Hewlett Packard Enterprise

PRACTICAL EXAMPLES OF PRODUCTIVITY AND PERFORMANCE IN CHAPEL

Brad Chamberlain

PASC'23
June 26, 2023

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

PRODUCTIVE PARALLEL PROGRAMMING: A POTENTIAL DEFINITION

Imagine a programming language for parallel computing that was as…

…**programmable** as Python

…yet also as…

…**fast** as Fortran/C/C++

…**scalable** as MPI/SHMEM

…**GPU-ready** as CUDA/OpenMP/OpenCL/OpenACC/…

…**portable** as C

OUTLINE

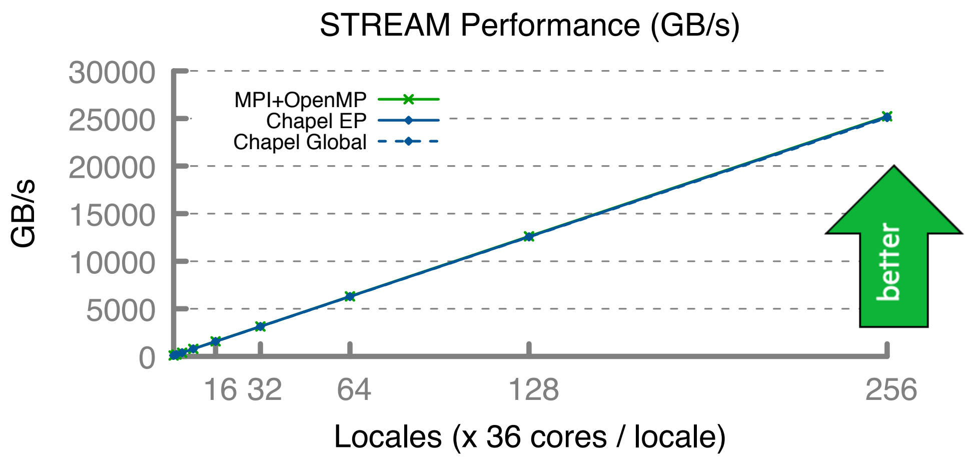
- What is Chapel?
- Applications of Chapel
- Chapel Intro on CPUs and GPUs
- Wrap-up

