Hewlett Packard Enterprise

Writing Parallel Research Software in Chapel

Brad Chamberlain

HiRSE Summer of Programming Languages

August 2025

Q: Why consider programming in Chapel?

A: It’s one of the few programming languages designed for scalable parallel computing from the start.

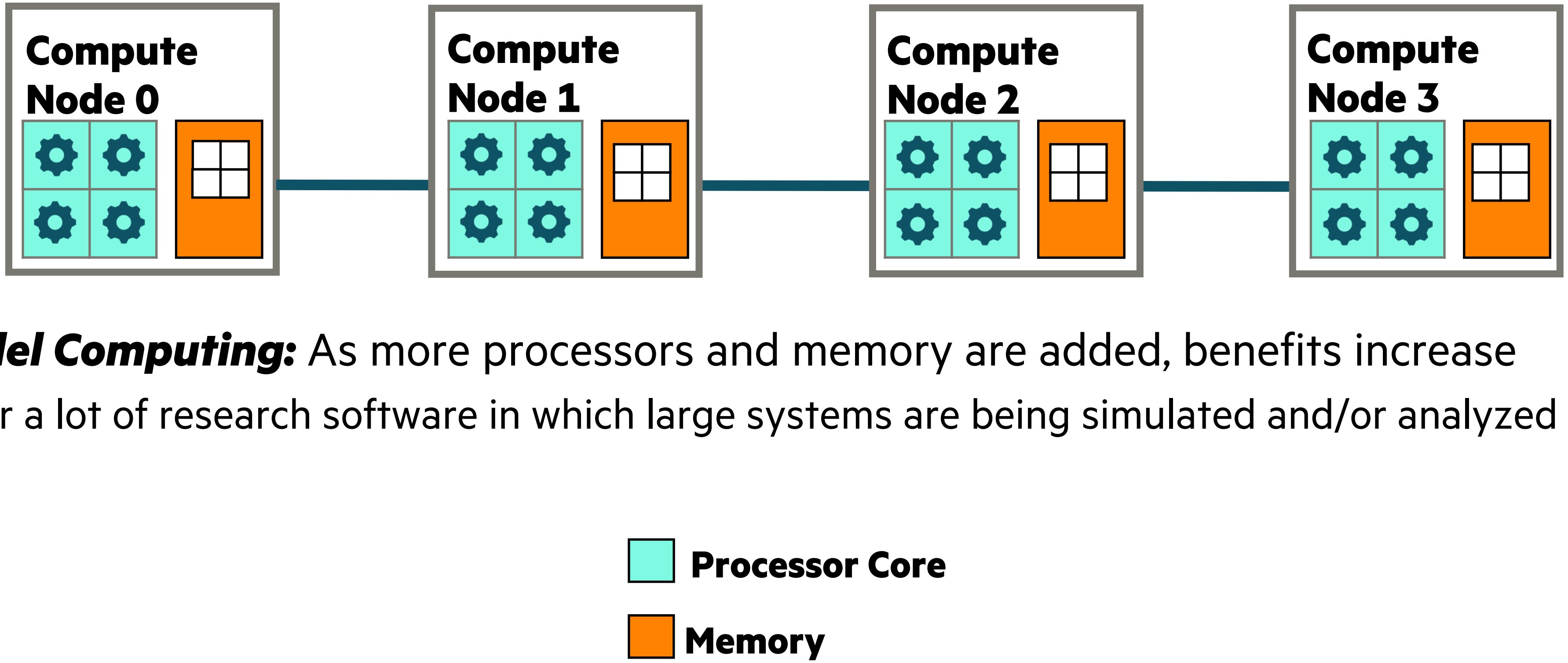
What is Scalable Parallel Computing?

Parallel Computing: Using the processors and memories of multiple compute resources

- Why? To run a program…

  …faster than we could otherwise

  …and/or using larger problem sizes

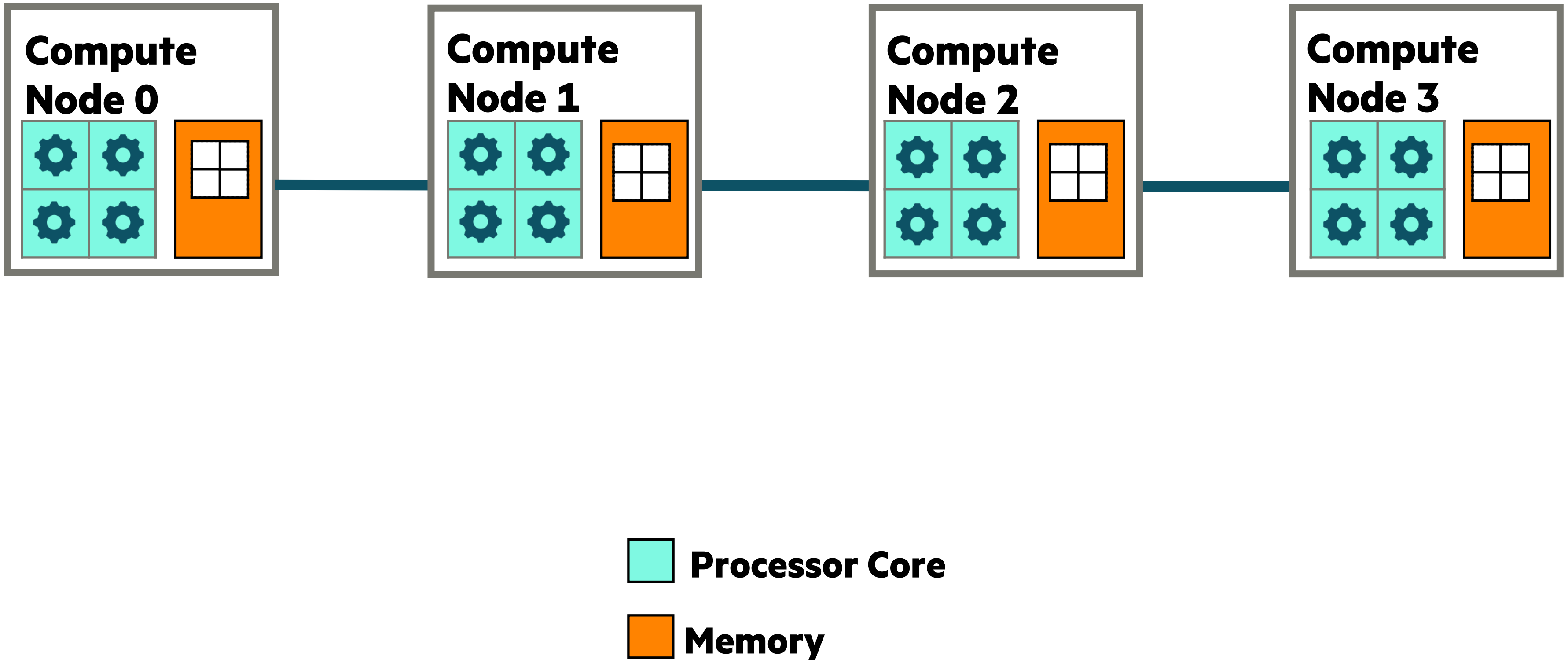


Scalable Parallel Computing: As more processors and memory are added, benefits increase

- important for a lot of research software in which large systems are being simulated and/or analyzed

Key Concerns for Scalable Parallel Computing

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



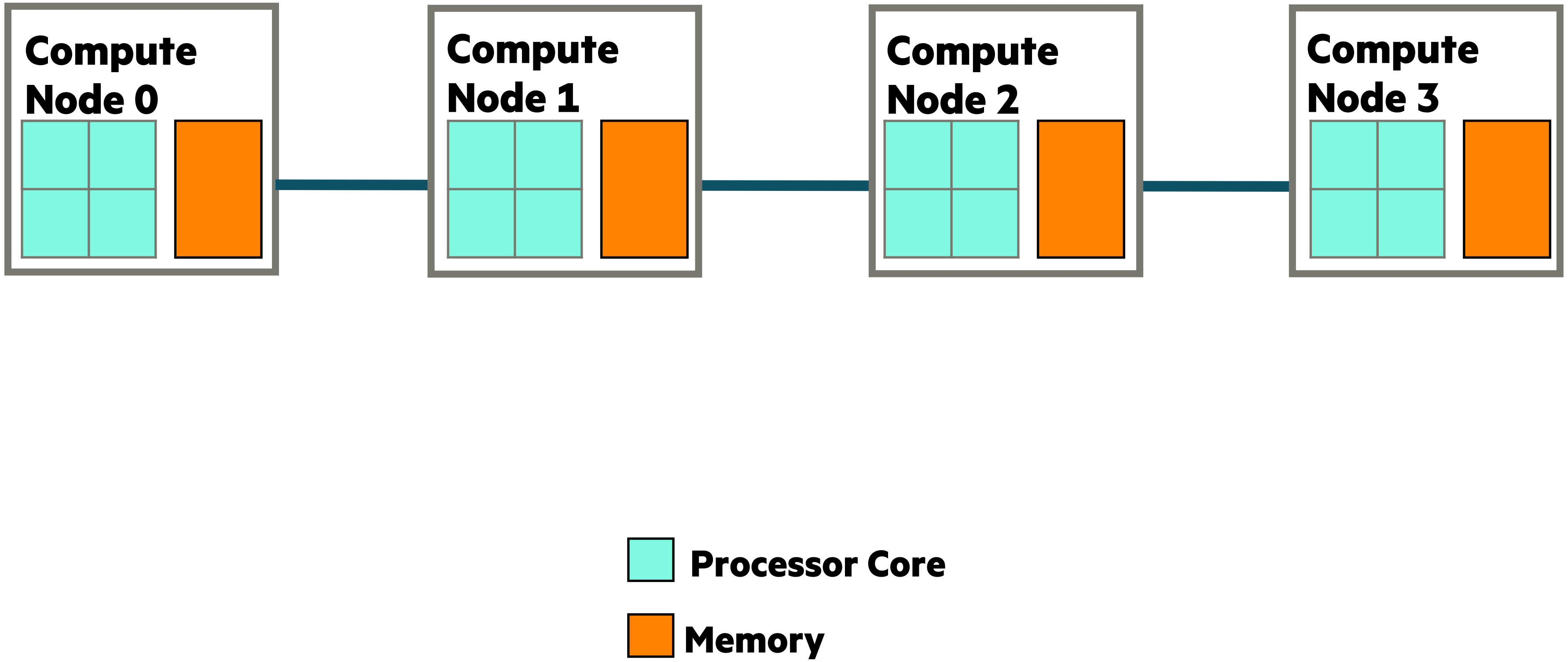
Parallel Computing has become Ubiquitous

Parallel computing, historically:

- supercomputers
- commodity clusters

Additional, ubiquitous parallelism today:

- multicore processors
- cloud computing
- GPUs



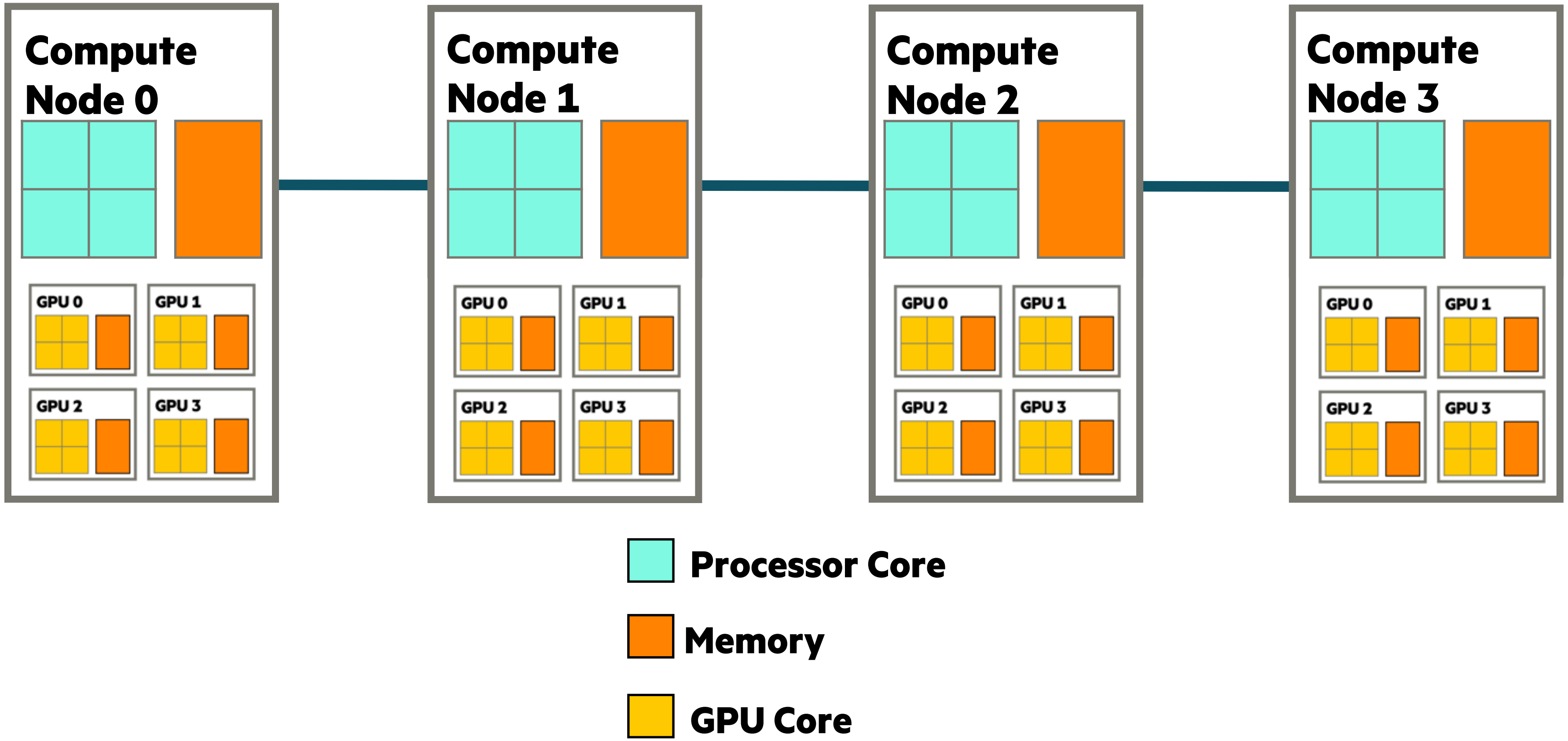
Compute Nodes with GPUs

Parallel computing, historically:

- supercomputers
- commodity clusters

Additional, ubiquitous parallelism today:

- multicore processors
- cloud computing
- GPUs



What is Chapel?

