



Productive Parallel Programming from Laptops to Supercomputers with Chapel

Brad Chamberlain

SeaGL 2025

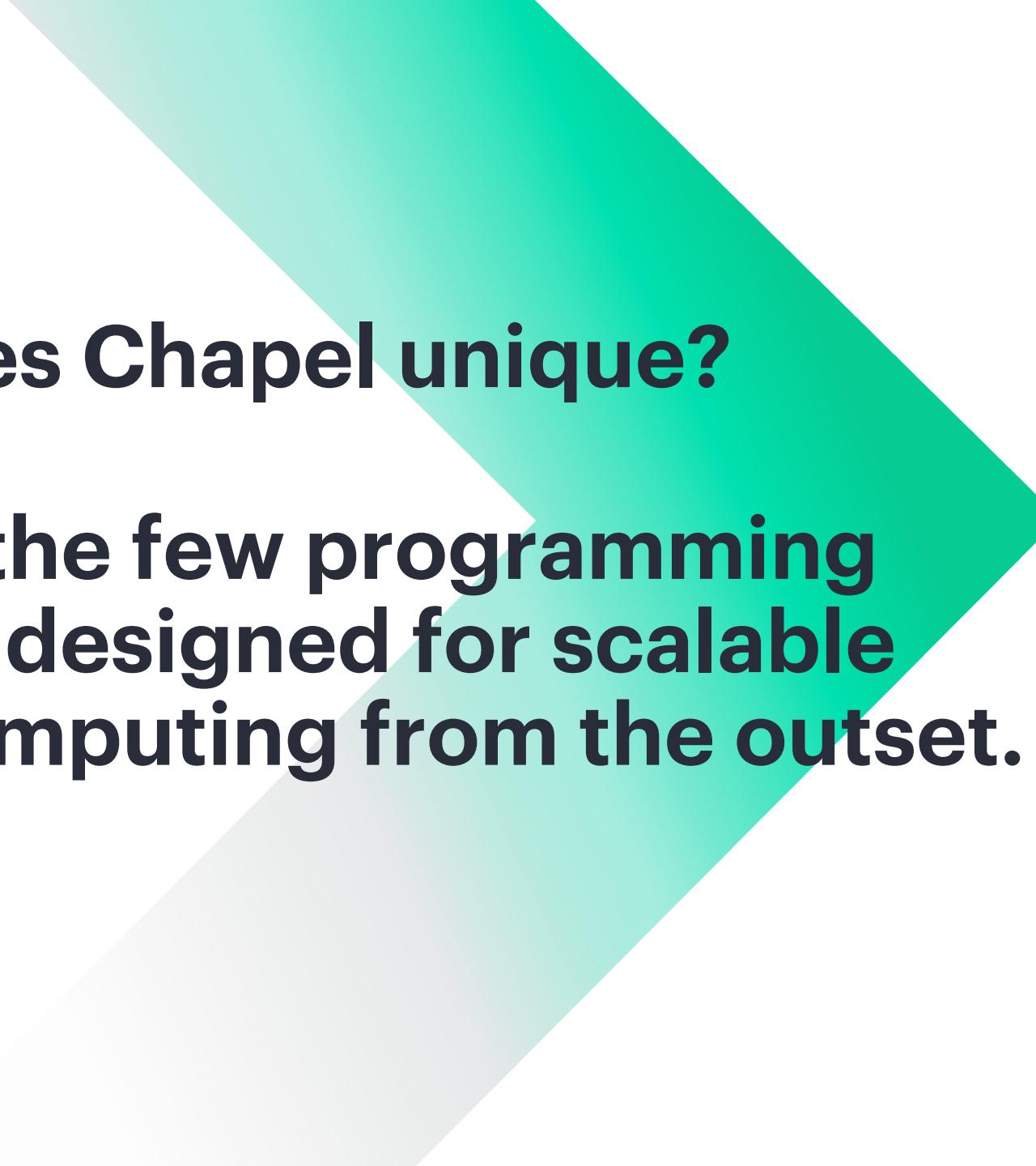
November 7, 2025

A photograph of a middle-aged man with short brown hair, smiling slightly. He is wearing a dark blue zip-up jacket over a green and white plaid shirt. He is seated at a desk, looking down at a white laptop screen. In the background, there is a yellow wall and another person whose back is to the camera, also working on a laptop.

A Bit About Me

A Bit About You?





Q: What makes Chapel unique?

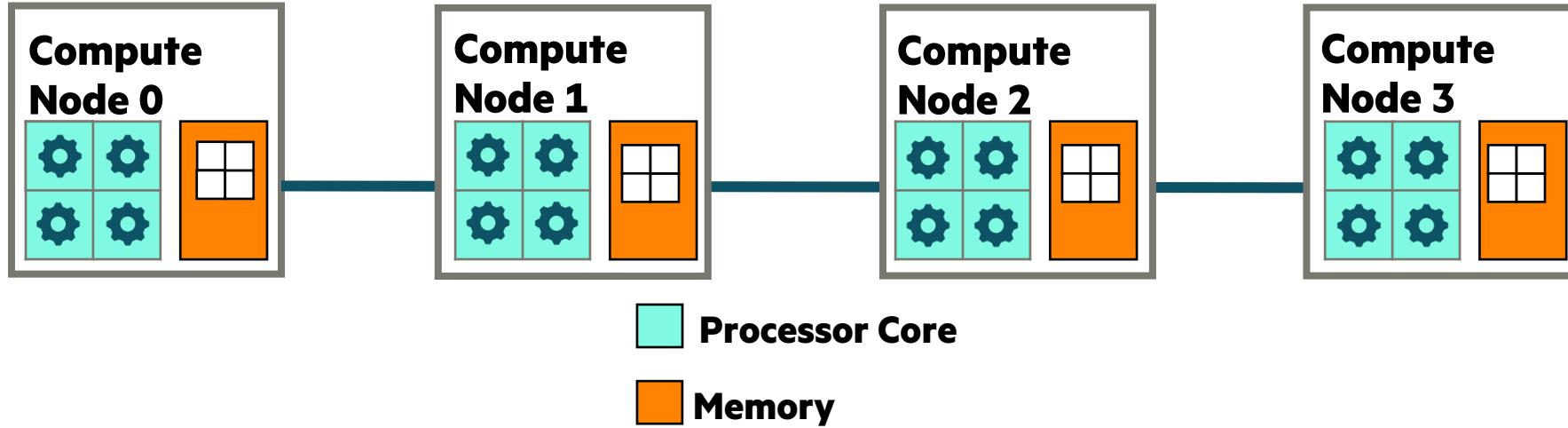
A: It's one of the few programming languages designed for scalable parallel computing from the outset.



What is [Scalable] Parallel Computing?

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

- 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

HPC = High Performance Computing

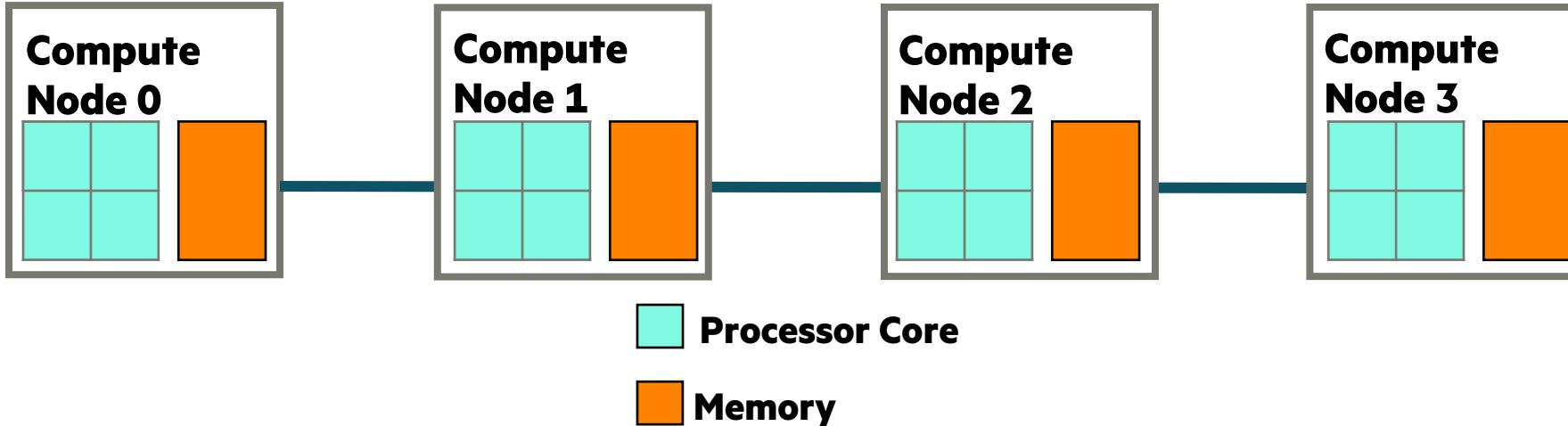
Parallel Computing has become Ubiquitous

Parallel computing, historically:

- supercomputers
- commodity clusters

Additional, ubiquitous parallelism today:

