Hewlett Packard Enterprise

MAKING PARALLEL COMPUTING AS EASY AS PY(THON), FROM LAPTOPS TO SUPERCOMPUTERS

Brad Chamberlain, distinguished technologist

HPE Dev Munch & Learn

April 20, 2022

PARALLEL COMPUTING BASICS

Q: What is parallel computing?

A: Running an application using multiple processors in order to…

…run it faster and/or…

…run it using larger data sets…

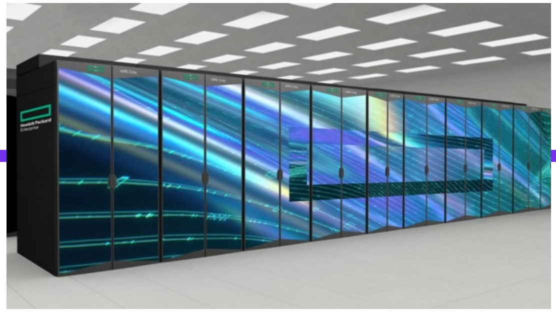
…than you could with just a single processor.

HPC =
High Performance Computing
(parallel computing at the largest scales)

Q: Where can I run parallel programs?

A: These days, everywhere:

- multi-core processors in laptops
- commodity clusters
- the cloud
- enterprise servers and supercomputers
  - HPE Apollo, HPE Superdome Flex, HPE Cray EX, …



Q: What are the main barriers to doing parallel computing?

A: Writing parallel programs is challenging by nature—and even more so for distributed memory systems

PARALLEL COMPUTING THAT'S AS EASY AS PYTHON?

Imagine having a programming language for parallel computing that was as…

…**programmable** as Python

…yet also as…

…**fast** as Fortran

…**scalable** as MPI or SHMEM

…**portable** as C

…**flexible** as C++

…**type-safe** as Fortran, C, C++, …

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

This is the motivation for the Chapel language

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

WHAT DO CHAPEL PROGRAMS LOOK LIKE?

helloTaskPar.chpl: print a message from each core in the system

```
coforall loc in Locales {
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n on %s\n",
                tid, numTasks, here.name);
  }
}
```

```
> chpl helloTaskPar.chpl
> ./helloTaskPar --numLocales=4
Hello from task 1 of 4 on n1032
Hello from task 4 of 4 on n1032
Hello from task 1 of 4 on n1034
Hello from task 2 of 4 on n1032
Hello from task 1 of 4 on n1033
Hello from task 3 of 4 on n1034
…
```

