



**Hewlett Packard  
Enterprise**

# **Parallel Programming from Desktops to Supercomputers with Chapel**

Brad Chamberlain  
Galois Tech Talk  
July 24, 2025

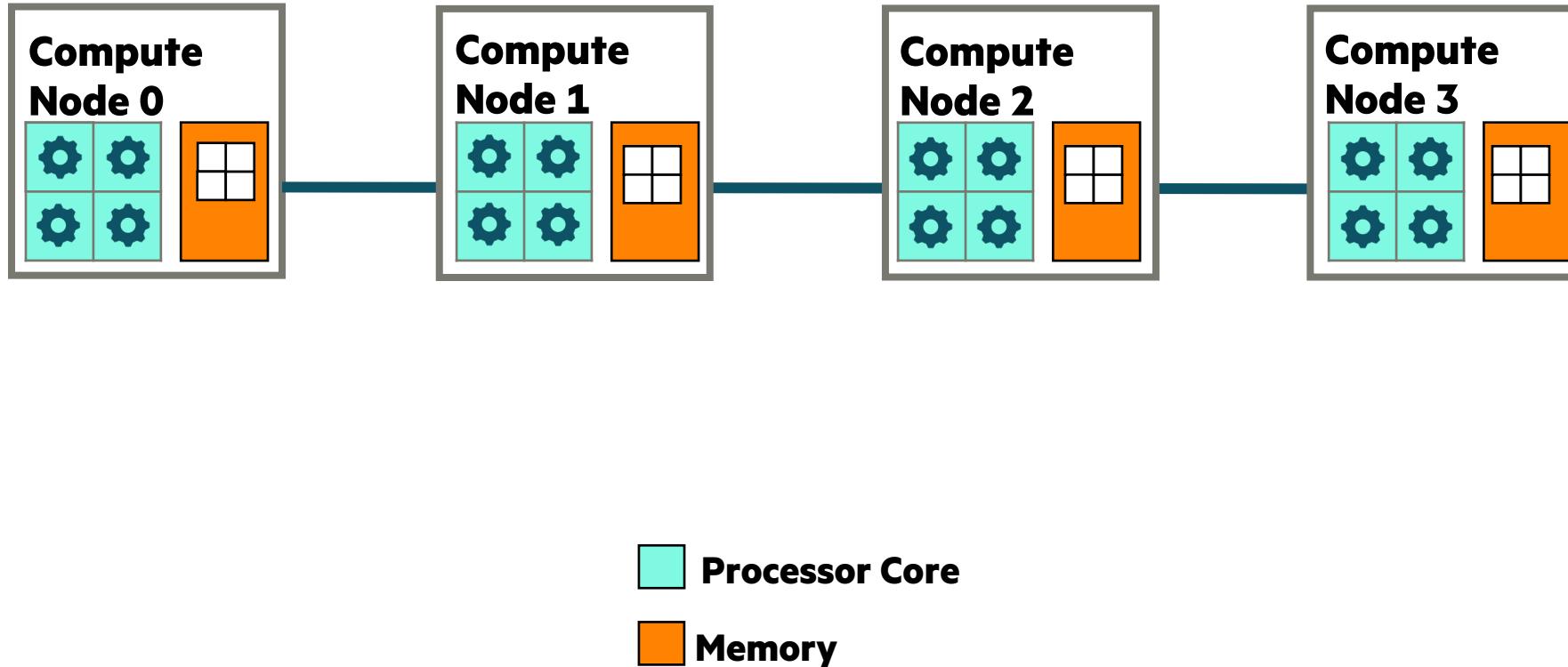
A close-up photograph of a middle-aged man with short, light-colored hair. He is smiling warmly at the camera. He is wearing a dark blue zip-up jacket over a green and white plaid shirt. A pair of glasses hangs from the neckline of his shirt. He is seated at a desk, facing a white laptop computer. In the background, there is a bright yellow wall and a blurred figure of another person, suggesting a casual office or study environment.

**A Bit About Me**

# Motivation for Parallel Computing

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

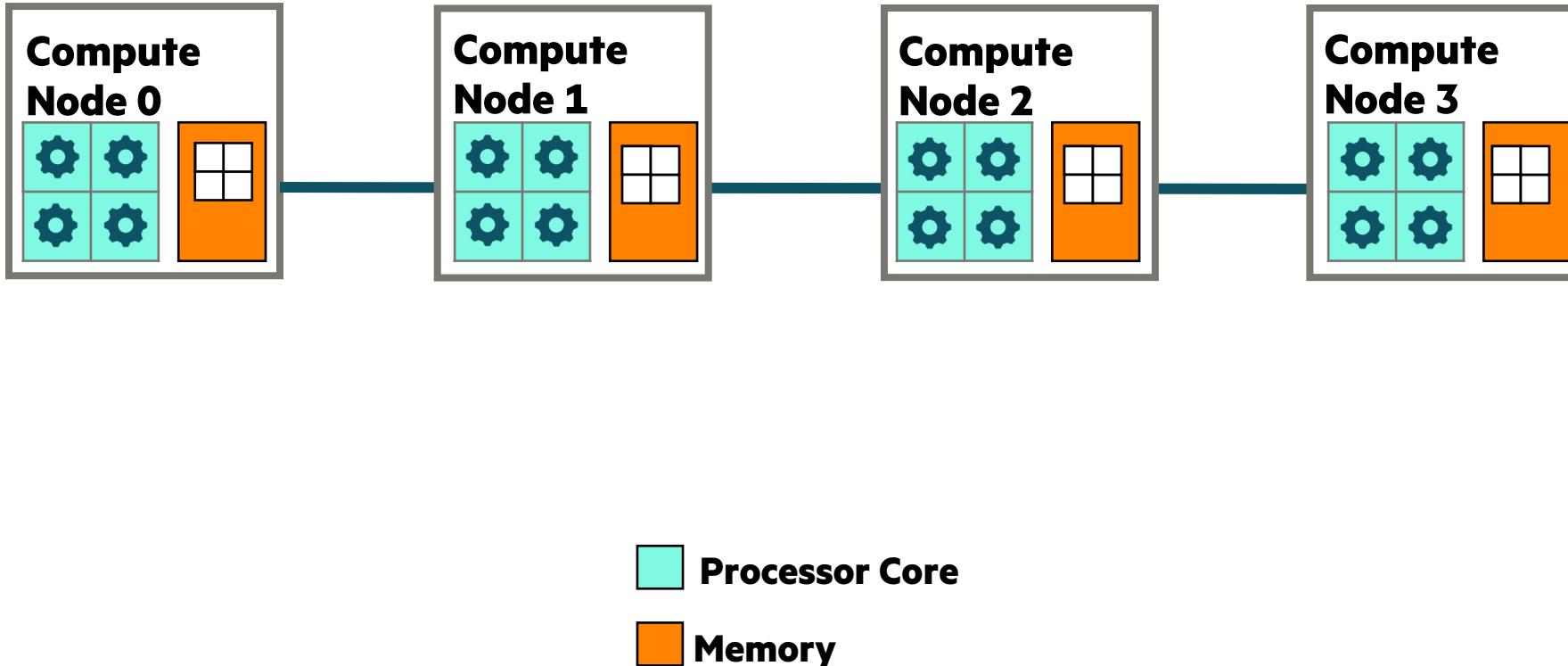
- in order to run a program...
  - faster than we could otherwise
  - and/or using larger problem sizes