Chapel: A modern parallel programming language
- Portable & scalable
- Open-source & collaborative
- Focused on expressing parallelism and locality

Goals:
- Support general parallel programming
- Make parallel programming at scale far more productive

Productive Parallel Programming: One Definition

Imagine a programming language for parallel computing that is as…

…**readable and writeable** as Python

…yet also as…

…**fast** as Fortran / C / C++

…**scalable** as MPI / SHMEM

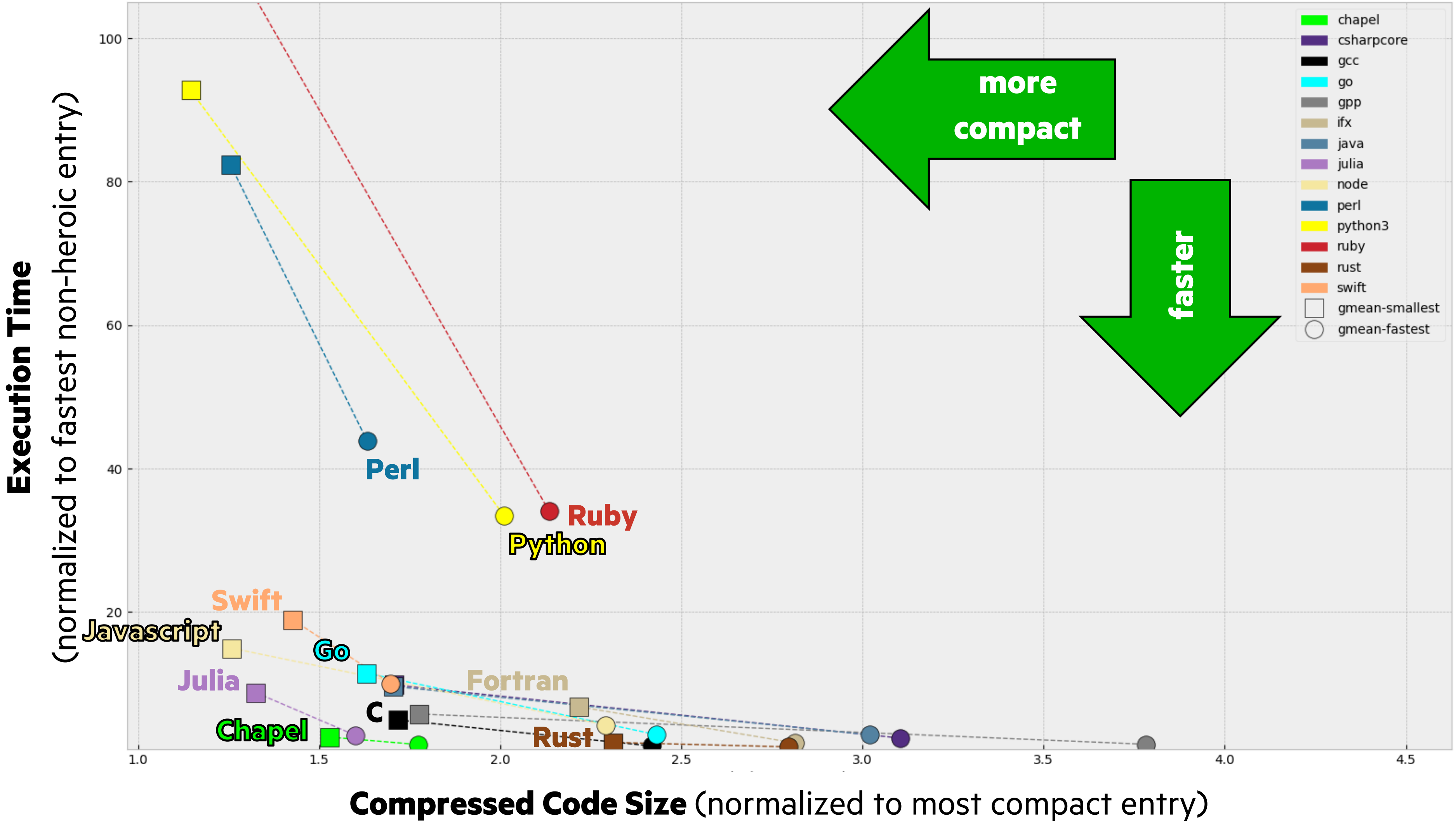
…**GPU-ready** as CUDA / HIP / OpenMP / Kokkos / OpenCL / OpenACC / …

…**portable** as C

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

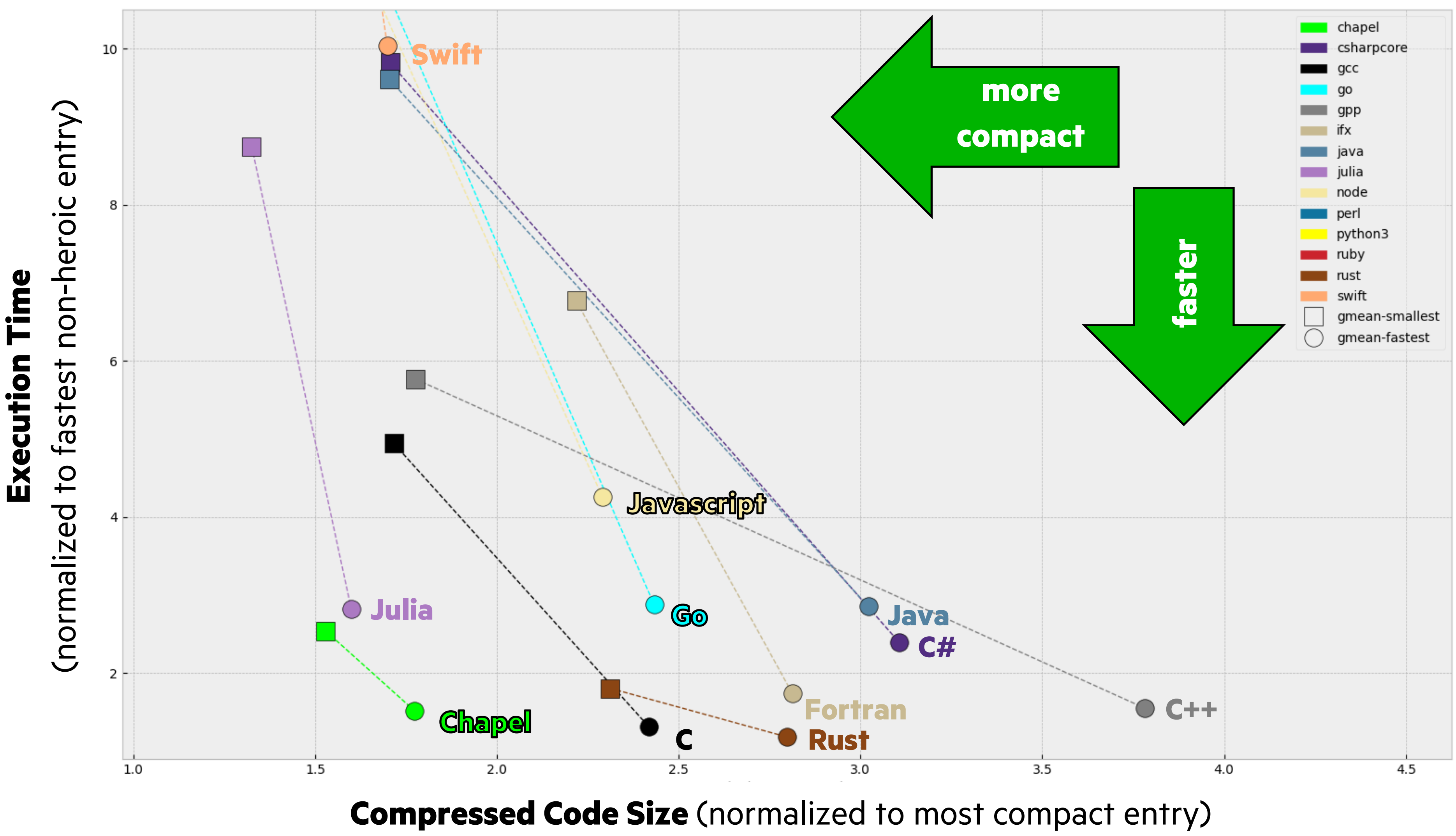
This is our motivation for Chapel

CLBG Language Comparison (selected languages, no heroic versions)



For more information, see this lightning talk: https://www.youtube.com/watch?v=U8KM8wv32js&list=PLuqM5RJ2KYFi2yV4sFLc6QeRYpS35UeKl&index=6

CLBG Language Comparison (selected languages, no heroic versions, zoomed in)



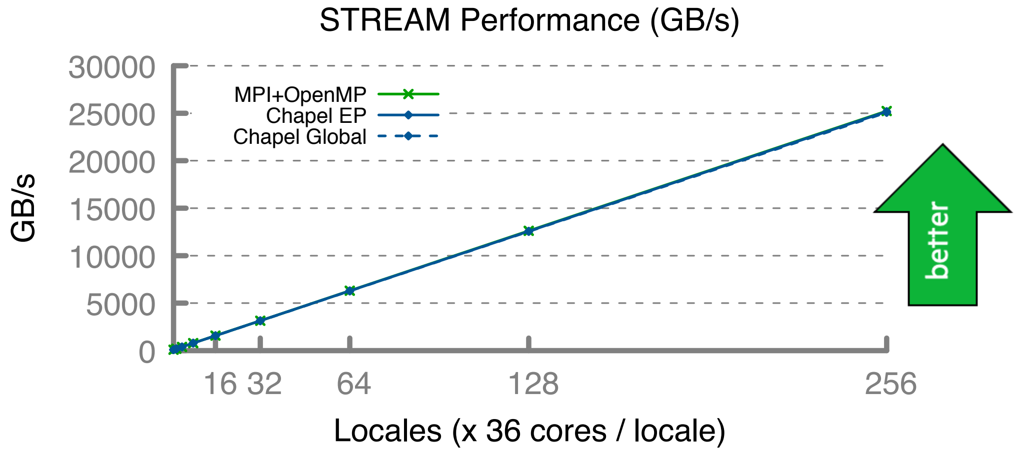
For more information, see this lightning talk: https://www.youtube.com/watch?v=U8KM8wv32js&list=PLuqM5RJ2KYFi2yV4sFLc6QeRYpS35UeKI&index=6

HPCC Stream Triad and RA in C + MPI + OpenMP vs. Chapel

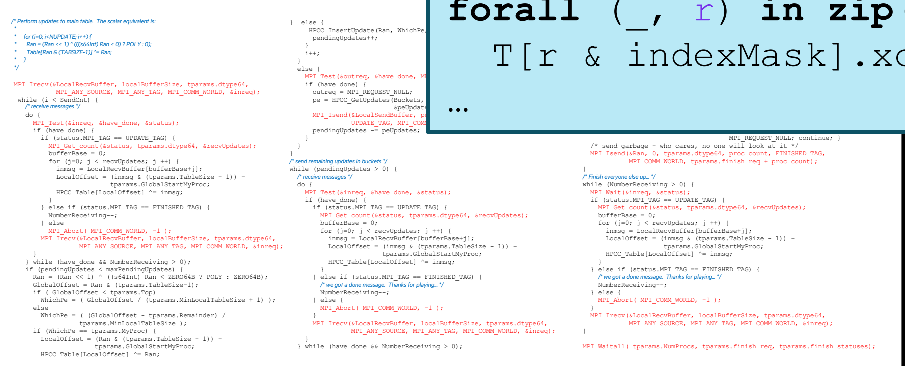
STREAM TRIAD: C + MPI + OPENMP

```
#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 );
```

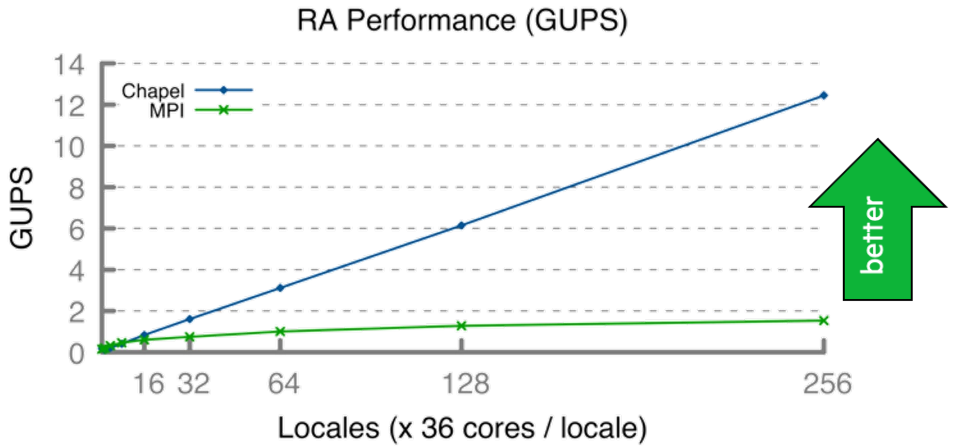
```
use BlockDist;

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

B = 2.0;
C = 1.0;

A = B + alpha * C;
```



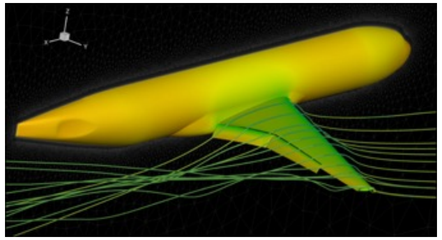
HPCC RA: MPI KERNEL



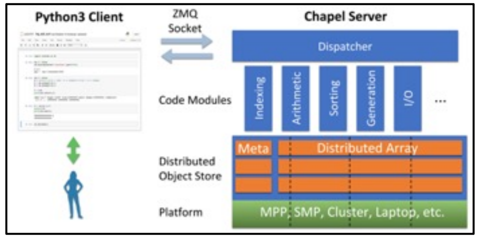
```
…
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask].xor(r);
…
```