fillArray.chpl: declare and initialize a distributed array

```
use CyclicDist;

config const n = 1000;

const D = {1..n, 1..n}
            dmapped Cyclic(startIdx = (1,1));
var A: [D] real;

forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;

writeln(A);
```

```
> chpl fillArray.chpl
> ./fillArray --n=5 --numLocales=4
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```

KEY CHARACTERISTICS OF CHAPEL

- **compiled:** to generate the best performance possible
- **statically typed:** to avoid simple errors after hours of execution
- **interoperable:** with C, Fortran, Python, ...
- **portable:** runs on laptops, clusters, the cloud, supercomputers
- **open-source:** to lower barriers to adoption and leverage community contributions

CHAPEL RELEASES

Q: What is provided in a Chapel release?

A: Chapel releases contain…

…**the Chapel compiler** (‘chpl’)**:** translates Chapel source code into optimized executables

…**runtime libraries:** help map Chapel programs to a system’s capabilities (e.g., processors, network, memory, …)

…**library modules:** provide standard algorithms, data types, capabilities, …

…**documentation:** also available online at: https://chapel-lang.org/docs/

…**sample programs:** primers, benchmarks, etc.

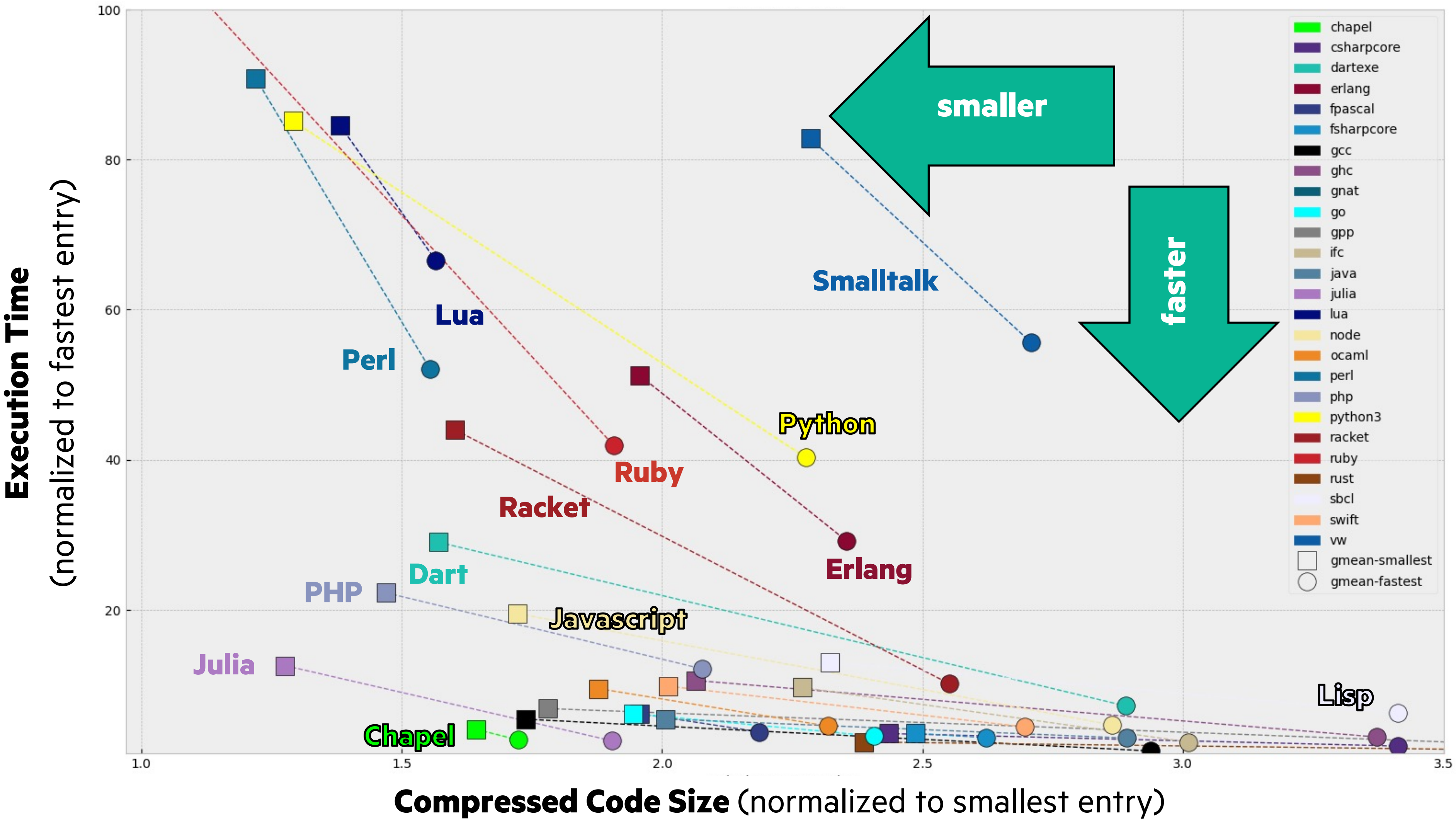
Q: How often is Chapel released? When is the next one?

A: New Chapel releases are made available every 3–6 months

- version 1.26.0 was released March 31, 2022

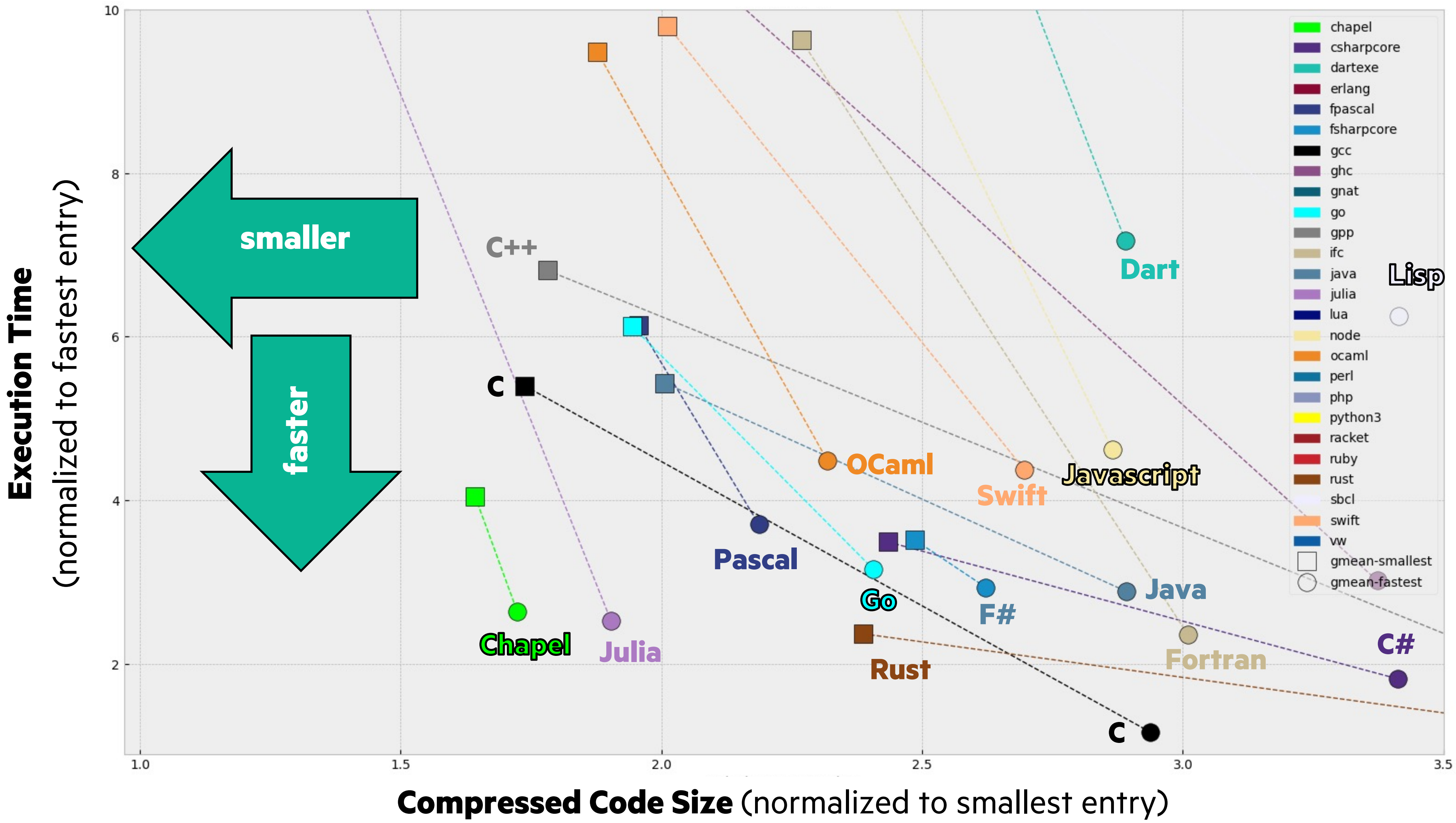
HOW DOES CHAPEL COMPARE TO OTHER PROGRAMMING LANGUAGES?

FOR DESKTOP BENCHMARKS, CHAPEL TENDS TO BE COMPACT AND FAST



(graph generated by scraping and summarizing data from the Computer Language Benchmarks Game on April 18, 2022)

FOR DESKTOP BENCHMARKS, CHAPEL TENDS TO BE COMPACT AND FAST

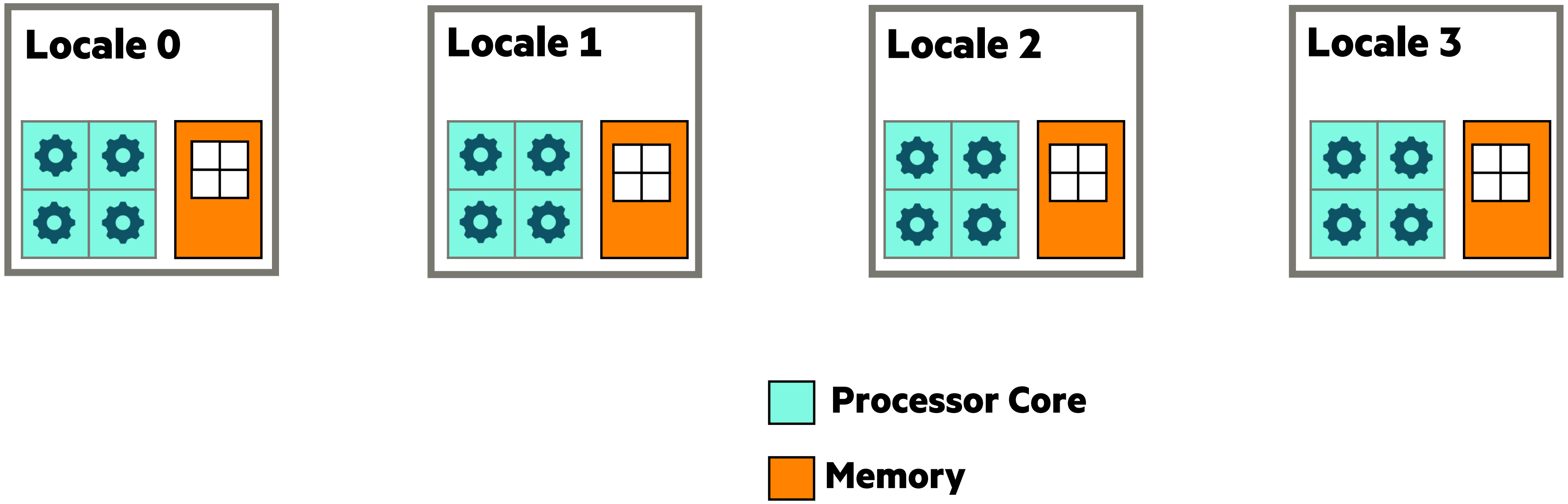


(graph generated by scraping and summarizing data from the Computer Language Benchmarks Game on April 18, 2022)

HOW DOES CHAPEL COMPARE TO PROGRAMMING APPROACHES USED FOR HPC?

KEY CONCERNS FOR SCALABLE PARALLEL COMPUTING

1. **parallelism:** What tasks should run simultaneously?
2. **locality:** Where should tasks run? Where should data be allocated?

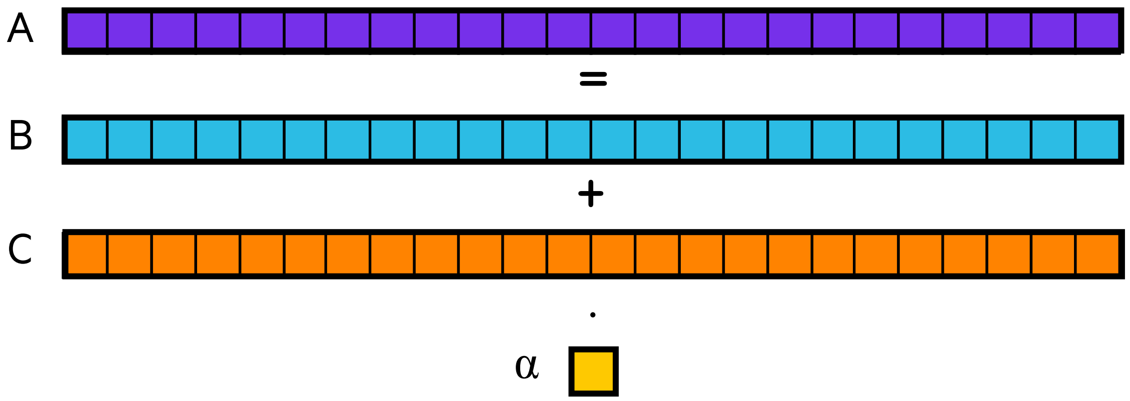