# Parallel Computing has become Ubiquitous

## Historical parallel computing:

- supercomputers
- commodity clusters



## Today, we also have:

- multicore processors
- GPUs
- cloud computing

# **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: 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**



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

# **STREAM TRIAD: C + MPI + OPENMP**

```
use BlockDist;

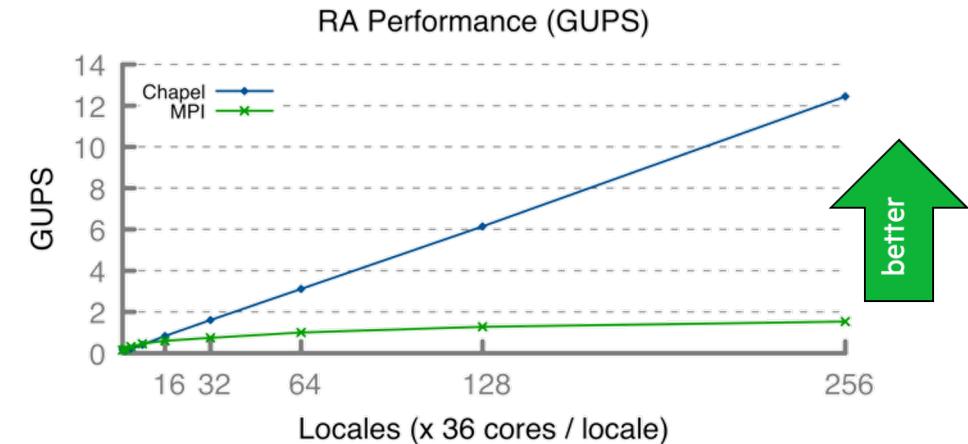
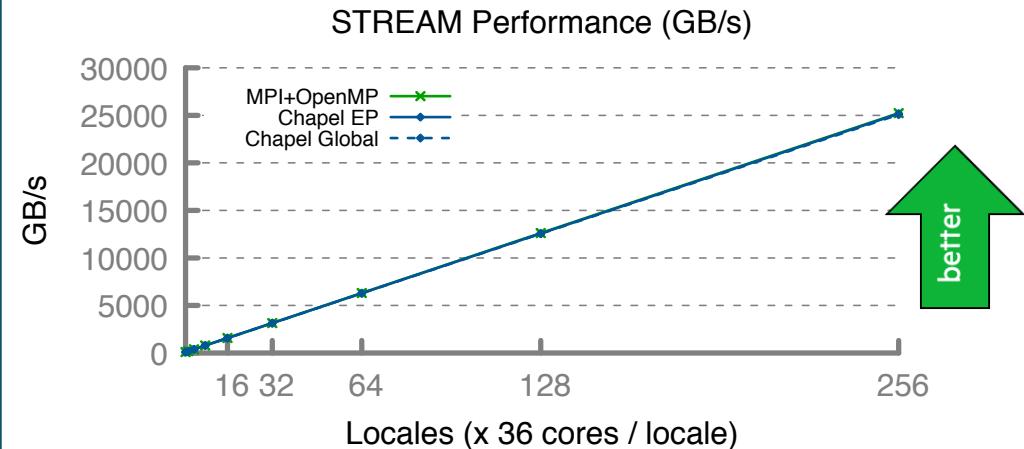
config const n = 1_000_000,
          alpha = 0.01;
const Dom = blockDist.createDomain({1..n});
var A, B, C: [Dom] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```

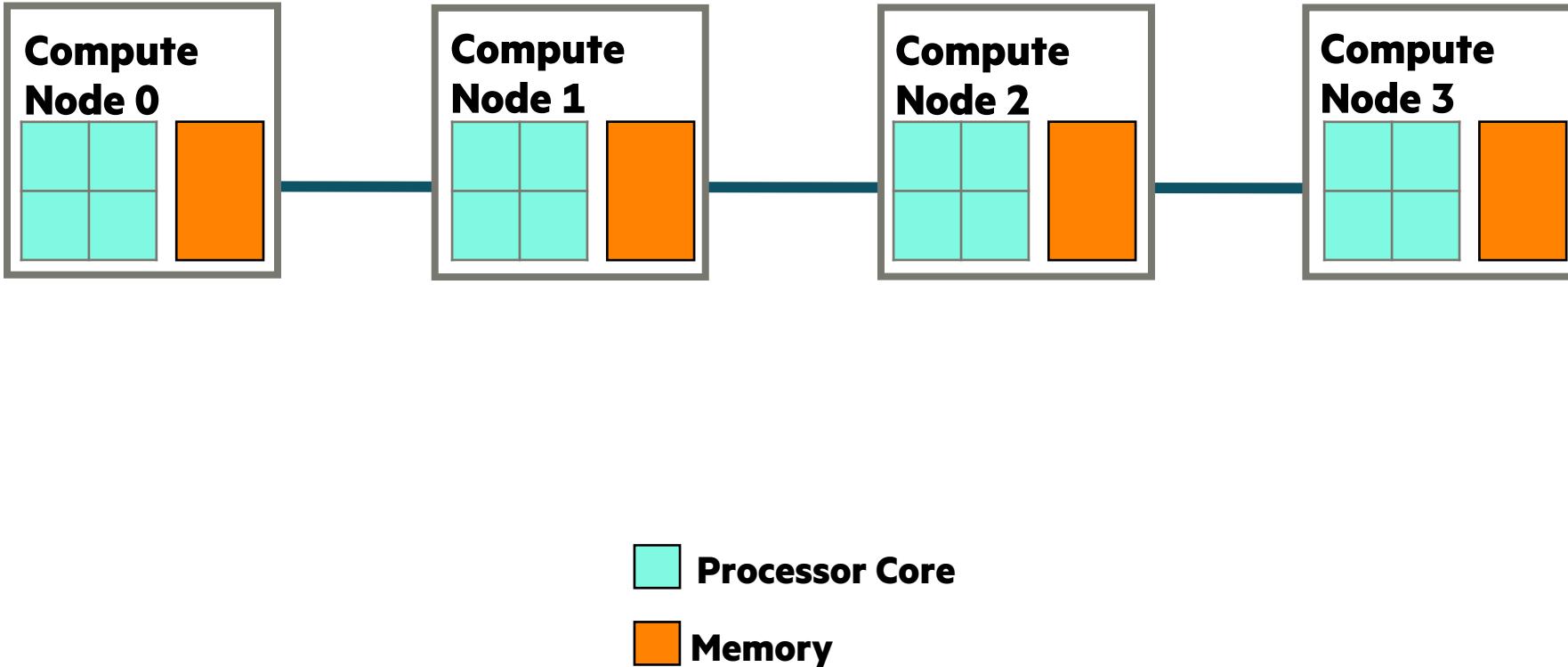
## **HPCC RA: MPI KERNEL**

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



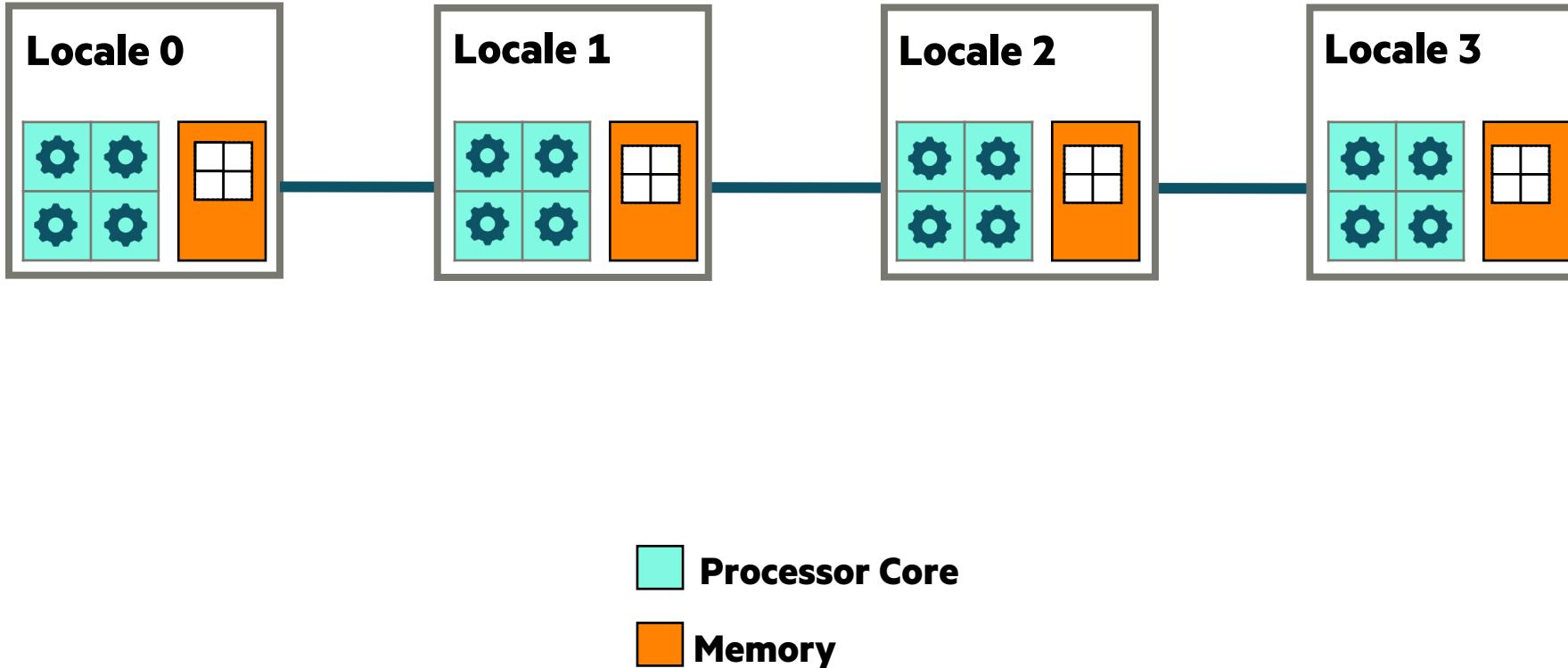
# Locales in Chapel

- In Chapel, a *locale* refers to a compute resource with...
  - processors, so it can run tasks
  - memory, so it can store variables
- For now, think of each compute node as being a locale



# Key Concerns for Scalable Parallel Computing

- parallelism:** What computational tasks should run simultaneously?
- locality:** Where should tasks run? Where should data be allocated?



# Outline

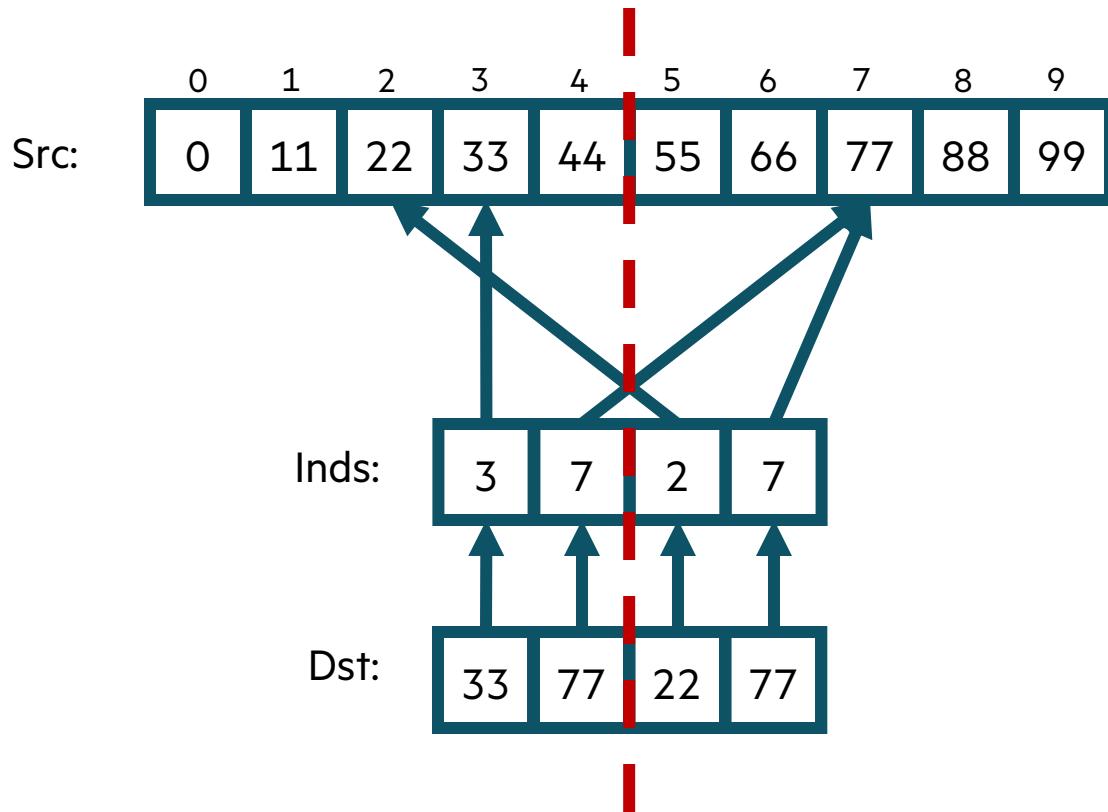
---

- Motivation and Context
- Chapel by Example
- Applications of Chapel
- Chapel on GPUs
- Wrap-up



# **Chapel by Example: Bale Index Gather (IG)**

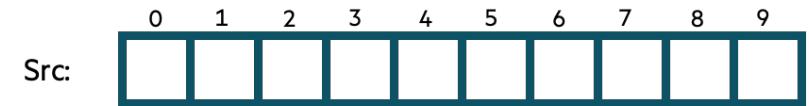
# Bale Index Gather (IG): In Pictures



# Bale IG in Chapel: Array Declarations

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

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

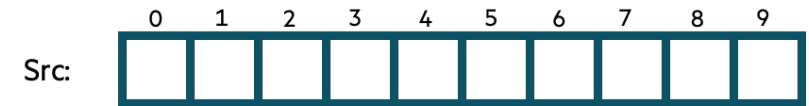


\$

# Bale IG in Chapel: Compiling

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

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

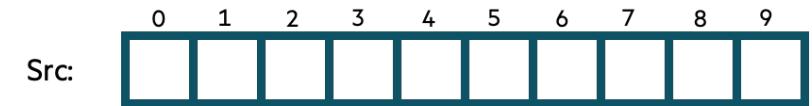


```
$ chpl bale-ig.chpl  
$
```

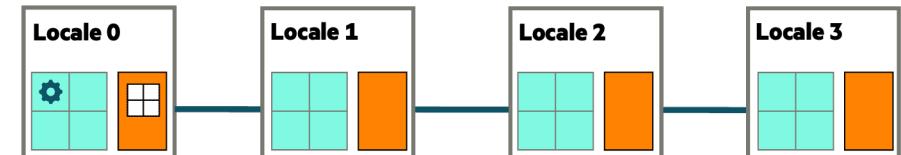
# Bale IG in Chapel: Executing

```
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: Executing, Overriding Configs

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

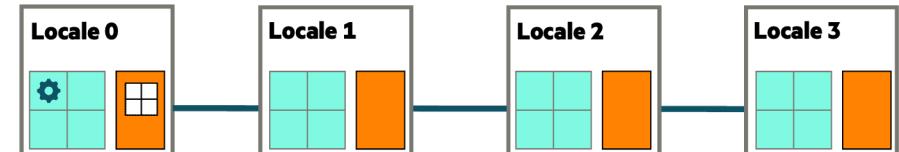
```
var Src: [0..<n] int,  
       Inds, Dst: [0..<m] int;
```

Src: 

Inds: 

Dst: 

```
$ chpl bale-ig.chpl  
$ ./bale-ig --n=1_000_000 --m=1_000_000  
$
```



# Bale IG in Chapel: Array Initialization

```
use Random;

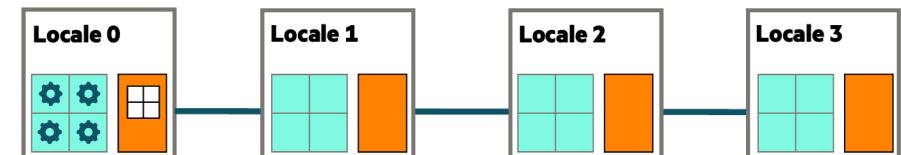
config const n = 10,
      m = 4;

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

Src = [i in 0..<n] i*11;
fillRandom(Inds, min=0, max=n-1);
```

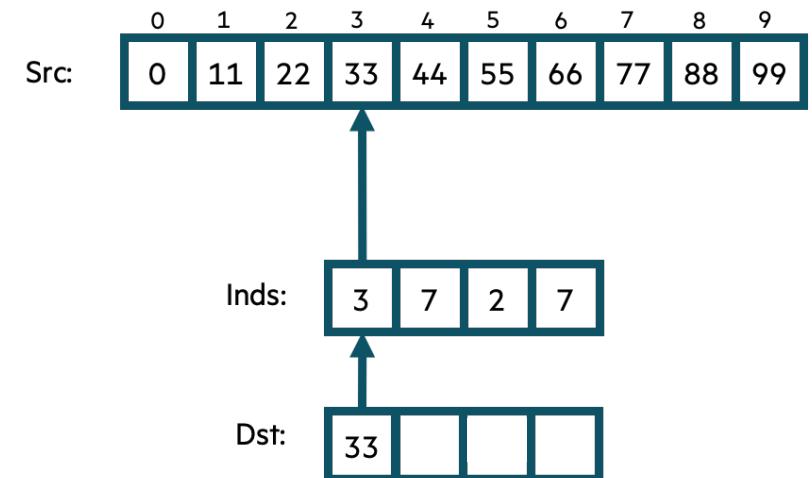


```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

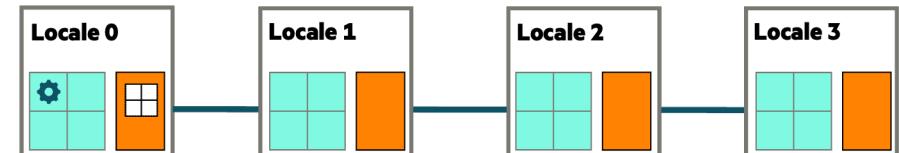


# Bale IG in Chapel: Serial Version

```
config const n = 10,  
      m = 4;  
  
var Src: [0..<n] int,  
    Inds, Dst: [0..<m] int;  
...  
for i in 0..<m do  
  Dst[i] = Src[Inds[i]];
```

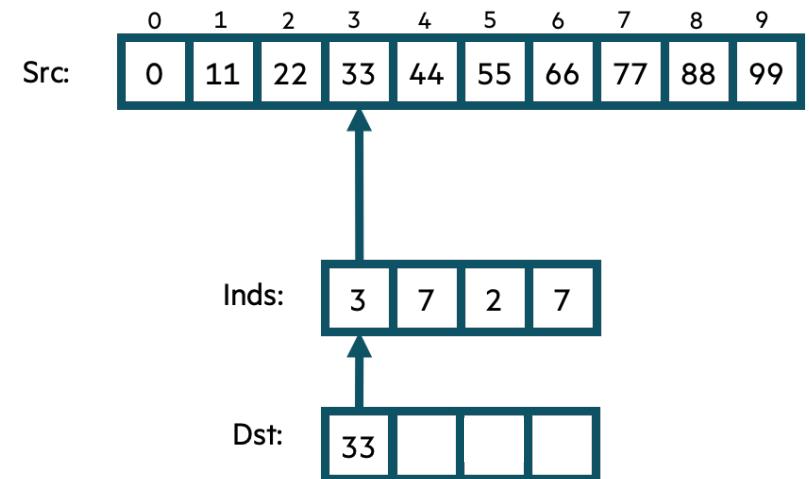