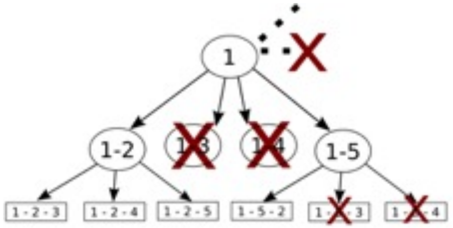
Applications of Chapel



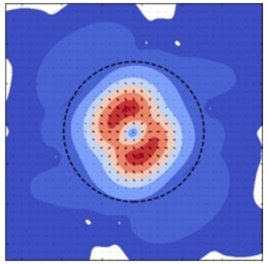
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



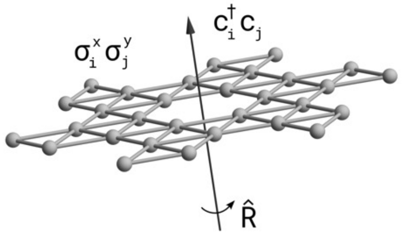
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



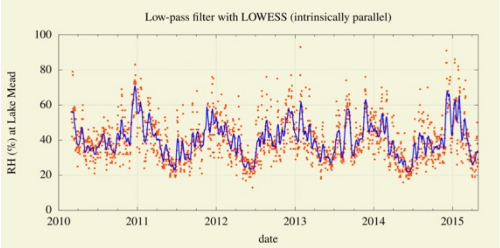
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

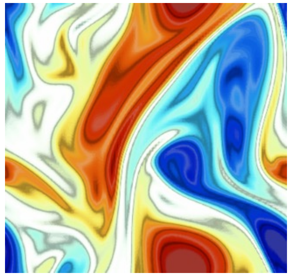


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

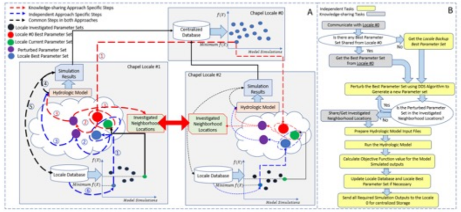


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

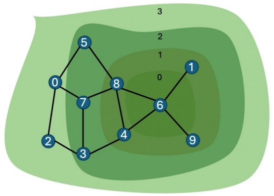
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



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

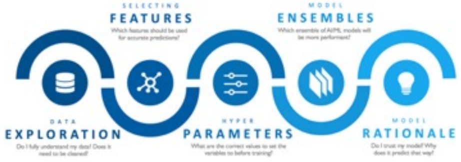


Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



Arachne Graph Analytics
Bader, Du, Rodriguez, et al.
New Jersey Institute of Technology

Modeling Ocean Carbon Dioxide Removal
Scott Bachman Brandon Neth, et al.
[C]Worthy



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

[images provided by their respective teams and used with permission]

Why use Chapel?

Some reasons to consider Chapel:

- You want to do parallel or distributed programming in a language designed for it
- You're using Python, R, or Matlab but need more performance or scalability
- You're doing parallelism in a conventional approach and want a more productive, integrated alternative
  - you're doing multicore / multithreaded parallelism using OpenMP or pthreads
  - you're doing GPU programming using CUDA, HIP, SYCL, Kokkos, OpenACC, OpenCL
  - you're doing scalable parallelism using MPI or SHMEM
  - you're using some combination of the above in a single program and it's become unwieldy

Things to be aware of:

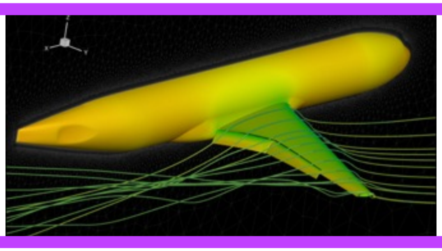
- Chapel's user community is much smaller than Python, C++, etc.
  - However, it's growing and the Chapel team is very active and communicative
- The current number of libraries is much smaller than Python, C++, etc.
  - Interoperating with existing libraries is a common way of dealing with this
- Like a lot of HPC software, it's a bit more rough-and-tumble than mainstream languages
  - However, development proceeds at a fast pace and user issues are typically resolved quickly

Outline

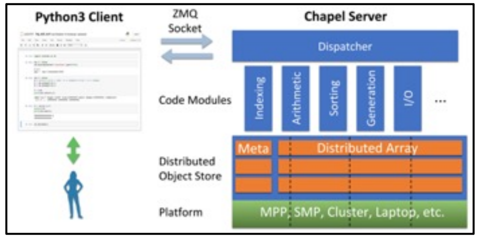
- Motivation for Chapel
- Sample Chapel Applications
- Chapel Characteristics
- Sample Computation
- Wrap-up

Sample Chapel Applications

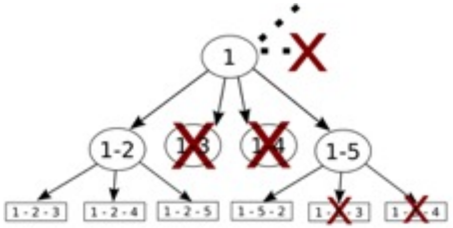
Applications of Chapel



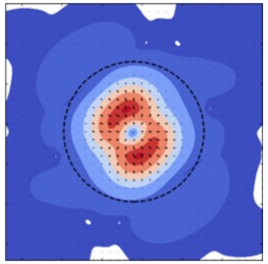
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



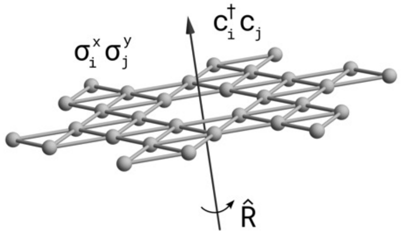
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



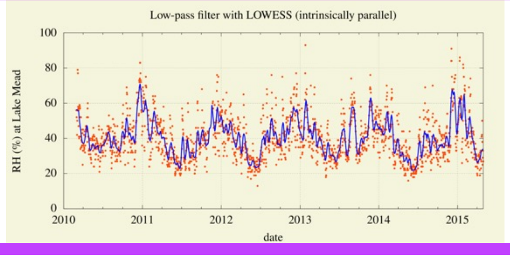
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

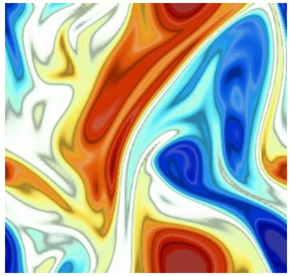


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

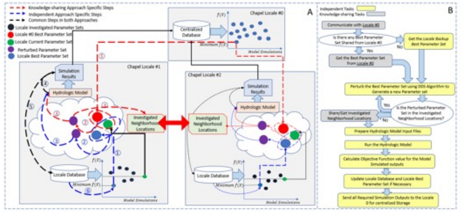


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

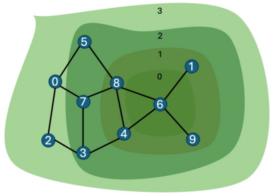
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



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

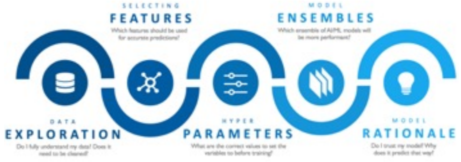


Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



Arachne Graph Analytics
Bader, Du, Rodriguez, et al.
New Jersey Institute of Technology

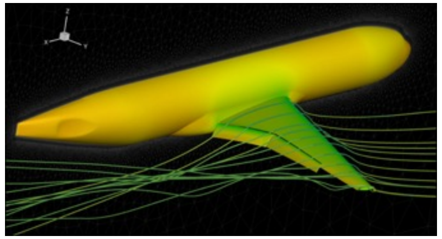
Modeling Ocean Carbon Dioxide Removal
Scott Bachman Brandon Neth, et al.
[C]Worthy



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

[images provided by their respective teams and used with permission]

Productivity Across Diverse Application Scales (code and system size)



Computation: Aircraft simulation / CFD
Code size: 100,000+ lines
Systems: Desktops, HPC systems

7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

Posted on September 17, 2024.

Tags: Computational Fluid Dynamics | User Experiences | Interviews

By: Engin Kayraklioglu, Brad Chamberlain

"Chapel worked as intended: the code maintenance is very much reduced, and its readability is astonishing. This enables undergraduate students to contribute, something almost impossible to think of when using very complex software."

Computation: Coral reef image analysis
Code size: ~300 lines
Systems: Desktops, HPC systems w/ GPUs

7 Questions for Scott Bachman: Analyzing Coral Reefs with Chapel

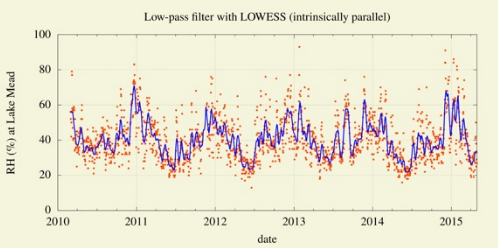
Posted on October 1, 2024.

Tags: Earth Sciences | Image Analysis | GPU Programming | User Experiences | Interviews

By: Brad Chamberlain, Engin Kayraklioglu

In this second installment of our Seven Questions for Chapel Users series, we're looking at a recent success story in which Scott Bachman used Chapel to unlock new scales of biodiversity analysis in coral reefs to study ocean health using satellite image processing. This is work that

"With the coral reef program, I was able to speed it up by a factor of 10,000. Some of that was algorithmic, but Chapel had the features that allowed me to do it."



Computation: Atmospheric data analysis
Code size: 5000+ lines
Systems: Desktops, sometimes w/ GPUs

7 Questions for Nelson Luís Dias: Atmospheric Turbulence in Chapel

Posted on October 15, 2024.

Tags: User Experiences | Interviews | Data Analysis | Computational Fluid Dynamics

By: Engin Kayraklioglu, Brad Chamberlain

In this edition of our Seven Questions for Chapel Users series, we turn to Dr. Nelson Luís Dias from Brazil who is using Chapel to analyze data generated by the Amazon Tall Tower Observatory (ATTO), a project dedicated to long-term, 24/7 monitoring of greenhouse gas fluctuations. Read on

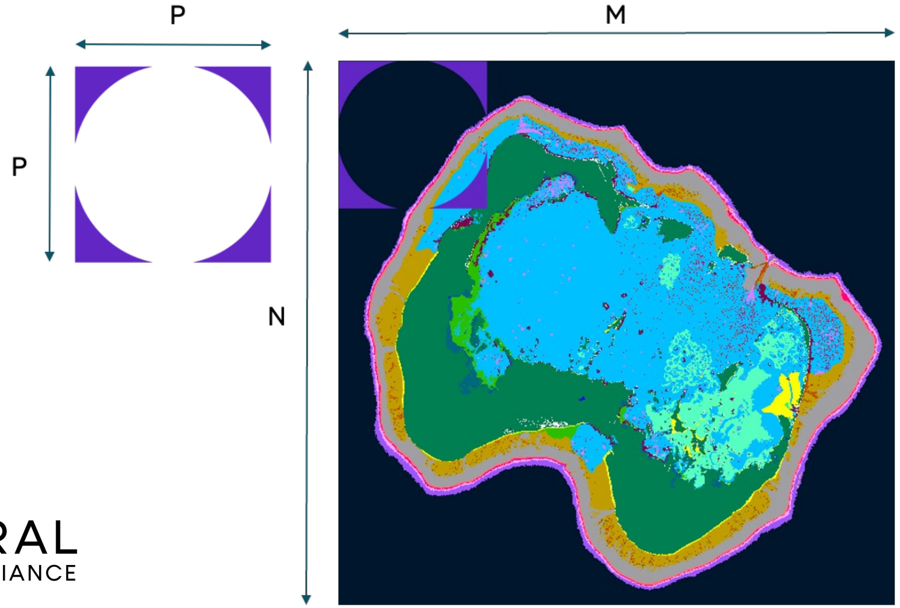
"Chapel allows me to use the available CPU and GPU power efficiently without low-level programming of data synchronization, managing threads, etc."

[read this interview series at: https://chapel-lang.org/blog/series/7-questions-for-chapel-users/]

RapidQ Coral Biodiversity Summary

What is it?

- Measures coral reef diversity using high-res satellite image analysis
- ~230 lines of Chapel code written in late 2022
- Initial code was CPU-only

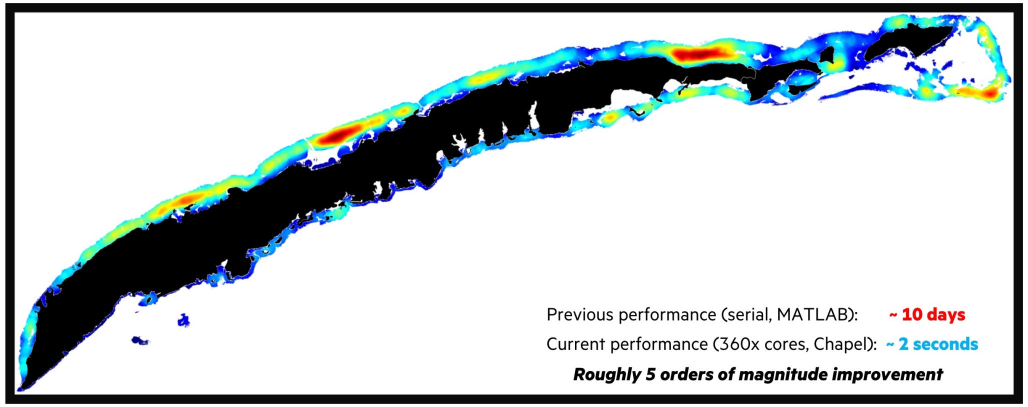


Who wrote it?

- Scott Bachman, NCAR/[C]Worthy
  - with Rebecca Green, Helen Fox, Coral Reef Alliance

Why Chapel?