STREAM TRIAD: A TRIVIAL CASE OF PARALLELISM + LOCALITY

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m,\ A_i = B_i + \alpha \cdot C_i$

In pictures:

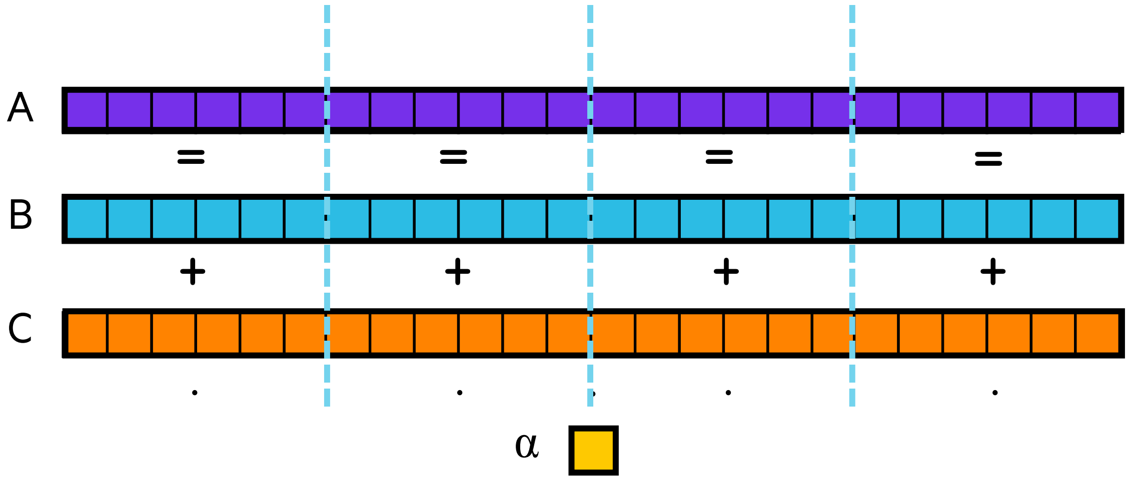


STREAM TRIAD: A TRIVIAL CASE OF PARALLELISM + LOCALITY

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m,\ A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel (shared memory / multicore)**:**

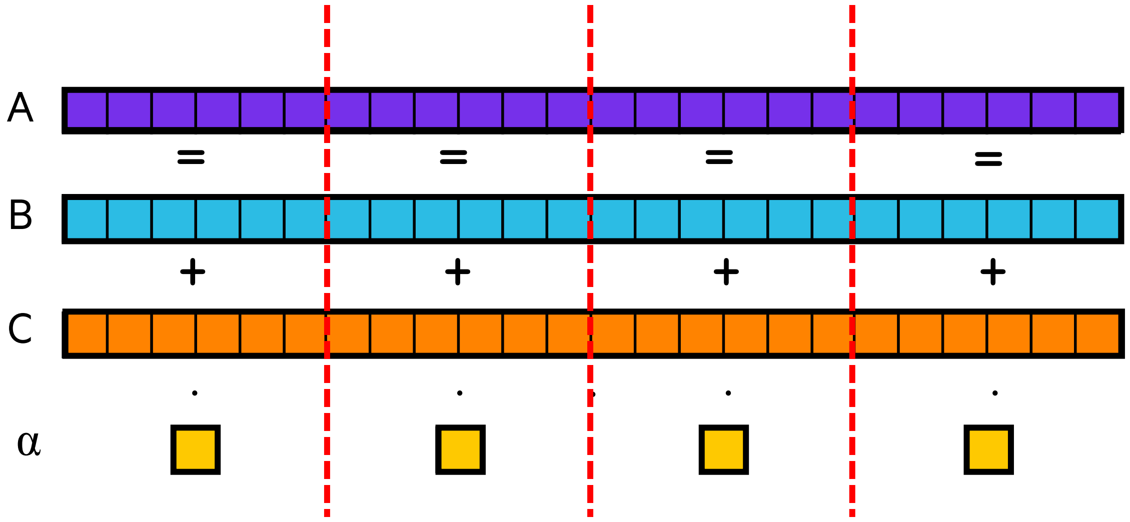


STREAM TRIAD: A TRIVIAL CASE OF PARALLELISM + LOCALITY

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m,\ A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel (distributed memory)**:**

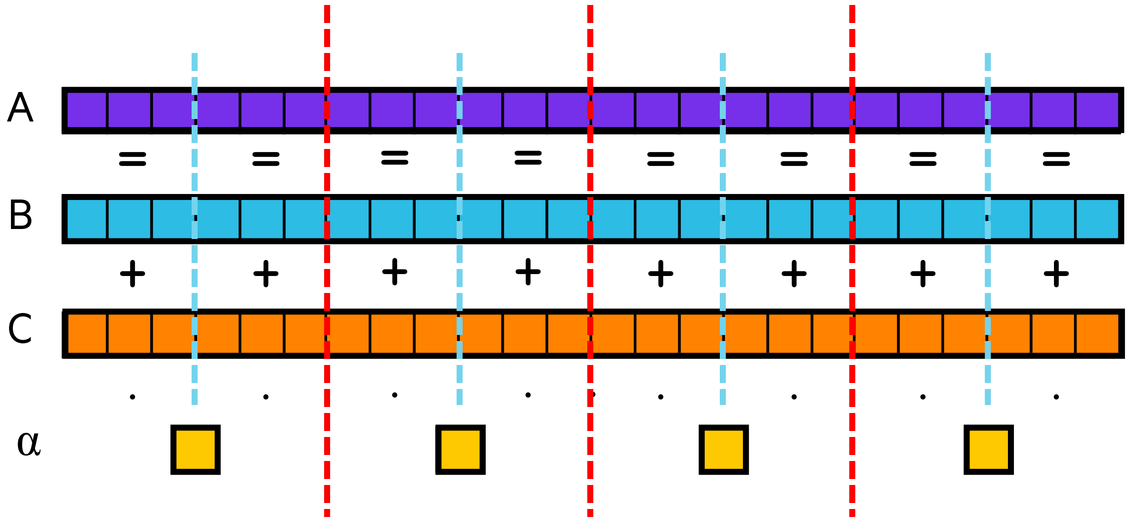


STREAM TRIAD: A TRIVIAL CASE OF PARALLELISM + LOCALITY

Given: m-element vectors A, B, C

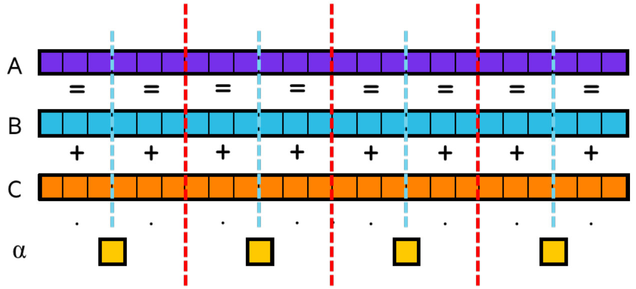
Compute: $\forall i \in 1..m,\ A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel (distributed memory multicore)**:**



STREAM TRIAD IN CONVENTIONAL HPC PROGRAMMING MODELS

Many Disparate Notations for Expressing Parallelism + Locality

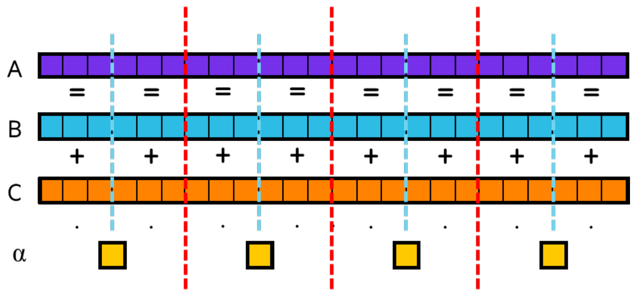


MPI