CHAPEL IS COMPACT, CLEAR, AND COMPETITIVE

STREAM TRIAD: C + MPI + OPENMP

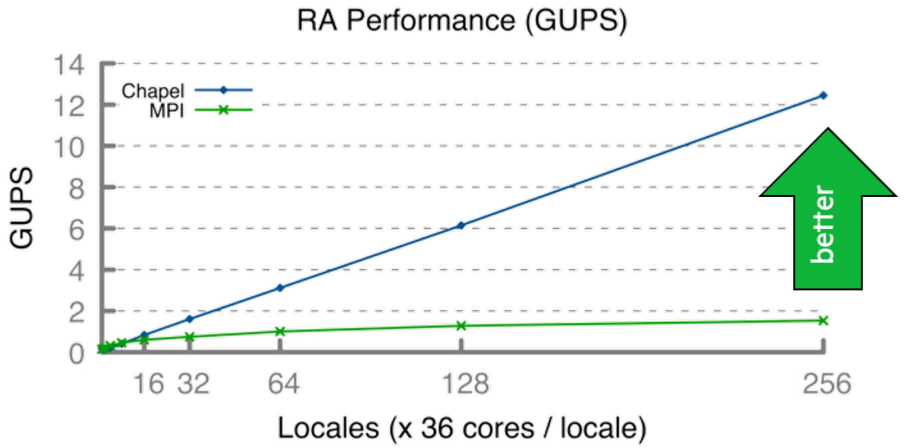
```
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

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 );
```

```
use BlockDist;

config const n = 1_000_000,
             alpha = 0.01;
const Dom = Block.createDomain({1..n});
var A, B, C: [Dom] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```

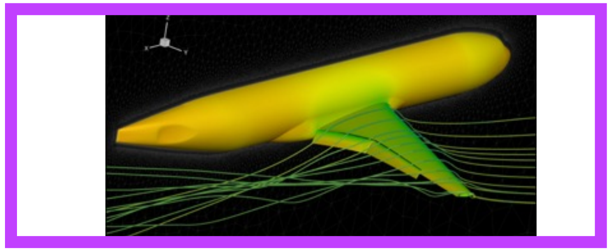


HPCC RA: MPI KERNEL

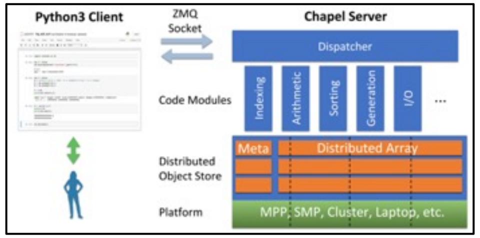
```
…
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask].xor(r);
…
```



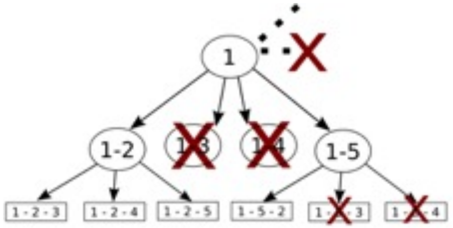
APPLICATIONS OF CHAPEL



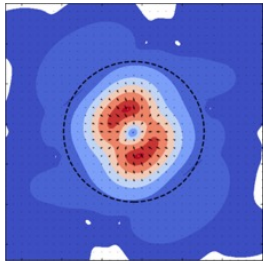
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



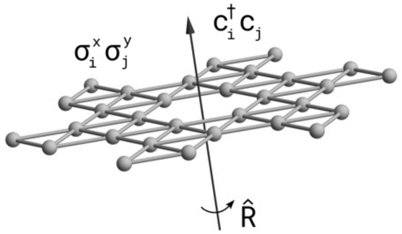
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



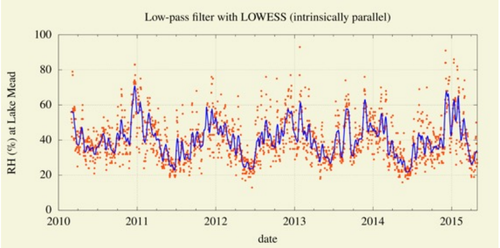
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

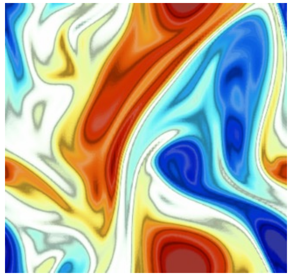


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

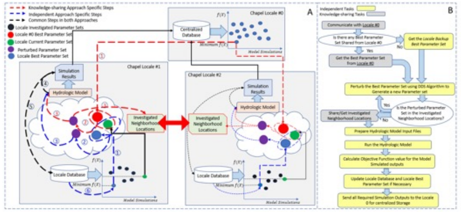


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

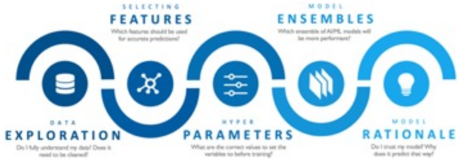
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



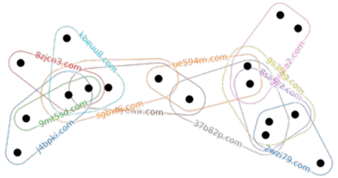
ChapQG: Layered Quasigeostrophic CFD
Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



CrayAI HyperParameter Optimization (HPO)
Ben Albrecht et al.
Cray Inc. / HPE



CHGL: Chapel Hypergraph Library
Louis Jenkins, Cliff Joslyn, Jesun Firoz, et al.
PNNL

?

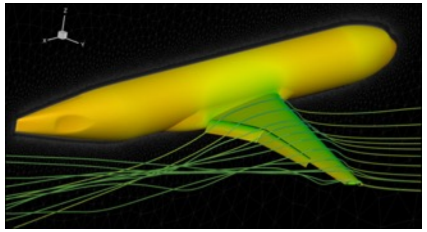
Your Application Here?

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

CHAMPS SUMMARY

What is it?

- 3D unstructured CFD framework for airplane simulation
- ~85k lines of Chapel written from scratch in ~3 years



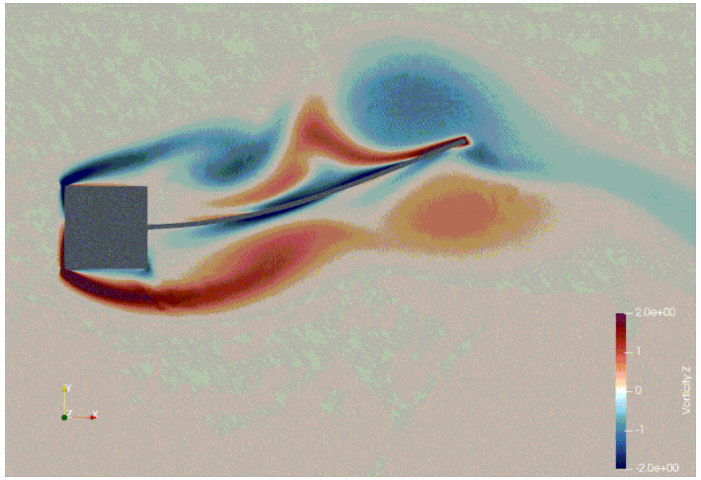
Who wrote it?

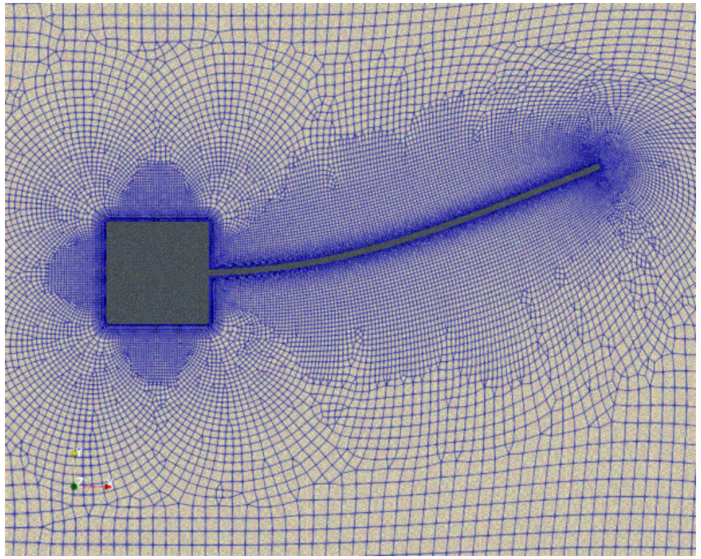
- Professor Éric Laurendeau’s students + postdocs at Polytechnique Montreal

POLYTECHNIQUE MONTRÉAL

Why Chapel?

- performance and scalability competitive with MPI + C++
- students found it far more productive to use
- enabled them to compete with more established CFD centers

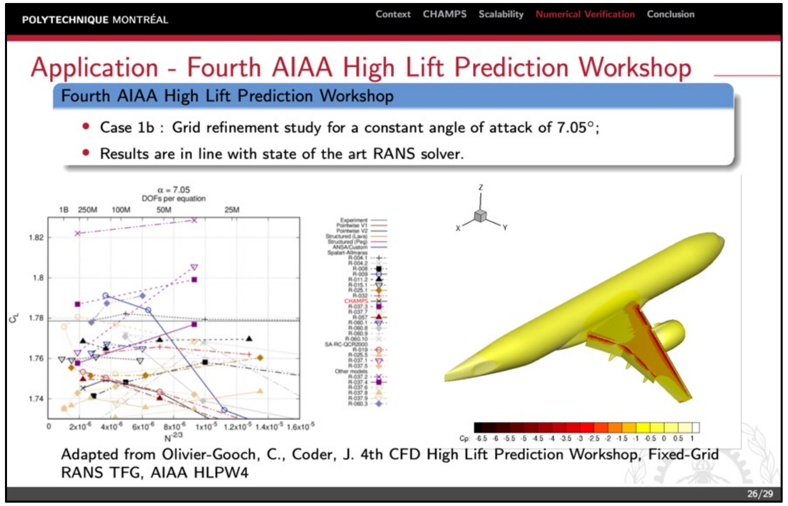


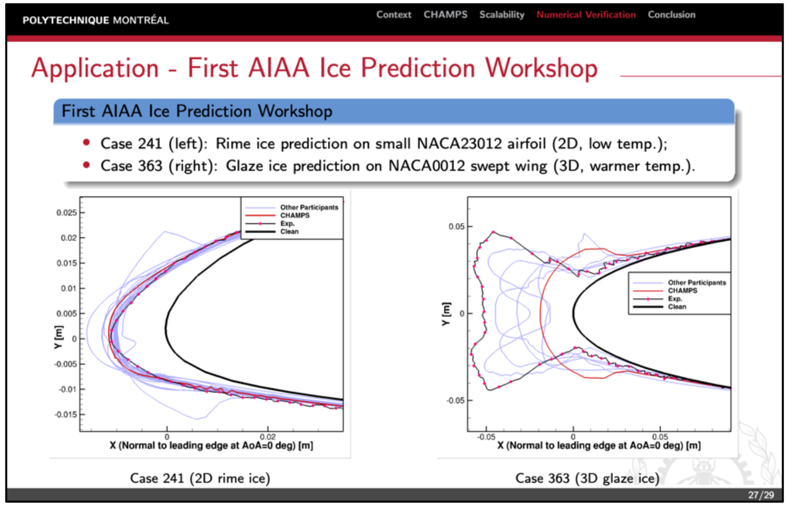


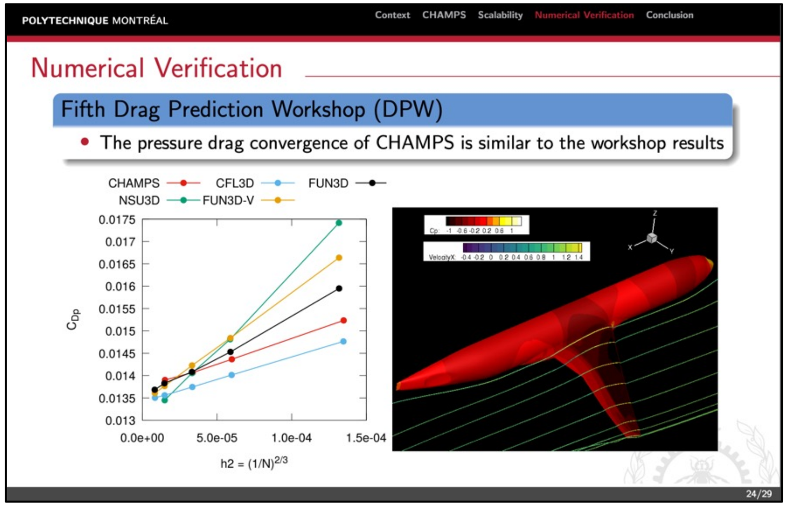
(images provided by the CHAMPS team and used with permission)

CHAMPS COMMUNITY HIGHLIGHTS

- Team participated in the **7^{th} AIAA High-lift Prediction Workshop** and **1^{st} AIAA Ice Prediction Workshop**
  - Generating comparable results to high-profile sites: Boeing, Lockheed Martin, NASA, JAXA, Georgia Tech, …
- Five papers published last summer at **2022 AIAA Aviation**
- While on sabbatical, Éric presented CHAMPS and Chapel at **ONERA**, **DLR**, **Université de Strasbourg**, …
- Student presentations at **CASI/IASC Aero 21 Conference** and to **CFD Society of Canada (CFDSC)**







CHAMPS: EXCERPT FROM ÉRIC'S CHIUW 2021 KEYNOTE (TRANSCRIPT)

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."*

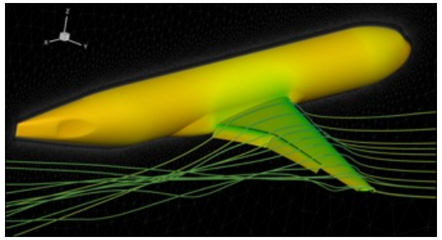
POLYTECHNIQUE MONTRÉAL