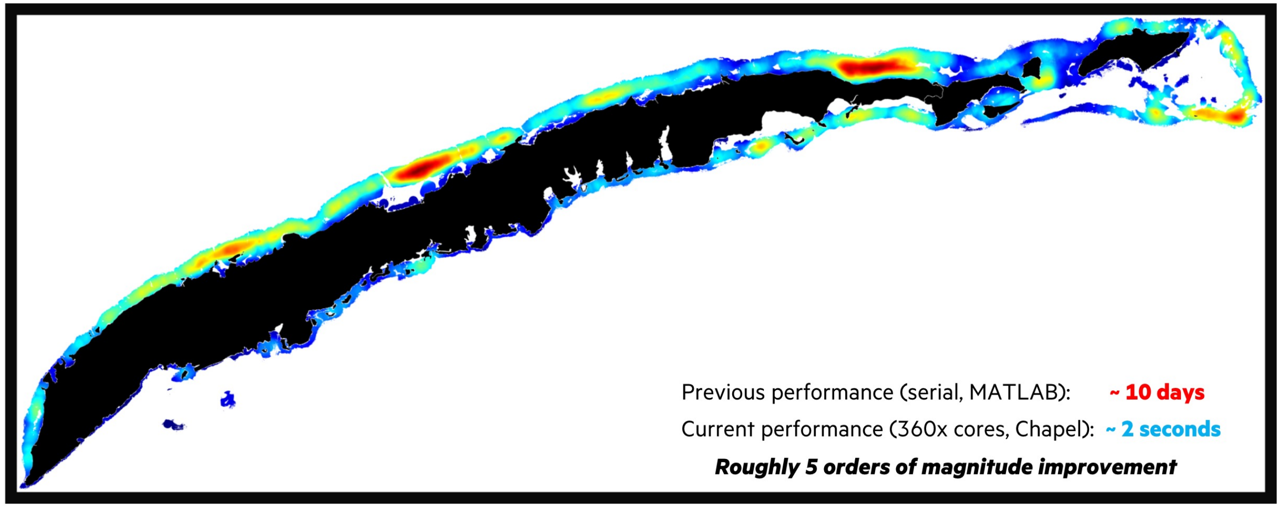
- easy transition from Matlab/Python, which were being used
- massive performance improvement:
  previous ~10-day run finished in ~2 seconds using 360 cores
- enabled unexpected algorithmic improvements



From Scott Bachman's CHIUW 2023 talk: https://youtu.be/lJhh9KLL2X0

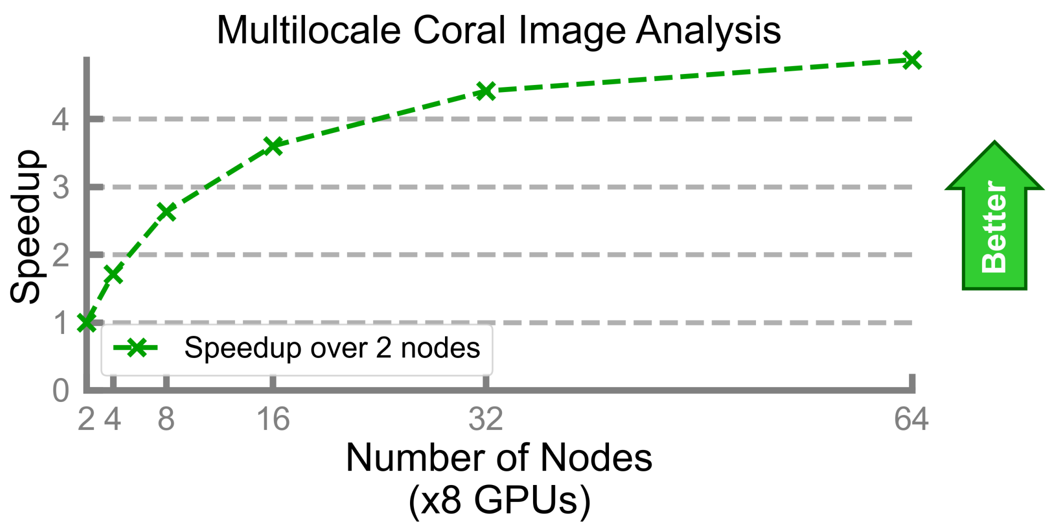
Coral Reef Spectral Biodiversity: Productivity and Performance

Original algorithm: Habitat Diversity, $O(M \cdot N \cdot P)$

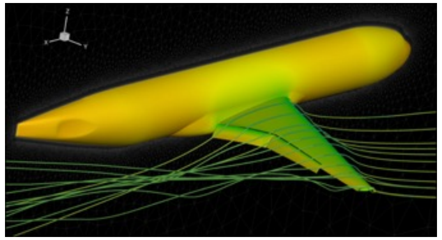


Improved algorithm: Spectral Diversity, $O(M \cdot N \cdot P^3)$

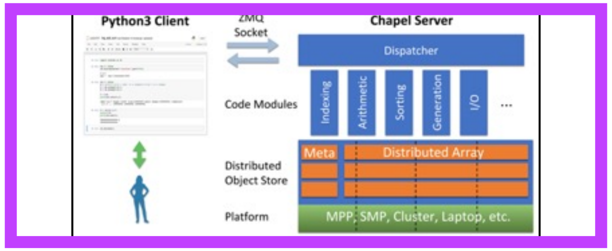
- Chapel run was estimated to require ~4 weeks on 8-core desktop
- updated code to leverage GPUs
  – required adding ~90 lines of code for a total of ~320
- ran in ~20 minutes on 64 nodes of Frontier
  – 512 NVIDIA K20X Kepler GPUs



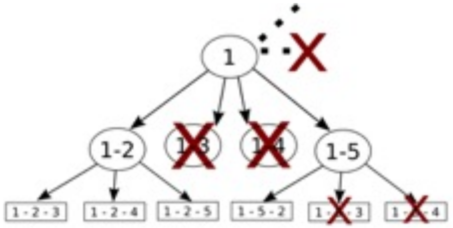
Applications of Chapel



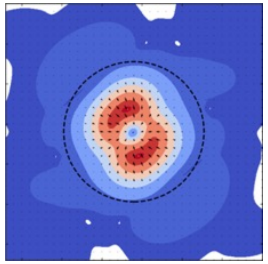
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



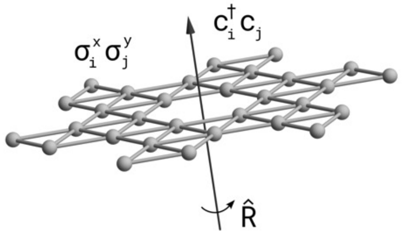
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



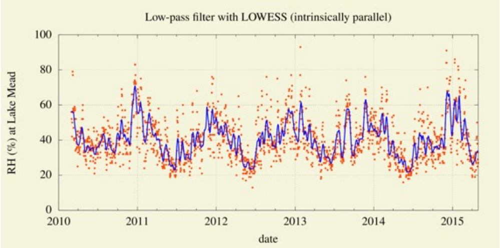
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

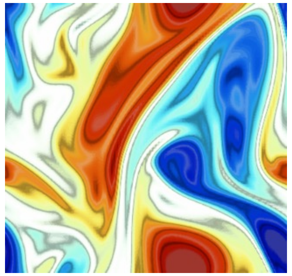


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

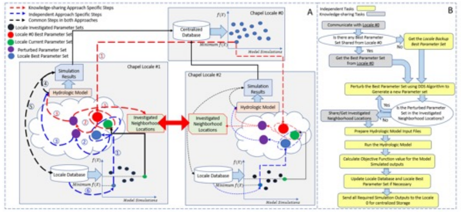


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

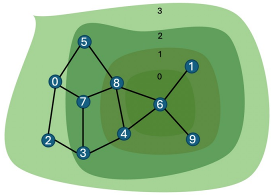
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



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

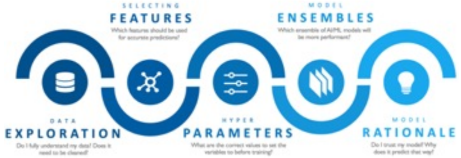


Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



Arachne Graph Analytics
Bader, Du, Rodriguez, et al.
New Jersey Institute of Technology

Modeling Ocean Carbon Dioxide Removal
Scott Bachman Brandon Neth, et al.
[C]Worthy



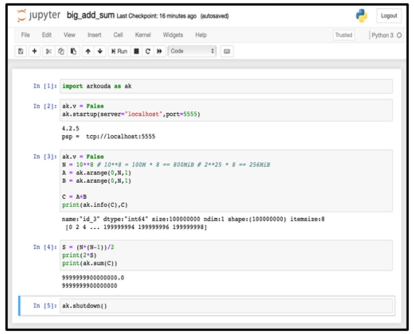
CrayAI HyperParameter Optimization (HPO)
Ben Albrecht et al.
Cray Inc. / HPE

[images provided by their respective teams and used with permission]

What is Arkouda?

Q: “What is Arkouda?”

Arkouda Client
(written in Python)

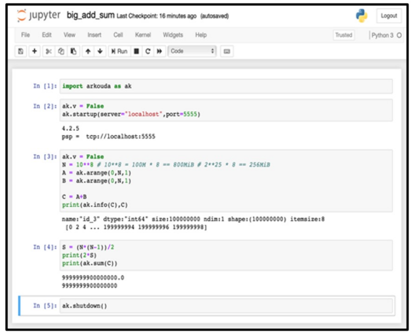


User writes Python code
making familiar NumPy/Pandas calls

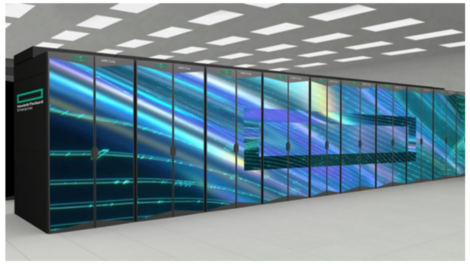
What is Arkouda?

Q: "What is Arkouda?"

Arkouda Client
(written in Python)



Arkouda Server
(written in Chapel)



User writes Python code
making familiar NumPy/Pandas calls

A: "A scalable version of NumPy / Pandas for data scientists"

Performance and Productivity: Arkouda Argsort

HPE Cray EX

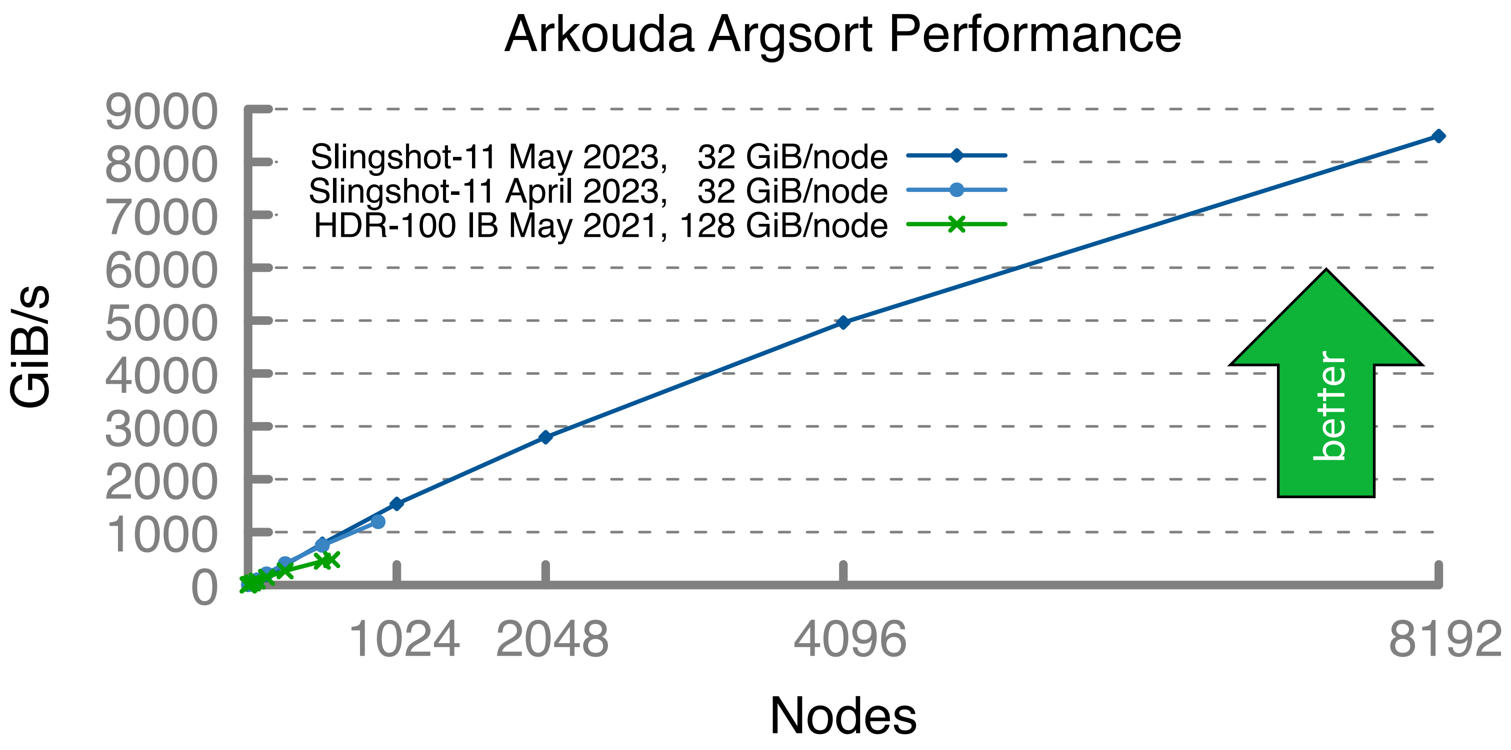
- Slingshot-11 network (200 Gb/s)
- 8192 compute nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)

HPE Cray EX

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

HPE Apollo

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

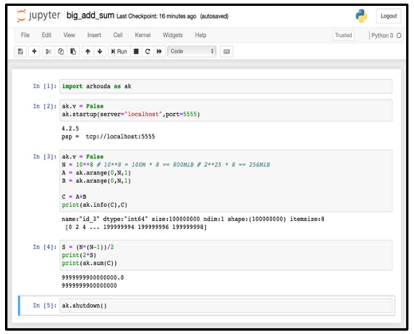


Implemented using ~100 lines of Chapel

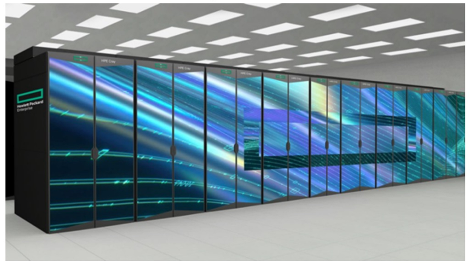
What is Arkouda?

Q: “What is Arkouda?”