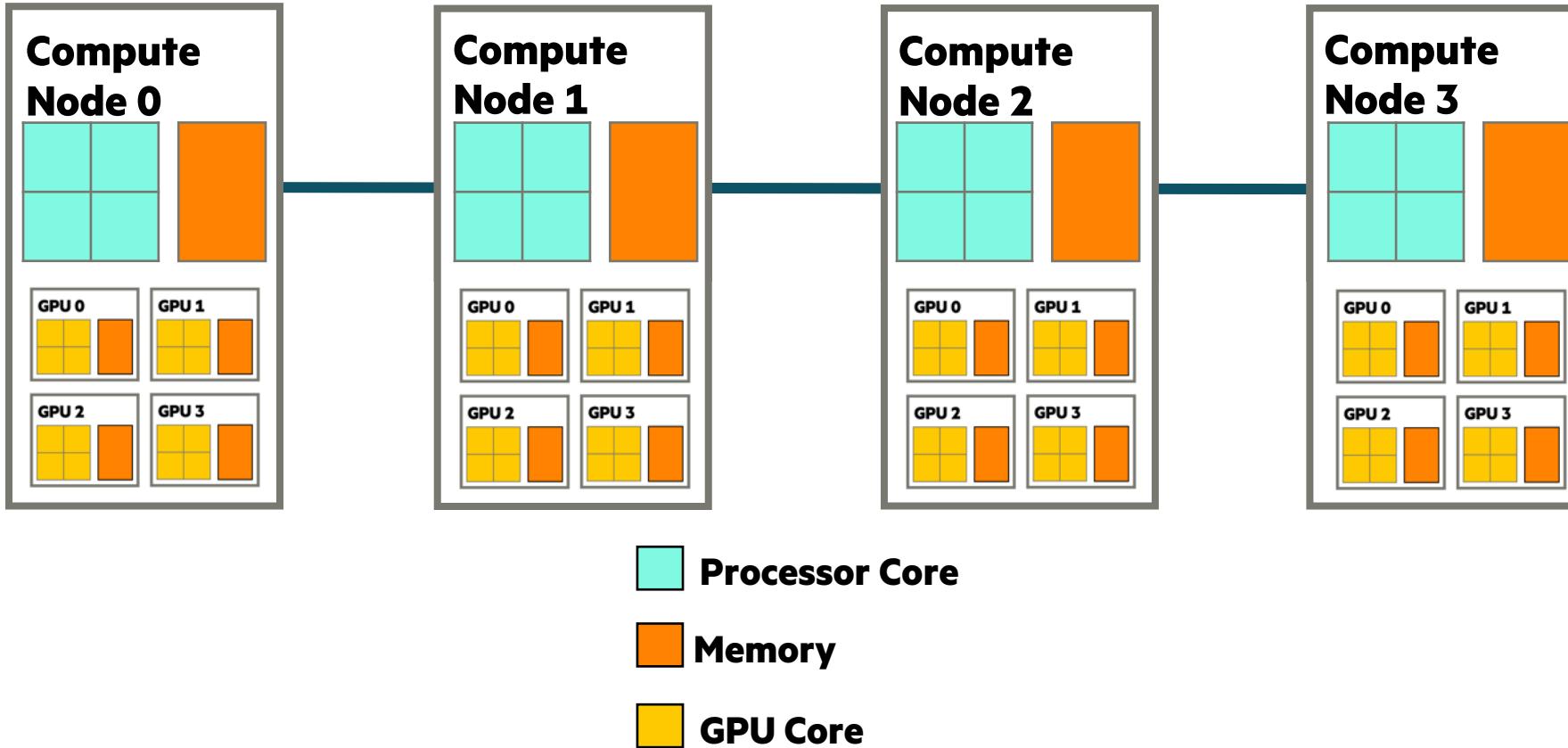
- multicore processors
- cloud computing
- GPUs



Parallel Computing has become Ubiquitous

Parallel computing, historically:

- supercomputers
- commodity clusters



What is Chapel?

Chapel: A modern parallel programming language

- Portable & scalable
- Open-source & collaborative
 - developed on GitHub
 - an HPSF / Linux Foundation project



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 / ...

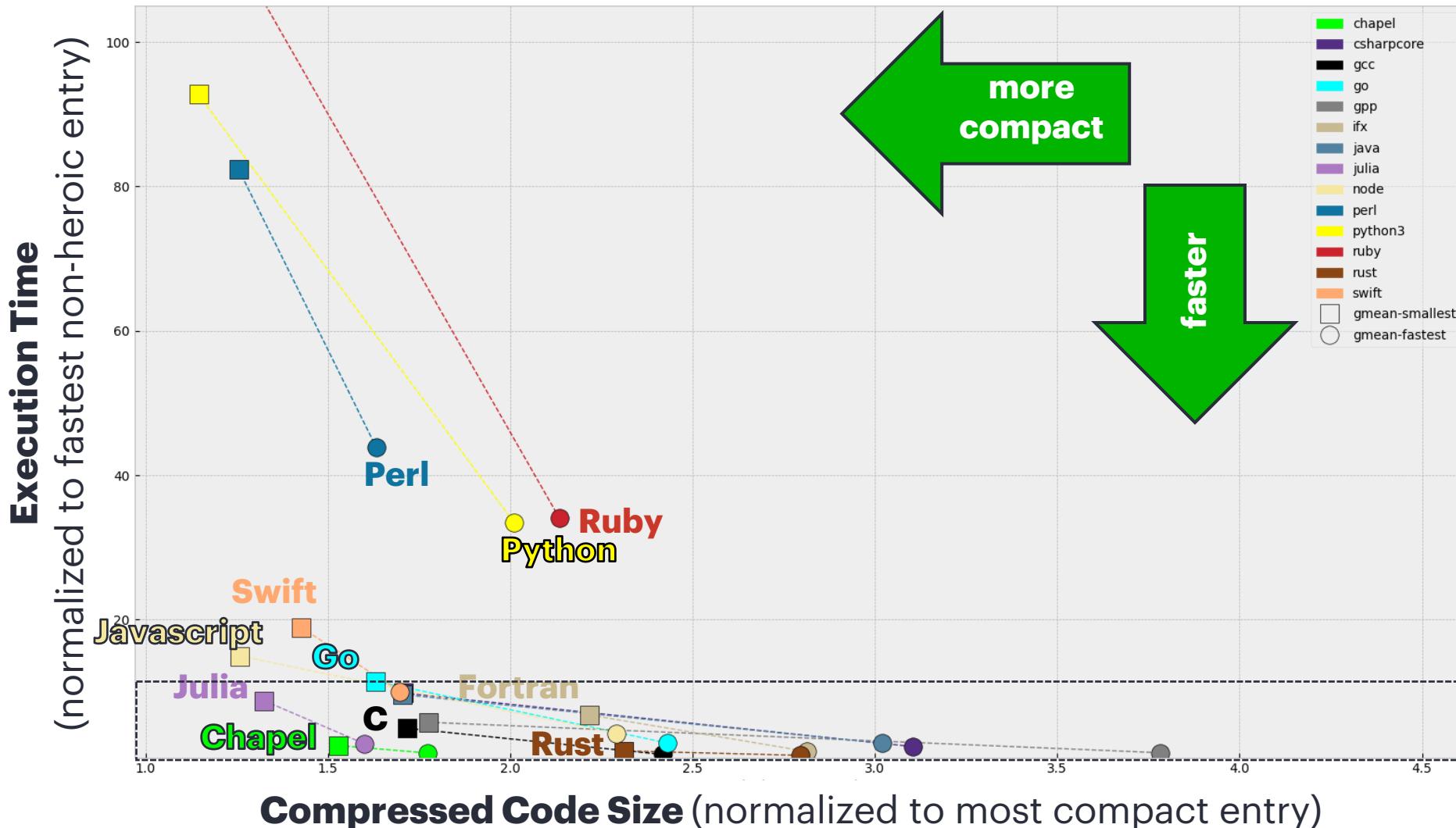
...**portable** as C

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

This is our motivation for Chapel

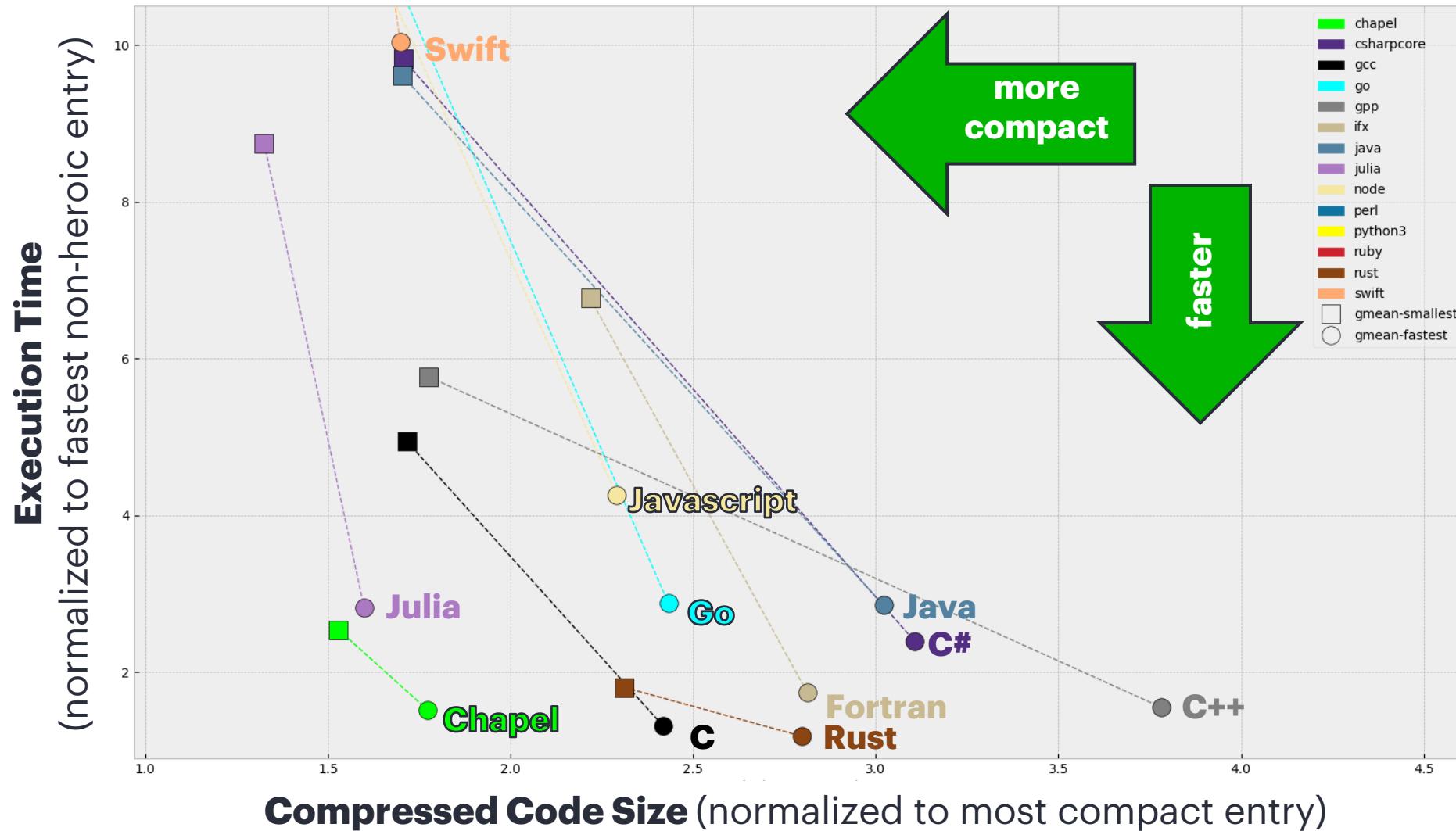


CLBG Language Comparison (selected languages, no heroic versions)