```c
#include <hpcc.h>

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 (%d).\n",
      VectorSize );
    fclose( outFile );
  }
  return 1;
}

for (j=0; j<VectorSize; j++) {
  b[j] = 2.0;
  c[j] = 1.0;
}
scalar = 3.0;

for (j=0; j<VectorSize; j++)
  a[j] = b[j]+scalar*c[j];

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

return 0;  }
```

STREAM TRIAD IN CONVENTIONAL HPC PROGRAMMING MODELS

Many Disparate Notations for Expressing Parallelism + Locality



MPI + OpenMP

```
#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 (%d).\n",
        VectorSize );
      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;  }
```

CUDA

```
#define N 2000000

int main() {
  float *d_a, *d_b, *d_c;
  float scalar;

  cudaMalloc((void**)&d_a, sizeof(float)*N);
  cudaMalloc((void**)&d_b, sizeof(float)*N);
  cudaMalloc((void**)&d_c, sizeof(float)*N);

  dim3 dimBlock(128);
  dim3 dimGrid(N/dimBlock.x );
  if( N % dimBlock.x != 0 ) dimGrid

  set_array<<<dimGrid,dimBlock>>>(d_b, .5f, N);
  set_array<<<dimGrid,dimBlock>>>(d_c, .5f, N);

  scalar=3.0f;
  STREAM_Triad<<<dimGrid,dimBlock>>>(d_b, d_c, d_a, scalar,  N);
  cudaThreadSynchronize();

  cudaFree(d_a);
  cudaFree(d_b);
  cudaFree(d_c);

__global__ void set_array(float *a,  float value, int len) {
  int idx = threadIdx.x + blockIdx.x * blockDim.x;
  if (idx < len) a[idx] = value;
}

__global__ void STREAM_Triad( float *a, float *b, float *c,
                              float scalar, int len) {
  int idx = threadIdx.x + blockIdx.x * blockDim.x;
  if (idx < len) c[idx] = a[idx]+scalar*b[idx];  }
```

Note: This is a trivial parallel computation—imagine the additional complexity for something more realistic…

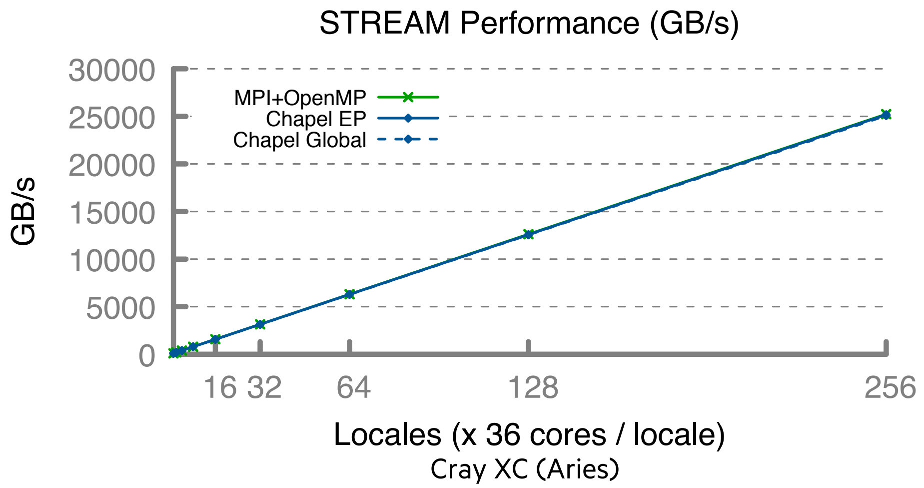
Challenge: Can we do better?

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

STREAM TRIAD: C + MPI + OPENMP

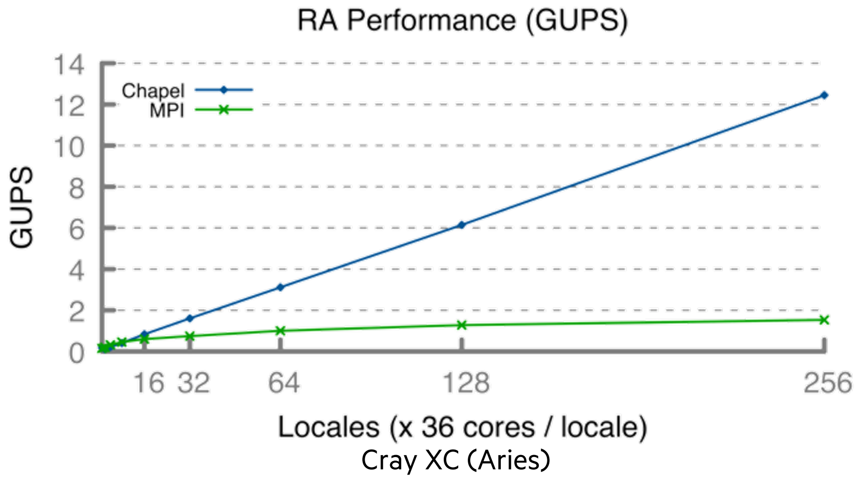
```c
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