```
$ chpl bale-ig.chpl  
$ ./bale-ig  
$
```

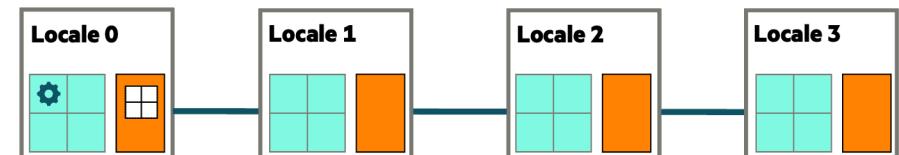


# Bale IG in Chapel: Serial, Zippered Version

```
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];
```

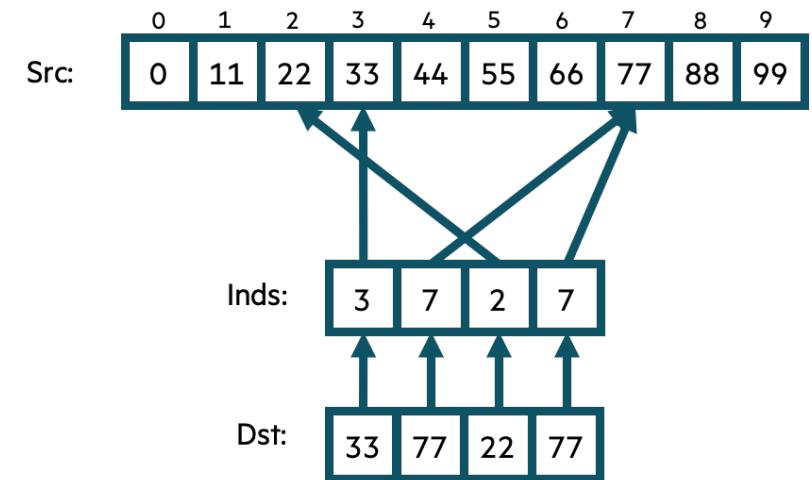


```
$ chpl bale-ig.chpl  
$ ./bale-ig  
$
```

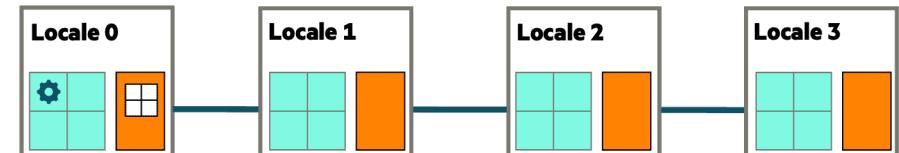


# Bale IG in Chapel: Parallel, Zippered Version (Vectorized)

```
config const n = 10,  
      m = 4;  
  
var Src: [0..<n] int,  
    Inds, Dst: [0..<m] int;  
...  
foreach (d, i) in zip(Dst, Inds) do  
    d = Src[i];
```

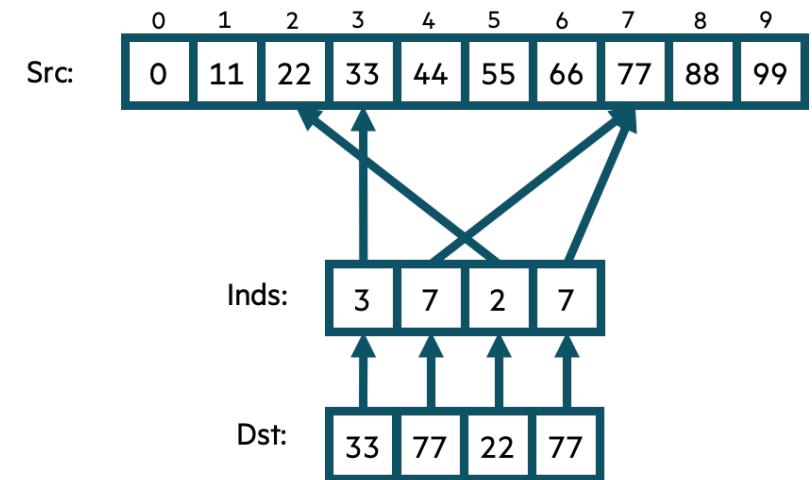


```
$ chpl bale-ig.chpl  
$ ./bale-ig  
$
```

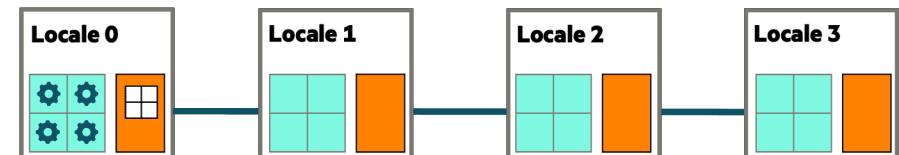


# Bale IG in Chapel: Parallel, Zippered Version (Multicore)

```
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];
```

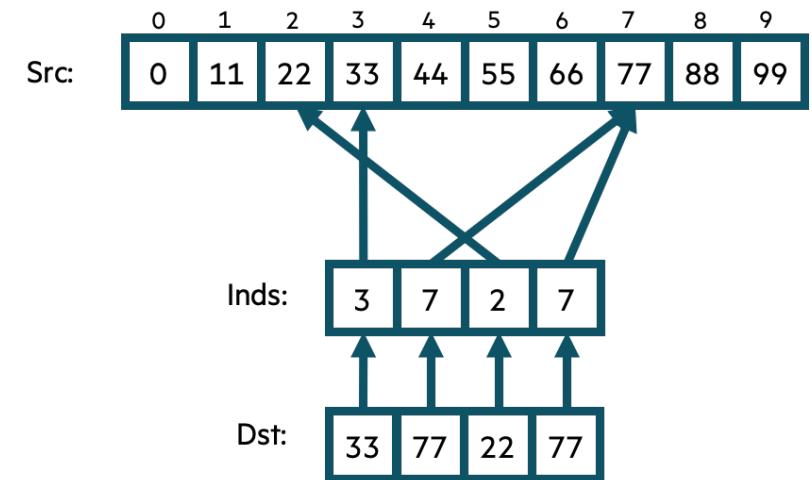


```
$ chpl bale-ig.chpl  
$ ./bale-ig  
$
```

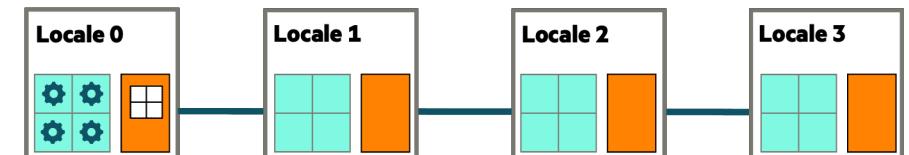


# Bale IG in Chapel: Parallel Promoted Version (equivalent to previous version)

```
config const n = 10,  
      m = 4;  
  
var Src: [0..<n] int,  
    Inds, Dst: [0..<m] int;  
...  
Dst = Src[Inds];
```

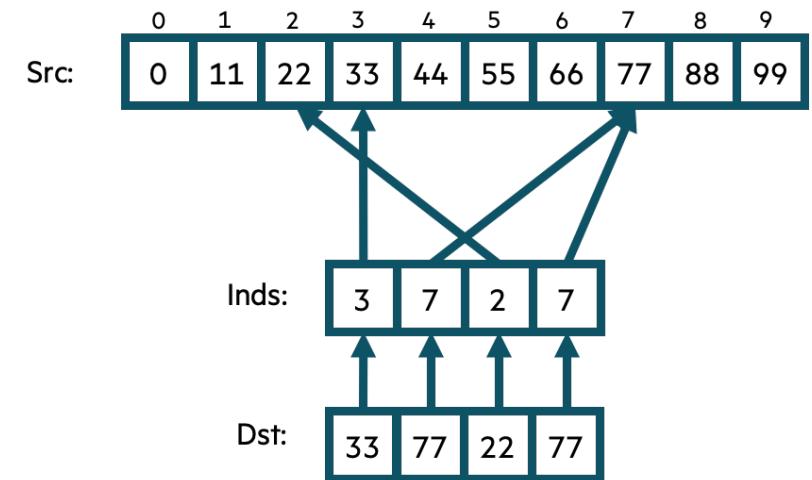


```
$ chpl bale-ig.chpl  
$ ./bale-ig  
$
```

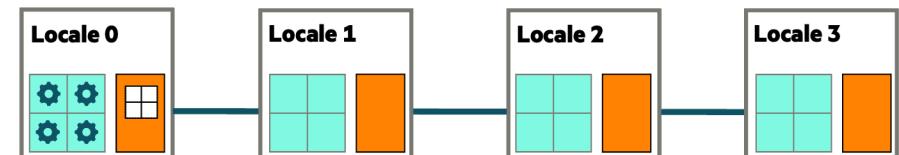


# Bale IG in Chapel: Parallel, Zippered Version (Multicore)

```
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];
```

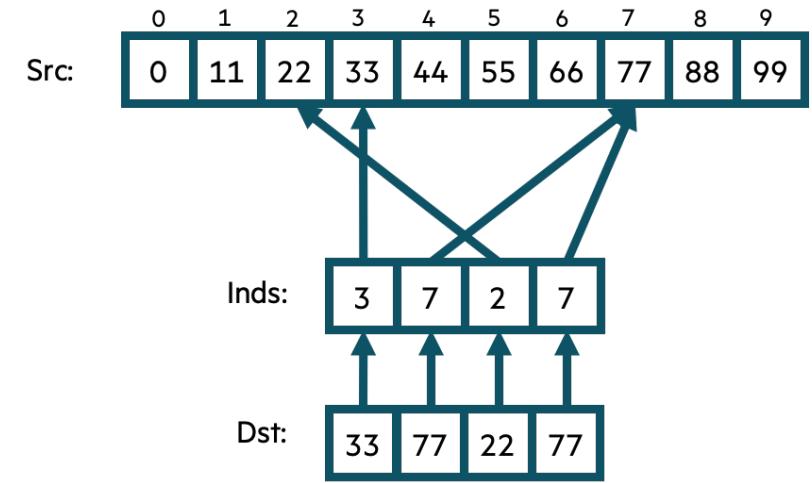


```
$ chpl bale-ig.chpl  
$ ./bale-ig  
$
```

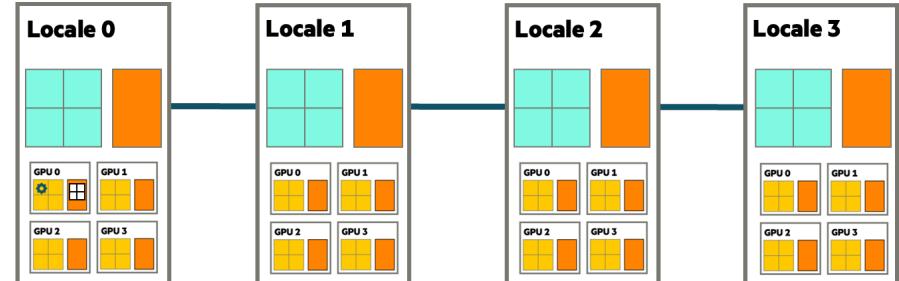


# Bale IG in Chapel: Parallel, Zippered Version for a GPU

```
config const n = 10,  
      m = 4;  
  
on here.gpus[0] {  
    var Src: [0..<n] int,  
        Inds, Dst: [0..<m] int;  
    ...  
    forall (d, i) in zip(Dst, Inds) do  
        d = Src[i];  
}
```

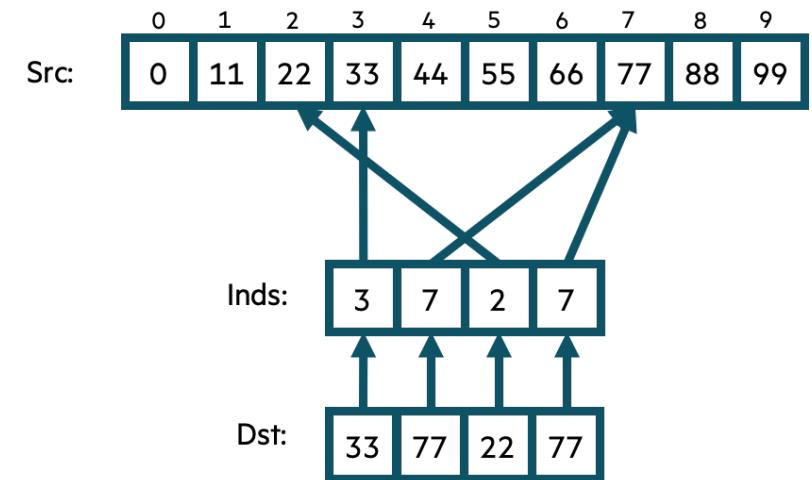


```
$ chpl bale-ig.chpl  
$ ./bale-ig  
$
```

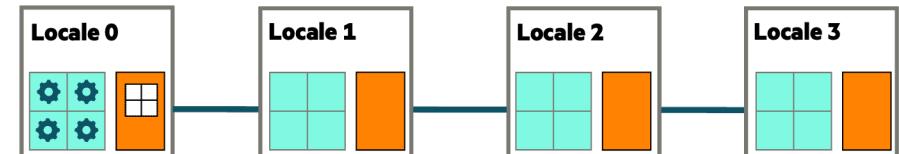


# Bale IG in Chapel: Parallel, Zippered Version (Multicore)