For further details, see my [ChapelCon '24 lightning talk](#)

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



For further details, see my [ChapelCon '24 lightning talk](#)

HPCC Benchmarks in C+MPI+OpenMP vs. Chapel

STREAM TRIAD: C + MPI + OPENMP

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

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

int HPCC_StartTeam(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

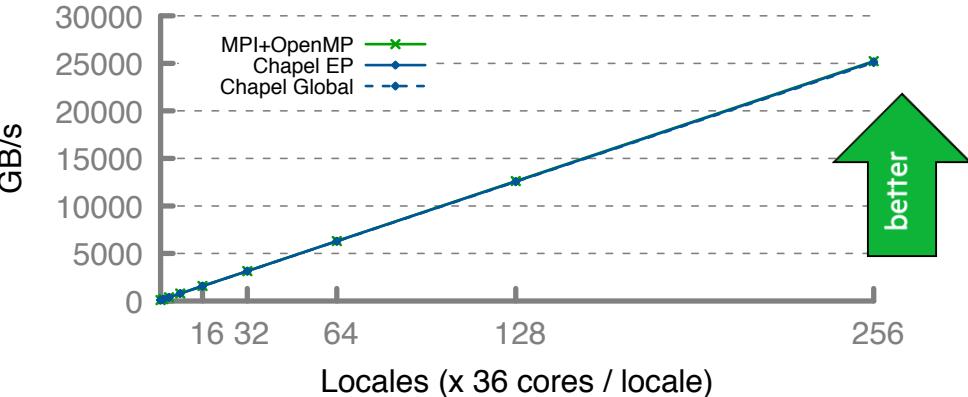
```
/* Perform updates to main table. The scalar equivalent is:
 * for(i=0;i<updates;i++)
 *   T[Updates[i].Index] = Updates[i].Value;
 */

MPI_Irecv(localBuff, localBuffSize, tparams.dtyped4, tparams.dtagged4,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &rcvreq);
while (1<=rcvreq) {
    if (status.MPI_TAG == UPDATE_TAG) {
        if (status.MPI_SOURCE == MPI_SELF) {
            if (status.MPI_STATUS == MPI_STATUS_INQUIRE) {
                MPI_Test(rcvreq, &have_done, &status);
                if (have_done) {
                    if (status.MPI_TAG == UPDATE_TAG) {
                        if (status.MPI_SOURCE == MPI_SELF) {
                            if (status.MPI_STATUS == MPI_STATUS_INQUIRE) {
                                MPI_Irecv(localBuff, localBuffSize, tparams.dtyped4, tparams.dtagged4,
                                          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &rcvreq);
                                while (1<=rcvreq) {
                                    if (status.MPI_TAG == UPDATE_TAG) {
                                        if (status.MPI_SOURCE == MPI_SELF) {
                                            if (status.MPI_STATUS == MPI_STATUS_INQUIRE) {
                                                MPI_Irecv(localBuff, localBuffSize, tparams.dtyped4, tparams.dtagged4,
                                                          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &rcvreq);
                                                while (1<=rcvreq) {
                                                    if (status.MPI_TAG == UPDATE_TAG) {
                                                        if (status.MPI_SOURCE == MPI_SELF) {
                                                            if (status.MPI_STATUS == MPI_STATUS_INQUIRE) {
                                                                MPI_Irecv(localBuff, localBuffSize, tparams.dtyped4, tparams.dtagged4,
                                                          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &rcvreq);
                                                          }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

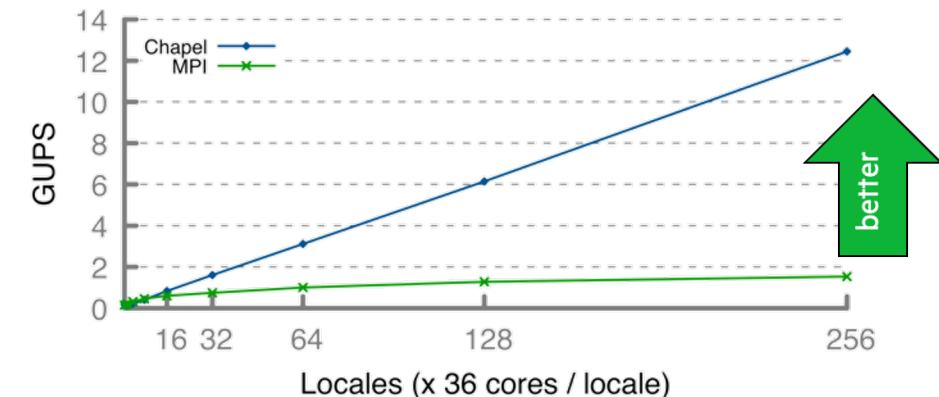
```
forall (_, r) in zip(Updates, RAStream()) do
    T[r & indexMask].xor(r);
```

72

STREAM Performance (GB/s)



RA Performance (GUPS)



Outline

Background & Motivation

Chapel by Example: Bale Index Gather

Chapel Applications

“Low-level” Features for Parallelism & Locality (time permitting)

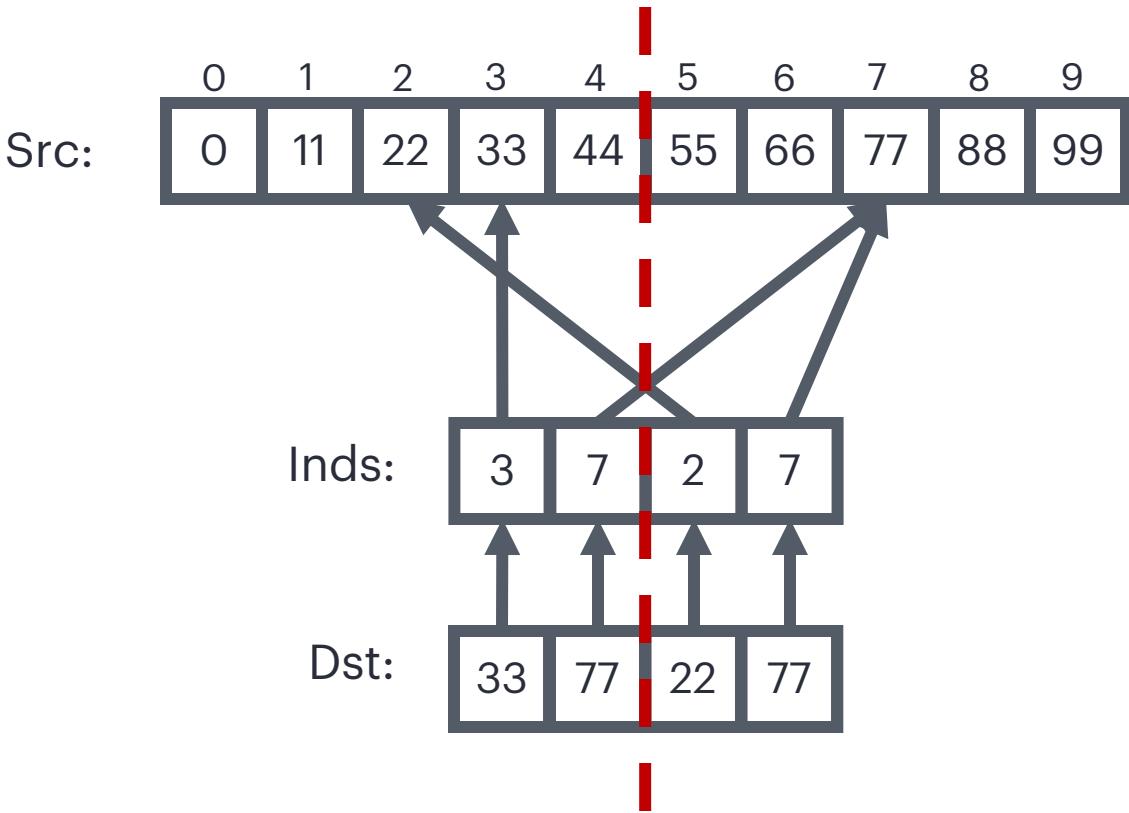
Wrap-up



Chapel by Example: Bale Index Gather



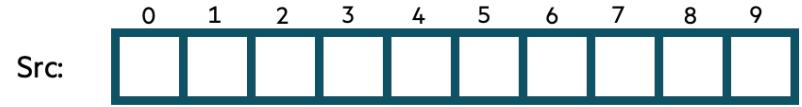
Bale Index Gather (IG): In Pictures



Bale IG in Chapel: Scalar and Array Declarations

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

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



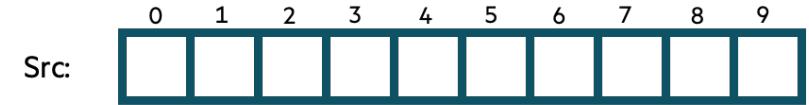
\$



Bale IG in Chapel: Compiling

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

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



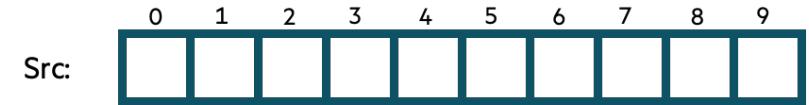
```
$ chpl bale-ig.chpl  
$
```



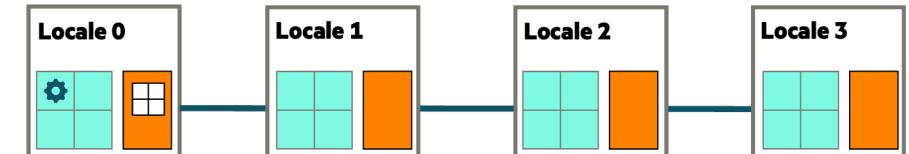
Bale IG in Chapel: Executing