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;
```



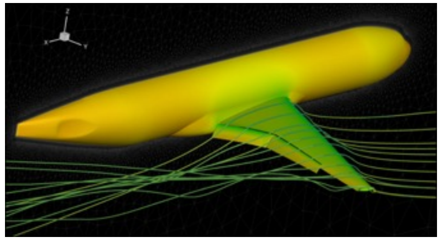
HPCC RA: MPI KERNEL

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

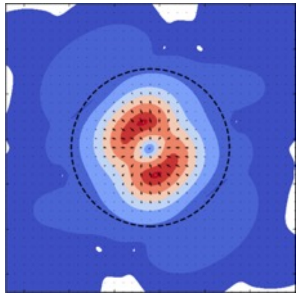


HOW IS CHAPEL BEING USED IN THE FIELD?

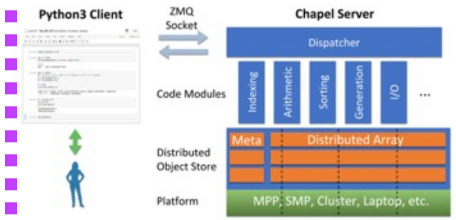
CURRENT FLAGSHIP CHAPEL APPLICATIONS



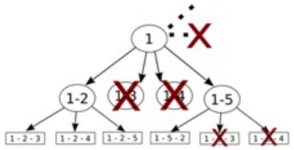
CHAMPS: 3D Unstructured CFD
Éric Laurendeau, Simon Bourgault-Côté, Matthieu Parenteau, et al.
École Polytechnique Montréal



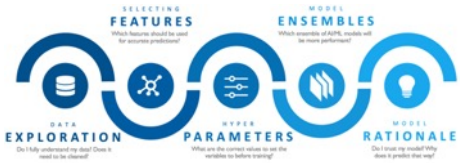
ChplUltra: Simulating Ultralight Dark Matter
Nikhil Padmanabhan, J. Luna Zagorac, *et al.*
Yale University / University of Auckland



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)

PARALLEL COMPUTING IN PYTHON?

Motivation: Say you've got…

…HPC-scale data science problems to solve

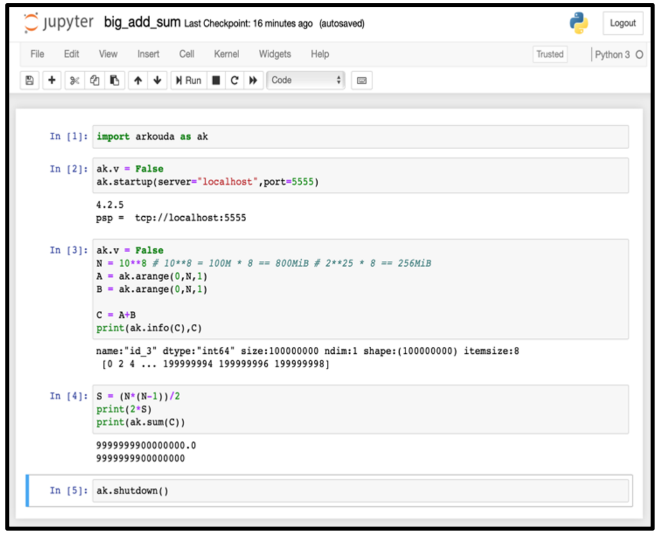
…a bunch of Python programmers

…access to HPC systems

How will you leverage your Python programmers to get your work done?

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

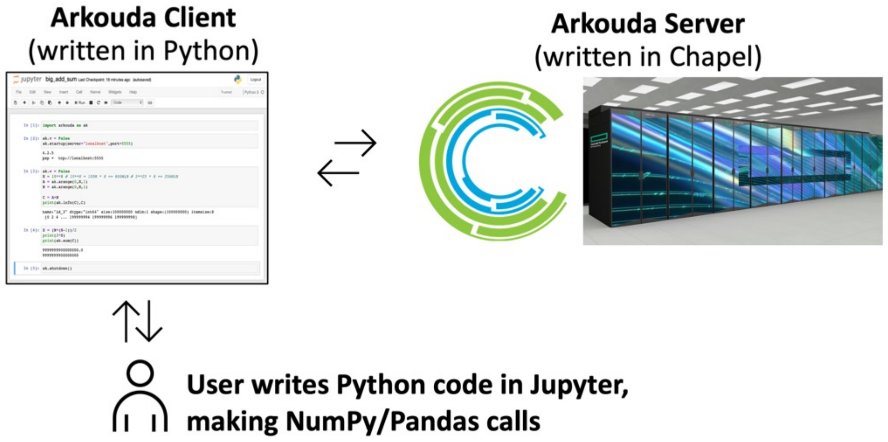
ARKOUDA SUMMARY

What is it?

- A Python library supporting a key subset of NumPy and Pandas for Data Science
  - Uses a Python-client/Chapel-server model to get scalability and performance
  - Computes massive-scale results (multi-TB-scale arrays) within the human thought loop (seconds to a few minutes)
- ~20k lines of Chapel, largely written in 2019, continually improved since then

Who wrote it?

- Mike Merrill, Bill Reus, *et al.*, US DoD
- Open-source: https://github.com/Bears-R-Us/arkouda



Why Chapel?

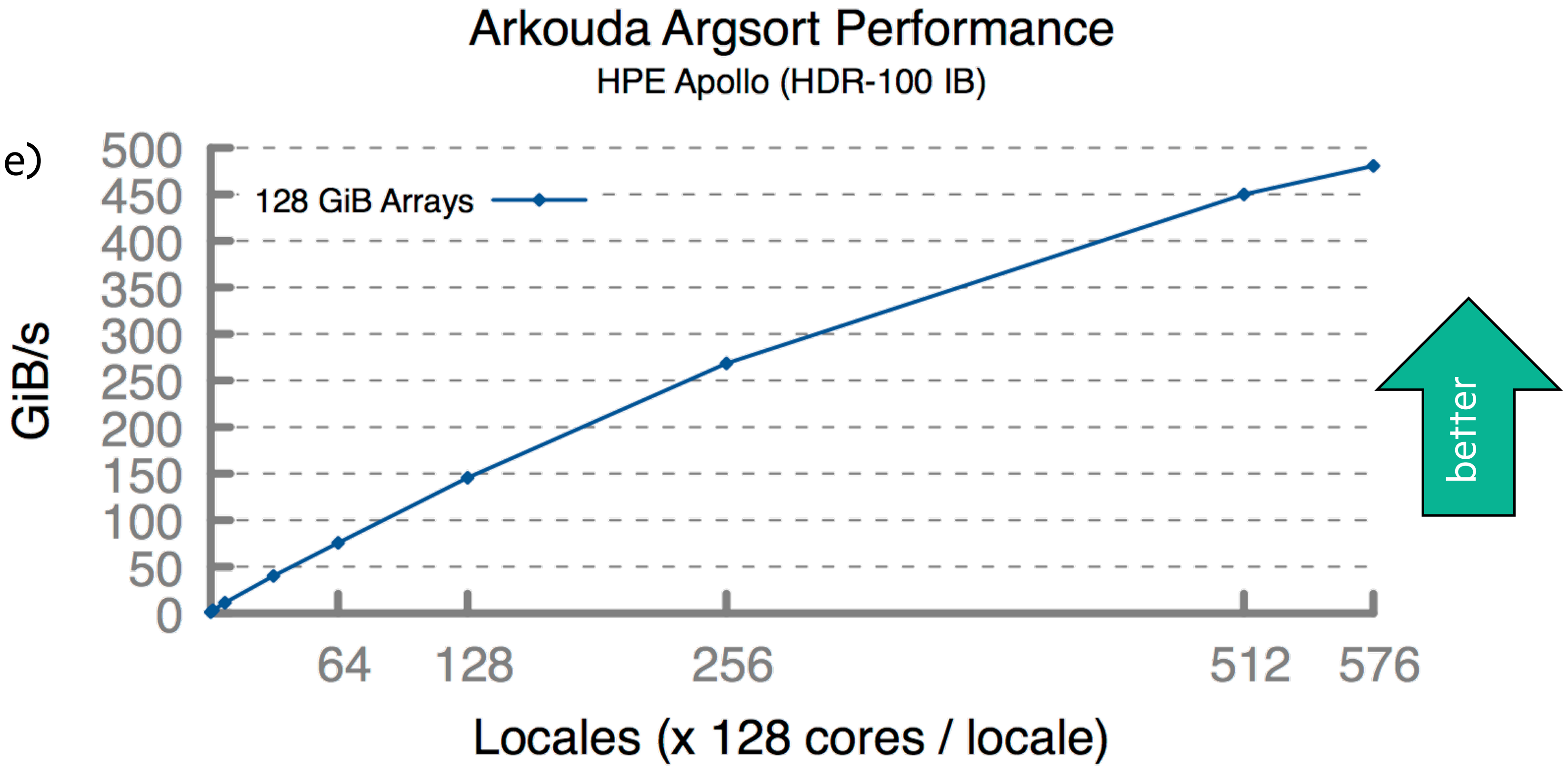
- high-level language with performance and scalability
- close to Pythonic
  - enabled writing Arkouda rapidly
  - doesn't repel Python users who look under the hood
- ports from laptop to supercomputer

ARKOUDA PERFORMANCE COMPARED TO NUMPY

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**

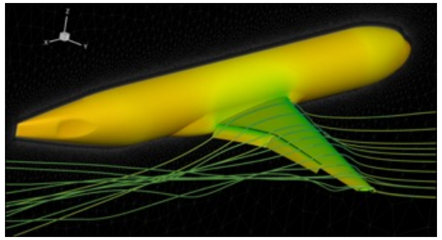
ARKOUDA ARGSORT AT MASSIVE SCALES

- Run 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

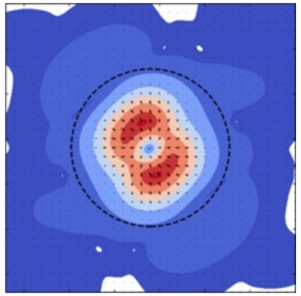


Close to world-record performance—quite likely a record for performance/SLOC

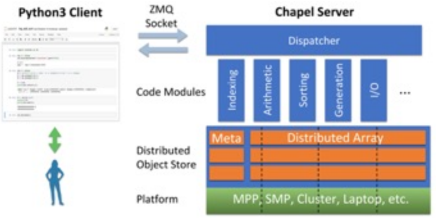
CURRENT FLAGSHIP CHAPEL APPLICATIONS