```
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];
```

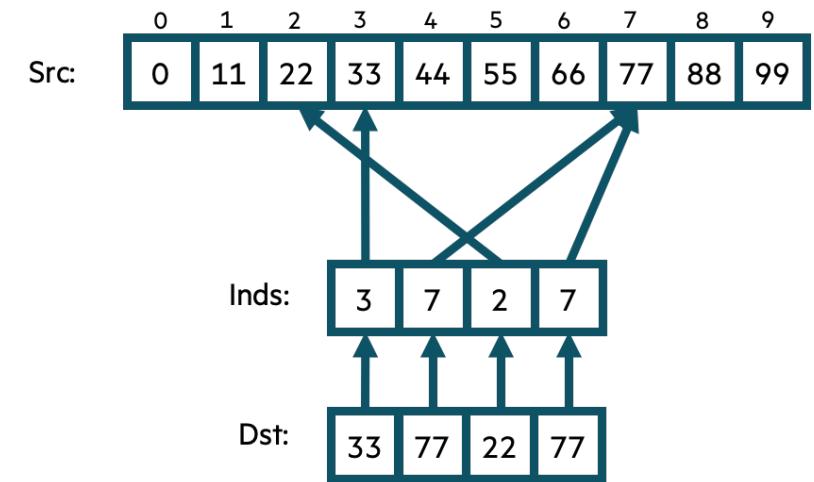


```
$ chpl bale-ig.chpl  
$ ./bale-ig  
$
```

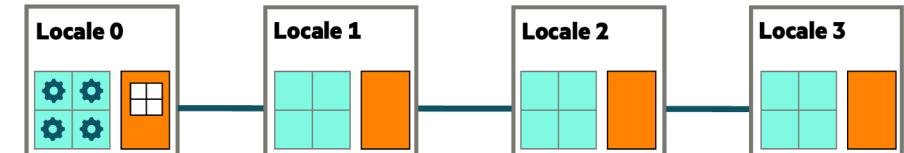


# Bale IG in Chapel: Parallel , Zippered Version with Named Domains (Multicore)

```
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  
$
```



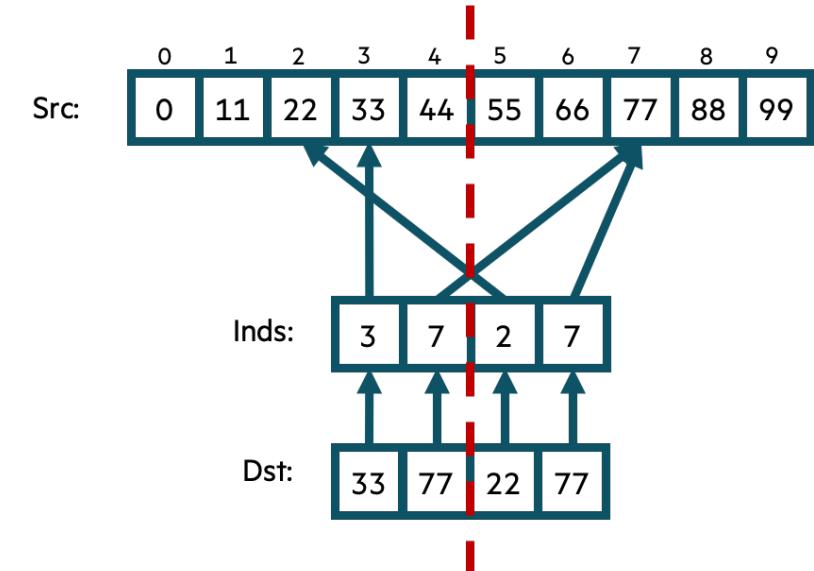
# Bale IG in Chapel: Distributed Parallel Version

```
use BlockDist;

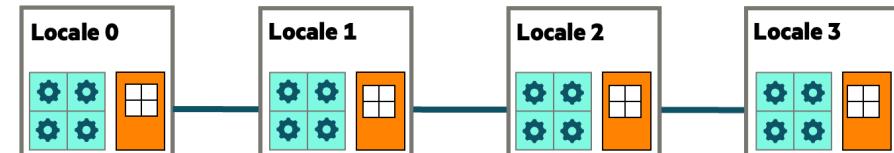
config const n = 10,
      m = 4;

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

```
use BlockDist;

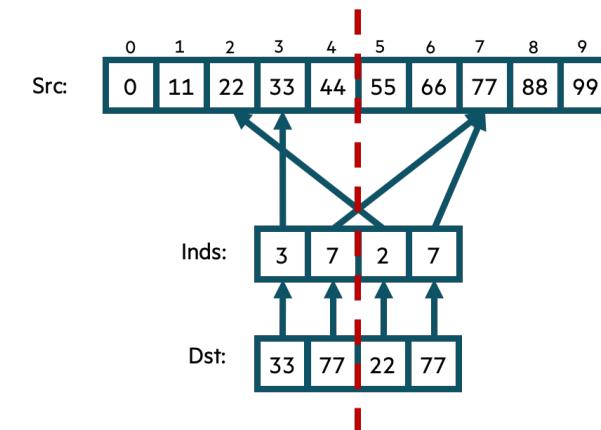
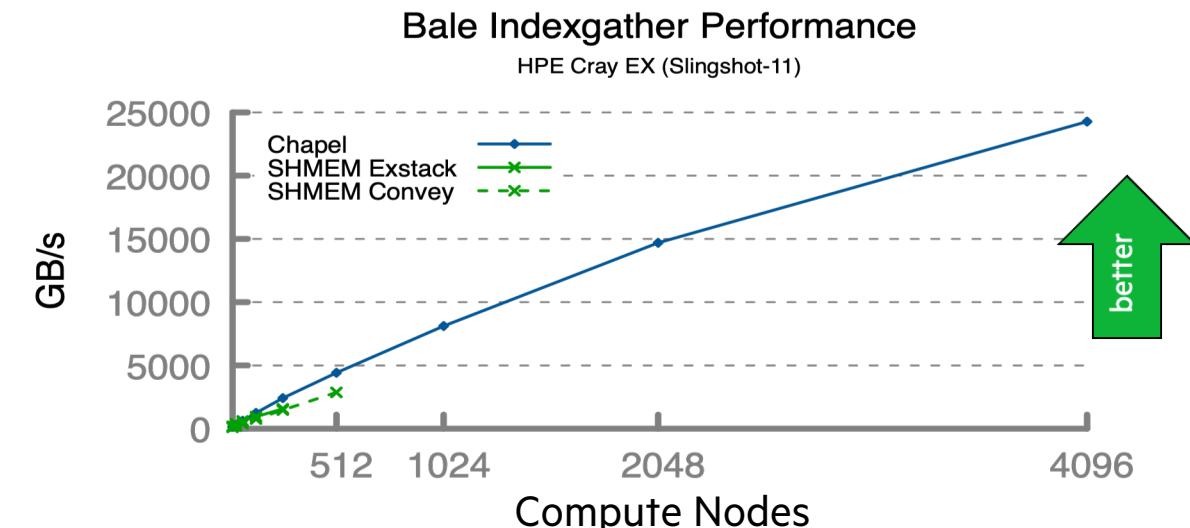
config const n = 10,
      m = 4;

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)

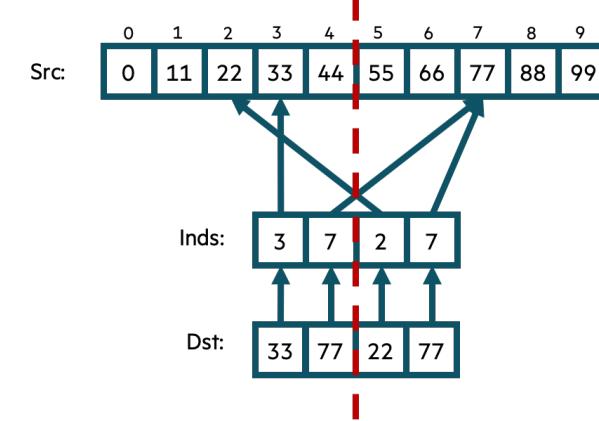
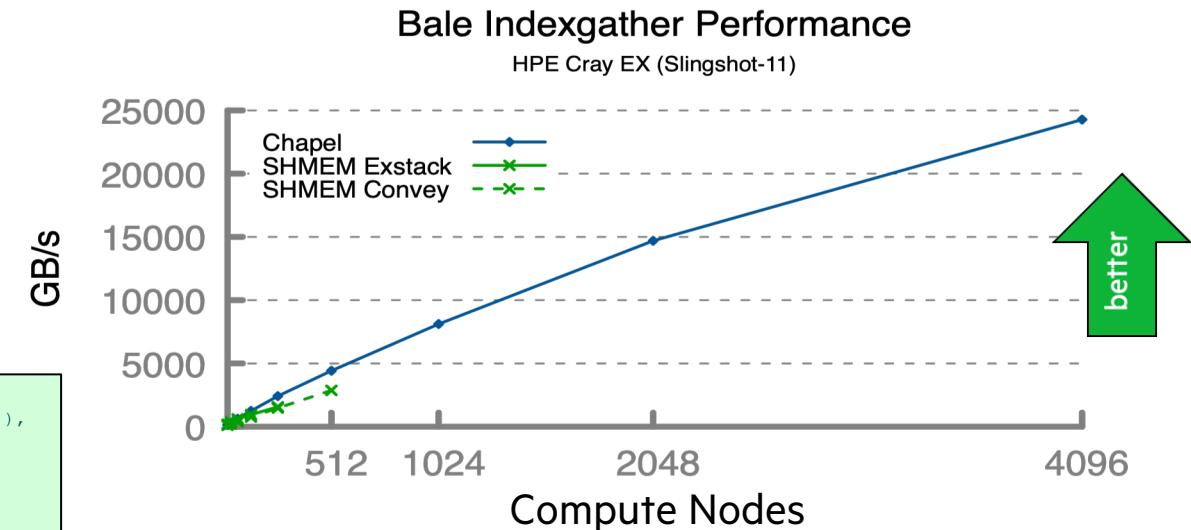
```
forall (d, i) in zip(Dst, Inds) do  
    d = Src[i];
```

## SHMEM (Exstack version)

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

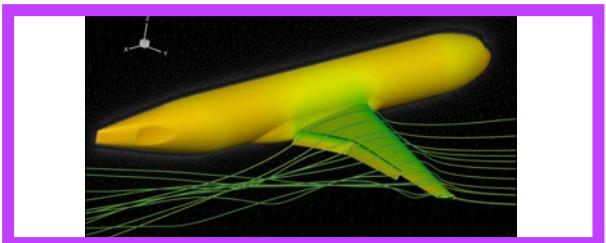
## SHMEM (Conveyors version)