```
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;
```

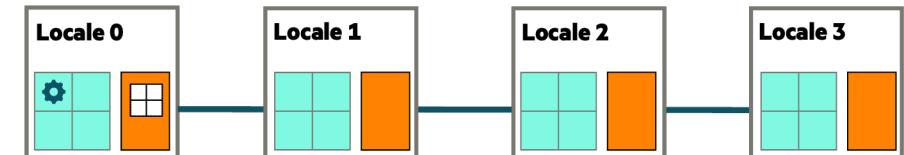
```
var Src: [0..<n] int,  
       Inds, Dst: [0..<m] int;
```

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;

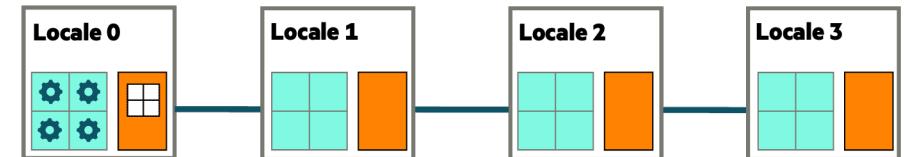
config const n = 10,
      m = 4;

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

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

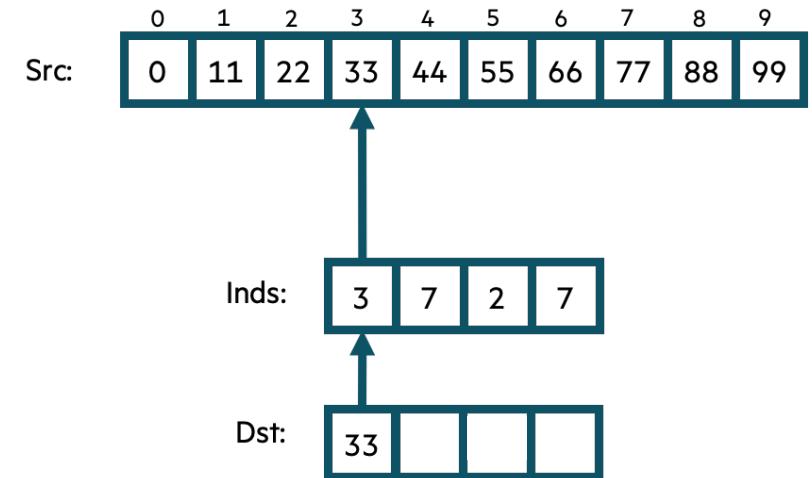


```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

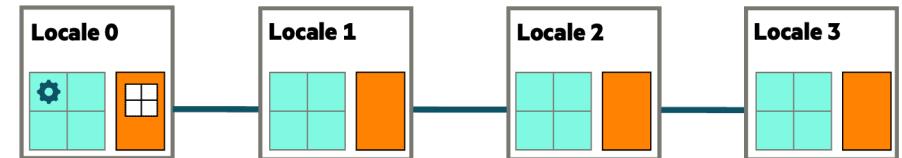


Bale IG in Chapel: Serial Version

```
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]];
```

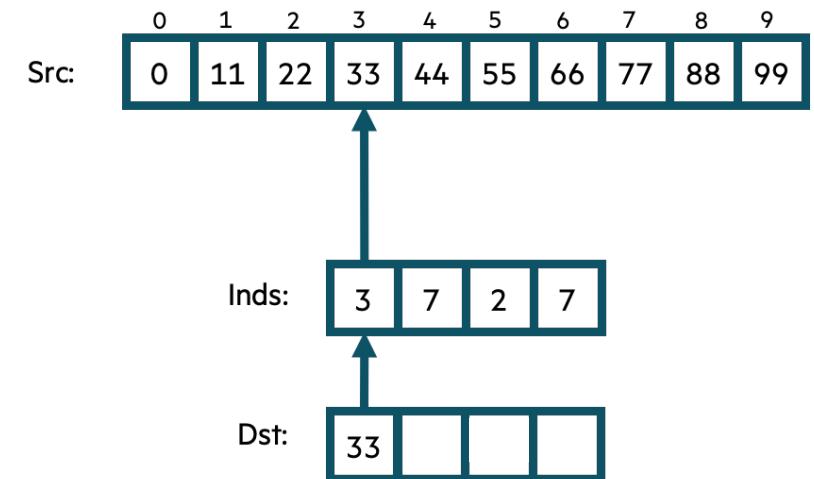


```
$ chpl bale-ig.chpl  
$ ./bale-ig
```

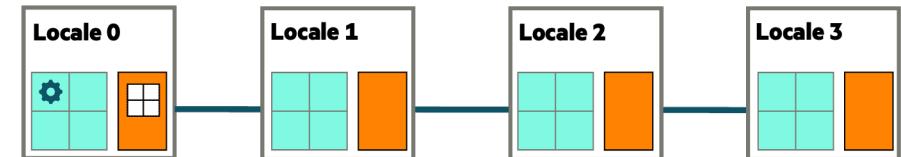


Bale IG in Chapel: Serial, Zippered Version

```
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];
```

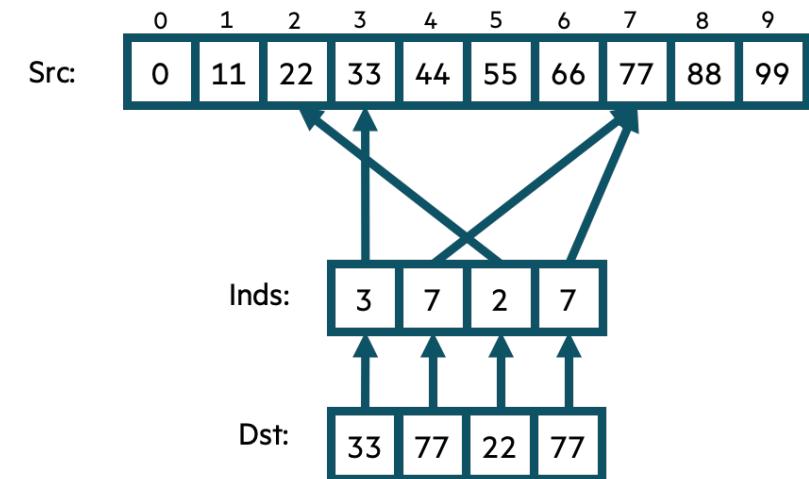


```
$ chpl bale-ig.chpl  
$ ./bale-ig
```

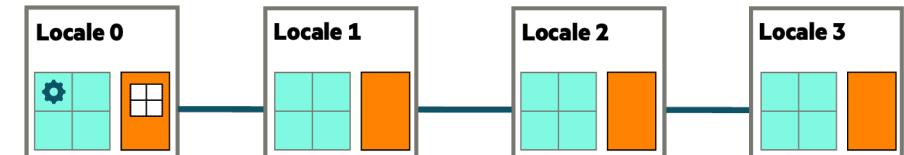


Bale IG in Chapel: Parallel, Zippered Version (vectorized)

```
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];
```

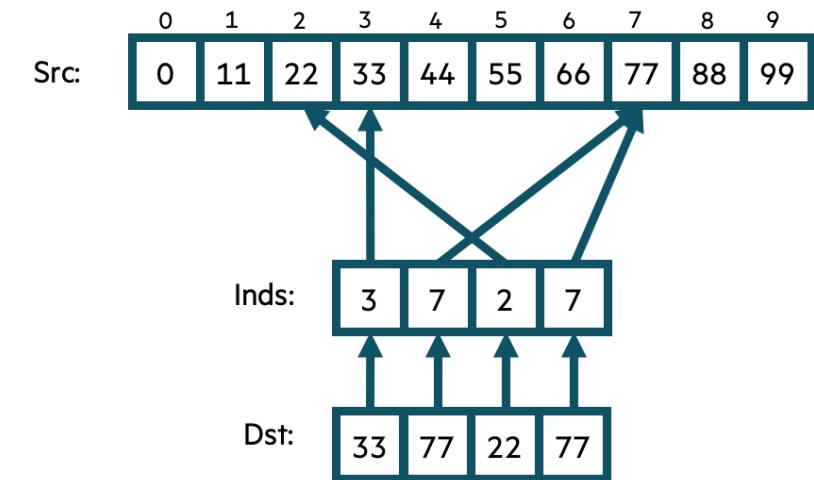


```
$ chpl bale-ig.chpl  
$ ./bale-ig
```

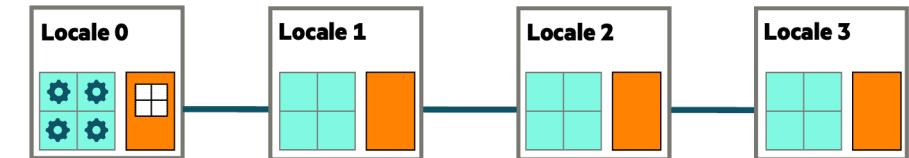


Bale IG in Chapel: Parallel, Zippered Version (multicore)

```
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];
```