*"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."*

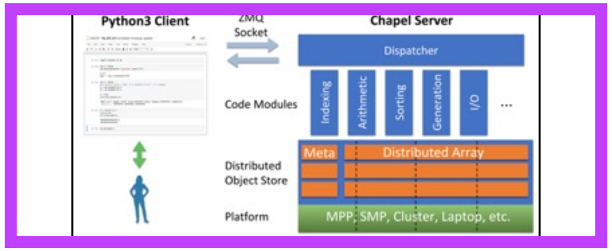
- Talk available online: https://youtu.be/wD-a_KyB8aI?t=1904 (hyperlink jumps to the section quoted here)

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

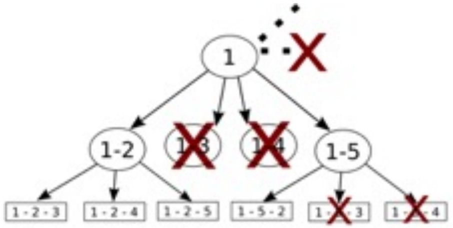
APPLICATIONS OF CHAPEL



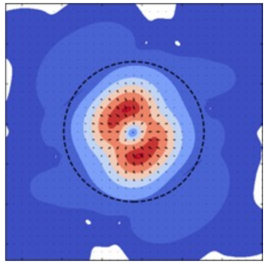
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



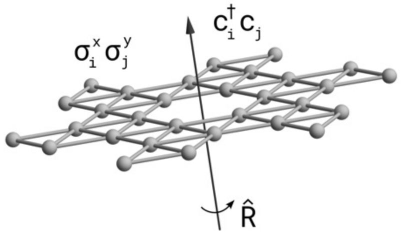
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



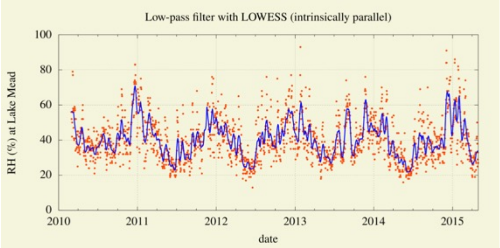
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

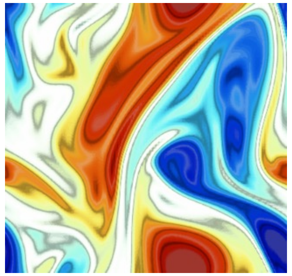


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

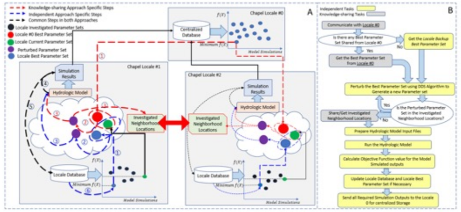


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

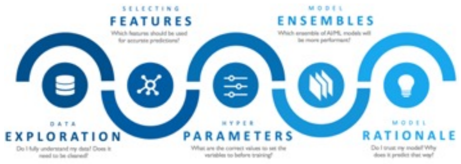
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



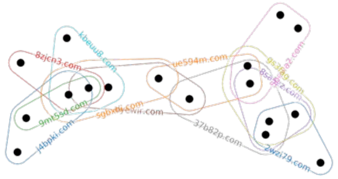
ChapQG: Layered Quasigeostrophic CFD
Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



CrayAI HyperParameter Optimization (HPO)
Ben Albrecht et al.
Cray Inc. / HPE



CHGL: Chapel Hypergraph Library
Louis Jenkins, Cliff Joslyn, Jesun Firoz, et al.
PNNL

?

Your Application Here?

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

DATA SCIENCE IN PYTHON AT SCALE?

Motivation: Imagine you've got…

…HPC-scale data science problems to solve

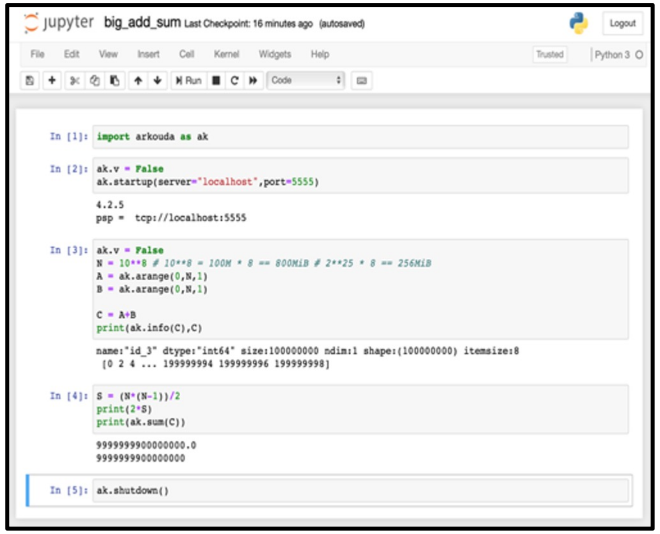
…a bunch of Python programmers

…access to HPC systems

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

ARKOUDA: A PYTHON FRAMEWORK FOR INTERACTIVE HPC

Arkouda Client
(written in Python)



Arkouda Server
(written in Chapel)

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

ARKOUDA SUMMARY

What is it?

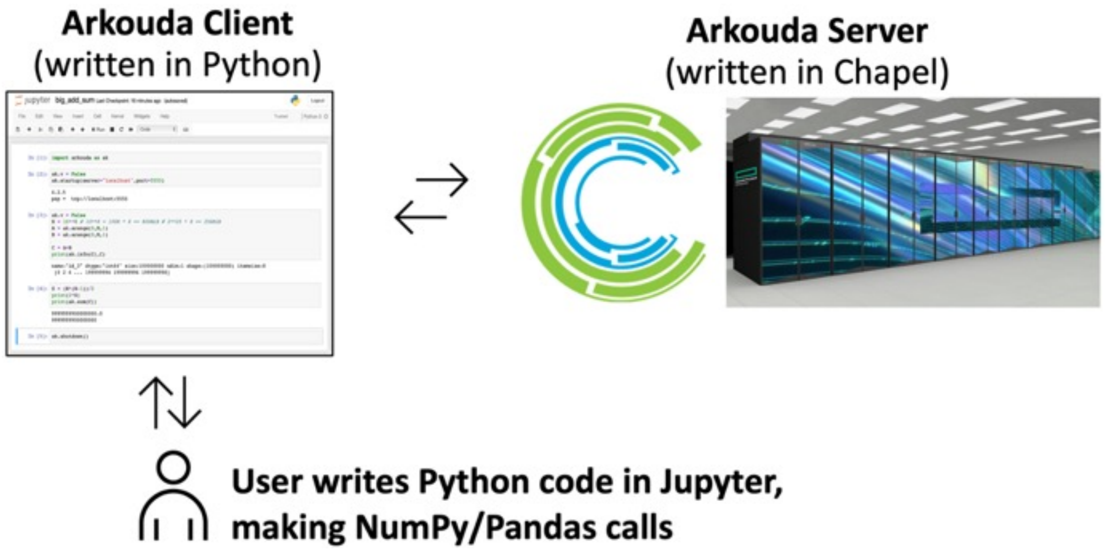
- A Python client-server framework supporting interactive supercomputing
  - Computes massive-scale results (TB-scale arrays) within the human thought loop (seconds to a few minutes)
  - Initial focus has been on a key subset of NumPy and Pandas for Data Science
- ~30k lines of Chapel + ~25k lines of Python, written since 2019
- Open-source: https://github.com/Bears-R-Us/arkouda

Who wrote it?

- Mike Merrill, Bill Reus, *et al.*, US DoD

Why Chapel?

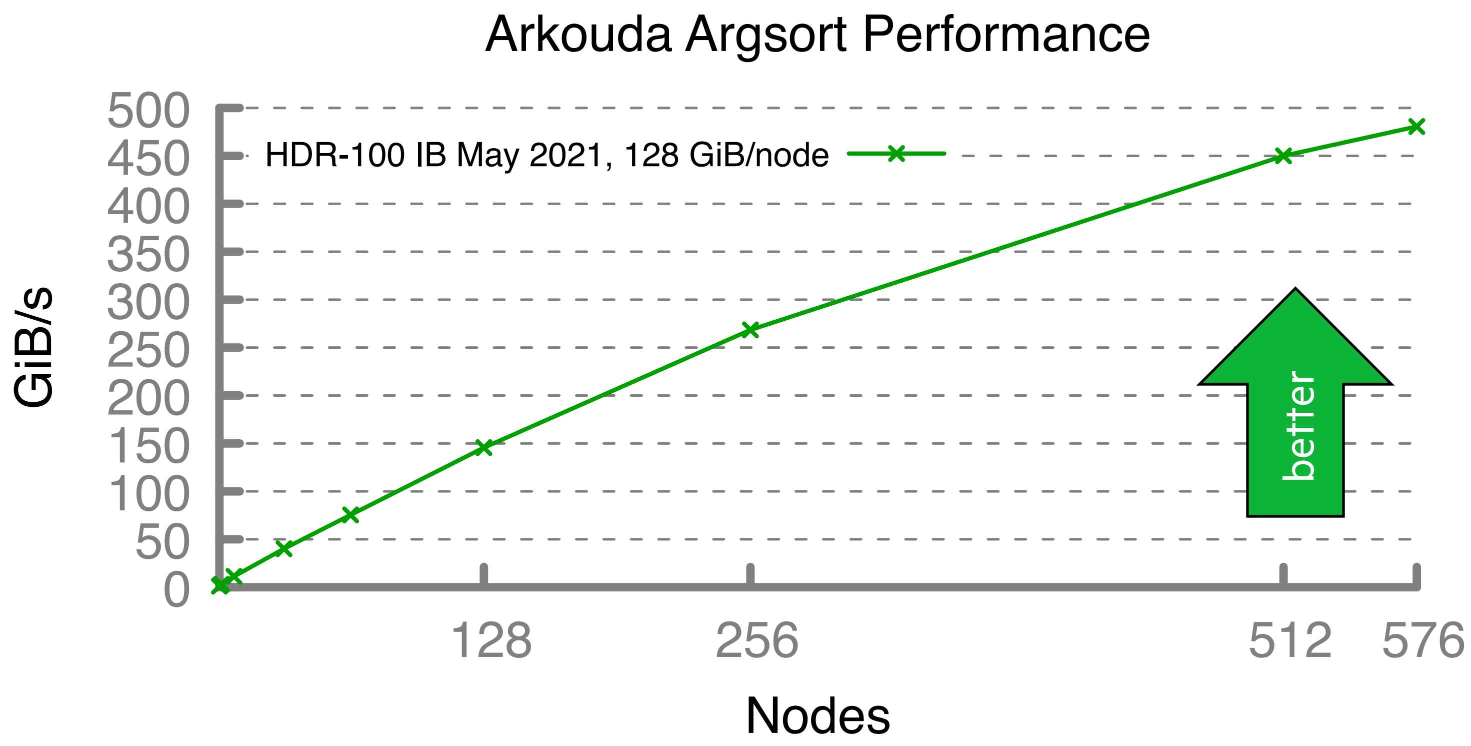
- close to Pythonic
  - enabled writing Arkouda rapidly
  - doesn't repel Python users who look under the hood
- achieved necessary performance and scalability
- ability to develop on laptop, deploy on supercomputer



ARKOUDA ARGSORT PERFORMANCE

HPE Apollo (May 2021)

- HDR-100 Infiniband network (100 Gb/s)
- 576 compute nodes
- 72 TiB of 8-byte values
- ~480 GiB/s (~150 seconds)



A notable performance achievement in ~100 lines of Chapel

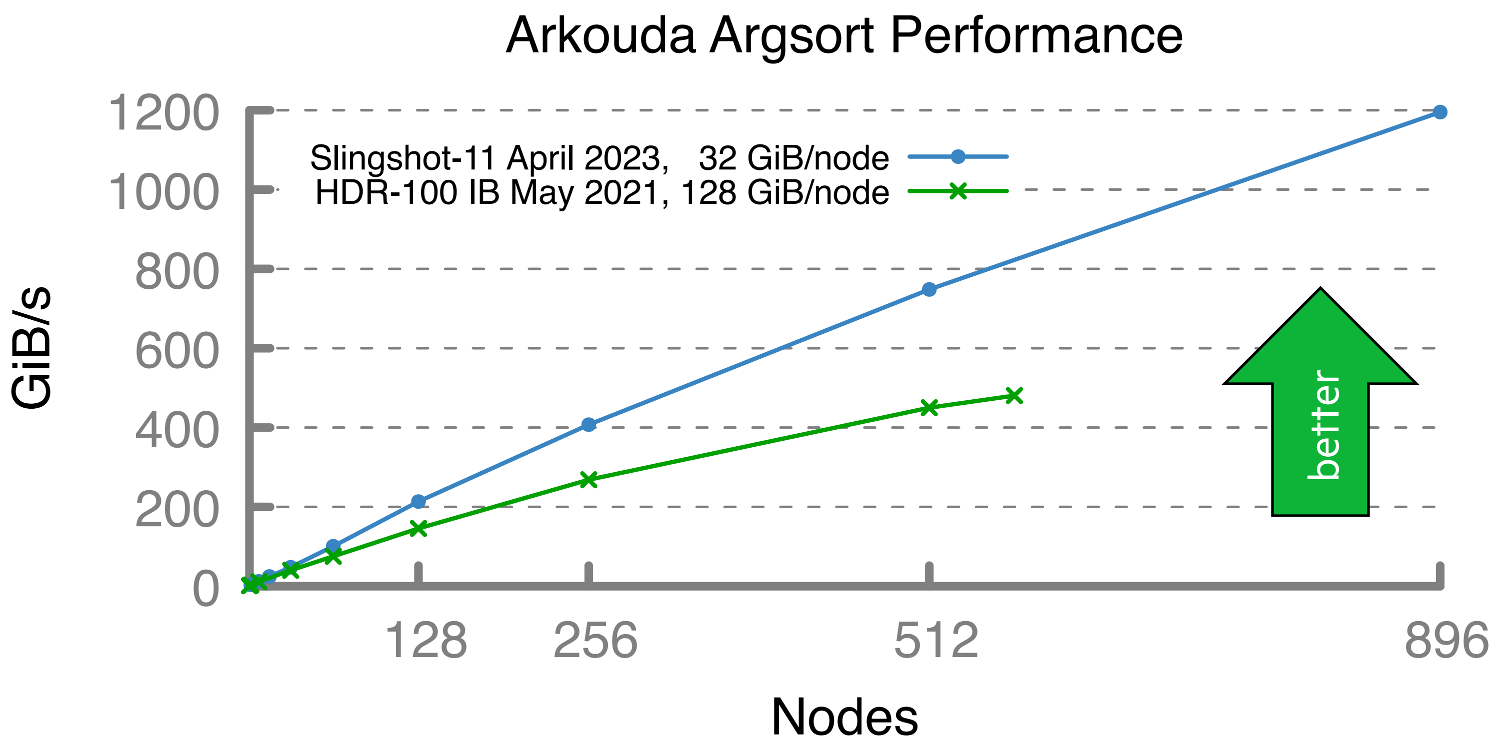
ARKOUDA ARGSORT PERFORMANCE

HPE Apollo (May 2021)

- HDR-100 Infiniband network (100 Gb/s)
- 576 compute nodes
- 72 TiB of 8-byte values
- ~480 GiB/s (~150 seconds)

HPE Cray EX (April 2023)

- Slingshot-11 network (200 Gb/s)
- 896 compute nodes
- 28 TiB of 8-byte values
- ~1200 GiB/s (~24 seconds)



A notable performance achievement in ~100 lines of Chapel