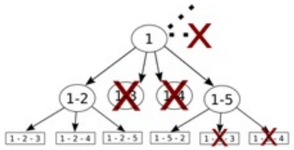
CHAMPS: 3D Unstructured CFD
Éric Laurendeau, Simon Bourgault-Côté, Matthieu Parenteau, et al.
École Polytechnique Montréal



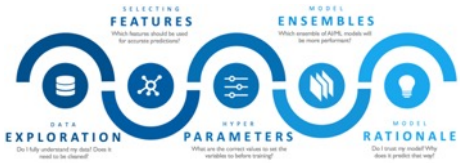
ChplUltra: Simulating Ultralight Dark Matter
Nikhil Padmanabhan, J. Luna Zagorac, *et al.*
Yale University / University of Auckland



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

?

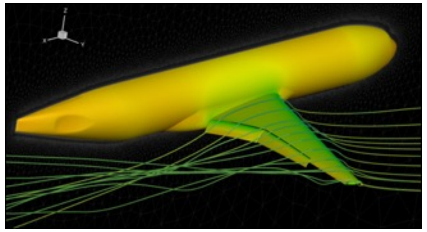
Your application here?

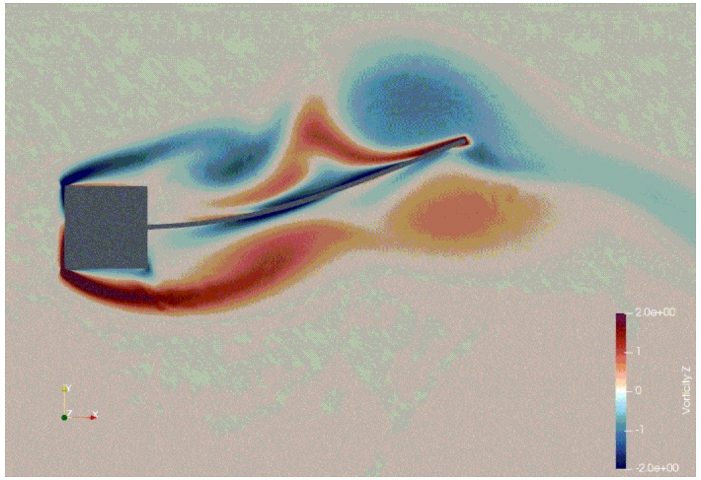
(images provided by their respective teams and used with permission)

CHAMPS SUMMARY

What is it?

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

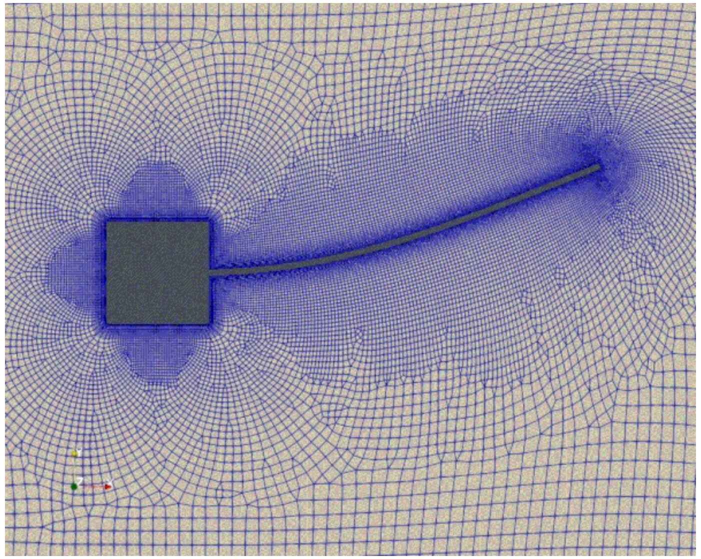




Who wrote it?

- 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."*

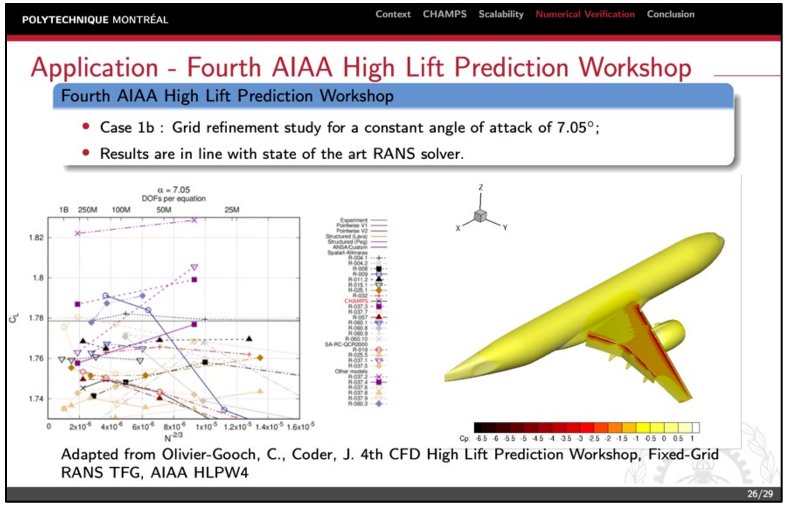
"So, for me, this is like the proof of the benefit of Chapel, ***plus the smiles I have on my students everyday in the lab because they love Chapel as well.*** *So that's the key, that's the takeaway."*

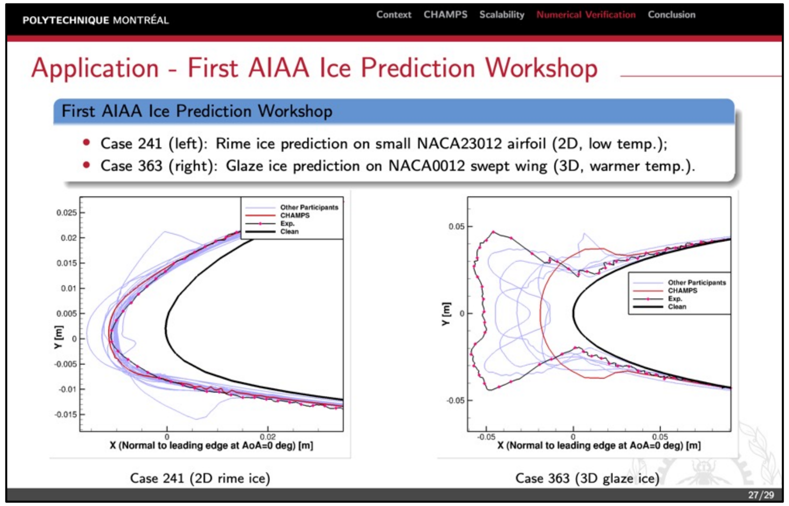
POLYTECHNIQUE MONTRÉAL

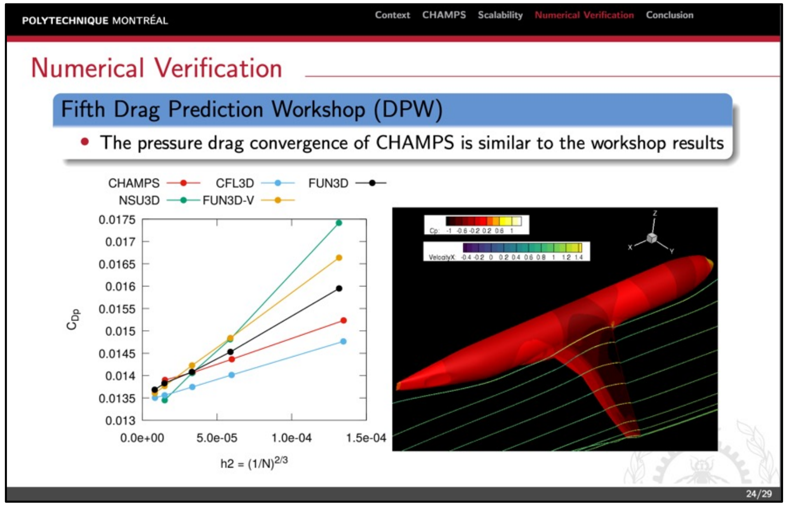
- Talk available online: https://youtu.be/wD-a_KyB8aI?t=1904 (hyperlink jumps to the section quoted here)

CHAMPS HIGHLIGHTS IN 2021

- Presented at CASI/IASC Aero 21 Conference
- Presented to CFD Society of Canada (CFDSC)
- Participated in 4th AIAA High-lift Prediction Workshops, 1st AIAA Ice Prediction Workshop
- Reproduced results from 5th AIAA Drag Prediction Workshop







- Generating results comparable to high-profile sites: Boeing, Lockheed Martin, NASA, JAXA, Georgia Tech, …

Looking ahead:

- giving 6–7 presentations at AIAA Aviation Forum and Exposition, June 2022
- participating in 7th AIAA Drag Prediction Workshop

(slide images taken from Éric Laurendeau's SIAM PP22 talk, A Case Study on the Impact of Chapel within an Academic Computational Aerodynamic Laboratory, with permission)

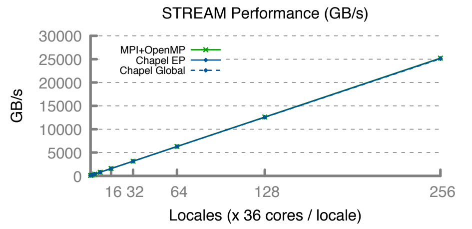
WRAPPING UP

SUMMARY

Chapel is unique among programming languages

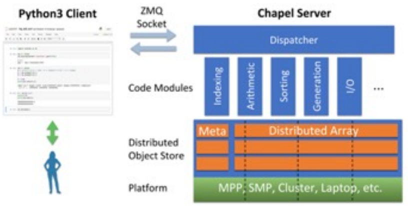
- built-in features for scalable parallel computing make it HPC-ready
- supports clean, concise code relative to conventional approaches
- ports and scales from laptops to supercomputers

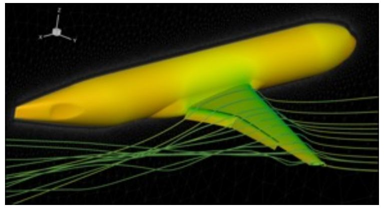
```
use BlockDist;