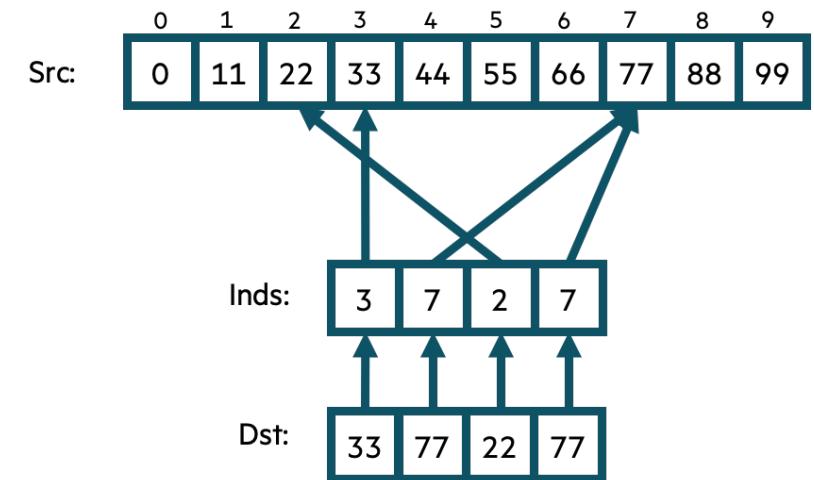


```
$ chpl bale-ig.chpl  
$ ./bale-ig  
$
```

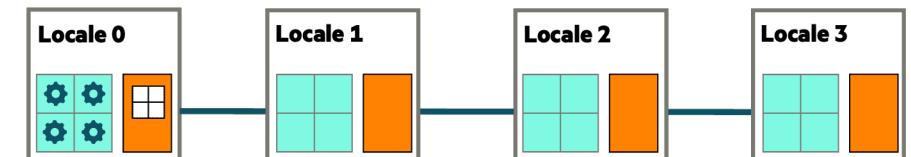


Bale IG in Chapel: Parallel Promoted Version (equivalent to previous version)

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

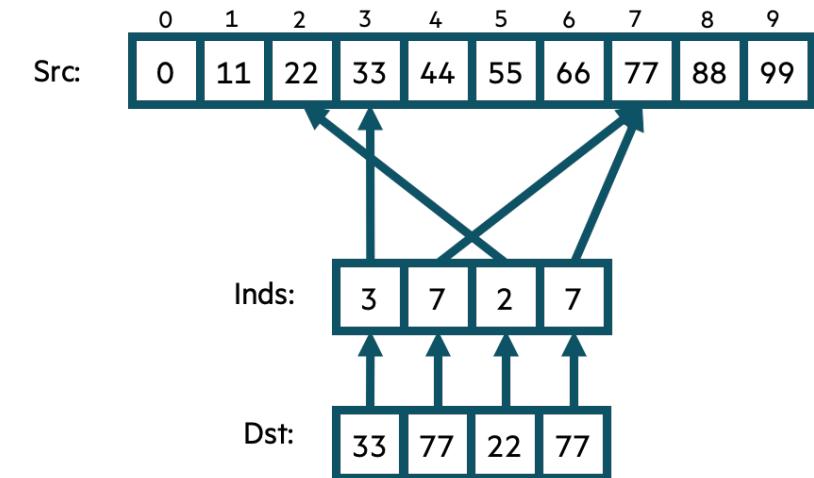


```
$ chpl bale-ig.chpl  
$ ./bale-ig  
$
```

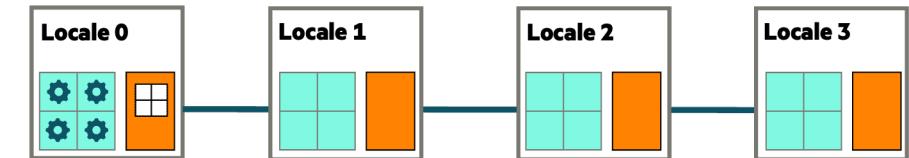


Bale IG in Chapel: Parallel, Zippered Version (Multicore, again)

```
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];
```

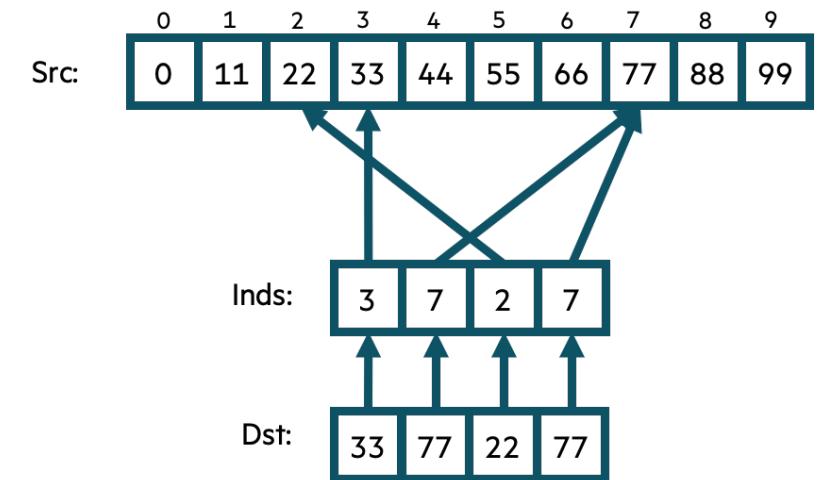


```
$ chpl bale-ig.chpl  
$ ./bale-ig
```

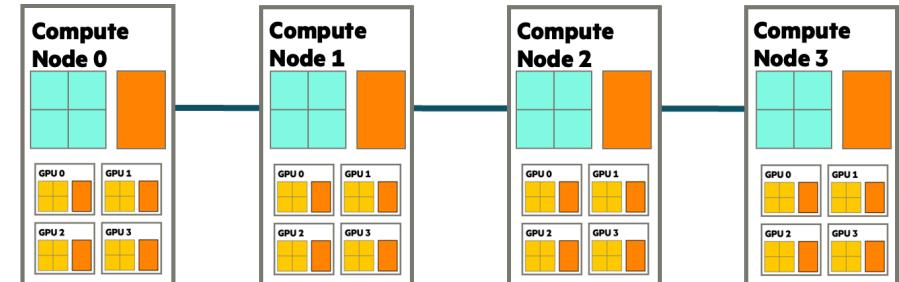


Bale IG in Chapel: Parallel, Zippered Version (GPU)

```
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];  
}
```

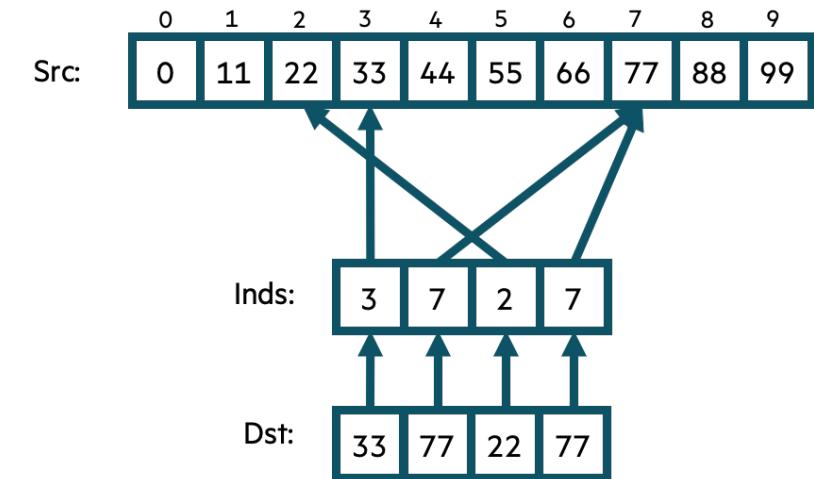


```
$ chpl bale-ig.chpl  
$ ./bale-ig
```

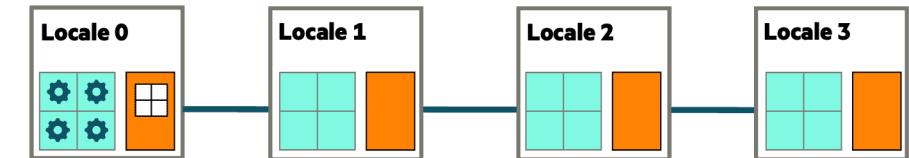


Bale IG in Chapel: Parallel, Zippered Version (Multicore, again)

```
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];
```

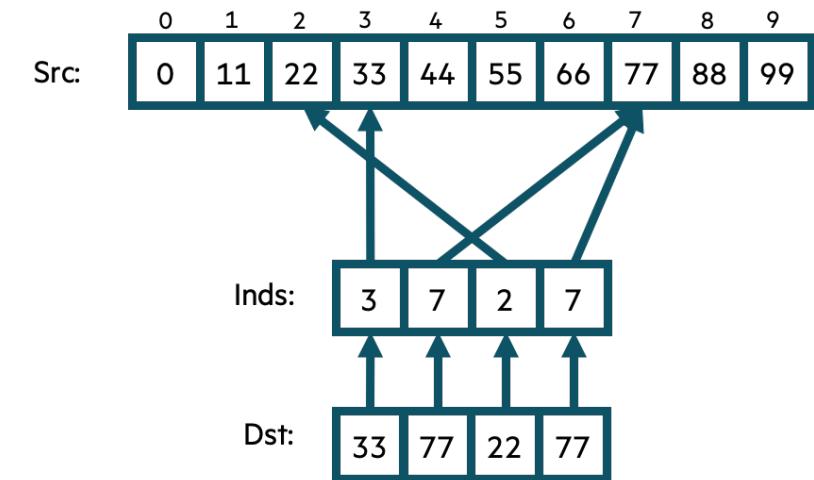


```
$ chpl bale-ig.chpl  
$ ./bale-ig
```