ARKOUDA ARGSORT PERFORMANCE

HPE Apollo (May 2021)

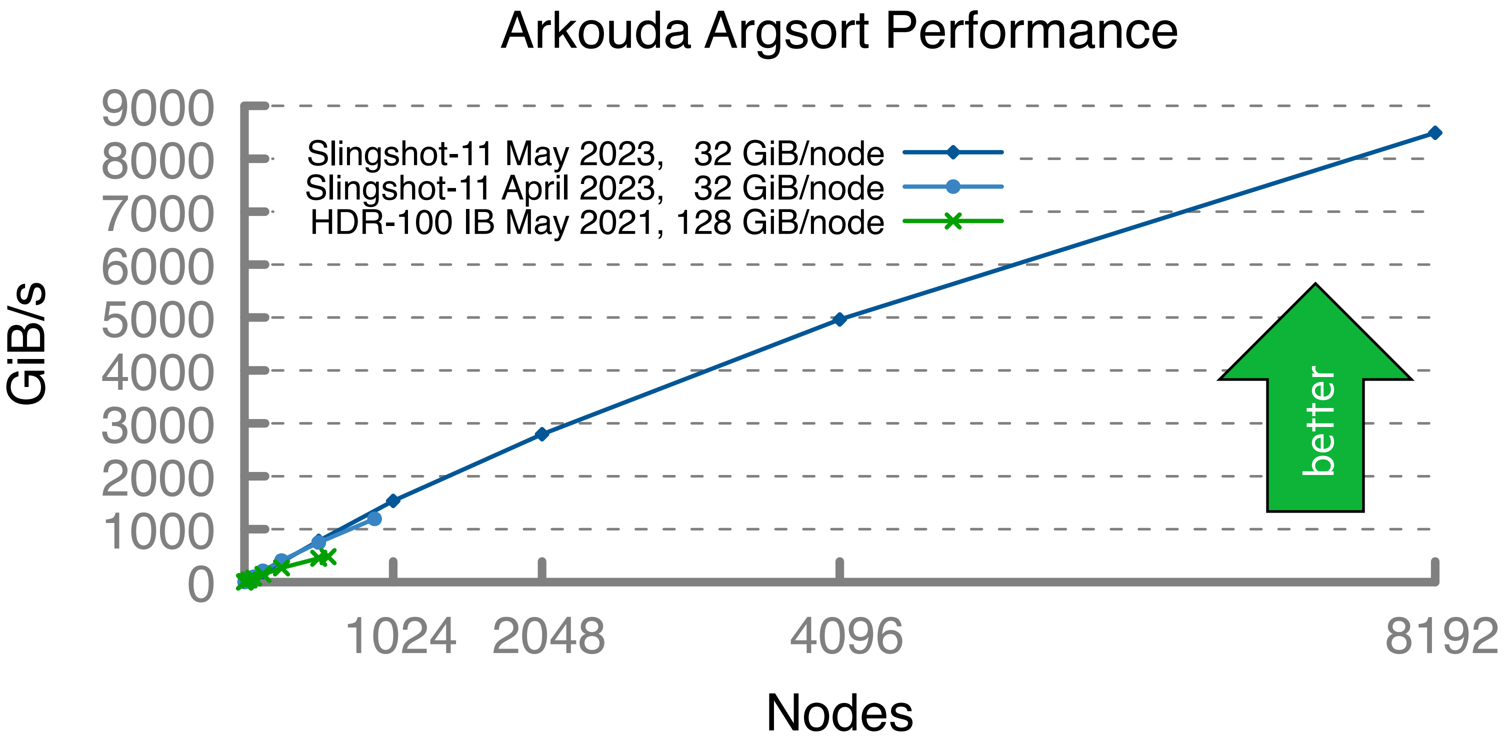
- HDR-100 Infiniband network (100 Gb/s)
- 576 compute nodes
- 72 TiB of 8-byte values
- ~480 GiB/s (~150 seconds)

HPE Cray EX (April 2023)

- Slingshot-11 network (200 Gb/s)
- 896 compute nodes
- 28 TiB of 8-byte values
- ~1200 GiB/s (~24 seconds)

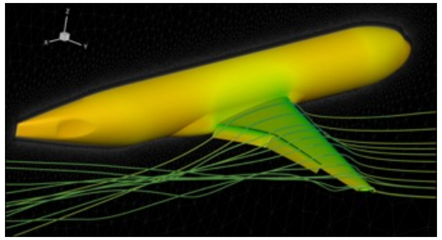
HPE Cray EX (May 2023)

- Slingshot-11 network (200 Gb/s)
- 8192 compute nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)



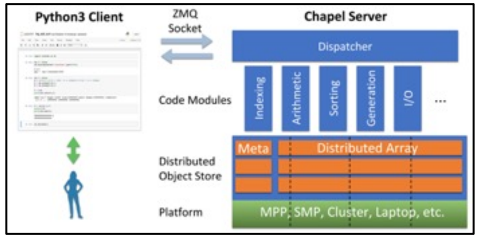
A notable performance achievement in ~100 lines of Chapel

APPLICATIONS OF CHAPEL: LINKS TO USERS' TALKS (SLIDES + VIDEO)



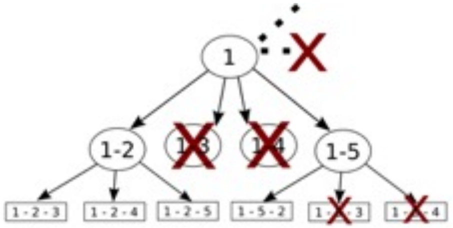
CHAMPS: 3D Unstructured CFD

CHIUW 2021 CHIUW 2022



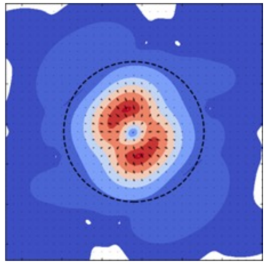
Arkouda: Interactive Data Science at Massive Scale

CHIUW 2020 CHIUW 2023



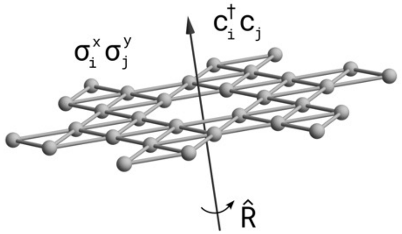
ChOp: Chapel-based Optimization

CHIUW 2021 CHIUW 2023



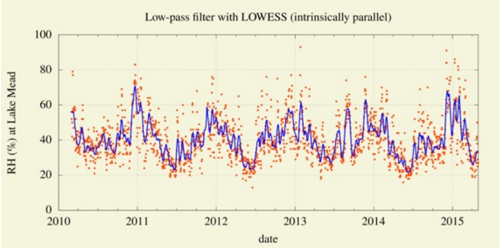
ChplUltra: Simulating Ultralight Dark Matter

CHIUW 2020 CHIUW 2022



Lattice-Symmetries: a Quantum Many-Body Toolbox

CHIUW 2022

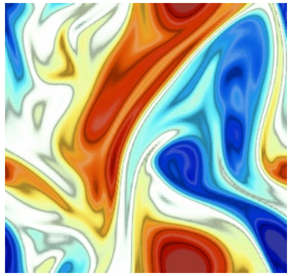


Desk dot chpl: Utilities for Environmental Eng.

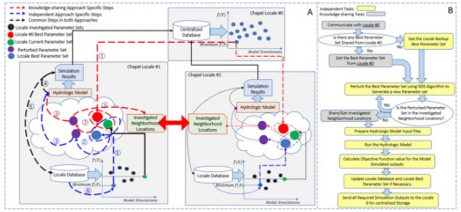
CHIUW 2022

RapidQ: Mapping Coral Biodiversity

CHIUW 2023

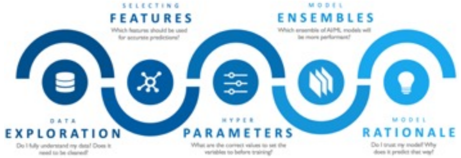


ChapQG: Layered Quasigeostrophic CFD



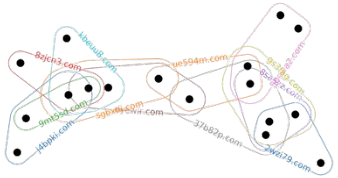
Chapel-based Hydrological Model Calibration

CHIUW 2023



CrayAI HyperParameter Optimization (HPO)

CHIUW 2021



CHGL: Chapel Hypergraph Library

CHIUW 2020

?

Your Application Here?

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

INTRODUCTION TO CHAPEL ON CPUS AND GPUS

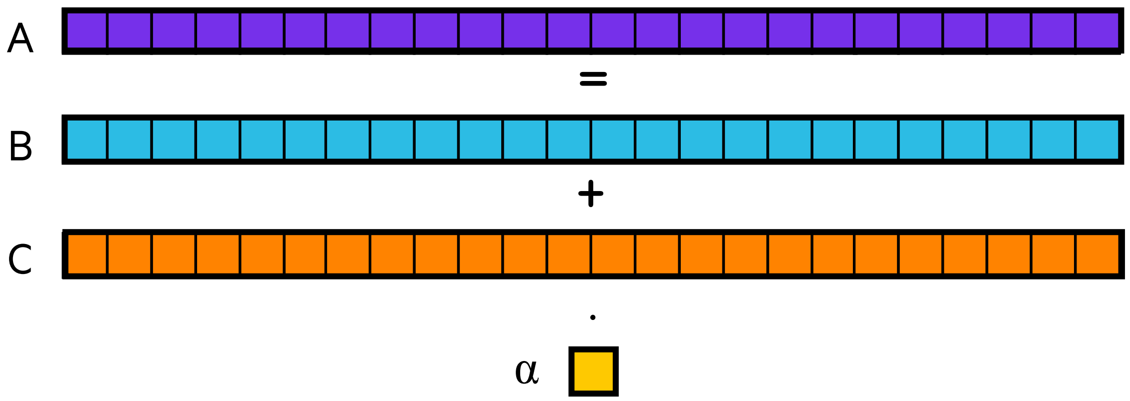
(BY EXAMPLE USING STREAM TRIAD)

STREAM TRIAD: A TRIVIAL CASE OF PARALLELISM + LOCALITY

Given: *n*-element vectors *A*, *B*, *C*

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

In pictures:

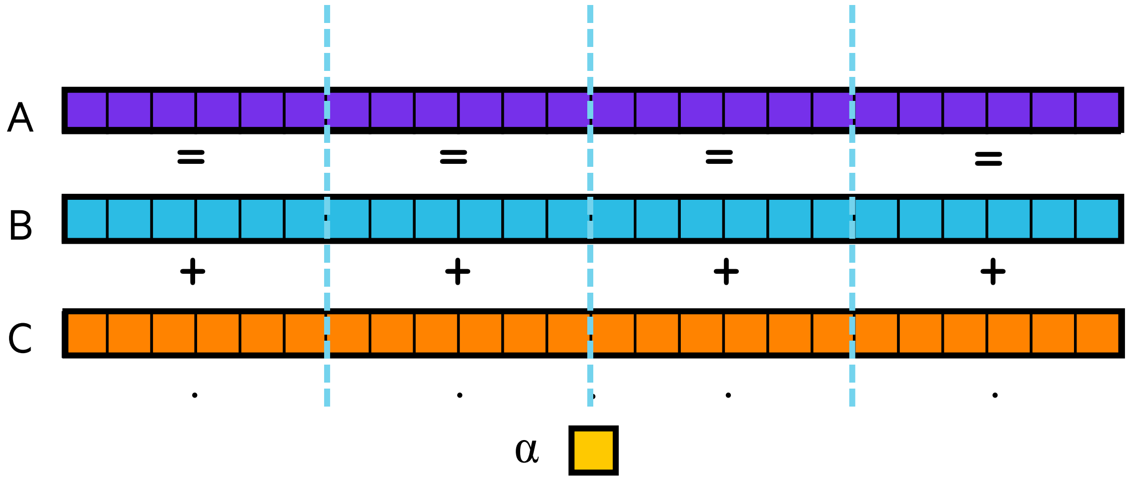


STREAM TRIAD: A TRIVIAL CASE OF PARALLELISM + LOCALITY

Given: n-element vectors A, B, C

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

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

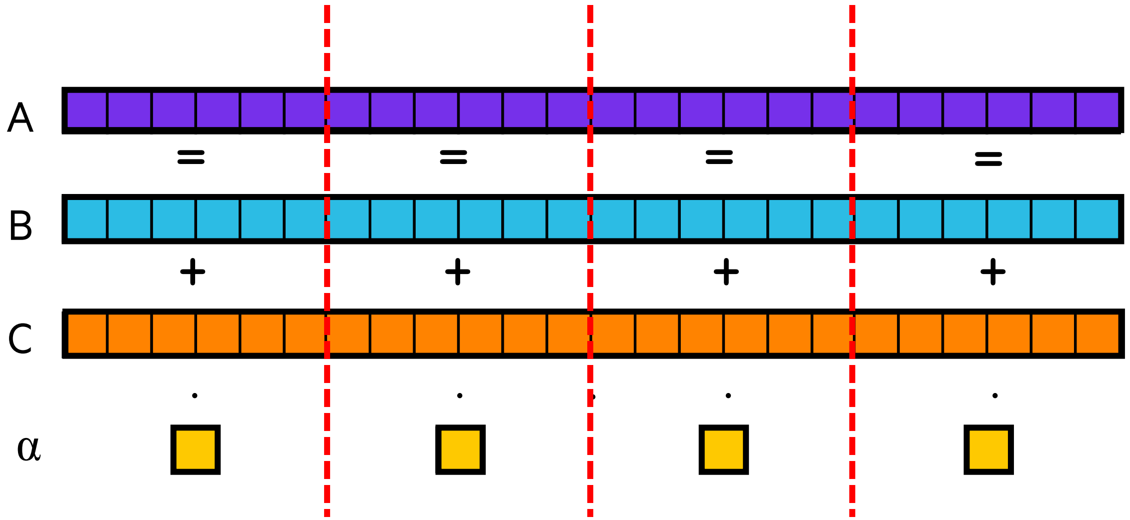


STREAM TRIAD: A TRIVIAL CASE OF PARALLELISM + LOCALITY

Given: *n*-element vectors *A*, *B*, *C*

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

In pictures, in parallel (distributed memory, global-view)**:**

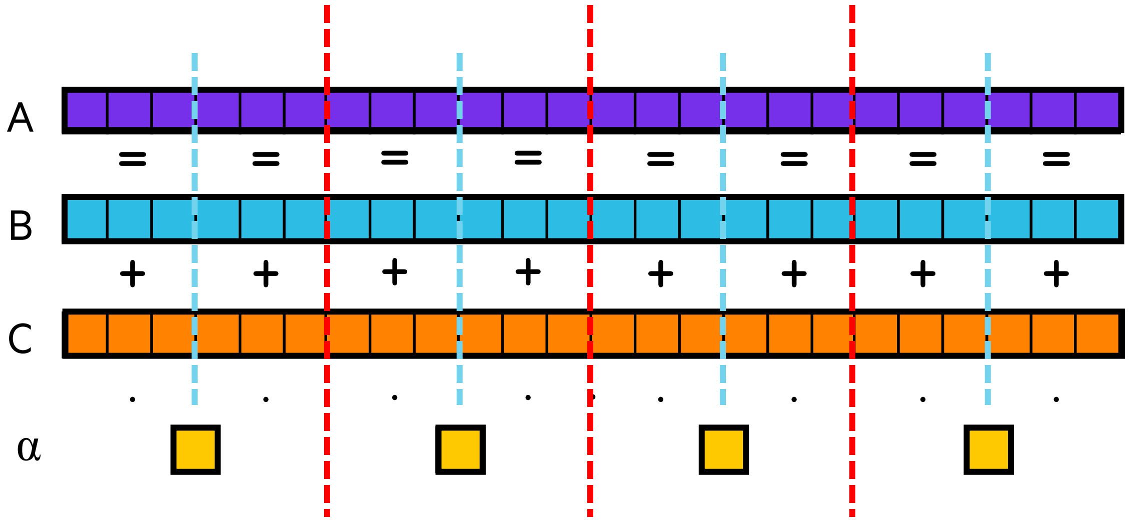


STREAM TRIAD: A TRIVIAL CASE OF PARALLELISM + LOCALITY

Given: *n*-element vectors *A*, *B*, *C*

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

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

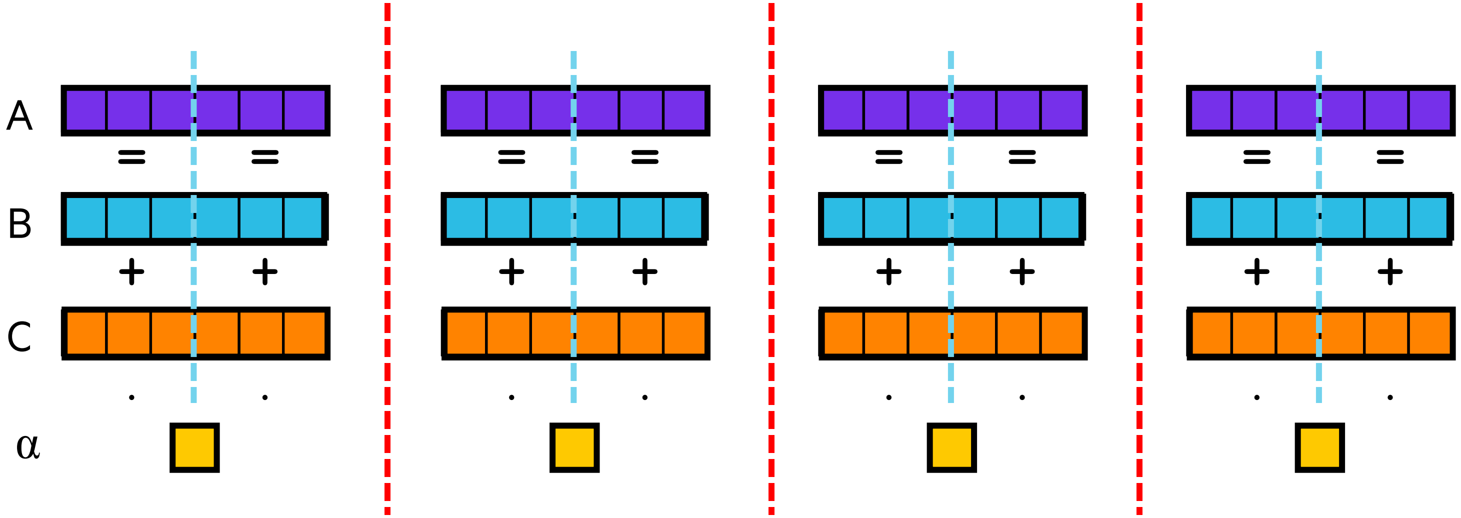


STREAM TRIAD: AN ALTERNATE APPROACH

Given: *n*-element vectors *A*, *B*, *C* on each locale

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

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



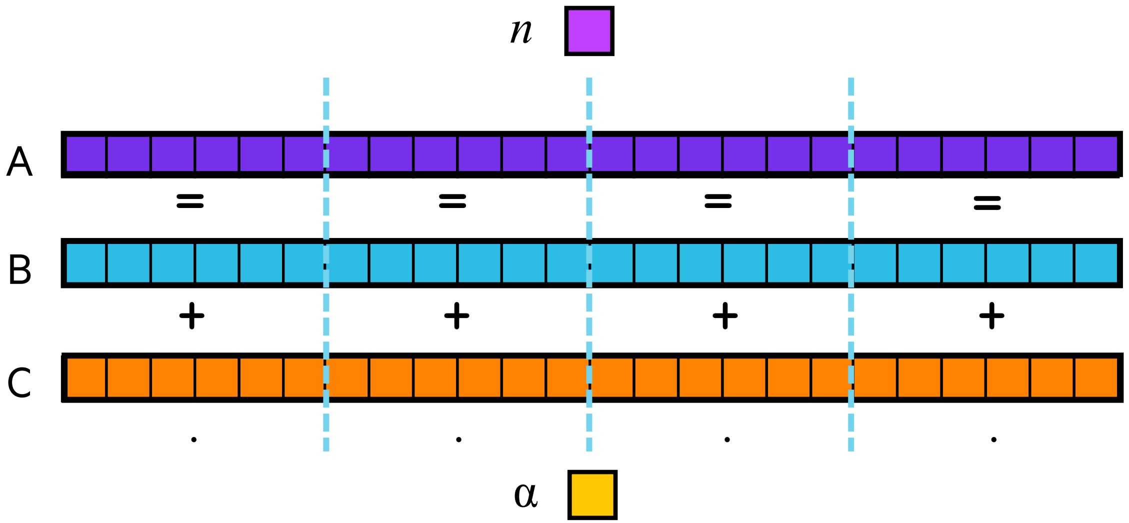
STREAM TRIAD: SHARED MEMORY VERSION

stream-ep.chpl