```
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;  
}
```



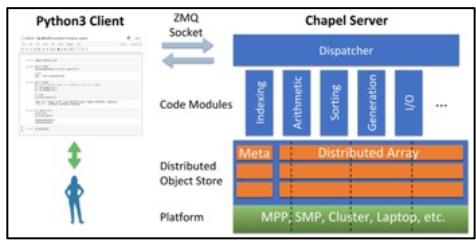
# **Applications of Chapel**

# Applications of Chapel



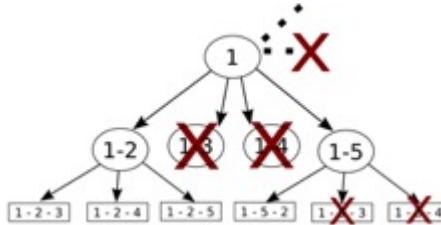
**CHAMPS: 3D Unstructured CFD**

Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.  
École Polytechnique Montréal



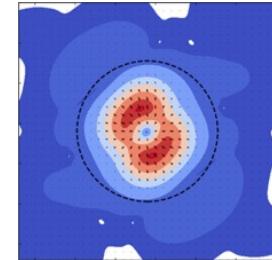
**Arkouda: Interactive Data Science at Massive Scale**

Mike Merrill, Bill Reus, et al.  
U.S. DoD

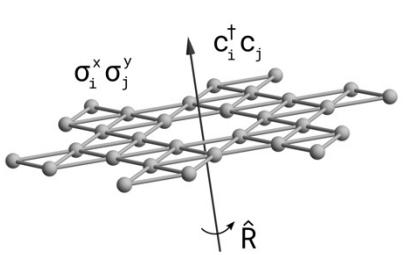


**ChOp: Chapel-based Optimization**

T. Carneiro, G. Helbecque, N. Melab, et al.  
INRIA, IMEC, et al.

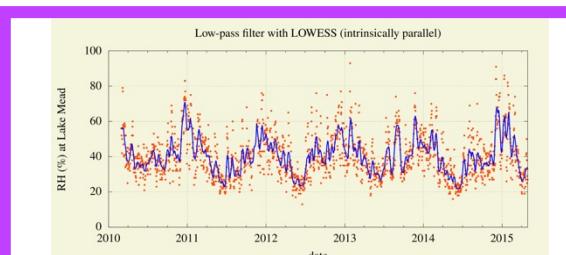


**ChplUltra: Simulating Ultralight Dark Matter**  
Nikhil Padmanabhan, J. Luna Zagorac, et al.  
Yale University et al.



**Lattice-Symmetries: a Quantum Many-Body Toolbox**

Tom Westerhout  
Radboud University



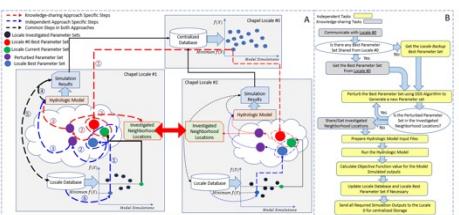
**Desk dot chpl: Utilities for Environmental Eng.**

Nelson Luis Dias  
The Federal University of Paraná, Brazil



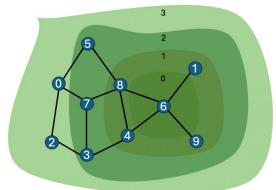
**RapidQ: Mapping Coral Biodiversity**

Rebecca Green, Helen Fox, Scott Bachman, et al.  
The Coral Reef Alliance



**Chapel-based Hydrological Model Calibration**

Marjan Asgari et al.  
University of Guelph



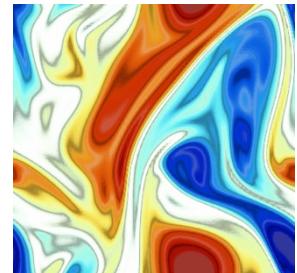
**Arachne Graph Analytics**

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



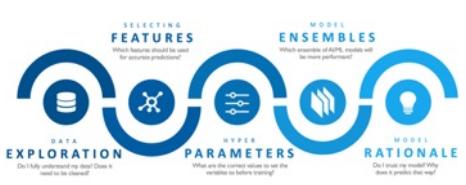
**Modeling Ocean Carbon Dioxide Removal**

Scott Bachman Brandon Neth, et al.  
[C]Worthy



**ChapQG: Layered Quasigeostrophic CFD**

Ian Grooms and Scott Bachman  
University of Colorado, Boulder et al.

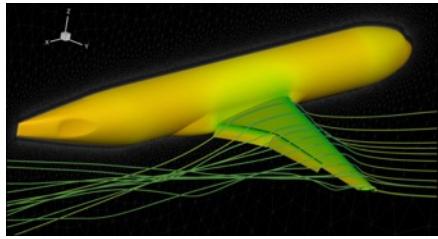


**CrayAI HyperParameter Optimization (HPO)**

Ben Albrecht et al.  
Cray Inc. / HPE

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

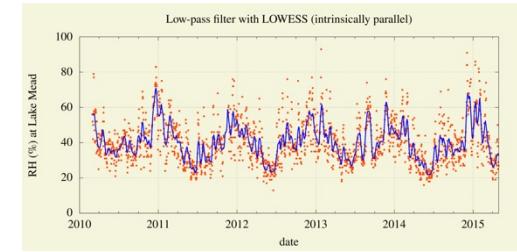
# Productivity Across Diverse Application Scales (code and system size)



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



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



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



## 7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

Posted on September 17, 2024.

Tags: Computational Fluid Dynamics, User Experiences, Interviews  
By: [Engin Kayraklıoglu](#), [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."*



## 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 Kayraklıoglu](#)

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."*



## 7 Questions for Nelson Luís Dias: Atmospheric Turbulence in Chapel

Posted on October 15, 2024.

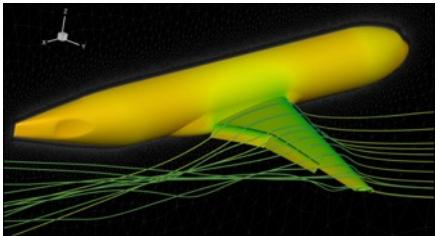
Tags: User Experiences, Interviews, Data Analysis, Computational Fluid Dynamics  
By: [Engin Kayraklıoglu](#), [Brad Chamberlain](#)

In this edition of our [Seven Questions for Chapel Users](#) series, we turn to Dr. Nelson Luis 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."*

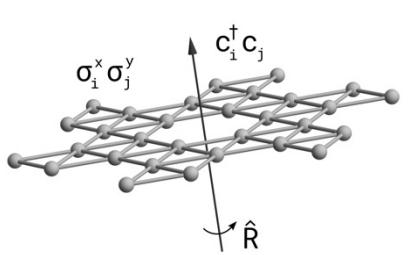
[read this interview series at: <https://chapel-lang.org/blog/series/7-questions-for-chapel-users/>]

# Applications of Chapel



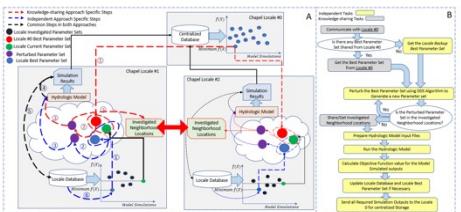
**CHAMPS: 3D Unstructured CFD**

Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.  
École Polytechnique Montréal



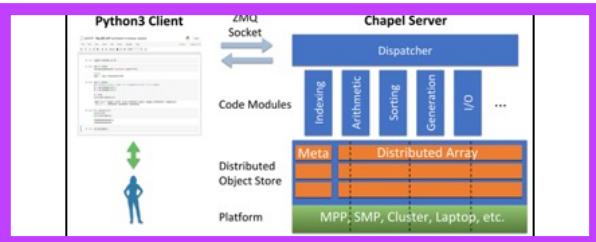
**Lattice-Symmetries: a Quantum Many-Body Toolbox**

Tom Westerhout  
Radboud University



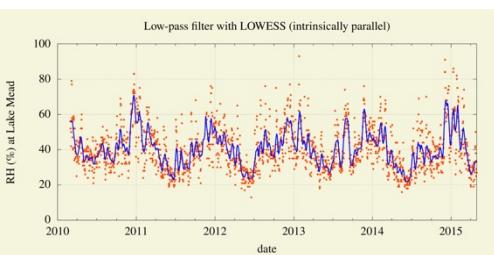
**Chapel-based Hydrological Model Calibration**

Marjan Asgari et al.  
University of Guelph



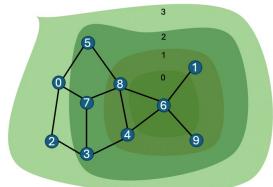
**Arkouda: Interactive Data Science at Massive Scale**

Mike Merrill, Bill Reus, et al.  
U.S. DoD



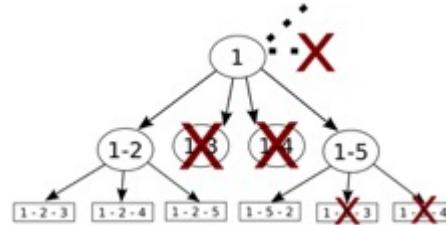
**Desk dot chpl: Utilities for Environmental Eng.**

Nelson Luis Dias  
The Federal University of Paraná, Brazil



**Arachne Graph Analytics**

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



**ChOp: Chapel-based Optimization**

T. Carneiro, G. Helbecque, N. Melab, et al.  
INRIA, IMEC, et al.



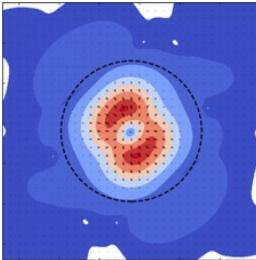
**RapidQ: Mapping Coral Biodiversity**

Rebecca Green, Helen Fox, Scott Bachman, et al.  
The Coral Reef Alliance



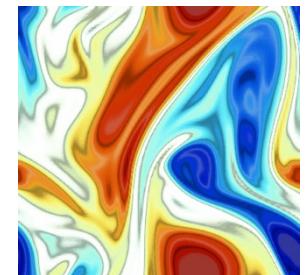
**Modeling Ocean Carbon Dioxide Removal**

Scott Bachman Brandon Neth, et al.  
[C]Worthy



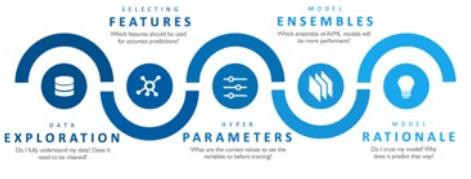
**ChplUltra: Simulating Ultralight Dark Matter**

Nikhil Padmanabhan, J. Luna Zagorac, et al.  
Yale University et al.



**ChapQG: Layered Quasigeostrophic CFD**

Ian Grooms and Scott Bachman  
University of Colorado, Boulder et al.



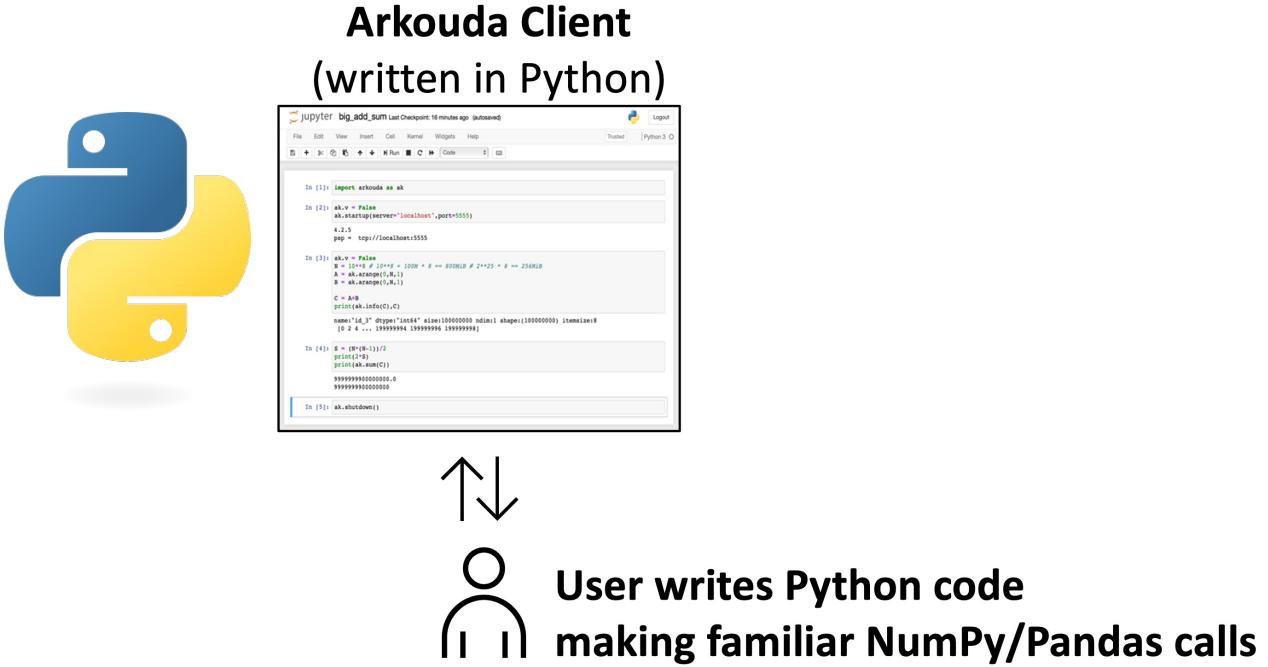
**CrayAI HyperParameter Optimization (HPO)**

Ben Albrecht et al.  
Cray Inc. / HPE