Arkouda Client
(written in Python)



Arkouda Server
(written in Chapel)



User writes Python code
making familiar NumPy/Pandas calls

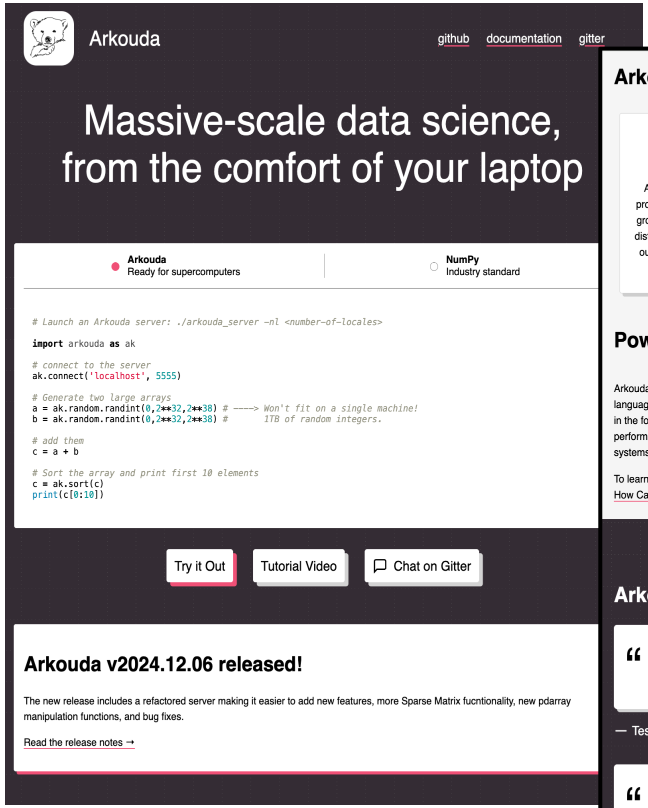
Classic Arkouda: “A scalable version of NumPy / Pandas for data scientists”

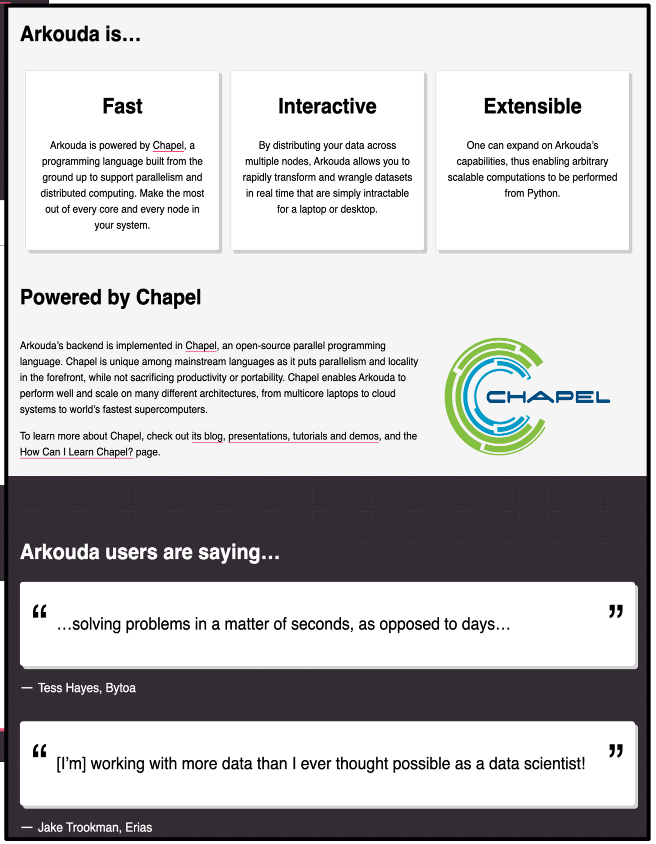
Current Arkouda: “An extensible framework for arbitrary HPC computations”

Both: “A way to drive HPC systems interactively from Python on a laptop”

For More Information

Arkouda website: https://arkouda-www.github.io/





Arkouda Interview

Blog: Interview with co-founding developer, Bill Reus: https://chapel-lang.org/blog/posts/7qs-reus/

Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts

7 Questions for Bill Reus: Interactive Supercomputing with Chapel for Cybersecurity

Posted on February 12, 2025.

Tags: User Experiences | Interviews | Data Analysis | Arkouda

By: Engin Kayraklioglu, Brad Chamberlain

Table of Contents

We're very excited to kick off the 2025 edition of our Seven Questions for Chapel Users series with the following interview with Bill Reus. Bill is one of the co-creators of Arkouda, which is one of Chapel's flagship applications. To learn more about Arkouda and its support for interactive data analysis at massive scales, read on!

1. Who are you?

My name is Bill Reus, and I live near Annapolis, MD and the beautiful Chesapeake Bay. I am currently a data scientist doing statistical modeling and simulation for the United States government, but I began my career as an experimental chemist. In graduate school, I measured electron transport through thin films of organic molecules using an apparatus that our group invented to collect large volumes of noisy data quickly and with low cost. This approach contrasted with the typical means of studying molecular electronics, which was to spend weeks or months collecting a small number of exquisite measurements in ultra-high vacuum and at ultra-low temperature.

"I was on the verge of resigning myself to learning MPI when I first encountered Chapel. After writing my first Chapel program, I knew I had found something much more appealing."

...

"Chapel's separation of concerns immediately felt like the most natural way to think about large-scale computing. I would highly encourage anyone wanting to get into HPC programming to start with Chapel."

Chapel Characteristics

Six Key Characteristics of Chapel

1. **portable:** runs on laptops, clusters, the cloud, supercomputers

Where Does Chapel Run?

In the Browser:
- GitHub Codespaces
- Attempt This Online (ATO)

Laptops/Desktops:
- Linux/UNIX
- Mac OS X
- Windows (leveraging WSL)

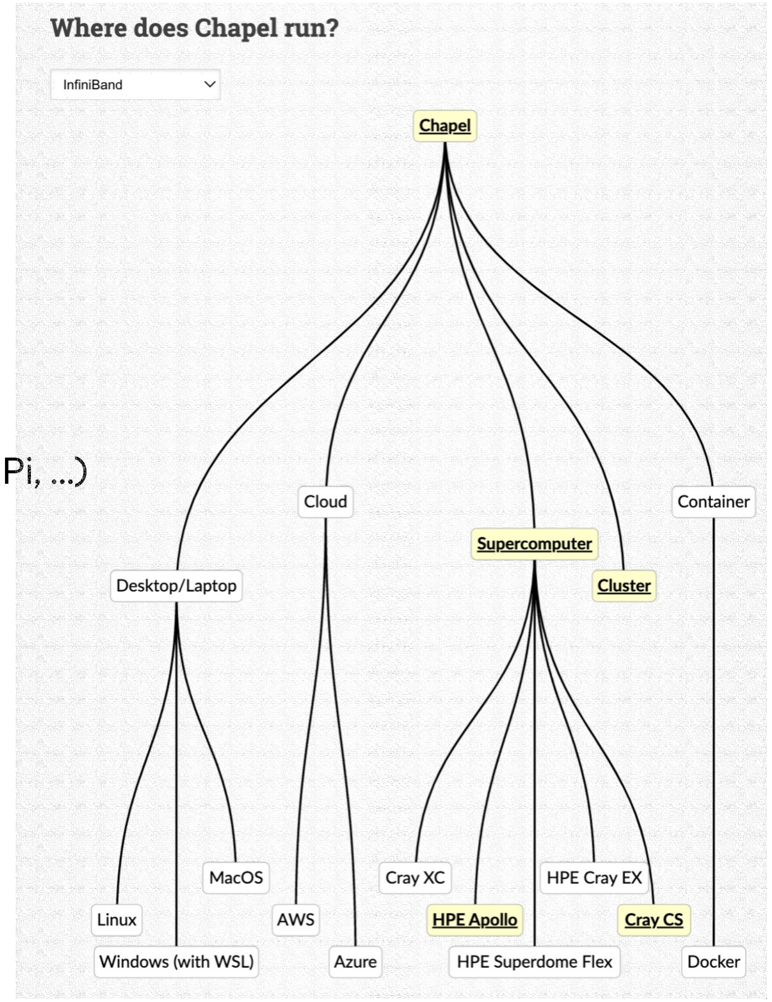
HPC Systems:
- Commodity clusters
- HPE/Cray supercomputers, such as:
  - Frontier
  - Perlmutter
  - Piz Daint
  - Polaris
  - ...
- Other vendors' supercomputers

Cloud:
- AWS
- Microsoft Azure (?)
- Google Cloud (?)

CPUs:
- Intel
- AMD
- Arm (M1/M2, Graviton, A64FX, Raspberry Pi, ...)

GPUs:
- NVIDIA
- AMD

Networks:
- Slingshot
- Aries/Gemini
- InfiniBand
- AWS EFA
- Ethernet

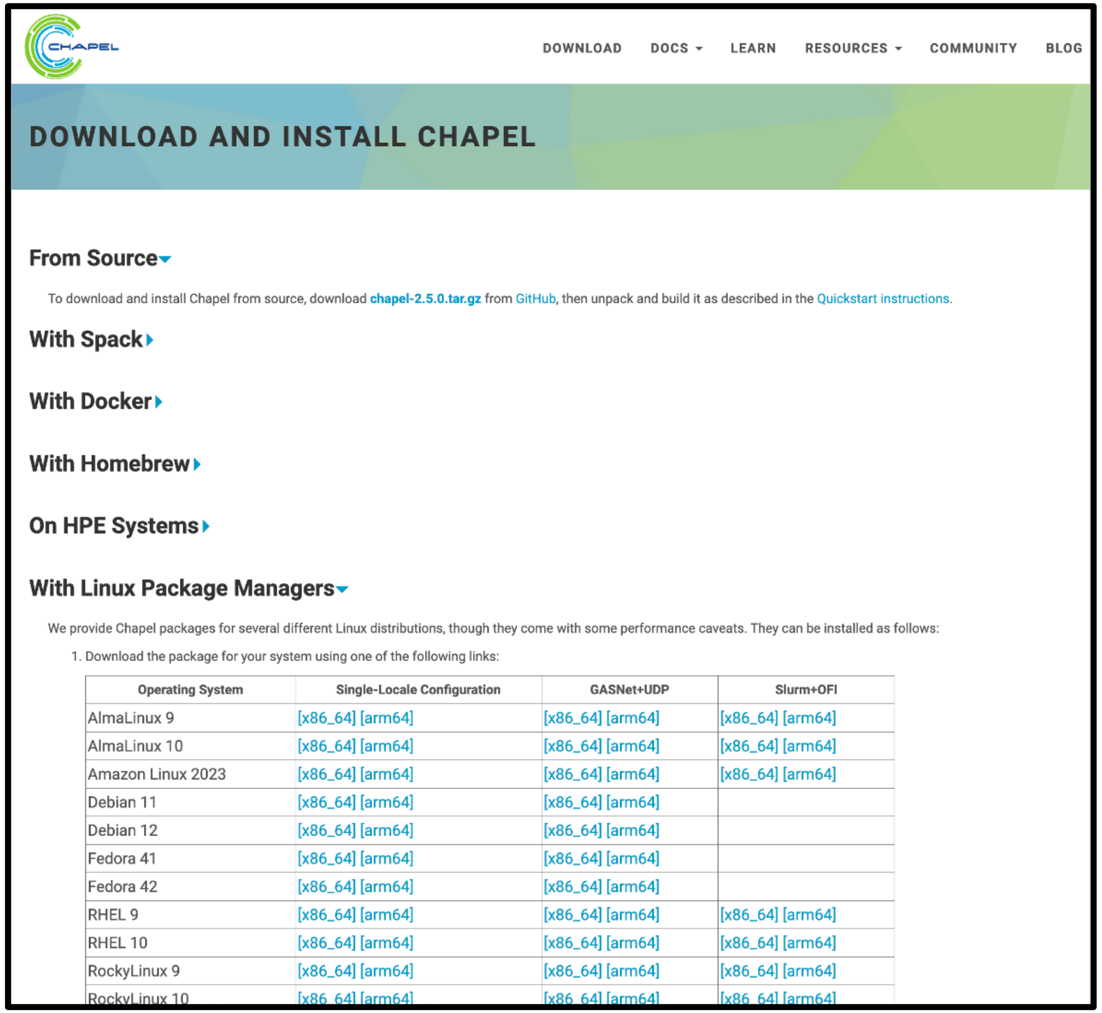


https://chapel-lang.org/docs/usingchapel/portability.html

Where can I get Chapel?

Release Formats:

- Source releases via GitHub
- Spack
- E4S
- Linux packages via apt/rpm
- Homebrew
- Docker
- Modules on HPE Cray systems
- ATO / GitHub Codespaces



CHAPEL
DOWNLOAD DOCS LEARN RESOURCES COMMUNITY BLOG

DOWNLOAD AND INSTALL CHAPEL

From Source

To download and install Chapel from source, download chapel-2.5.0.tar.gz from GitHub, then unpack and build it as described in the Quickstart instructions.