```
config const n = 1_000_000,
             alpha = 0.01;



    var A, B, C: [1..n] real;
    A = B + alpha * C;
```

declare three arrays of size 'n'

whole-array operations result in parallel computation



So far, this is simply a multi-core program

Nothing refers to remote locales, explicitly or implicitly

STREAM TRIAD: DISTRIBUTED MEMORY, EP VERSION

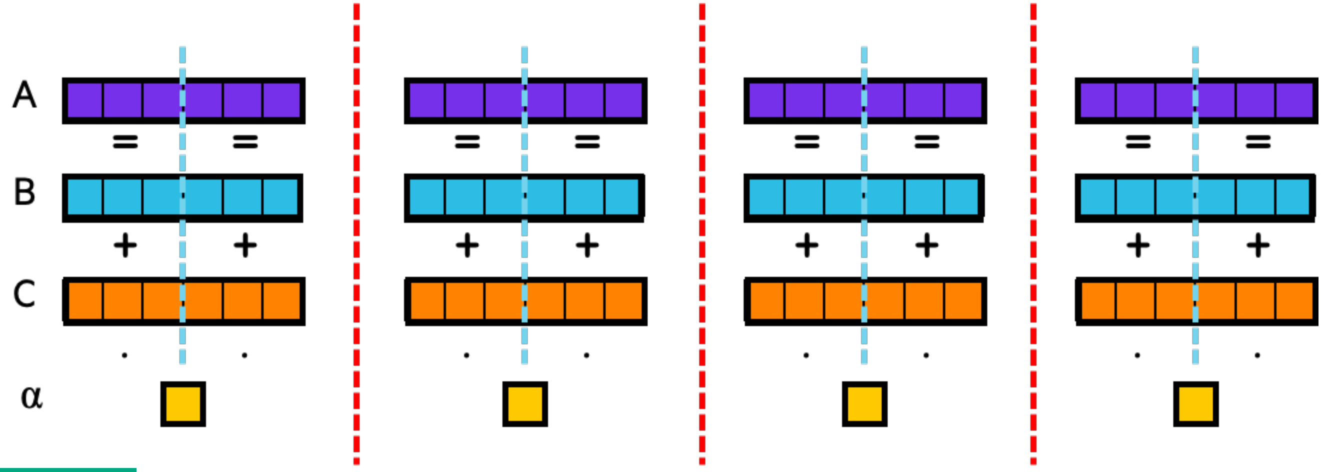
stream-ep.chpl