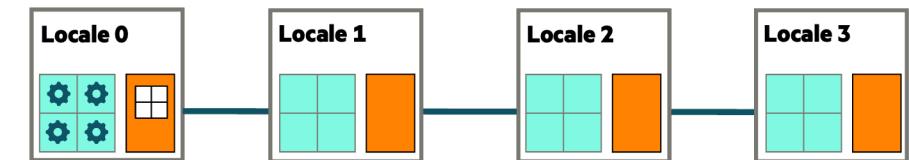


Bale IG in Chapel: Parallel , Zippered Version with Named Domains (Multicore)

```
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
```



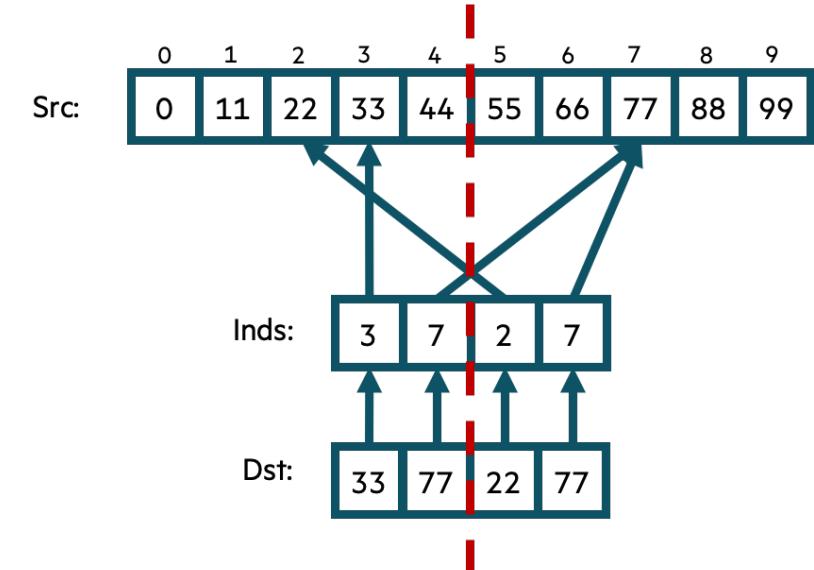
Bale IG in Chapel: Distributed Parallel Version

```
use BlockDist;

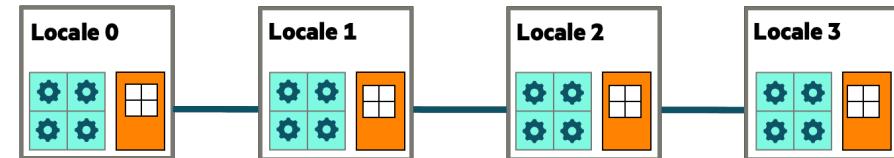
config const n = 10,
          m = 4;

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;

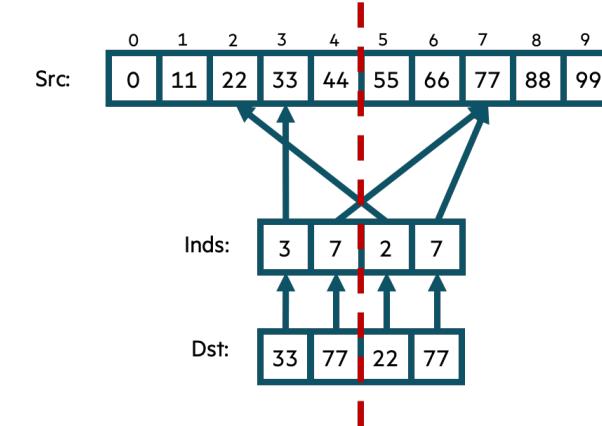
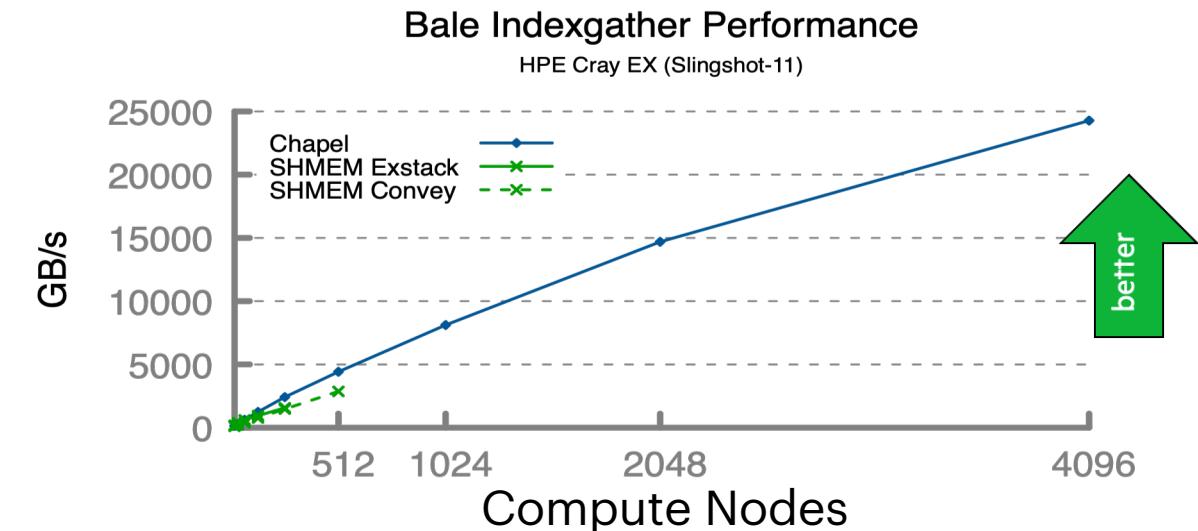
config const n = 10,
      m = 4;

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

```
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();
}
```

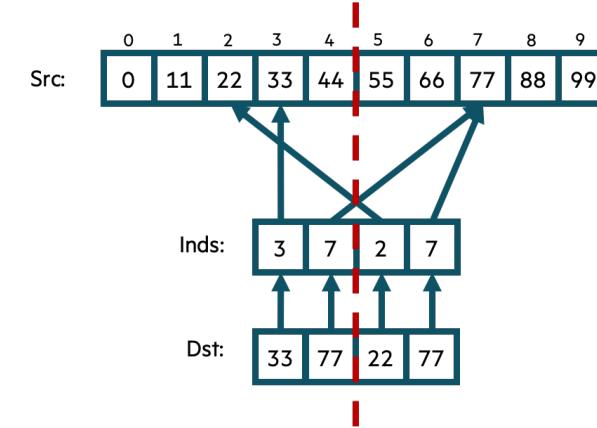
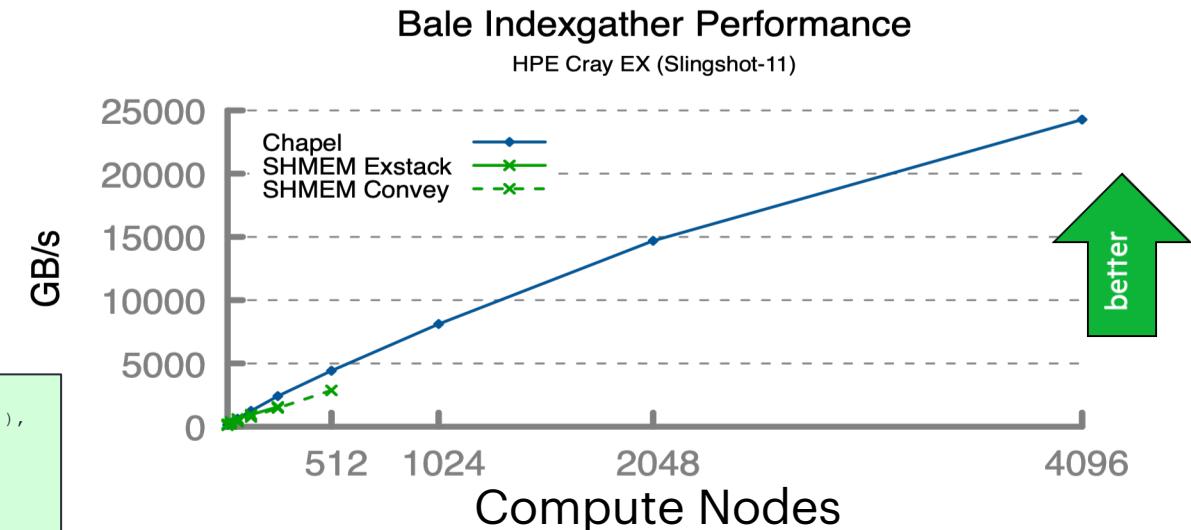
SHMEM (Conveyors version)

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

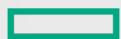
    for (; i < l_num_req; i++) {
        pkg.idx = i;
        pkg.val = pckindx[i] >> 16;
        pe = pckindx[i] & 0xffff;
        if (!convey_push(requests, &pkg, pe))
            break;
    }

    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;
        }
    }

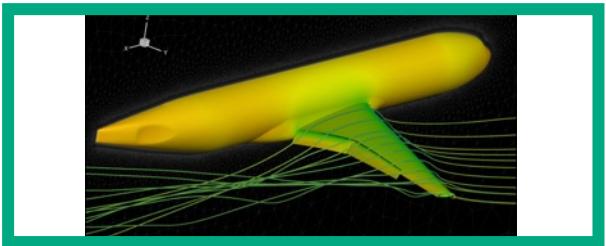
    while (convey_pull(replies, ptr, NULL) == convey_OK)
        tgt[ptr->idx] = ptr->val;
}
```