With Spack

With Docker

With Homebrew

On HPE Systems

With Linux Package Managers

We provide Chapel packages for several different Linux distributions, though they come with some performance caveats. They can be installed as follows:

1. Download the package for your system using one of the following links:

Operating System	Single-Locale Configuration	GASNet+UDP	Slurm+OFI
AlmaLinux 9	[x86_64] [arm64]	[x86_64] [arm64]	[x86_64] [arm64]
AlmaLinux 10	[x86_64] [arm64]	[x86_64] [arm64]	[x86_64] [arm64]
Amazon Linux 2023	[x86_64] [arm64]	[x86_64] [arm64]	[x86_64] [arm64]
Debian 11	[x86_64] [arm64]	[x86_64] [arm64]	
Debian 12	[x86_64] [arm64]	[x86_64] [arm64]	
Fedora 41	[x86_64] [arm64]	[x86_64] [arm64]	
Fedora 42	[x86_64] [arm64]	[x86_64] [arm64]	
RHEL 9	[x86_64] [arm64]	[x86_64] [arm64]	[x86_64] [arm64]
RHEL 10	[x86_64] [arm64]	[x86_64] [arm64]	[x86_64] [arm64]
RockyLinux 9	[x86_64] [arm64]	[x86_64] [arm64]	[x86_64] [arm64]
RockyLinux 10	[x86_64] [arm64]	[x86_64] [arm64]	[x86_64] [arm64]

https://chapel-lang.org/download/

Six Key Characteristics of Chapel

1. **portable:** runs on laptops, clusters, the cloud, supercomputers

2. **open-source:** to reduce barriers to adoption and leverage community contributions

Chapel is Open-Source

- Developed at, and released through, GitHub
  - Using the Apache 2.0 license
  - Free to download and use
- Accepts and benefits from contributions from the community
- Recently accepted into the Linux Foundation and HPSF (High Performance Software Foundation)

Six Key Characteristics of Chapel

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

Chapel Interoperability

- Like most modern languages, Chapel is designed to interoperate with others
  - In practice, the most common tend to be C, C++, Fortran, and Python

- Two modes:
  - Chapel owns 'main( )' and calls out to routines in other languages
  - User builds a Chapel library and invokes it from another language

- Resources:
  - Interoperability technical notes: https://chapel-lang.org/docs/technotes/index.html#interoperability
  - Library for calling out to Python: https://chapel-lang.org/docs/modules/packages/Python.html

Six Key Characteristics of Chapel

1. **portable:** runs on laptops, clusters, the cloud, supercomputers
2. **open-source:** to reduce barriers to adoption and leverage community contributions
3. **compiled:** to generate the best performance possible
4. **statically typed:** to avoid simple errors after hours of execution
5. **interoperable:** with C, C++, Fortran, Python, …
6. **from scratch:** not a dialect or extension of another language
   (though inspiration was taken from many)

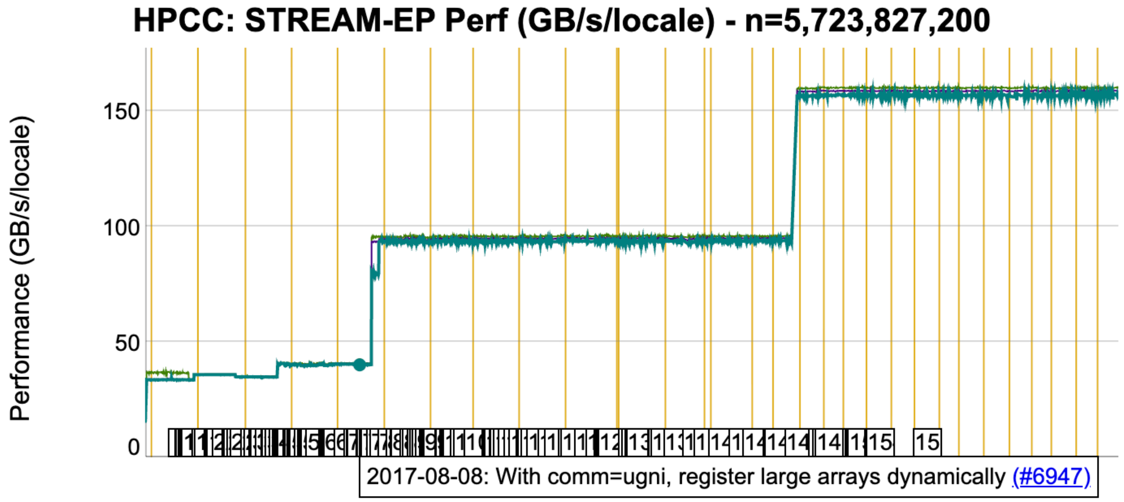
Chapel Tools

- **VSCode support:** see https://marketplace.visualstudio.com/items?itemName=chpl-hpe.chapel-vscode
- **chpl-language-server:** provides Chapel code intelligence for most editors (VSCode, vim, emacs, …)
- **chplcheck:** a Chapel linter
- **c2chapel:** converts C header files to Chapel extern declarations in support of interoperability
- **chpldoc:** HTML-based rendering of comment-based documentation
- **chapel-py:** Python bindings to the Chapel compiler front-end
- **mason:** Chapel's package manager

More information available at https://chapel-lang.org/docs/tools/

Frameworks for Testing Chapel Code

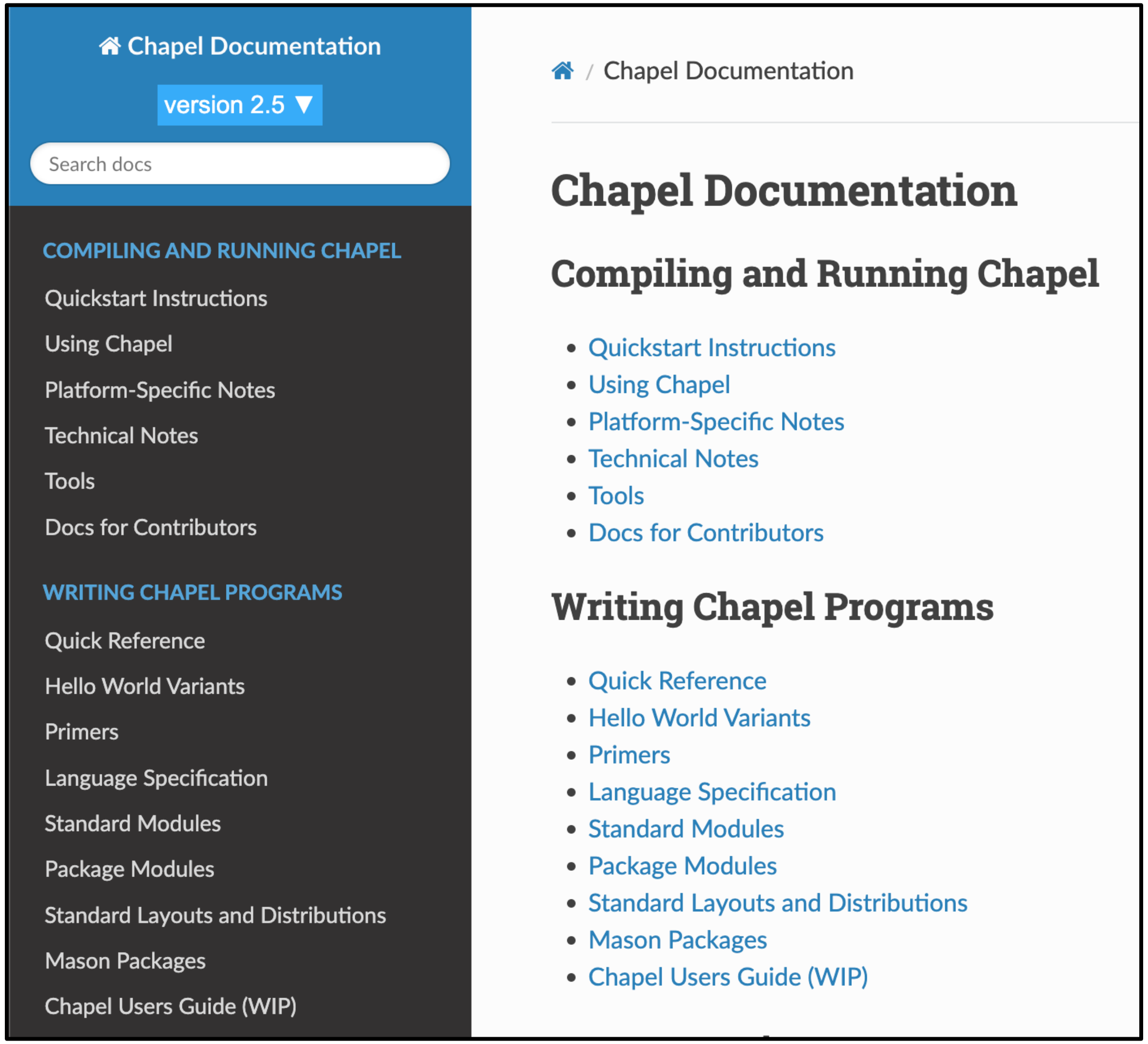
1. Chapel testing system (https://chapel-lang.org/docs/developer/bestPractices/TestSystem.html):
   - forms the basis of Chapel's nightly regression testing
   - supports two modes of testing
     - **correctness:** compiles and runs program, capturing output and comparing to a '.good' file
     - **performance:** tracks key-value pairs and supports plotting values over time



2. UnitTest package library: https://chapel-lang.org/docs/modules/packages/UnitTest.html

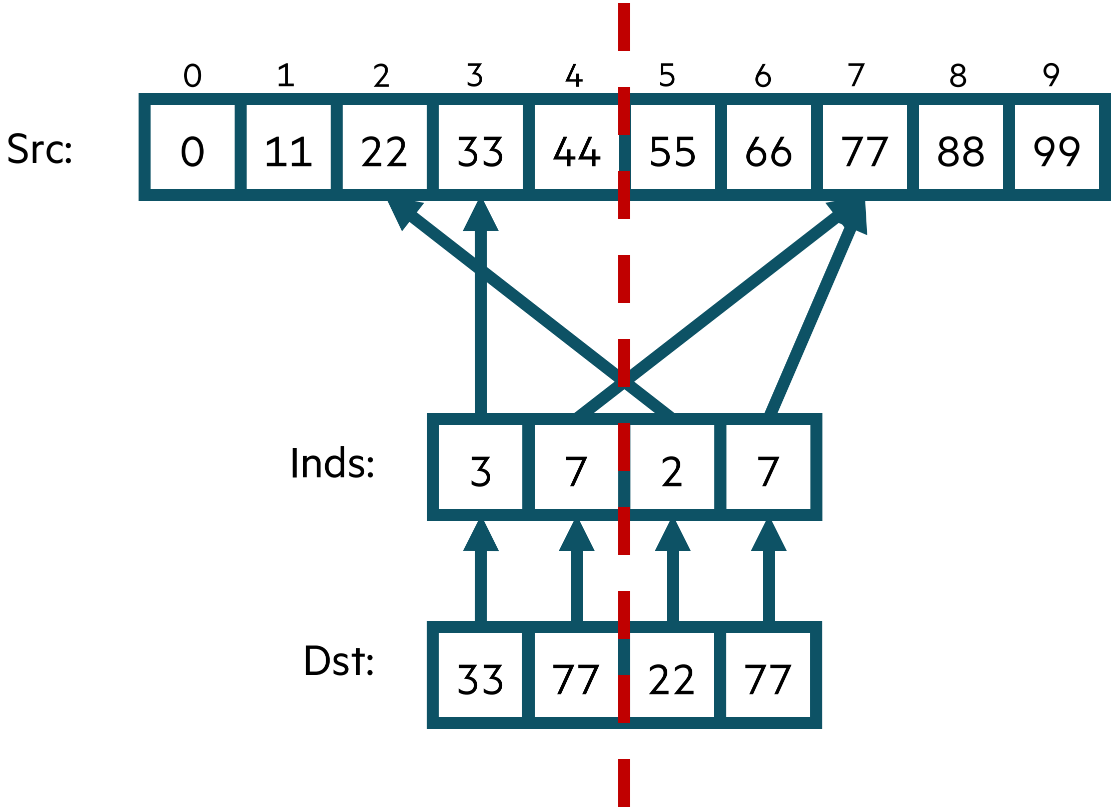
Chapel Documentation

- Docs hierarchy: https://chapel-lang.org/docs/
- Key elements:
  - Getting started with Chapel (building and configuring)
  - Platform-specific notes (Mac, Windows, AWS, HPE, …)
  - Primers (programs that teach specific features)
  - Library module documentation:
    - standard modules
    - package modules
  - Language specification



Sample Computation:
Bale Index Gather (IG)

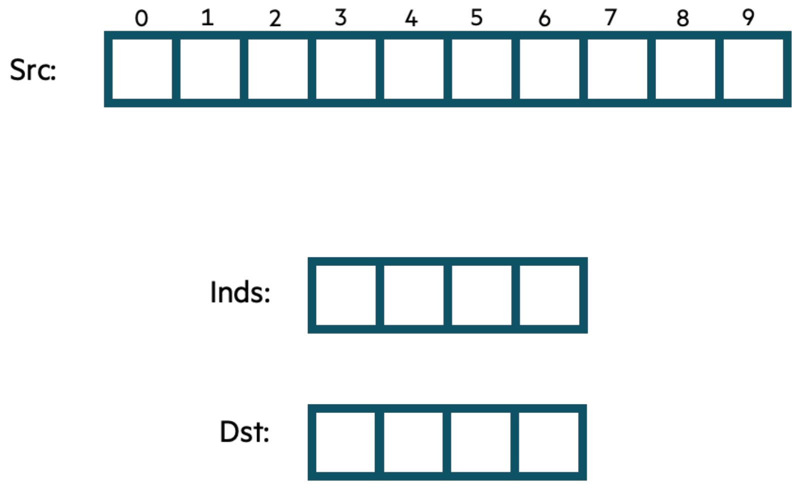
Bale Index Gather (IG): In Pictures



Bale IG in Chapel: Array Declarations

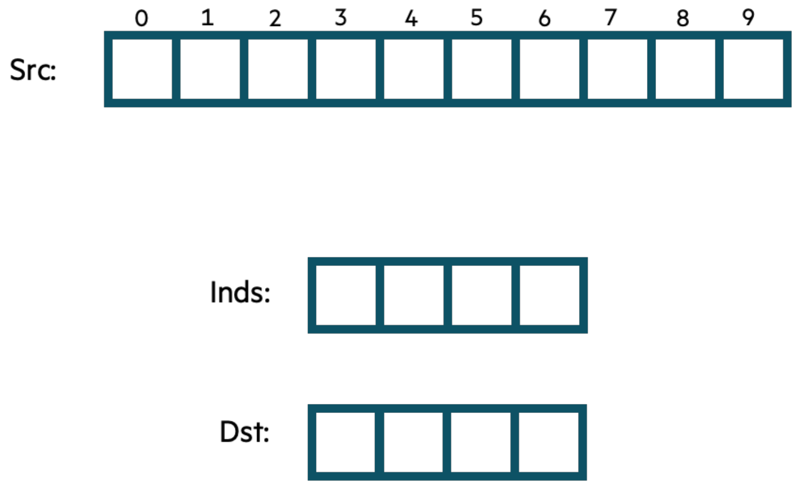
```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
```

```
$
```



Bale IG in Chapel: Compiling

```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
```