```
config const n = 1_000_000,
             alpha = 0.01;

coforall loc in Locales {
  on loc {
    var A, B, C: [1..n] real;
    A = B + alpha * C;
  }
}
```

create a task per locale…

…running 'on' its locale

then run multi-core Stream on local arrays, as before



STREAM TRIAD: DISTRIBUTED MEMORY, GLOBAL VERSION

stream-glbl.chpl

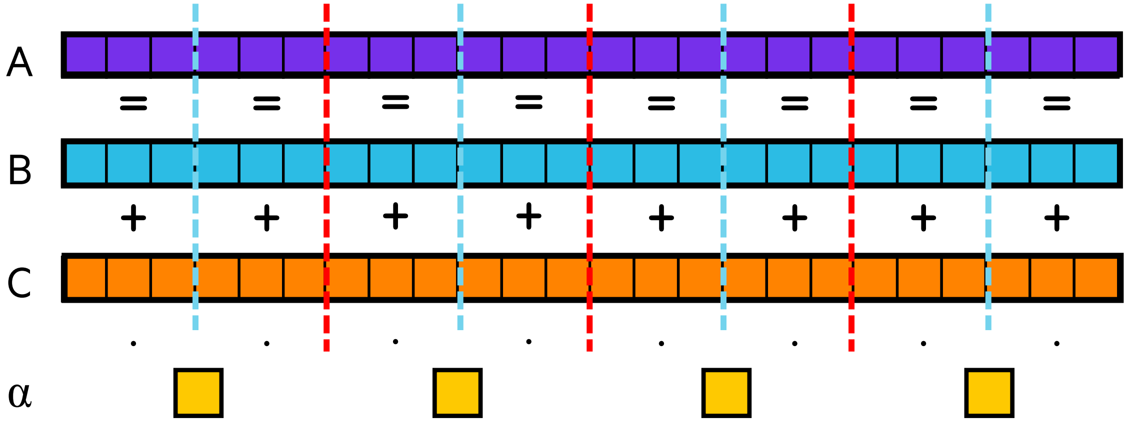
```
config const n = 1_000_000,
             alpha = 0.01;

use BlockDist;

const Dom = Block.createDomain({1..n});
var A, B, C: [Dom] real;

A = B + alpha * C;
```

'use' the standard block-distribution module

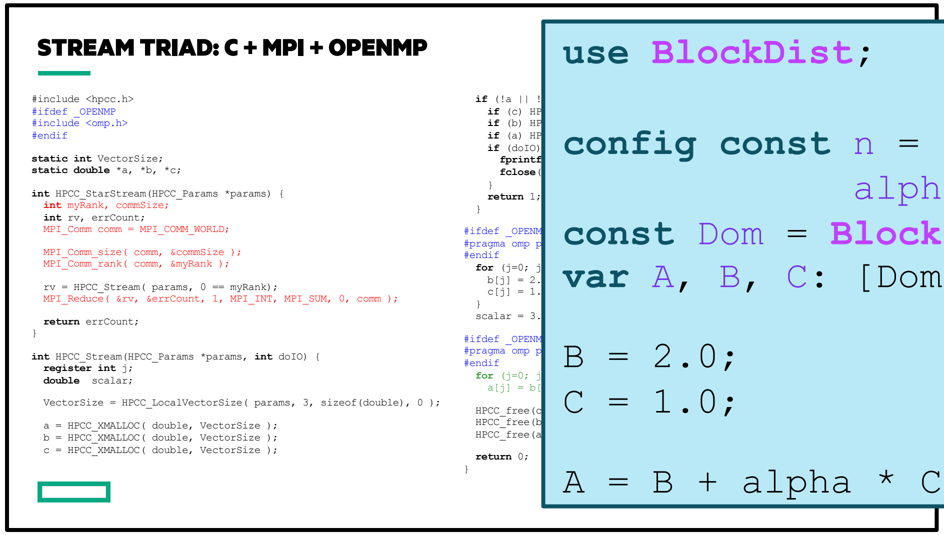
create a distributed domain (index set)…

…and distributed arrays

these whole-array operations will use all cores on all locales

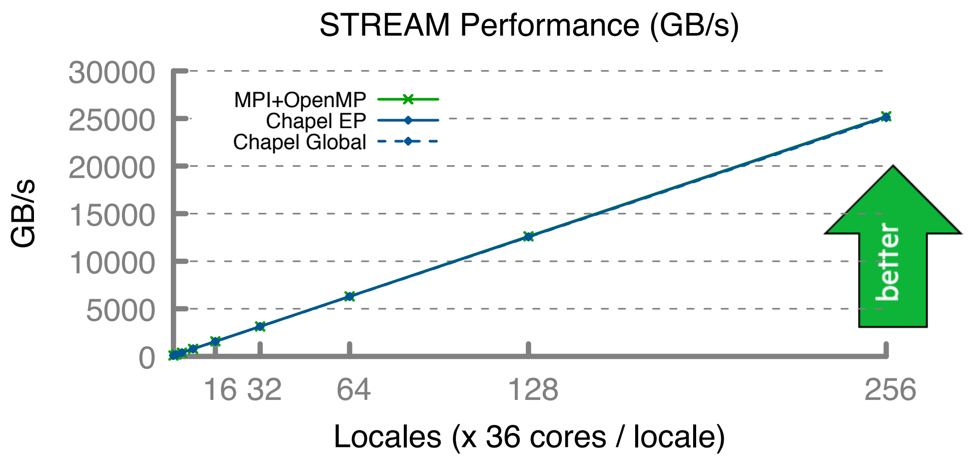


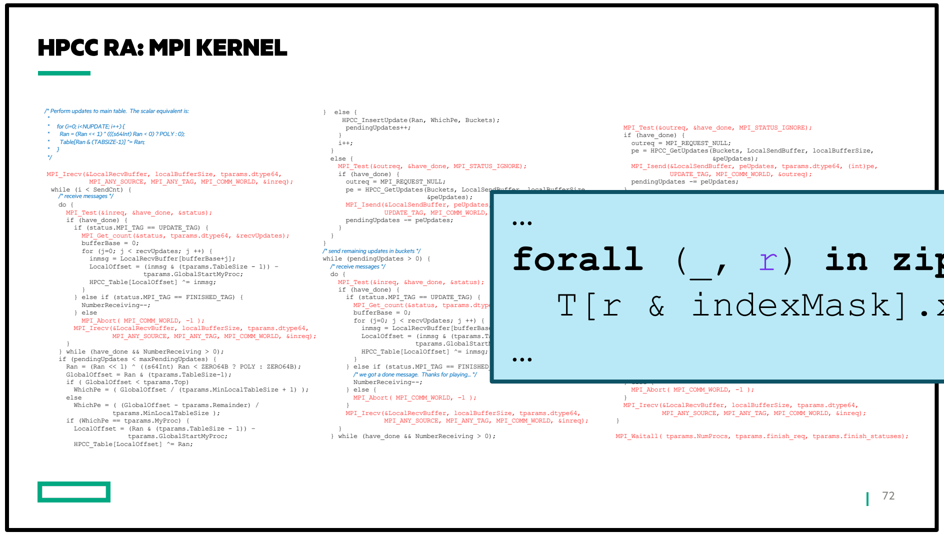
HPC BENCHMARKS: CONVENTIONAL APPROACHES VS. CHAPEL



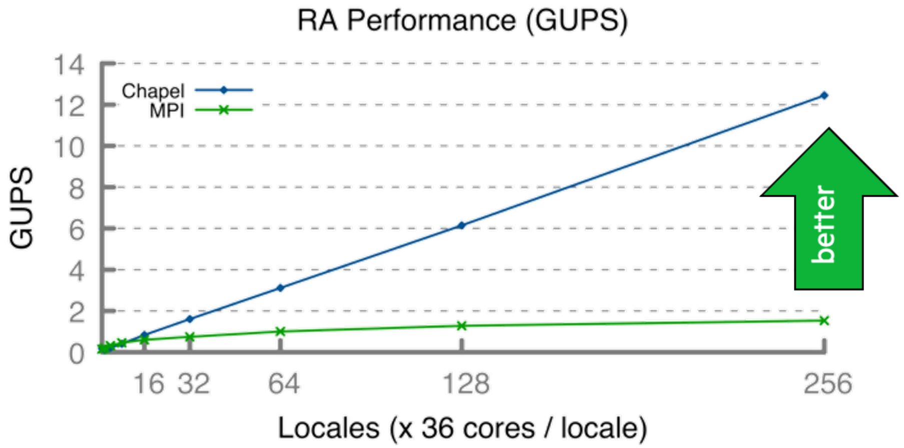