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

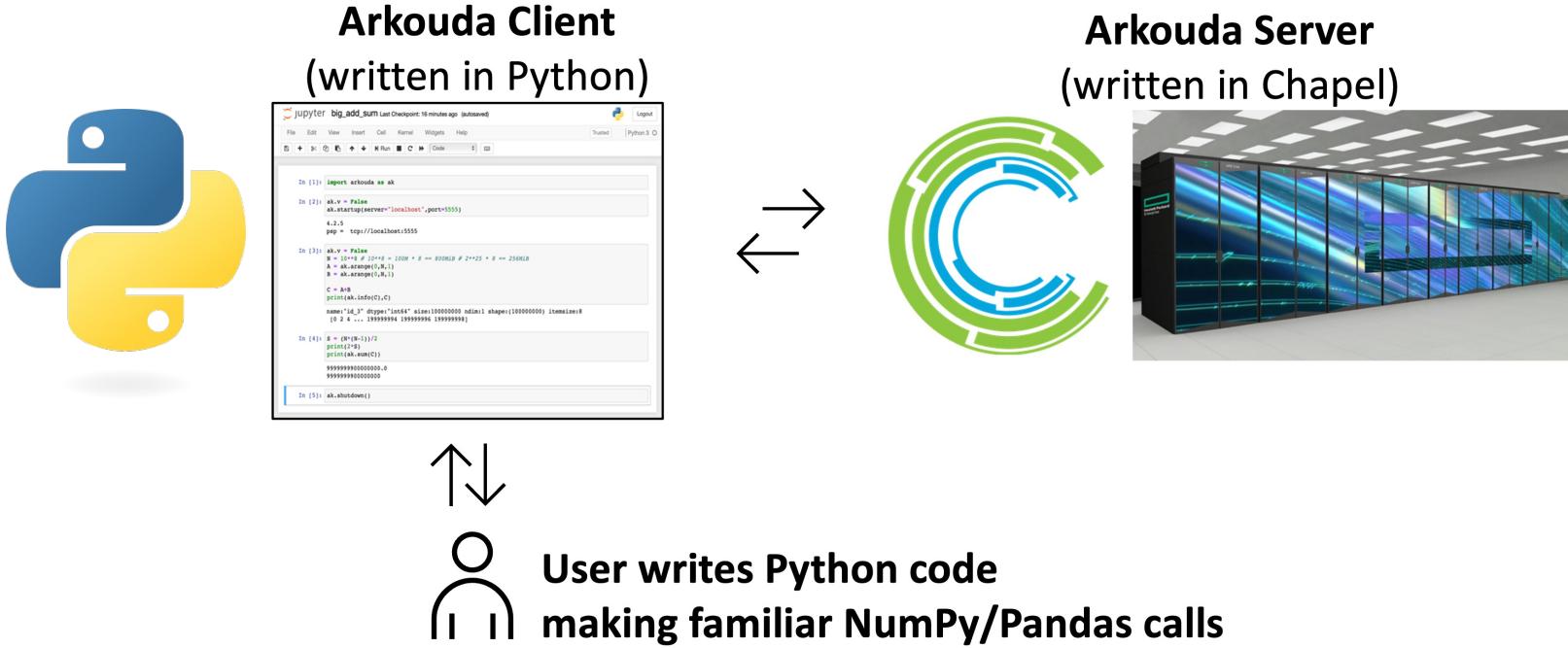
# What is Arkouda?

Q: “What is Arkouda?”



# What is Arkouda?

Q: “What is Arkouda?”



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

# Performance and Productivity: Arkouda Argsort

## HPE Cray EX

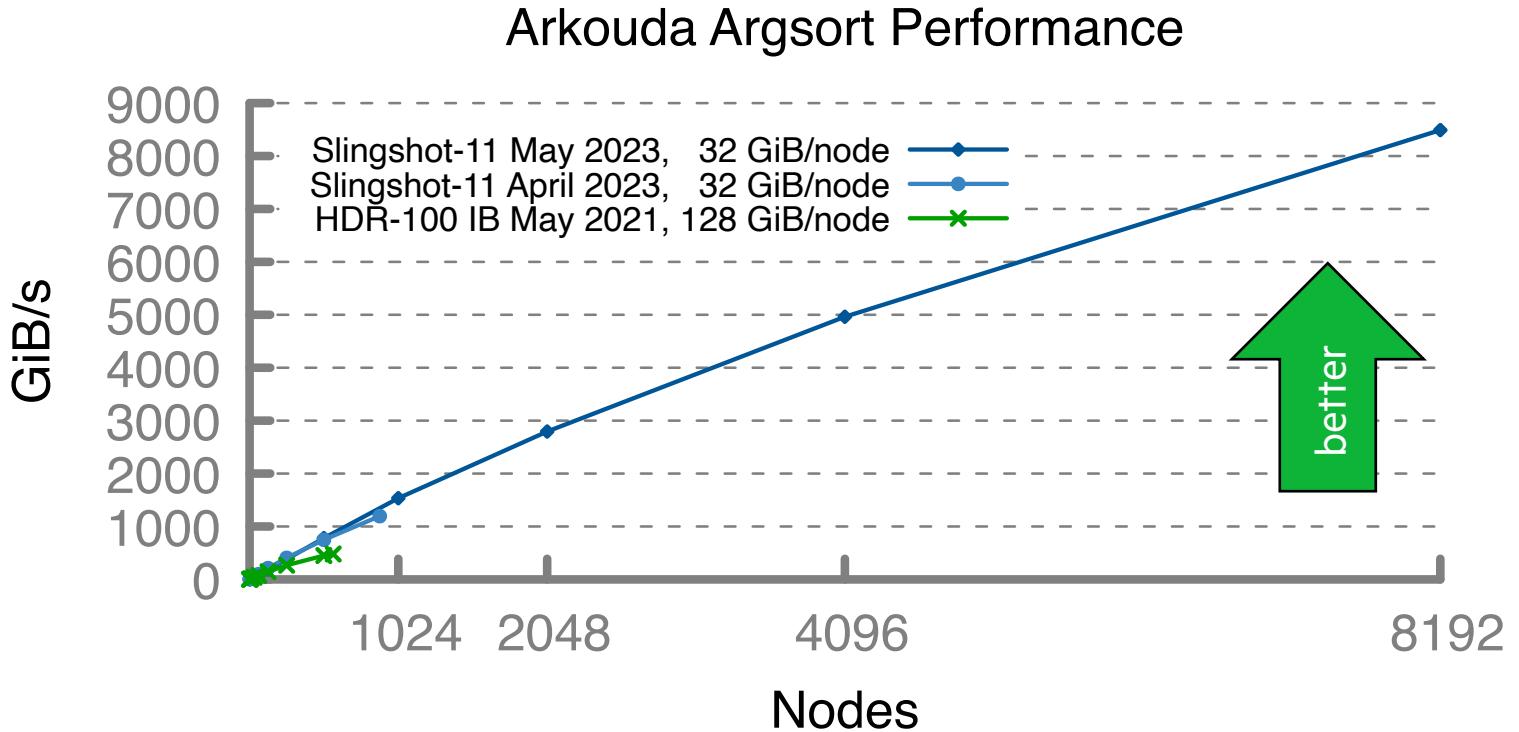
- Slingshot-11 network (200 Gb/s)
- 8192 compute nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)

## 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)

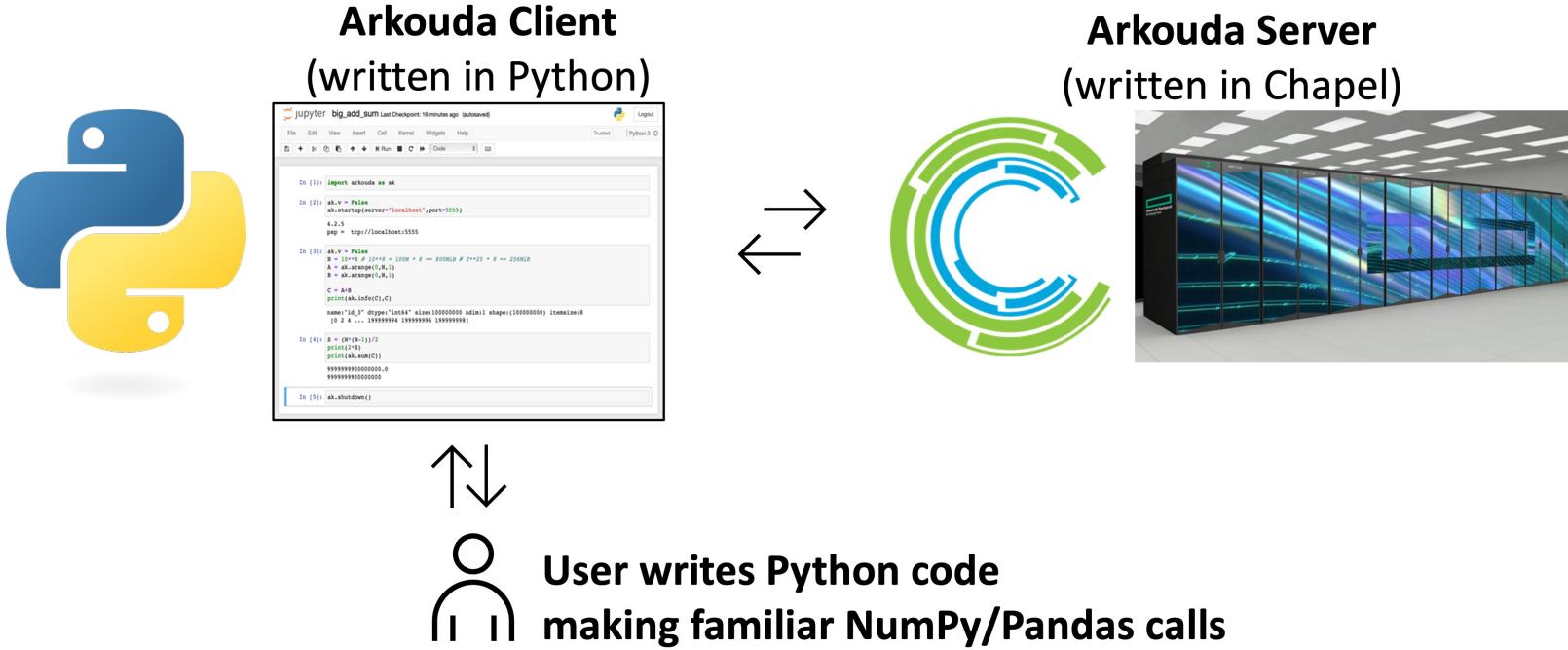


**Implemented using ~100 lines of Chapel**



# What is Arkouda?

Q: “What is Arkouda?”



**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”

# Arkouda Resources

**Website:** <https://arkouda-www.github.io/>

The Arkouda website homepage features a dark header with the Arkouda logo, navigation links for GitHub, documentation, and Gitter, and a search bar. Below the header, a main banner highlights "Massive-scale data science, from the comfort of your laptop". It includes sections for "Arkouda is..." (Fast, Interactive, Extensible), "Powered by Chapel", and "Arkouda users are saying...". A code snippet demonstrates Arkouda's performance on supercomputers compared to NumPy. A "Try it Out" button is at the bottom.

**Arkouda is...**

- Fast**: Arkouda is powered by Chapel, a programming language built from the ground up to support parallelism and distributed computing. Make the most out of every core and every node in your system.
- Interactive**: By distributing your data across multiple nodes, Arkouda allows you to rapidly transform and wrangle datasets in real time that are simply intractable for a laptop or desktop.
- Extensible**: One can expand on Arkouda's capabilities, thus enabling arbitrary scalable computations to be performed from Python.

**Powered by Chapel**

Arkouda's backend is implemented in Chapel, an open-source parallel programming language. Chapel is unique among mainstream languages as it puts parallelism and locality in the forefront, while not sacrificing productivity or portability. Chapel enables Arkouda to perform well and scale on many different architectures, from multicore laptops to cloud systems to world's fastest supercomputers.

To learn more about Chapel, check out its blog, presentations, tutorials and demos, and the How Can I Learn Chapel? page.

**Arkouda users are saying...**

“...solving problems in a matter of seconds, as opposed to days...”  
— Tess Hayes, Bytoa

“[I'm] working with more data than I ever thought possible as a data scientist!”  
— Jake Trockman, Erias

**Arkouda v2024.12.06 released!**

The new release includes a refactored server making it easier to add new features, more Sparse Matrix functionality, new pdarray manipulation functions, and bug fixes.

[Read the release notes →](#)

**GitHub:** <https://github.com/Bears-R-Us/arkouda>

The Arkouda GitHub repository page shows the README and License tabs. The README features a hand-drawn illustration of a bear with the text "arkouda massive scale data science". Below the illustration, the repository title "Arkouda (αρκούδα) 🐻" and subtitle "Interactive Data Analytics at Supercomputing Scale" are displayed. The page also includes sections for Online Documentation, Nightly Arkouda Performance Charts, Gitter channels, and Talks on Arkouda.

**README** | **License**

**Arkouda (αρκούδα) 🐻**  
Interactive Data Analytics at Supercomputing Scale

[CI passing](#) [docs passing](#) [license MIT](#) [code style black](#)

**Online Documentation**

[Arkouda docs at Github Pages](#)

**Nightly Arkouda Performance Charts**

[Arkouda nightly performance charts](#)

**Gitter channels**

[Arkouda Gitter channel](#) [Chapel Gitter channel](#)

**Talks on Arkouda**

[Mike Merrill's SIAM PP-22 Talk](#) [Arkouda Hack-a-thon videos](#)

# 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](#)

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."*

...

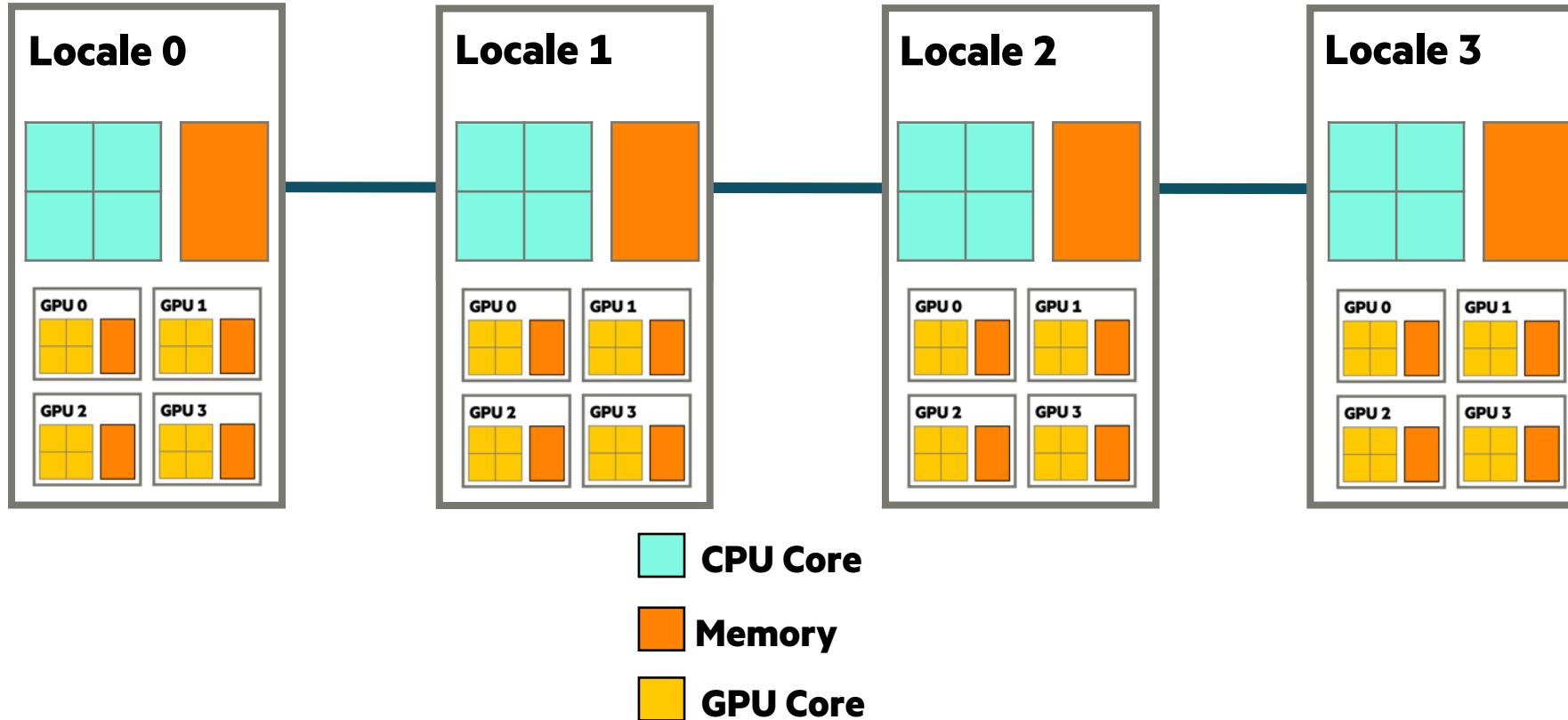
*"Chapel's separation of concerns immediately felt like the most natural way to think about large-scale computing. I would highly encourage anyone wanting to get into HPC programming to start with Chapel."*



**Chapel on GPUs**

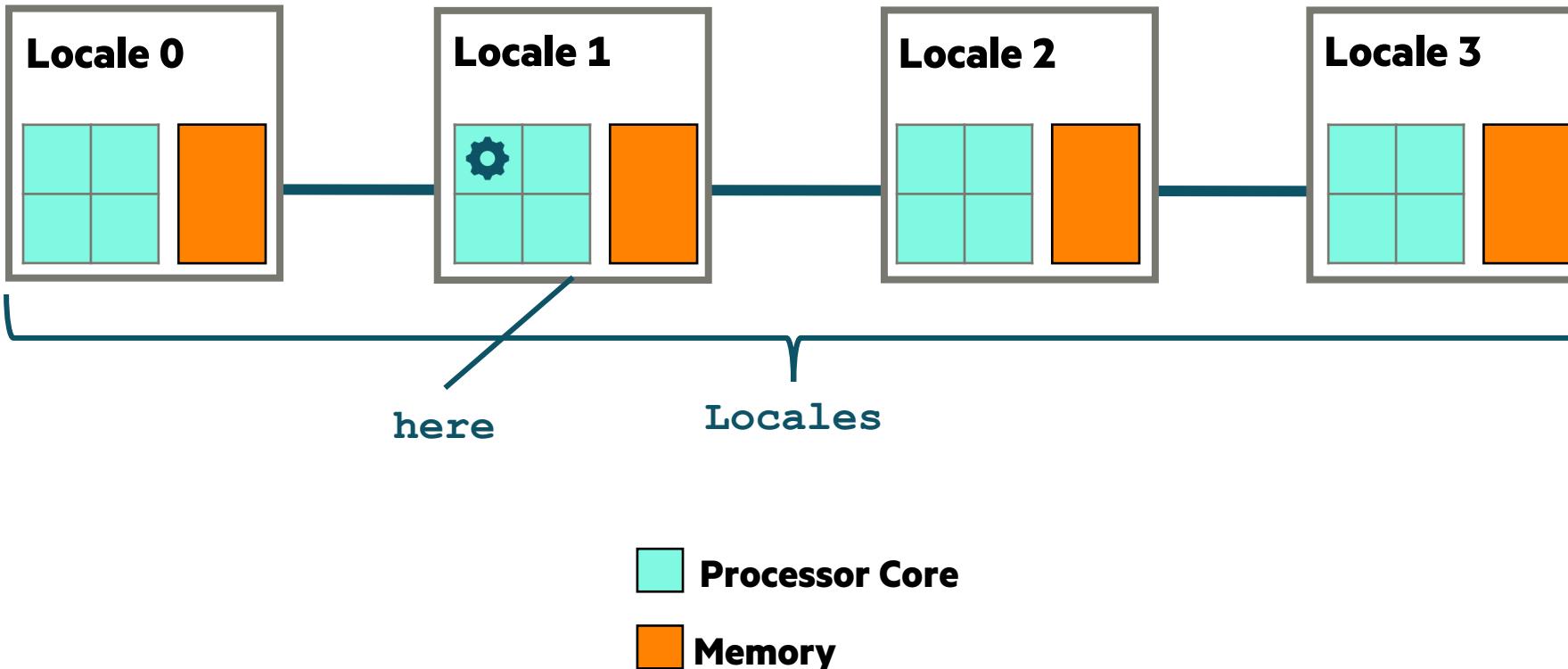
# Compute Nodes with GPUs

- Many modern HPC systems have GPUs in addition to CPUs
  - These GPUs have their own distinct cores and memory



# Built-In Locale Variables in Chapel

- Two key built-in variables for referring to Chapel programs:
  - **Locales**: An array of locale values representing the system resources on which the program is running
  - **here**: The locale on which the current task is executing



# Basic Features for Locality

basics-on.chpl