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;
```



Chapel is being used for productive parallel programming at scale

- users are reaping its benefits in practical, cutting-edge applications
- Arkouda lets Python programmers drive supercomputers from Jupyter





If you're interested in taking Chapel for a spin, let us know!

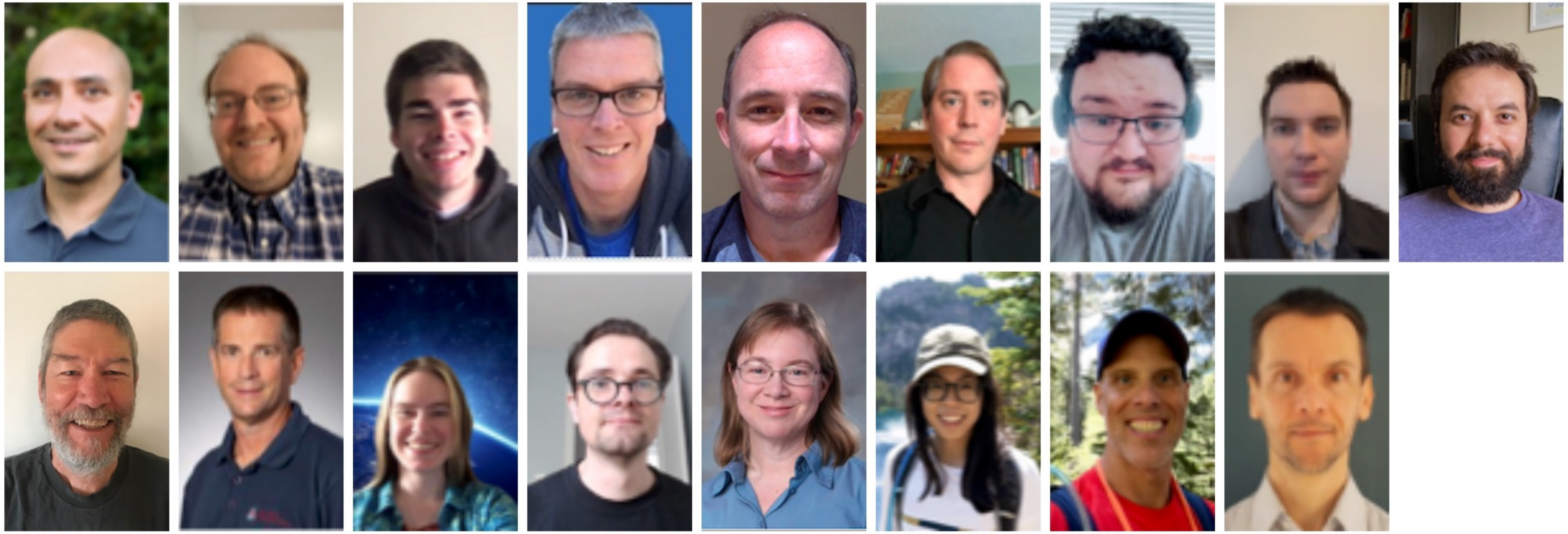
- we're happy to work with users and user groups to ease the learning curve

THE CHAPEL TEAM

Chapel is a team effort—currently made up of 14 full-time employees, 2 part-time, and our director

- we also have 3 more full-time engineers joining in the next few months, and 2 open positions

Chapel Development Team at HPE



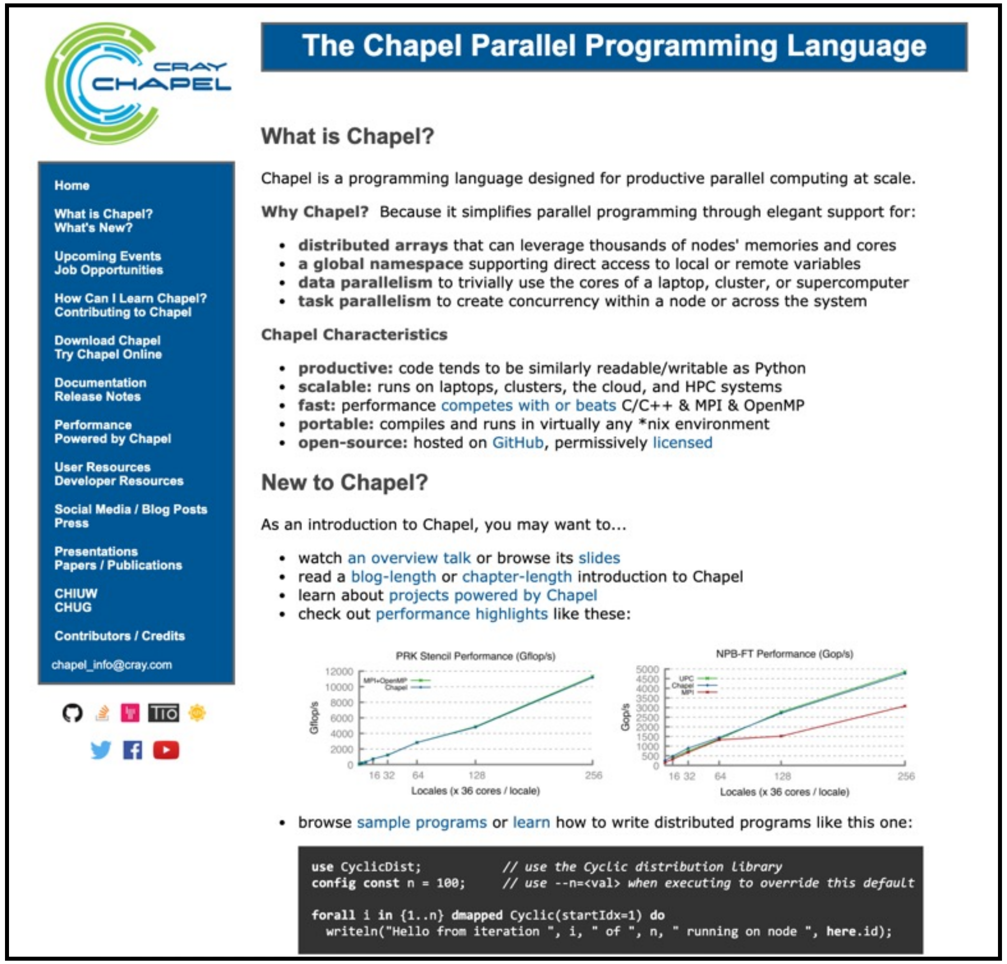
see: https://chapel-lang.org/contributors.html
and https://chapel-lang.org/jobs.html

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

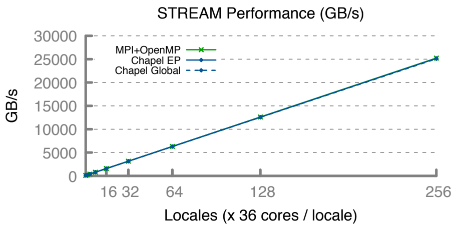


SUMMARY

Chapel is unique among programming languages

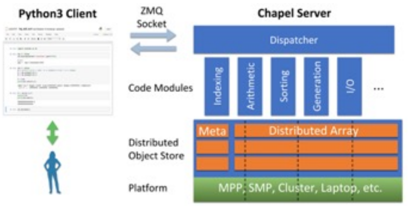
- built-in features for scalable parallel computing make it HPC-ready
- supports clean, concise code relative to conventional approaches
- ports and scales from laptops to supercomputers

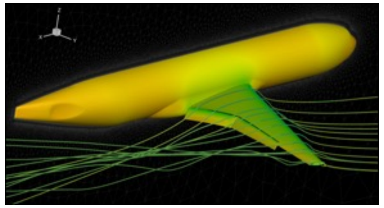
```
use BlockDist;

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;
```



Chapel is being used for productive parallel programming at scale

- users are reaping its benefits in practical, cutting-edge applications
- Arkouda lets Python programmers drive supercomputers from Jupyter





If you're interested in taking Chapel for a spin, let us know!

- we're happy to work with users and user groups to ease the learning curve

THANK YOU

https://chapel-lang.org
@ChapelLanguage