Chapel Applications

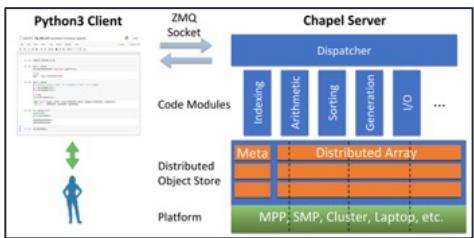


Applications of Chapel



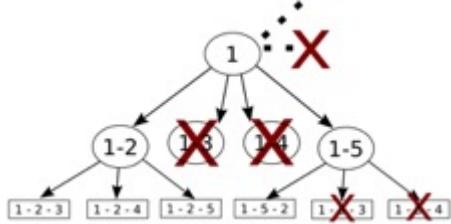
CHAMPS: 3D Unstructured CFD

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



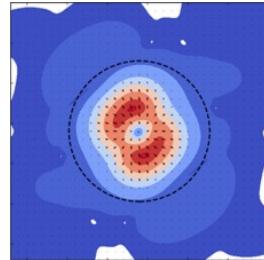
Arkouda: Interactive Data Science at Massive Scale

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

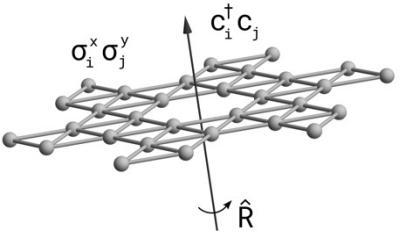


ChOp: Chapel-based Optimization

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

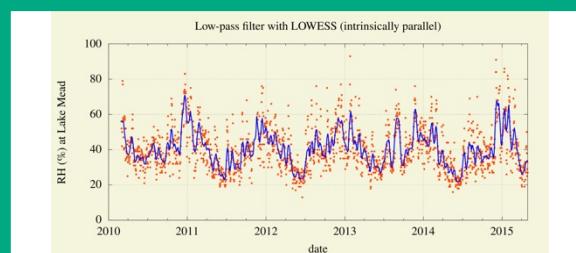


ChplUltra: Simulating Ultralight Dark Matter
Nikhil Padmanabhan, J. Luna Zagorac, et al.
Yale University et al.



Lattice-Symmetries: a Quantum Many-Body Toolbox

Tom Westerhout
Radboud University



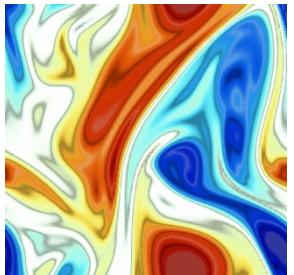
Desk dot chpl: Utilities for Environmental Eng.

Nelson Luis Dias
The Federal University of Paraná, Brazil

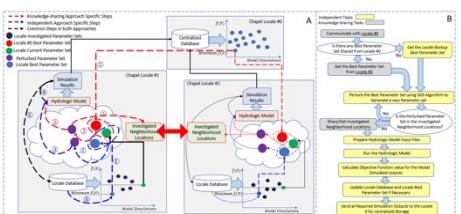


RapidQ: Mapping Coral Biodiversity

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

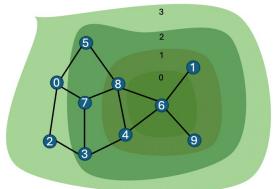


ChapQG: Layered Quasigeostrophic CFD
Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



Chapel-based Hydrological Model Calibration

Marjan Asgari, et al.
University of Guelph

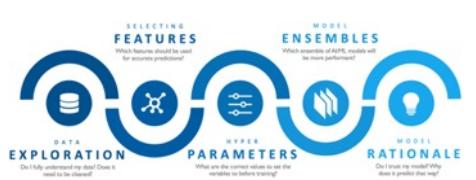


Arachne Graph Analytics
Bader, Du, Rodriguez, et al.
New Jersey Institute of Technology



Modeling Ocean Carbon Dioxide Removal

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



CrayAI HyperParameter Optimization (HPO)

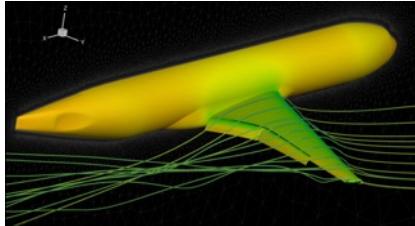
Ben Albrecht, et al.
Cray Inc. / HPE

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

CHAMPS Summary

What is it?

- 3D unstructured CFD framework for airplane simulation
- ~100+k lines of Chapel written since 2019



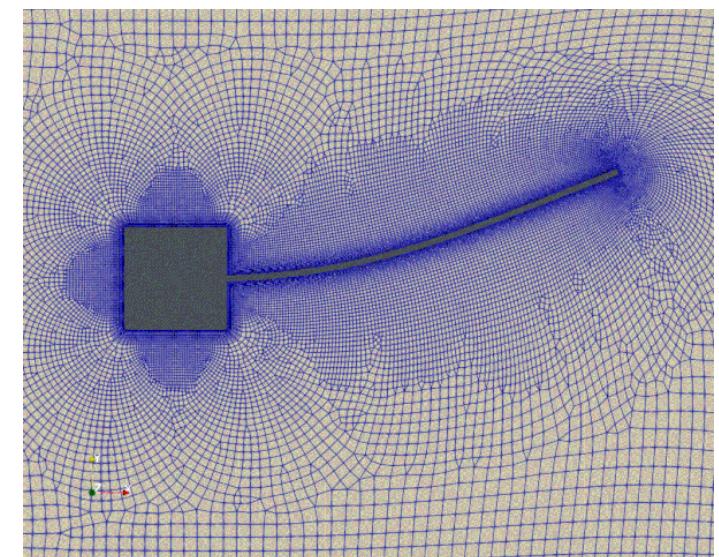
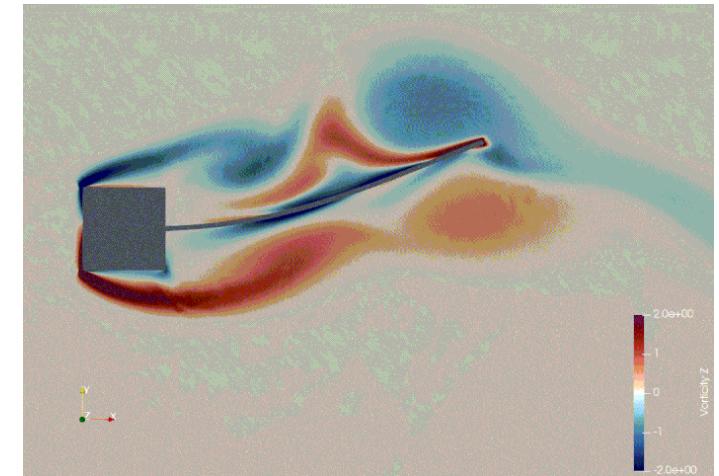
Who wrote it?

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



Why Chapel?

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



[images provided by the CHAMPS team and used with permission]



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

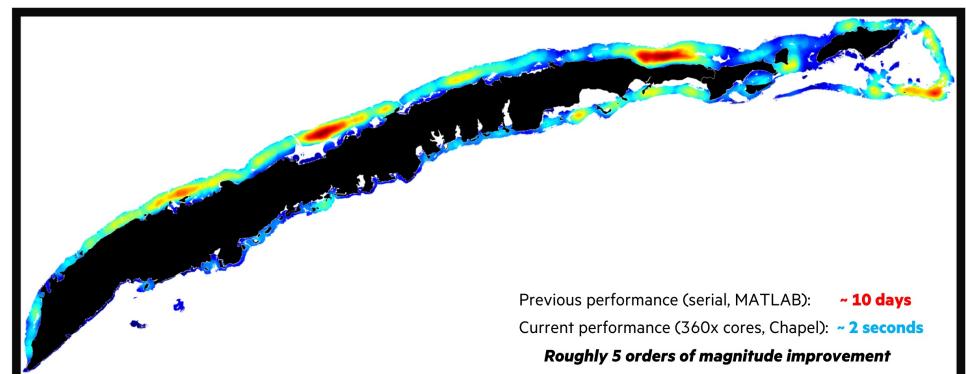
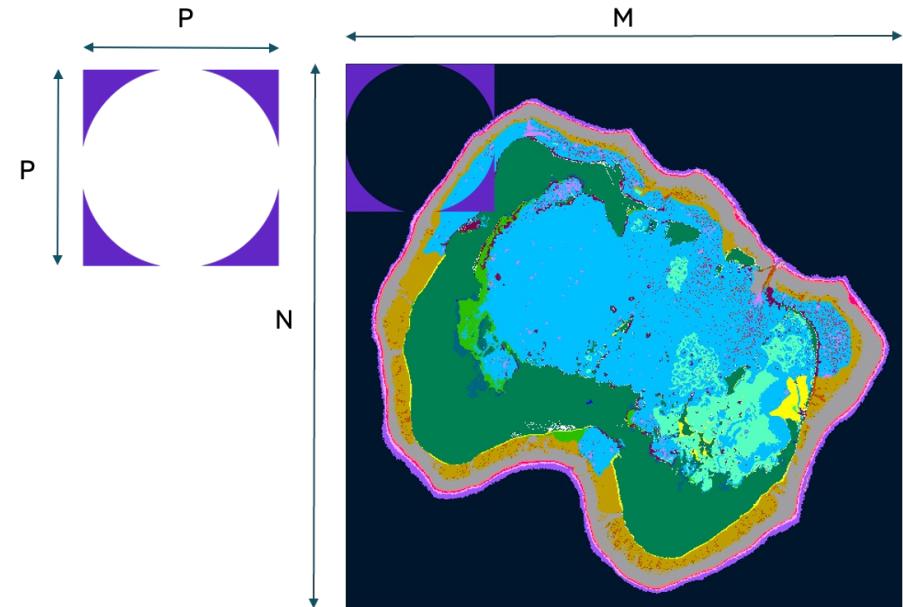
Who wrote it?

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



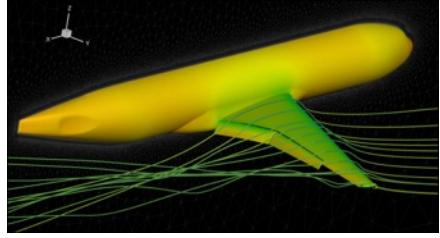
Why Chapel?