```
writeln("Hello from locale ", here.id);  
  
var A: [1..2, 1..2] real;  
  
for loc in Locales {  
    on loc {  
        var B = A;  
    }  
}
```

All Chapel programs begin running as a single task on locale 0

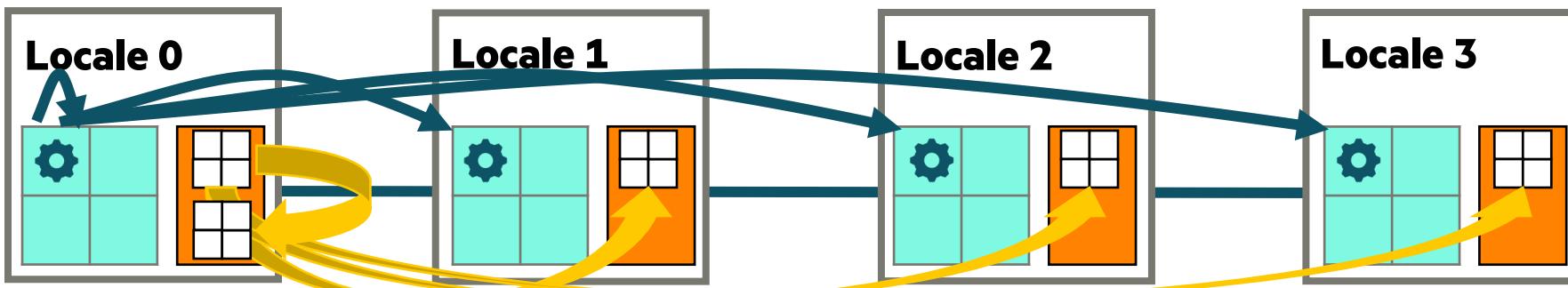
Variables are stored using the memory local to the current task

This loop will serially iterate over the program's locales

on-clauses move tasks to target locales

remote variables can be accessed directly

This is a distributed, yet serial, computation



# Mixing Locality with Task Parallelism

basics-coforall.chpl

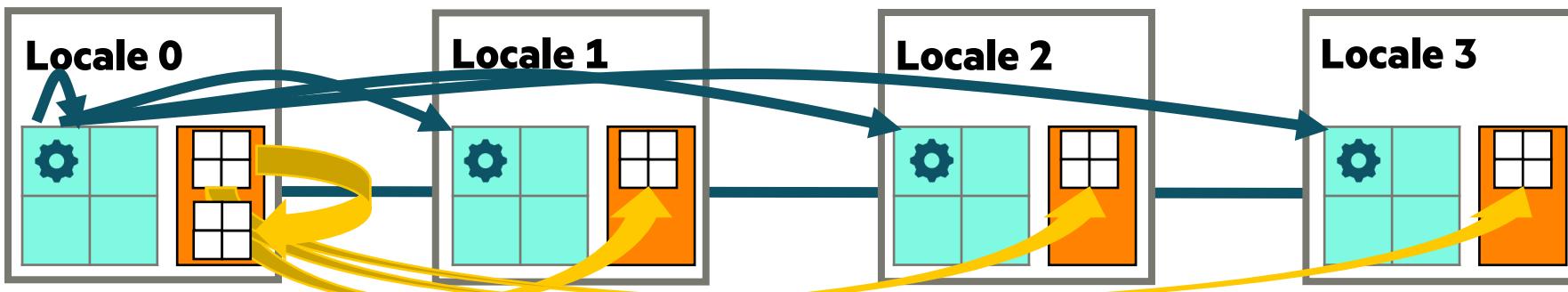
```
writeln("Hello from locale ", here.id);

var A: [1..2, 1..2] real;

coforall loc in Locales { ←
  on loc {
    var B = A;
  }
}
```

The forall loop creates a parallel task per iteration (in this case, a task per locale)

This results in a distributed parallel computation

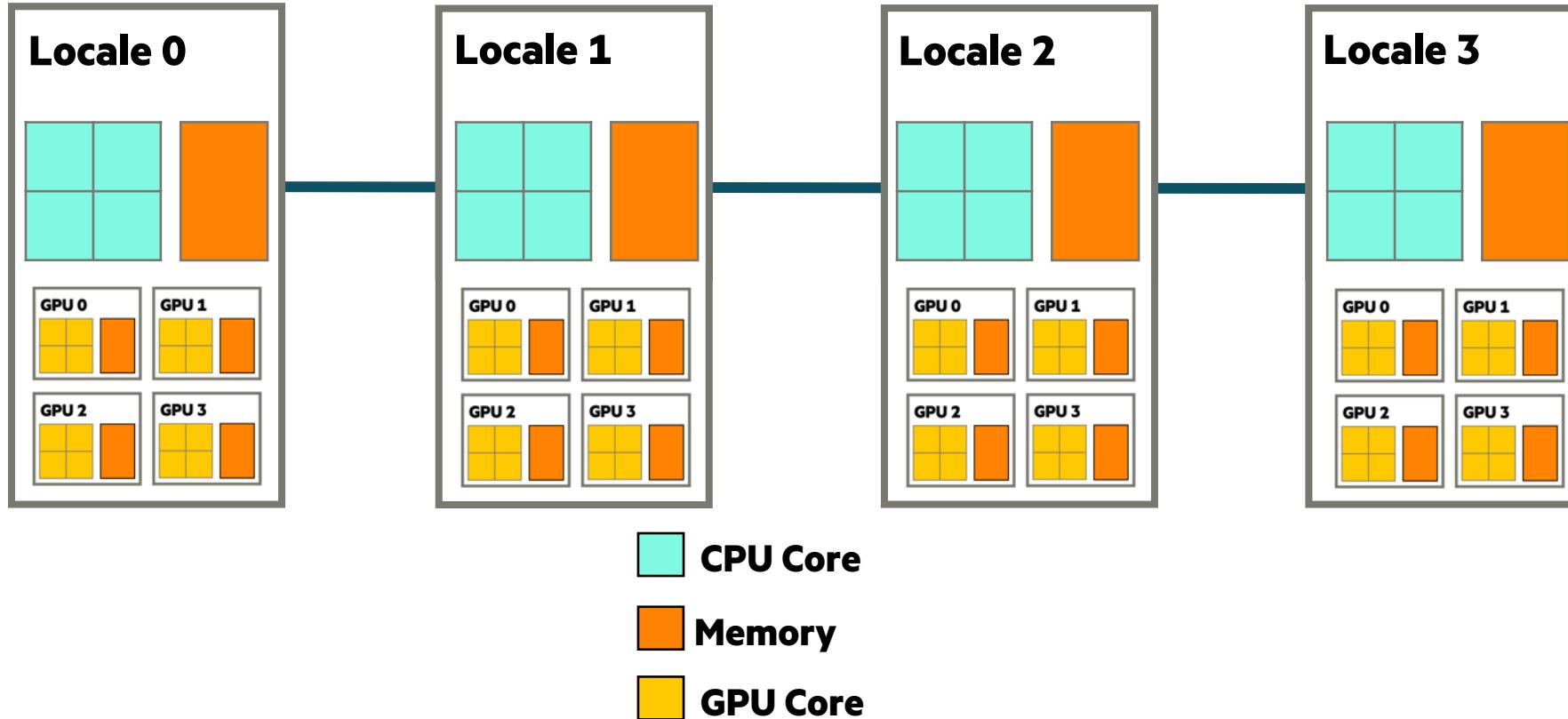


# Representing GPUs in Chapel

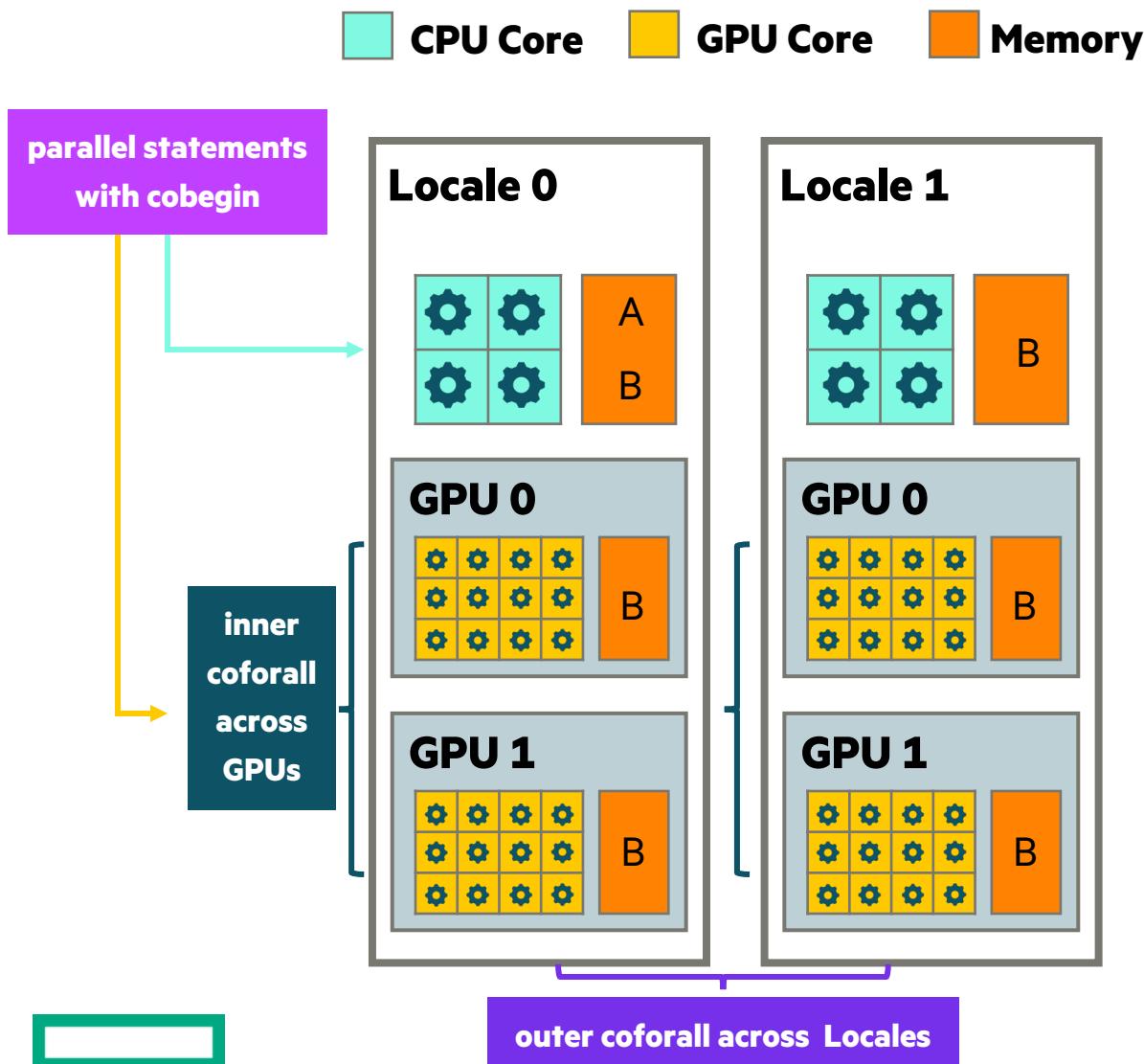
- In Chapel, we represent GPUs as *sub-locales*
  - Each top-level locale may have an array of locales called ‘gpus’
  - We can then target them using Chapel’s traditional features for parallelism + locality