```
use BlockDist;

config const n = 1_000_000,
             alpha = 0.01;

const Dom = Block.createDomain({1..n});
var A, B, C: [Dom] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```

Chapel Global






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



HPC BENCHMARKS: CONVENTIONAL APPROACHES VS. CHAPEL

STREAM TRIAD: C + 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 || !
   if (c) HP
   if (b) HP
   if (a) HP
   if (doIO)
     fprintf
     fclose(
   }
   return 1;
 }

#ifdef _OPENM
#pragma omp p
#endif
 for (j=0; j
   b[j] = 2.
   c[j] = 1.
 }
 scalar = 3.

#ifdef _OPENM
#pragma omp p
#endif
 for (j=0; j
   a[j] = b[

 HPCC_free(c
 HPCC_free(b
 HPCC_free(a

 return 0;
}
```

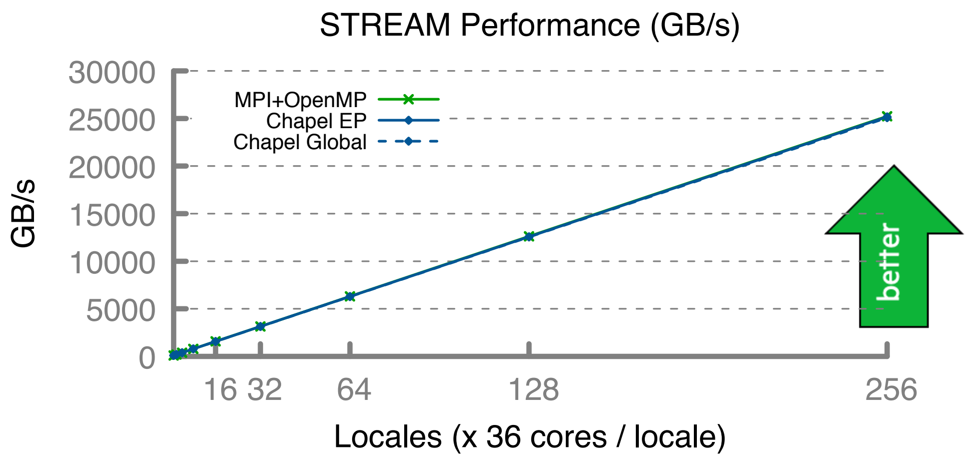
```
use BlockDist;

config const n = 1_000_000,
             alpha = 0.01;
const Dom = Block.createDomain({1..n});
var A, B, C: [Dom] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```

MPI+OpenMP

Chapel Global

Chapel EP

```
config const n = 1_000_000,
             alpha = 0.01;

coforall loc in Locales {
  on loc {
    var A, B, C: [1..n] real;
    A = B + alpha * C;
  }
}
```



These programs are all CPU-only

Nothing refers to GPUs, explicitly or implicitly

STREAM TRIAD: DISTRIBUTED MEMORY, GPUS ONLY

stream-ep.chpl

```
config const n = 1_000_000,
             alpha = 0.01;

coforall loc in Locales do on loc {

    coforall gpu in here.gpus do on gpu {
      var A, B, C: [1..n] real;
      A = B + alpha * C;
    }




}
```

Use a similar 'coforall' + 'on' idiom to run a Triad concurrently on each of this locale's GPUs

This is a GPU-only program

Nothing other than coordination code runs on the CPUs

STREAM TRIAD: DISTRIBUTED MEMORY, GPUS AND CPUS

stream-ep.chpl

```
config const n = 1_000_000,
             alpha = 0.01;

coforall loc in Locales do on loc {
  cobegin {
    coforall gpu in here.gpus do on gpu {
      var A, B, C: [1..n] real;
      A = B + alpha * C;
    }
    {
      var A, B, C: [1..n] real;
      A = B + alpha * C;
    }
  }
}
```

'cobegin { … }' creates a task per child statement

one task runs our multi-GPU triad

the other runs the multi-CPU triad

This program uses all CPUs and GPUs across all of our compute nodes

STREAM TRIAD: DISTRIBUTED MEMORY, GPUS AND CPUS (REFACTOR)

stream-ep.chpl

```
config const n = 1_000_000,
             alpha = 0.01;

coforall loc in Locales do on loc {
  cobegin {
    coforall gpu in here.gpus do on gpu {
      runTriad();
    }
    runTriad();
  }
}

proc runTriad() {
   var A, B, C: [1..n] real;
   A = B + alpha * C;
}
```

we can also refactor the repeated code into a procedure for re-use

the compiler creates CPU and GPU versions of this procedure

CHAPEL ON GPUS: STATUS

Status: Compiling Chapel to GPUs is still reasonably new:

	NVIDIA					AMD				
	via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?	via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?
pre-2021	✓					✓				

CHAPEL ON GPUS: STATUS

Status: Compiling Chapel to GPUs is still reasonably new:

		NVIDIA					AMD				
		via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?	via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?
pre-2021		✓					✓				
Mar 2021	v1.24	✓	✓				✓				

CHAPEL ON GPUS: STATUS

Status: Compiling Chapel to GPUs is still reasonably new:

		NVIDIA					AMD				
		via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?	via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?
pre-2021		✓					✓				
Mar 2021	v1.24	✓	✓				✓				
Sep 2021	v1.25	✓	✓	✓			✓				

CHAPEL ON GPUS: STATUS

Status: Compiling Chapel to GPUs is still reasonably new:

		NVIDIA					AMD				
		via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?	via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?
pre-2021		✓					✓				
Mar 2021	v1.24	✓	✓				✓				
Sep 2021	v1.25	✓	✓	✓			✓				
Mar 2022	v1. 26	✓	✓	✓	✓		✓				

CHAPEL ON GPUS: STATUS

Status: Compiling Chapel to GPUs is still reasonably new:

		NVIDIA					AMD				
		via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?	via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?
pre-2021		✓					✓				
Mar 2021	v1.24	✓	✓				✓				
Sep 2021	v1.25	✓	✓	✓			✓				
Mar 2022	v1. 26	✓	✓	✓	✓		✓				
Jun 2022	v1.27	✓	✓	✓	✓	✓	✓				

CHAPEL ON GPUS: STATUS

Status: Compiling Chapel to GPUs is still reasonably new:

		NVIDIA					AMD				
		via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?	via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?
pre-2021		✓					✓				
Mar 2021	v1.24	✓	✓				✓				
Sep 2021	v1.25	✓	✓	✓			✓				
Mar 2022	v1. 26	✓	✓	✓	✓		✓				
Jun 2022	v1.27	✓	✓	✓	✓	✓	✓				
Dec 2023	v1.29	✓	✓	✓	✓	✓	✓	✓			

CHAPEL ON GPUS: STATUS

Status: Compiling Chapel to GPUs is still reasonably new:

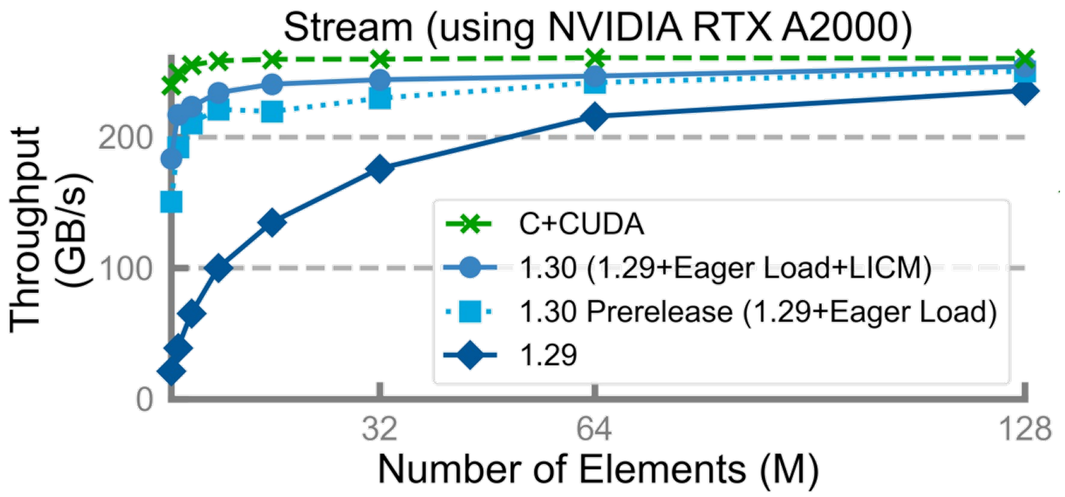
		NVIDIA					AMD				
		via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?	via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?
pre-2021		✓					✓				
Mar 2021	v1.24	✓	✓				✓				
Sep 2021	v1.25	✓	✓	✓			✓				
Mar 2022	v1. 26	✓	✓	✓	✓		✓				
Jun 2022	v1.27	✓	✓	✓	✓	✓	✓				
Dec 2023	v1.29	✓	✓	✓	✓	✓	✓	✓			
Mar 2023	v1.30	✓	✓	✓	✓	✓	✓	✓	✓	✓	