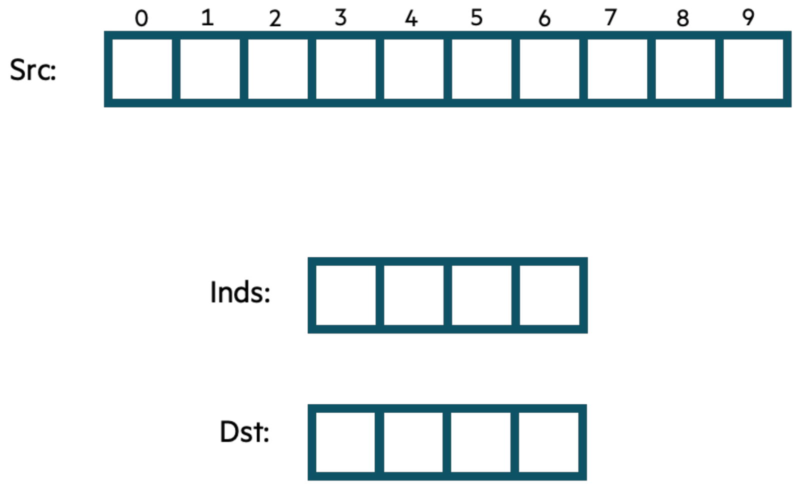


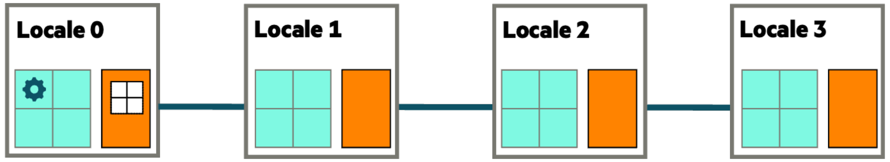

```
$ chpl bale-ig.chpl
$
```

Bale IG in Chapel: Executing

```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
```

```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

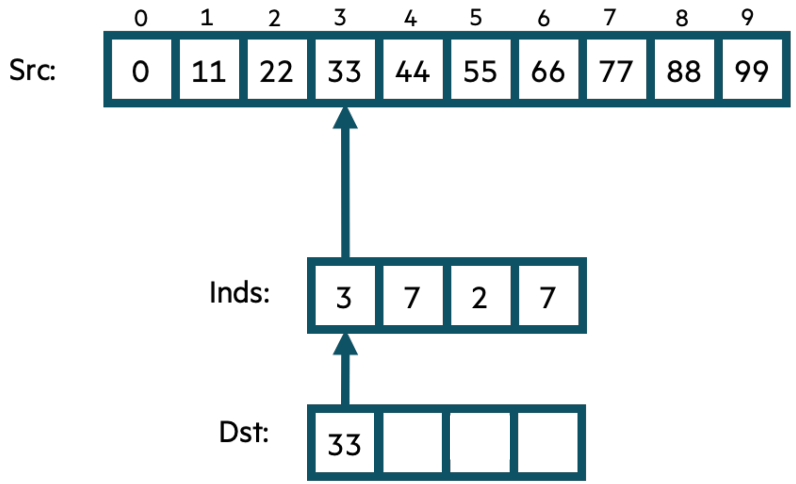


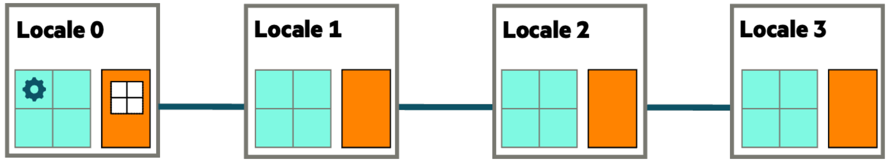


Bale IG in Chapel: Serial, Zippered Version

```
config const n = 10,
             m = 4;




var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
…
for (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

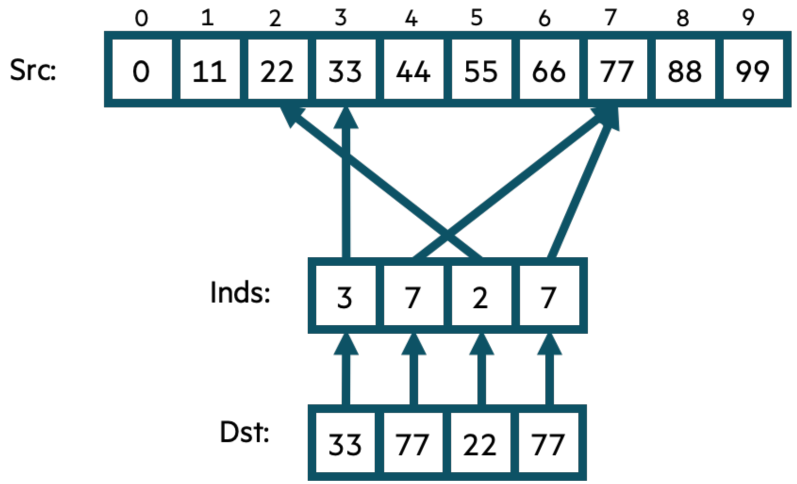


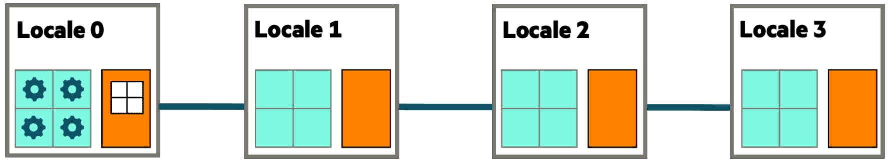


Bale IG in Chapel: Parallel, Zippered Version (Multicore)

```
config const n = 10,
             m = 4;




var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

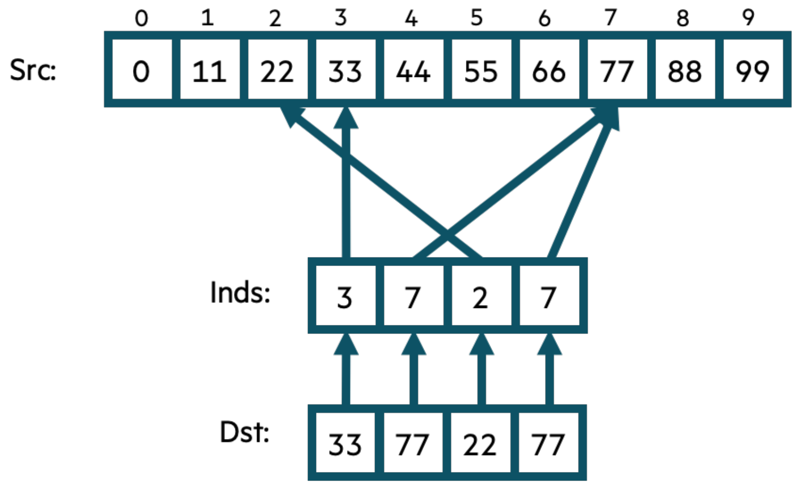


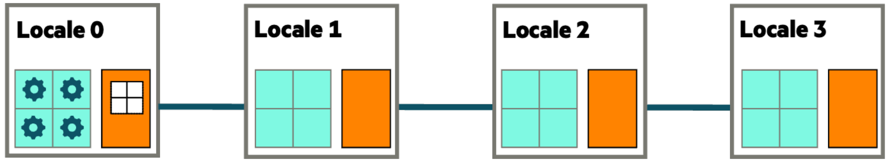


Bale IG in Chapel: Parallel , Zippered Version with Named Domains (Multicore)

```
config const n = 10,
             m = 4;

const SrcInds = {0..<n},
      DstInds = {0..<m};

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

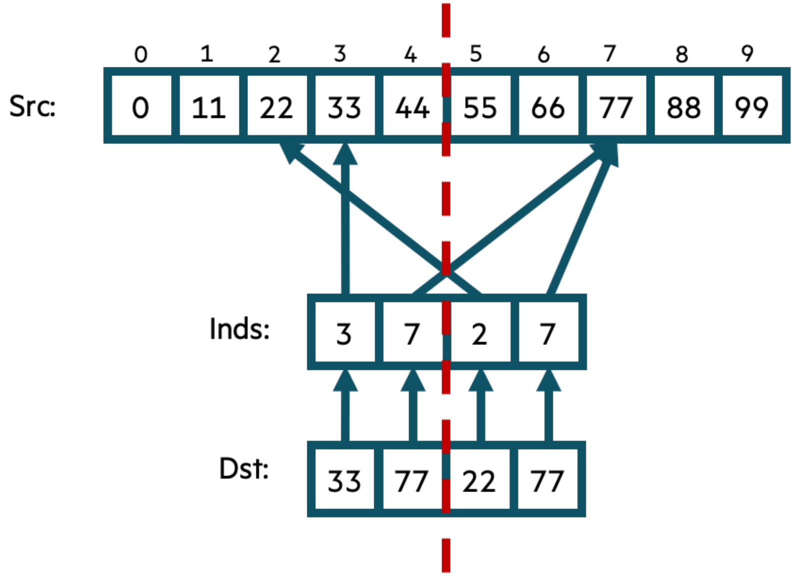


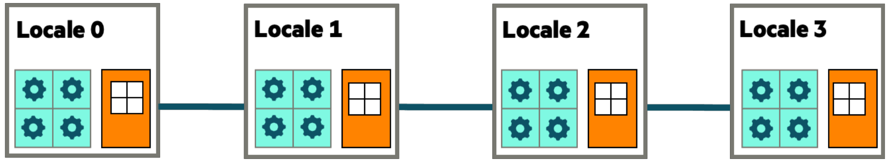


Bale IG in Chapel: Distributed Parallel Version

```
use BlockDist;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

```
$ chpl bale-ig.chpl
$ ./bale-ig --n=… --m=… -nl 4
$
```

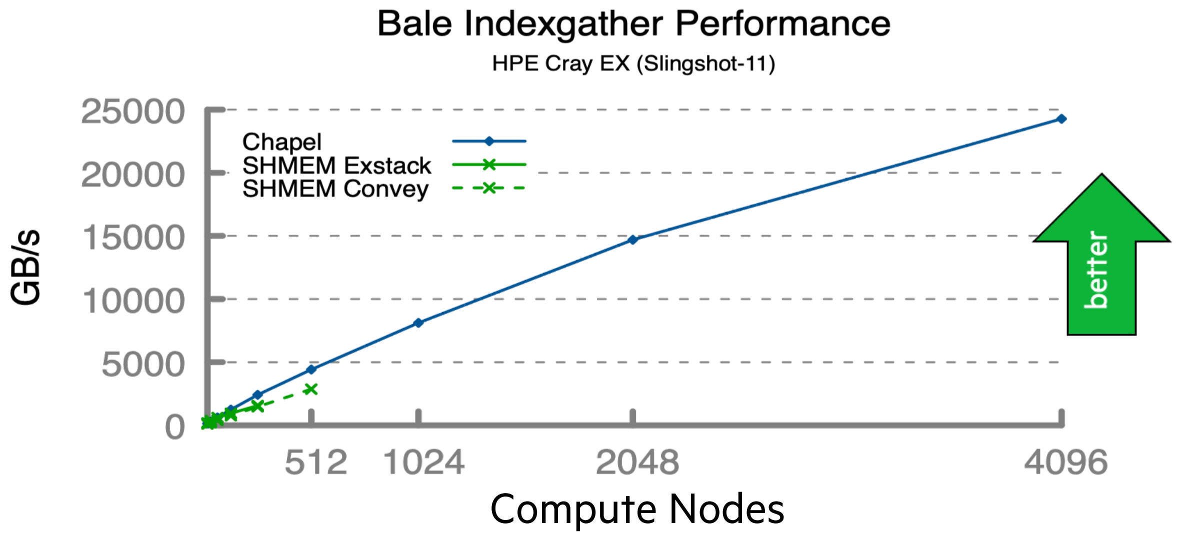


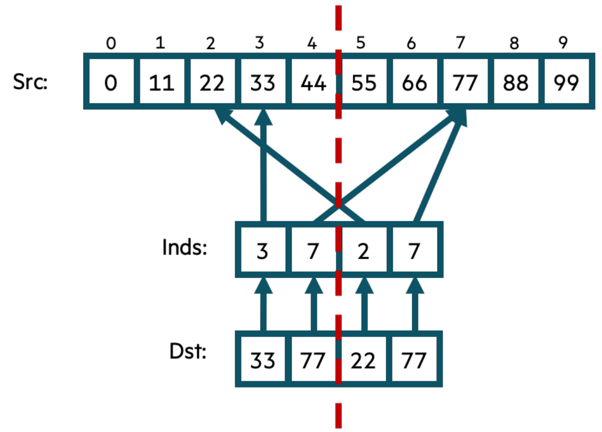


Bale IG in Chapel: Distributed Parallel Version on HPE Cray EX (Slingshot-11)

```
use BlockDist;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

```
$ chpl bale-ig.chpl --fast --auto-aggregation
$ ./bale-ig --n=… --m=… -nl 4096
$
```





Bale IG in Chapel vs. SHMEM on HPE Cray EX (Slingshot-11)

Chapel (Simple / Auto-Aggregated version)

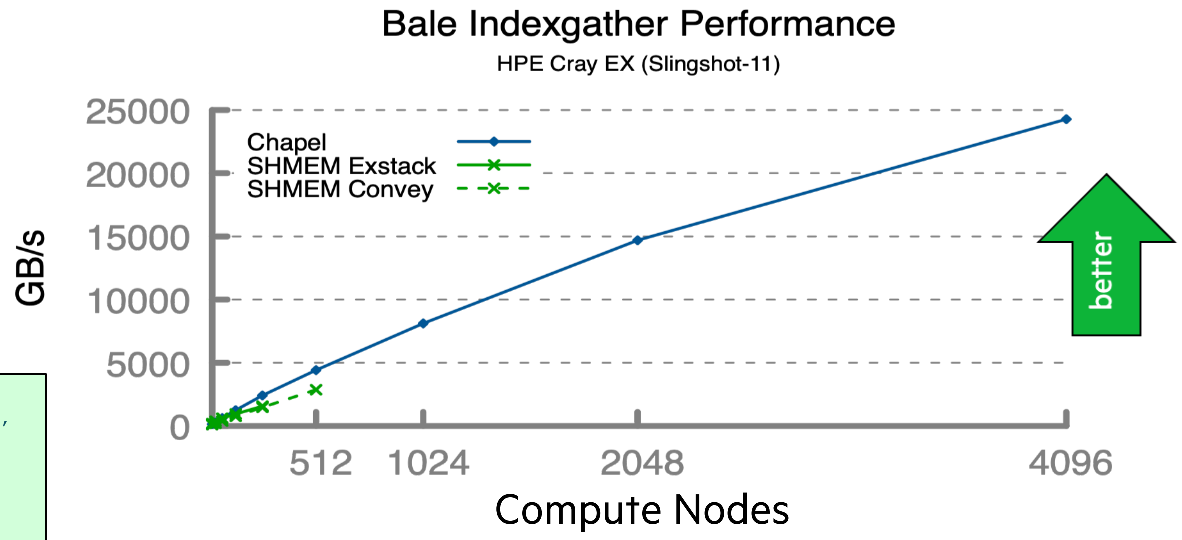
```
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

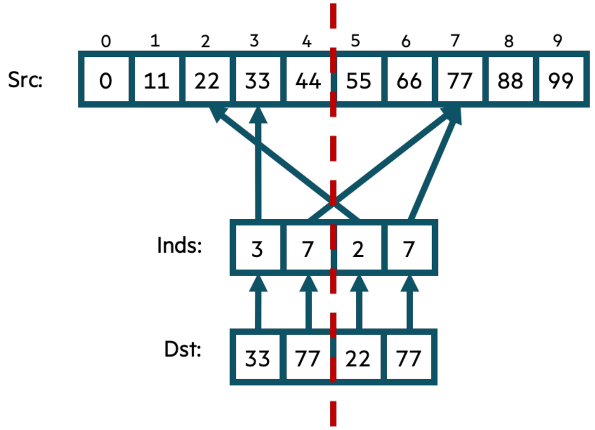
SHMEM (Exstack version)