```
on here.gpus[0] { ... }
```

```
coforall gpu in here.gpus do on gpu { ... }
```



# Targeting CPUs and GPUs using Parallelism and Locality



```
var A: [1..n, 1..n] real;
coforall l in Locales do on l {
    cobegin {
        {
            var B: [1..n, 1..n] real;
            B = 2;
            A = B;
        }
    coforall g in here.gpus do on g {
        var B: [1..n, 1..n] real;
        B = 2;
        A = B;
    }
}
writeln(A);
```

# **Parallel Programming and Chapel: Then and Now**

# 20 Years Ago vs. Today

---

## 2005:

- **Popular Languages:** C++, Java, Perl, Python
- **HPC Hardware:**
  - processors: single- or dual-core
  - nodes: single-socket, UMA, single-NIC, no GPUs
  - Top 5 systems from the Top-500:
    - core counts: 4,800–65,536
    - RMax perf: 27.91–136.80 TFlop/s
- **HPC Programming:**
  - languages: Fortran, C, C++, UPC
  - libraries: MPI, SHMEM
  - directives: OpenMP
  - GPUs: N/A
- **Chapel:** no public releases; only serial programs
  - draft language spec defines locality + parallel features

## 2025:

- **Popular Languages:** Python, Rust, Go, Julia, Swift
- **HPC Hardware:**
  - processors: multicore
  - nodes: multi-socket, NUMA, multi-NIC, multi-GPU
  - Top-5 systems from the Top-500:
    - core counts: 2,073,600–11,039,616 (**31x–2,300x**)
    - RMax perf: 477.9–1,742.0 PFlop/s (**3,493x–62,415x**)
- **HPC Programming:**
  - languages: Fortran, C, C++, Python (as glue code)
  - libraries: MPI, SHMEM
  - directives: OpenMP
  - GPUs: CUDA, HIP, SYCL, Kokkos, OpenACC, OpenCL, ...
- **Chapel:** 42 public releases; production apps
  - locality + parallel features essentially the same



# 20 Years Ago vs. Today

## 2005:

- **Popular Languages:** C++, Java, Perl, Python

- **HPC Hardware:**

- processors: single- or dual-core
- nodes: single-socket, UMA, single-NIC, no GPUs
- Top 5 systems from the Top-500:
  - core counts: 4,800–65,536
  - RMax perf: 27.91–136.80 TFlop/s

- **HPC Programming:**

- languages: Fortran, C, C++, UPC
- libraries: MPI, SHMEM
- directives: OpenMP
- GPUs: N/A

- **Chapel:** no public releases; only serial programs

- draft language spec defines locality + parallel features

Chapel has adapted well, due to its focus on parallelism and locality

## 2025:

- **Popular Languages:** Python, Rust, Go, Julia, Swift

- **HPC Hardware:**

- processors: multicore
- nodes: multi-socket, NUMA, multi-NIC, multi-GPU
- Top-5 systems from the Top-500:
  - core counts: 2,073,600–11,039,616 (**31x–2,300x**)
  - RMax perf: 477.9–1,742.0 PFlop/s (**3,493x–62,415x**)

- **HPC Programming:**

- languages: Fortran, C, C++, Python (as glue code)
- libraries: MPI, SHMEM
- directives: OpenMP
- GPUs: CUDA, HIP, SYCL, Kokkos, OpenACC, OpenCL, ...

- **Chapel:** 42 public releases; production apps

- locality + parallel features essentially the same

# **Wrap-up**

# Summary

## Chapel is unique among programming languages

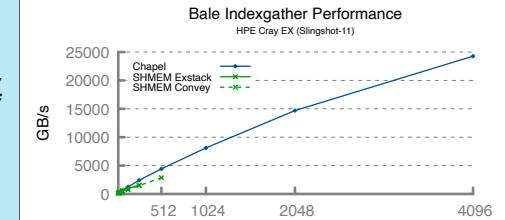
- features first-class concepts for parallelism and locality
- ports and scales from laptops to supercomputers
- supports clean, concise code relative to conventional approaches
- supports GPUs in a vendor-neutral manner

```
use BlockDist;

config const n = 10,
      m = 4;

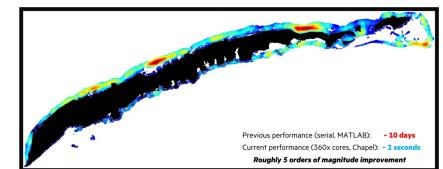
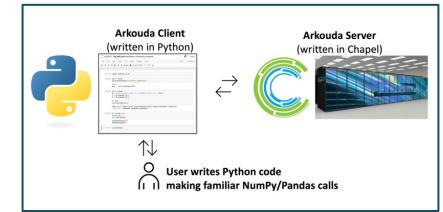
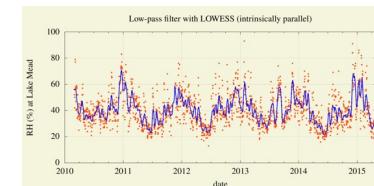
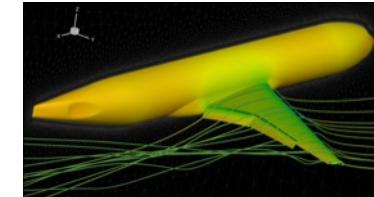
const SrcInds = blockDist.createDomain(0..<n>,
                                       DstInds = blockDist.createDomain(0..<m>);

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

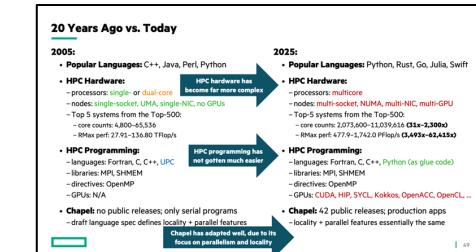


## Chapel is being used for productive parallel computing at all scales

- 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



## Chapel's general features for parallelism and locality have permitted it to adapt to the evolving parallel hardware landscape



# The Advanced Programming Team at HPE



# Ways to Engage with the Chapel Community

## “Live” (Virtual) Community Events

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

## Community / User Forums

- [Discord](#)
- [Discourse](#)  
chapel+qs@discoursemail.com
- Email Contact Alias
- [GitHub Issues](#)
- [Gitter](#)
- [Reddit](#)
- [Stack Overflow](#)



chapel+qs@discoursemail.com



GITTER



stackoverflow

## Electronic Broadcasts

- [Chapel Blog](#), 1–4 articles per month
- [Community Newsletter](#), quarterly
- [Announcement Emails](#), around big events

## Social Media

- [Bluesky](#)
- [Facebook](#)
- [LinkedIn](#)
- [Mastodon](#)
- [X / Twitter](#)
- [YouTube](#)



# Chapel Website

DOWNLOAD DOCS LEARN RESOURCES COMMUNITY BLOG

## The Chapel Programming Language

Productive parallel computing at every scale.

- Hello World
- Distributed Hello World
- Parallel File IO
- 1D Heat Diffusion
- GPU Kernel

[TRY CHAPEL](#) [GET CHAPEL](#) [LEARN CHAPEL](#)

**PRODUCTIVE**  
Concise and readable without compromising speed or expressive power. Consistent concepts for parallel computing make it easier to learn.

**PARALLEL**  
Built from the ground up to implement parallel algorithms at your desired level of abstraction. No need to trade low-level control for convenience.

**FAST**  
Chapel is a compiled language generating efficient machine code that matches or beats the performance of C/C++ languages.

**SCALABLE**  
Chapel enables application performance at any scale, from laptops to clusters, the cloud, and the largest supercomputers in the world.

**GPU-ENABLED**  
Chapel supports vendor-neutral GPU programming with the same language features used for distributed execution. No boilerplate. No cryptic APIs.

**OPEN**  
Entirely open-source using the MIT license. Built by a great community of developers. Join us!

### CHAMPS

World-class multiphysics simulation

Written by students and postdocs in Eric Laurendeau's lab at Polytechnique Montreal. Outperformed its C/OpenMP predecessor using far fewer lines of code. Dramatically accelerated the progress of grad students while also supporting contributions from undergrads for the first time.

[Learn More](#)

### CHAPEL IN PRODUCTION

• • • • •

### USERS LOVE IT

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

- Éric Laurendeau, Professor, Polytechnique Montréal

"A lot of the nitty gritty is hidden from you until you need to know it. ... I like the complexity grows as you get more comfortable – rather than having to learn everything at once."

- Tess Hayes, Chapel developer

### ChapelCon '25 CFP Released!

on June 26, 2025  
ChapelCon '25 is coming this fall. Check out the webpage and the newly released CFP today.

[CONTINUE READING](#)

### 10 Myths About Scalable Parallel Programming Languages (Redux), Part 3: New Languages vs. Language Extensions

By Brad Chamberlain on June 25, 2025  
A third archival post from the 2012 IEEE TCSC blog series with a current reflection on it.

[CONTINUE READING](#)

### v2.5

Highlights from the June 2025 release of Chapel 2.5

[CONTINUE READING](#)

### Paper and Presentation Refresh

on June 10, 2025  
We've just completed a long-overdue refresh of Chapel-related papers and presentations from the past year or so.

[CONTINUE READING](#)

### Public Weekly Deep-Dive / Demo Meeting Launched

on May 20, 2025  
In addition to our regular demos

[CONTINUE READING](#)

**FOLLOW US**

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

**GET IN TOUCH**

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

**GET STARTED**

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

chapel-lang.org

# Summary

## Chapel is unique among programming languages

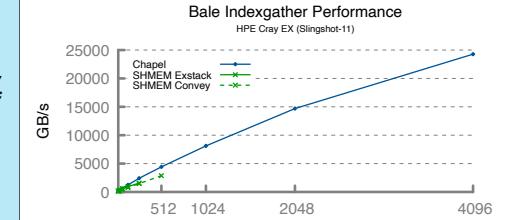
- features first-class concepts for parallelism and locality
- ports and scales from laptops to supercomputers
- supports clean, concise code relative to conventional approaches
- supports GPUs in a vendor-neutral manner

```
use BlockDist;

config const n = 10,
      m = 4;

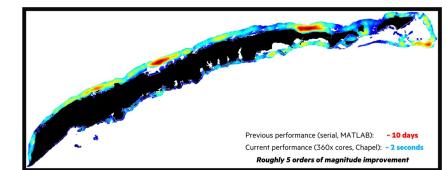
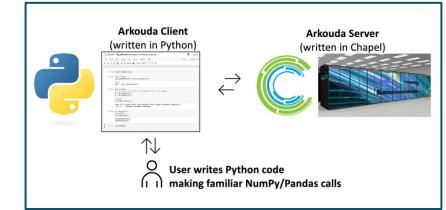
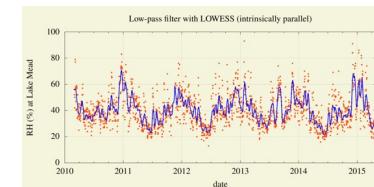
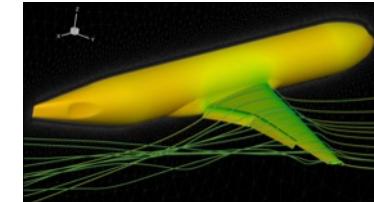
const SrcInds = blockDist.createDomain(0..<n>,
                                       DstInds = blockDist.createDomain(0..<m>);

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

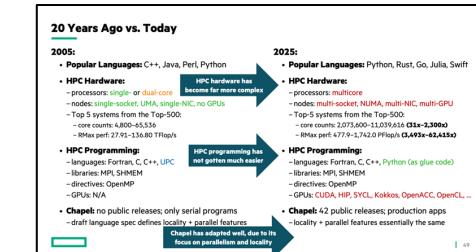


## Chapel is being used for productive parallel computing at all scales

- 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



## Chapel's general features for parallelism and locality have permitted it to adapt to the evolving parallel hardware landscape



# Thank you

---

<https://chapel-lang.org>  
@ChapelLanguage