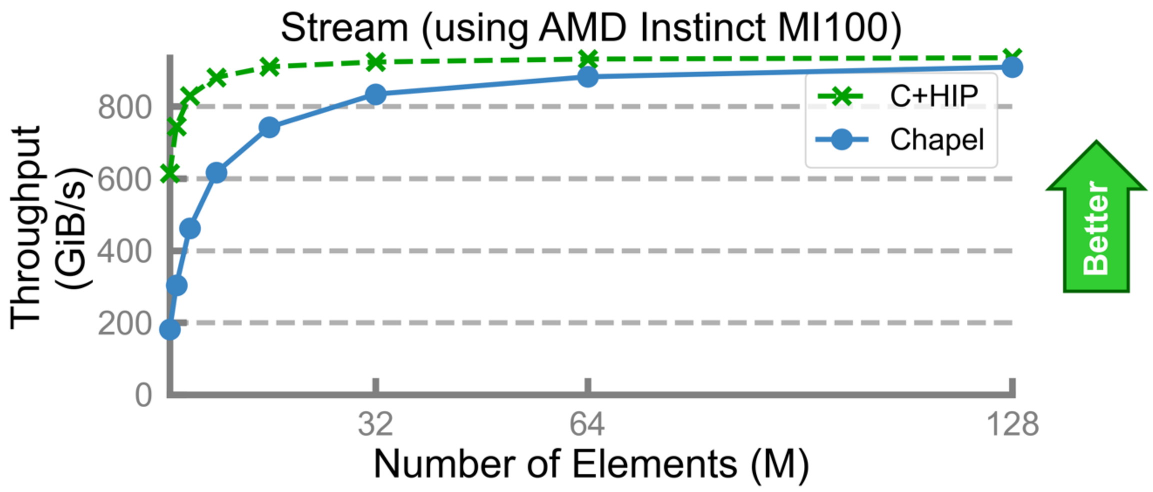
CHAPEL ON GPUS: STATUS

Status: Compiling Chapel to GPUs is still reasonably new:

		NVIDIA					AMD				
		via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?	via interop?	heroic?	Chapel loops?	multi-GPU?	multi-node?
pre-2021		✓					✓				
Mar 2021	v1.24	✓	✓				✓				
Sep 2021	v1.25	✓	✓	✓			✓				
Mar 2022	v1. 26	✓	✓	✓	✓		✓				
Jun 2022	v1.27	✓	✓	✓	✓	✓	✓				
Dec 2023	v1.29	✓	✓	✓	✓	✓	✓	✓			
Mar 2023	v1.30	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Jun 2023	v1.31	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

STREAM TRIAD: CHAPEL GPU PERFORMANCE VS. REFERENCE VERSIONS





Performance vs. CUDA has become increasingly competitive over the past 6 months

WRAP-UP

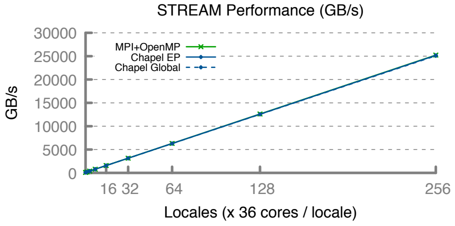
THE CHAPEL TEAM AT HPE, JUNE 2023

SUMMARY

Chapel is unique among programming languages

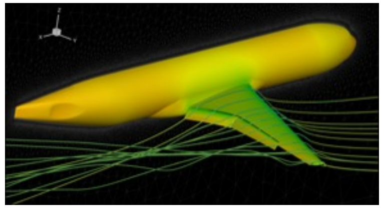
- built-in features for scalable parallel computing make it HPC-ready
- supports clean, concise code relative to conventional approaches
- ports and scales from laptops to supercomputers

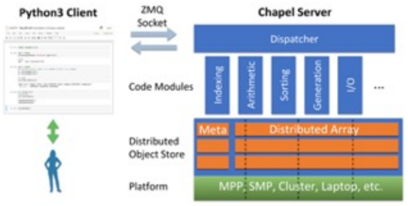
```
use BlockDist;

config const n = 1_000_000,
             alpha = 0.01;
const Dom = Block.createDomain({1..n});
var A, B, C: [Dom] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```



Chapel is being used for productive parallel computing at scale

- users are reaping its benefits in practical, cutting-edge applications
- in diverse application domains: from physical simulation to data science
- scaling to thousands of nodes / millions of processor cores





Vendor-neutral GPU support is maturing rapidly

- fleshes out an overdue aspect of "any parallel hardware"

```
coforall gpu in here.gpus do on gpu {
  var A, B, C: [1..n] real;
  A = B + alpha * C;
}
```

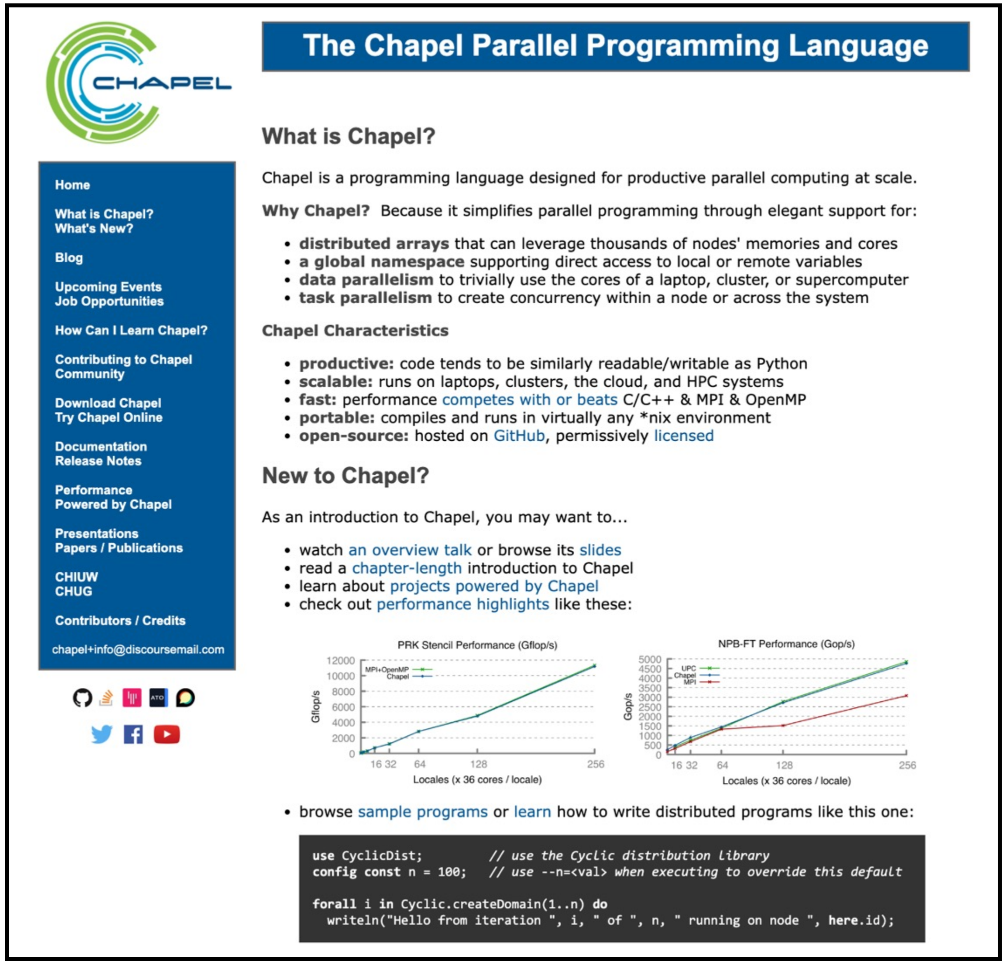
We're interested in helping new users and fostering new collaborations

CHAPEL RESOURCES

Chapel homepage: https://chapel-lang.org
- (points to all other resources)

Social Media:
- Blog: https://chapel-lang.org/blog/
- Twitter: @ChapelLanguage
- Facebook: @ChapelLanguage
- YouTube: @ChapelLanguage

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

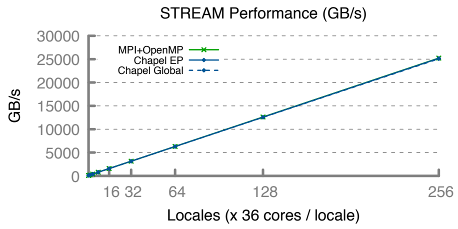


SUMMARY

Chapel is unique among programming languages

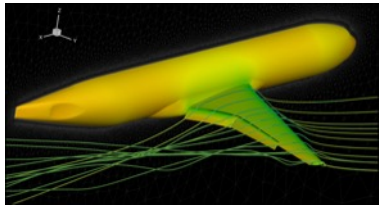
- built-in features for scalable parallel computing make it HPC-ready
- supports clean, concise code relative to conventional approaches
- ports and scales from laptops to supercomputers

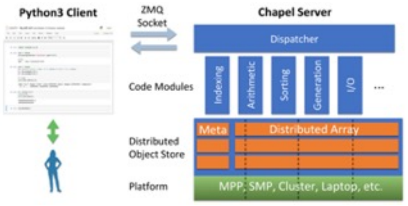
```
use BlockDist;

config const n = 1_000_000,
             alpha = 0.01;
const Dom = Block.createDomain({1..n});
var A, B, C: [Dom] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```



Chapel is being used for productive parallel computing at scale

- users are reaping its benefits in practical, cutting-edge applications
- in diverse application domains: from physical simulation to data science
- scaling to thousands of nodes / millions of processor cores



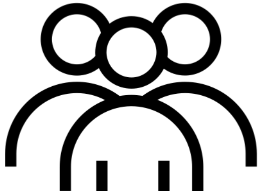


Vendor-neutral GPU support is maturing rapidly

- fleshes out an overdue aspect of “any parallel hardware”

```
coforall gpu in here.gpus do on gpu {
  var A, B, C: [1..n] real;
  A = B + alpha * C;
}
```

We’re interested in helping new users and fostering new collaborations



THANK YOU

https://chapel-lang.org
@ChapelLanguage