I've tried other languages I believe were targeting supercomputing - ParaSail, X10, Bigloo. All were a struggle to do anything. Chapel just works, and works well.

CHAPEL RESOURCES

Chapel homepage: https://chapel-lang.org

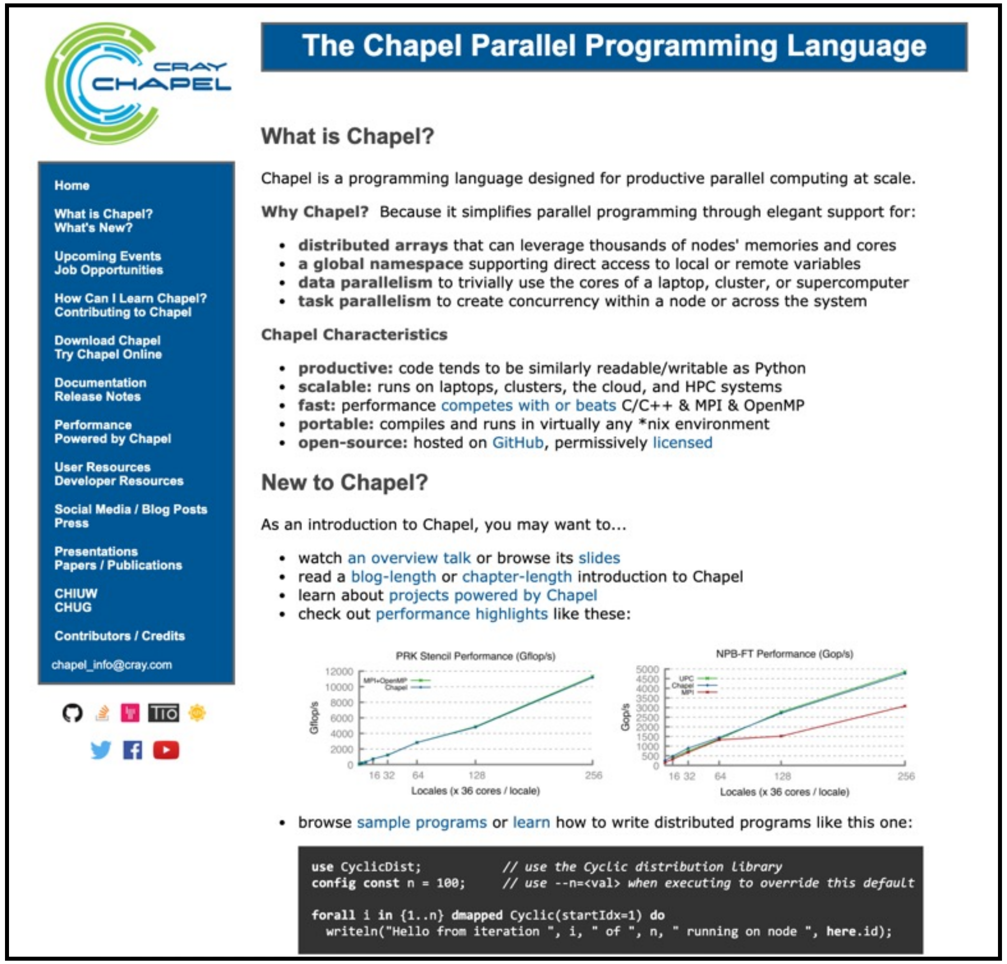
- (points to all other resources)

Social Media:

- Twitter: @ChapelLanguage
- Facebook: @ChapelLanguage
- YouTube: http://www.youtube.com/c/ChapelParallelProgrammingLanguage

Community Discussion / Support:

- Discourse: https://chapel.discourse.group/
- Gitter: https://gitter.im/chapel-lang/chapel
- Stack Overflow: https://stackoverflow.com/questions/tagged/chapel
- GitHub Issues: https://github.com/chapel-lang/chapel/issues



THANK YOU

https://chapel-lang.org
@ChapelLanguage