```
i=0;
while( exstack_proceed(ex, (i==l_num_req)) ) {
  i0 = i;
  while(i < l_num_req) {
    l_indx = pckindx[i] >> 16;
    pe  = pckindx[i] & 0xffff;
    if(!exstack_push(ex, &l_indx, pe))
      break;
    i++;
  }

  exstack_exchange(ex);

  while(exstack_pop(ex, &idx , &fromth)) {
    idx  = ltable[idx];
    exstack_push(ex, &idx, fromth);
  }
  lgp_barrier();
  exstack_exchange(ex);

  for(j=i0; j<i; j++) {
    fromth = pckindx[j] & 0xffff;
    exstack_pop_thread(ex, &idx, (uint64_t)fromth);
    tgt[j] = idx;
  }
  lgp_barrier();
}
```

SHMEM (Conveyors version)

```
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
       more | convey_advance(replies, !more)) {

  for (; i < l_num_req; i++) {
    pkg.idx = i;
    pkg.val = pckindx[i] >> 16;
    pe = pckindx[i] & 0xffff;
    if (! convey_push(requests, &pkg, pe))
      break;
  }

  while (convey_pull(requests, ptr, &from) == convey_OK) {
    pkg.idx = ptr->idx;
    pkg.val = ltable[ptr->val];
    if (! convey_push(replies, &pkg, from)) {
      convey_unpull(requests);
      break;
    }
  }

  while (convey_pull(replies, ptr, NULL) == convey_OK)
    tgt[ptr->idx] = ptr->val;
}
```





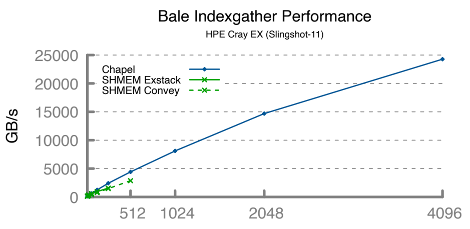
Wrap-up

Summary

Chapel is unique among programming languages

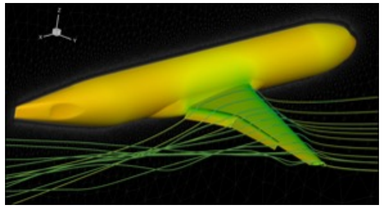
- features first-class concepts for parallelism and locality
- ports and scales from laptops to supercomputers
- supports clean, concise code relative to conventional approaches
- supports GPUs in a vendor-neutral manner

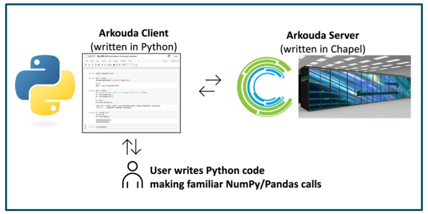
```
use BlockDist;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

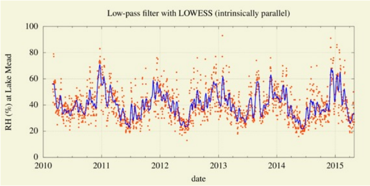


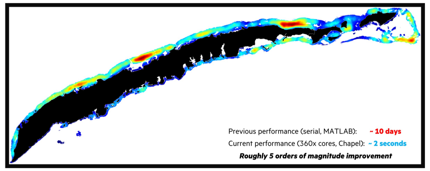
Chapel is being used for productive parallel computing at all scales

- users are reaping its benefits in practical, cutting-edge applications
- applicable to domains as diverse as physical simulations and data science
- Arkouda is a notable case, supporting extensible, interactive HPC









Ways to interact with or follow the Chapel Community

“Live” (Virtual) Community Events

- Project Meetings, weekly
- Deep Dive / Demo Sessions, weekly timeslot
- ChapelCon (formerly CHIUW), annually

Electronic Broadcasts

- Chapel Blog, typically 1–4 articles per month
- Community Newsletter, quarterly
- Announcement Emails, around big events

Social Media

FOLLOW US

- BlueSky
- Facebook
- LinkedIn
- Mastodon
- Reddit
- X (Twitter)
- YouTube

Discussion Forums

GET IN TOUCH

- Discord
- Discourse
- Email
- GitHub Issues
- Gitter
- Stack Overflow

Ways to Use Chapel

GET STARTED

- Attempt This Online
- Docker
- E4S
- GitHub Releases
- Homebrew
- Spack

(from the footer of chapel-lang.org)

TIMELINE SESSIONS ORGANIZATION CONTACT CHAPEL WEBSITE

ChapelCon '25

October 7-10, 2025

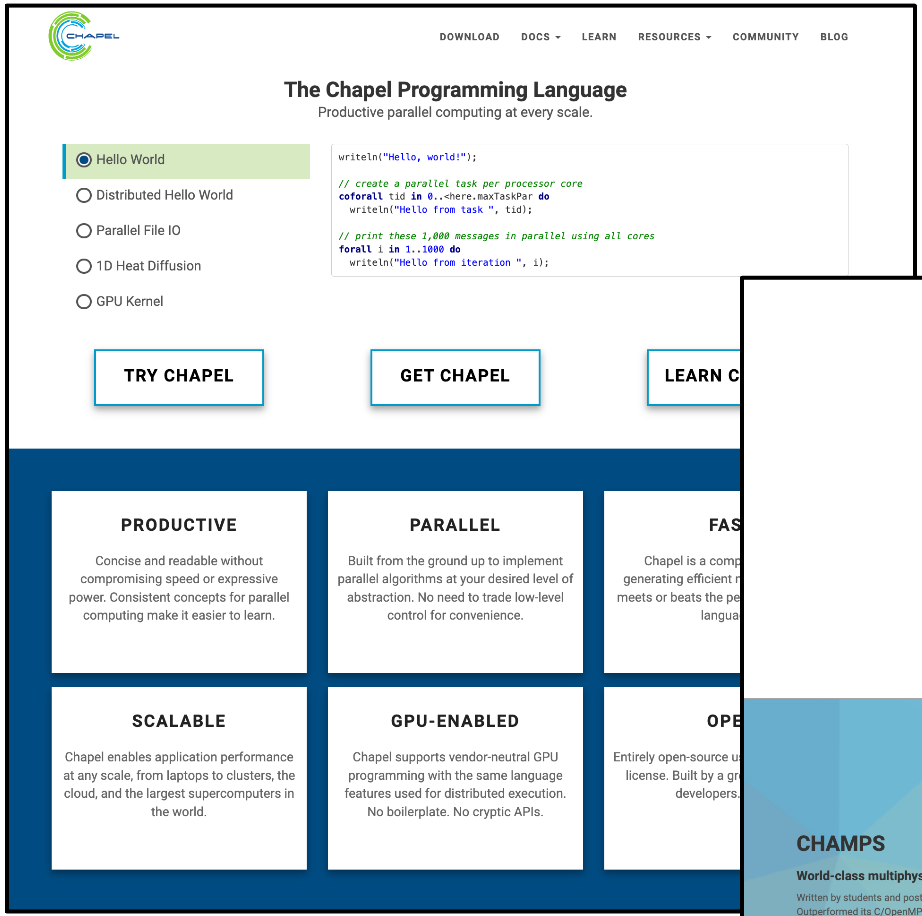
Join us at *the* annual Chapel Programming Language event to talk about the language, libraries, and applications! ChapelCon is free to attend and will be held virtually.

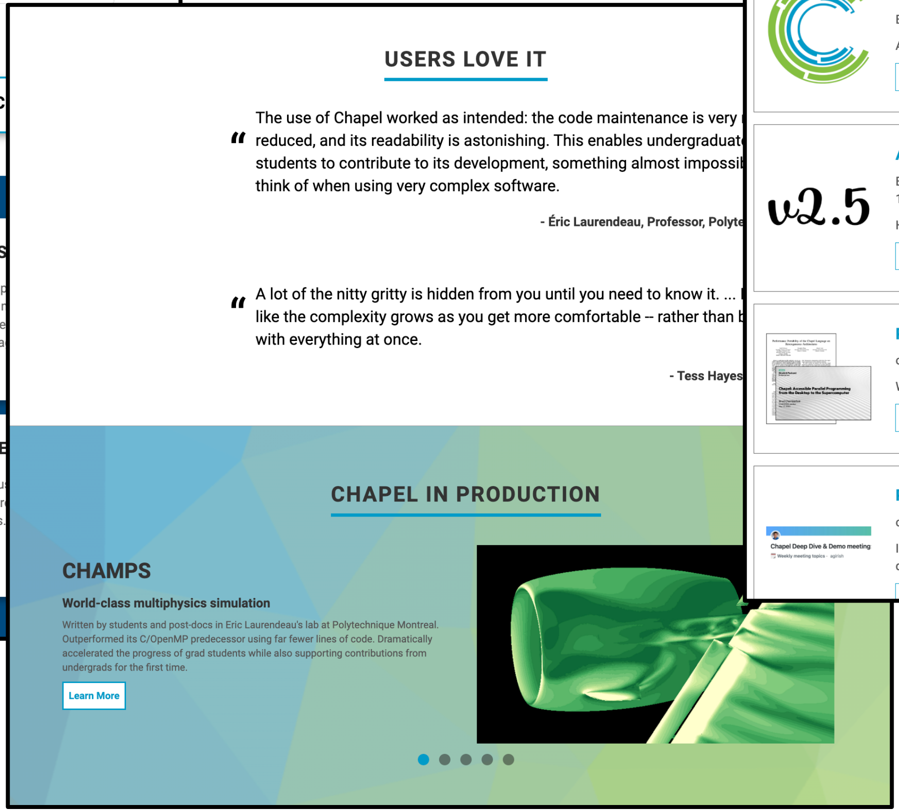
Registration | Office Hours Signup | Tutorial Topic Requests

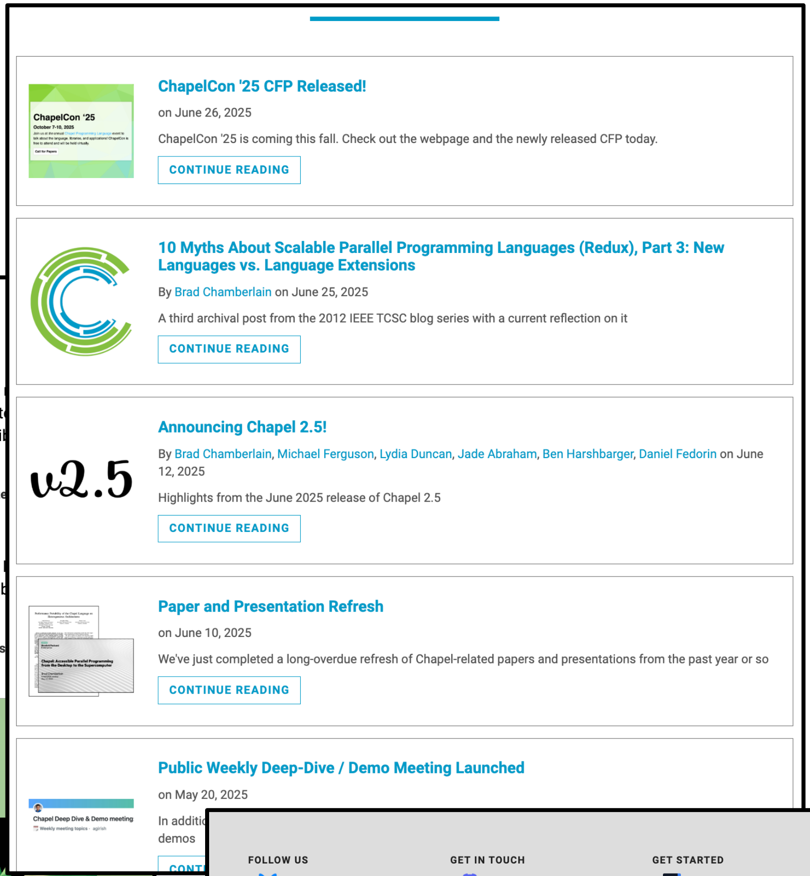
ChapelCon '25 welcomes anyone with computing challenges that demand performance, particularly through parallelism and scalability. ChapelCon '25 brings together Chapel users, enthusiasts, researchers, and developers to exchange ideas, present their work, and forge new collaborations. Anyone interested in parallel programming, programming languages, or high performance computing is encouraged to attend. A wide range of sessions support all levels of experience, with Tutorials and Free Coding sessions for those looking to hone their skills, Office Hour sessions for those looking for help from Chapel developers, and Conference sessions for those looking to share and discuss their work. **ChapelCon '25 is free to attend and will be held virtually.**

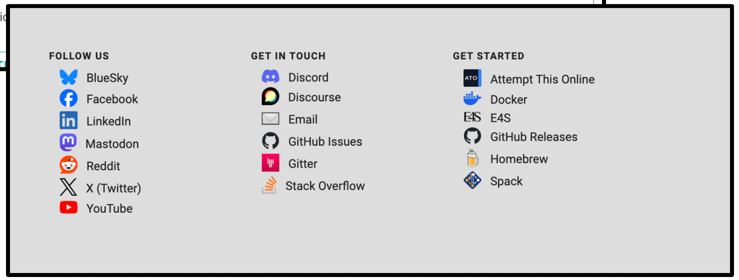
https://chapel-lang.org/chapelcon25/

Chapel Website









chapel-lang.org

Thank you

https://chapel-lang.org
@ChapelLanguage