- easy transition from Matlab, which they had been using
- massive performance improvement:
 - previous ~10-day run finished in ~2 seconds using 360 cores
- enabled unanticipated algorithmic improvements
 - from $O(M \cdot N \cdot P)$ habitat diversity to $O(M \cdot N \cdot P^3)$ spectral diversity
 - Added another ~90 lines of code to make it GPU-enabled
 - ~4-week desktop run → ~20 minutes on 20 nodes / 512 GPUs



[images provided by Scott Bachman from his [CHIUW 2023 talk](#) and used with permission]

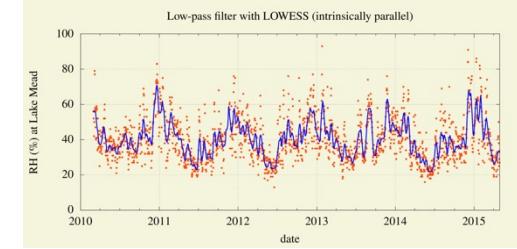
Diversity in Chapel Application Scales (Code Size and Systems)



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



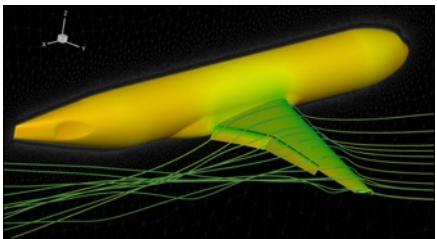
Computation: Coral reef image analysis
Code size: ~320 lines
Systems: Desktops, HPC systems w/ GPUs



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

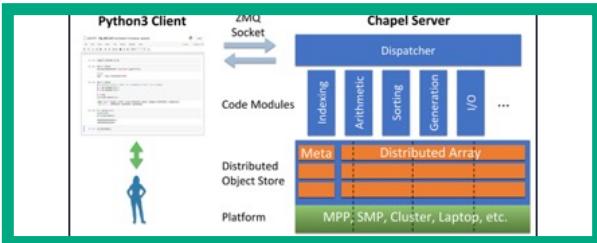
[images provided by their respective teams and used with permission]

Applications of Chapel



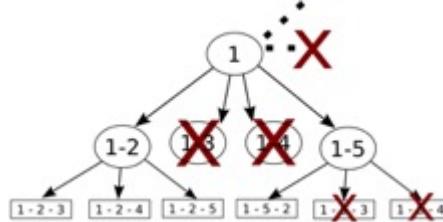
CHAMPS: 3D Unstructured CFD

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



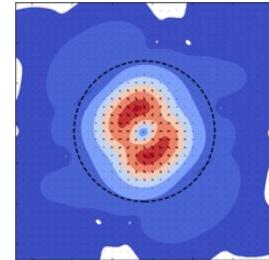
Arkouda: Interactive Data Science at Massive Scale

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



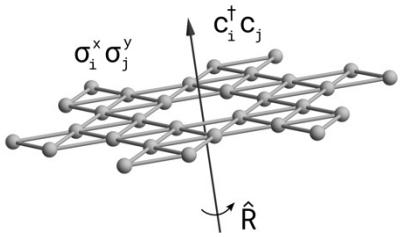
ChOp: Chapel-based Optimization

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



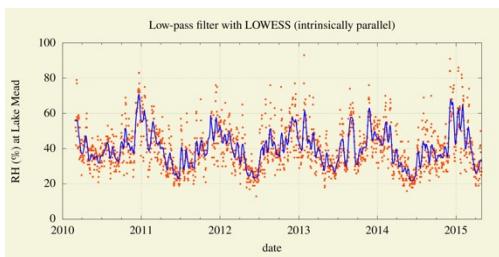
ChplUltra: Simulating Ultralight Dark Matter

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



Lattice-Symmetries: a Quantum Many-Body Toolbox

Tom Westerhout
Radboud University



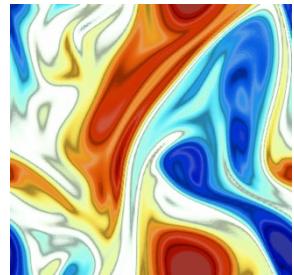
Desk dot chpl: Utilities for Environmental Eng.

Nelson Luis Dias
The Federal University of Paraná, Brazil



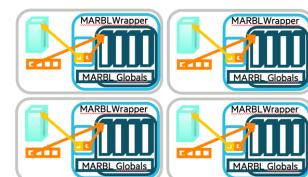
RapidQ: Mapping Coral Biodiversity

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



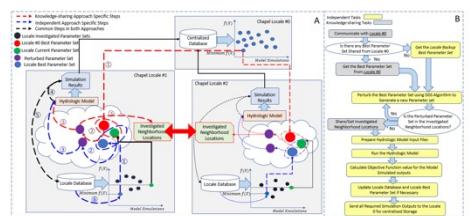
ChapQG: Layered Quasigeostrophic CFD

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



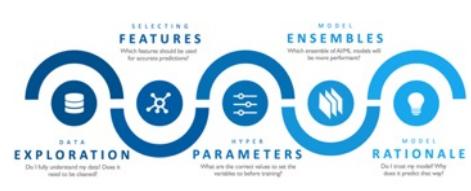
Chapel-based Hydrological Model Calibration

Marjan Asgari et al.
University of Guelph



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



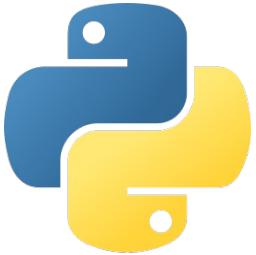
CrayAI HyperParameter Optimization (HPO)

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)

A screenshot of a Jupyter Notebook interface titled "jupyter big_add_sum Last Checkpoint: 16 minutes ago (abuseven)". The notebook shows the following code in cell In [1]:

```
import arkouda as ak
```

In [2]:

```
ak.v = False  
ak.startupserver("localhost",port=5555)
```

In [3]:

```
ak.v = False  
E = ak.randint(0, 1000 * E == 65536L * 2**12 * E == 2560L  
A = ak.arange(0, E)  
B = ak.arange(0, E)  
C = A+B  
print(ak.sum(C))
```

In [4]:

```
name="A.npy" dtype="int64" size:1000000000 ndim:1 shape:(1000000000) itemsize:8  
[0 2 ... 19999994 19999996 19999998]
```

In [5]:

```
ak.shutdown()
```



User writes Python code
making familiar NumPy/Pandas calls

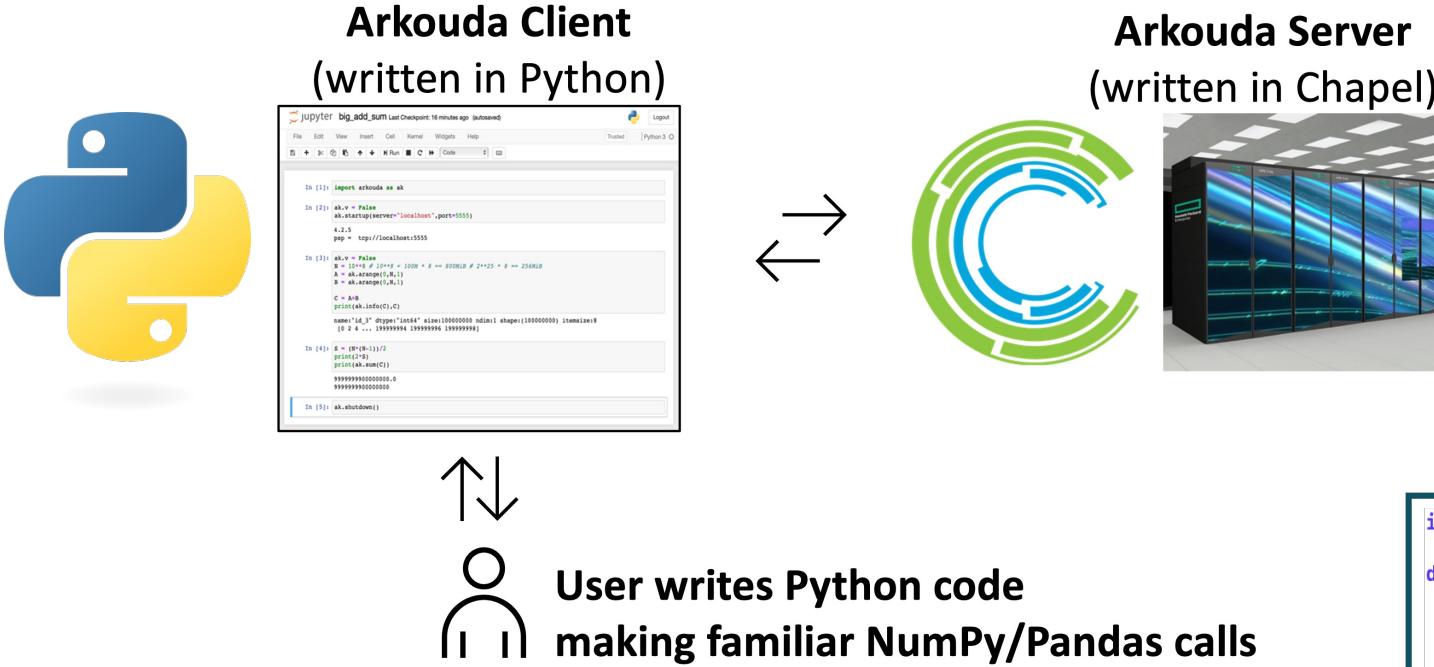
```
import arkouda as ak

def ak_argsort(N, seed):
    a = ak.randint(0, 2**64, N, dtype=ak.uint64, seed=seed)
    perm = ak.argsort(a)

    assert ak.is_sorted(a[perm])
```

What is Arkouda?

Q: “What is Arkouda?”



A1: “A scalable version of NumPy / Pandas for data scientists”

A2: “A framework for driving supercomputers interactively from Python”

Performance and Productivity: Arkouda Argsort

HPE Cray EX

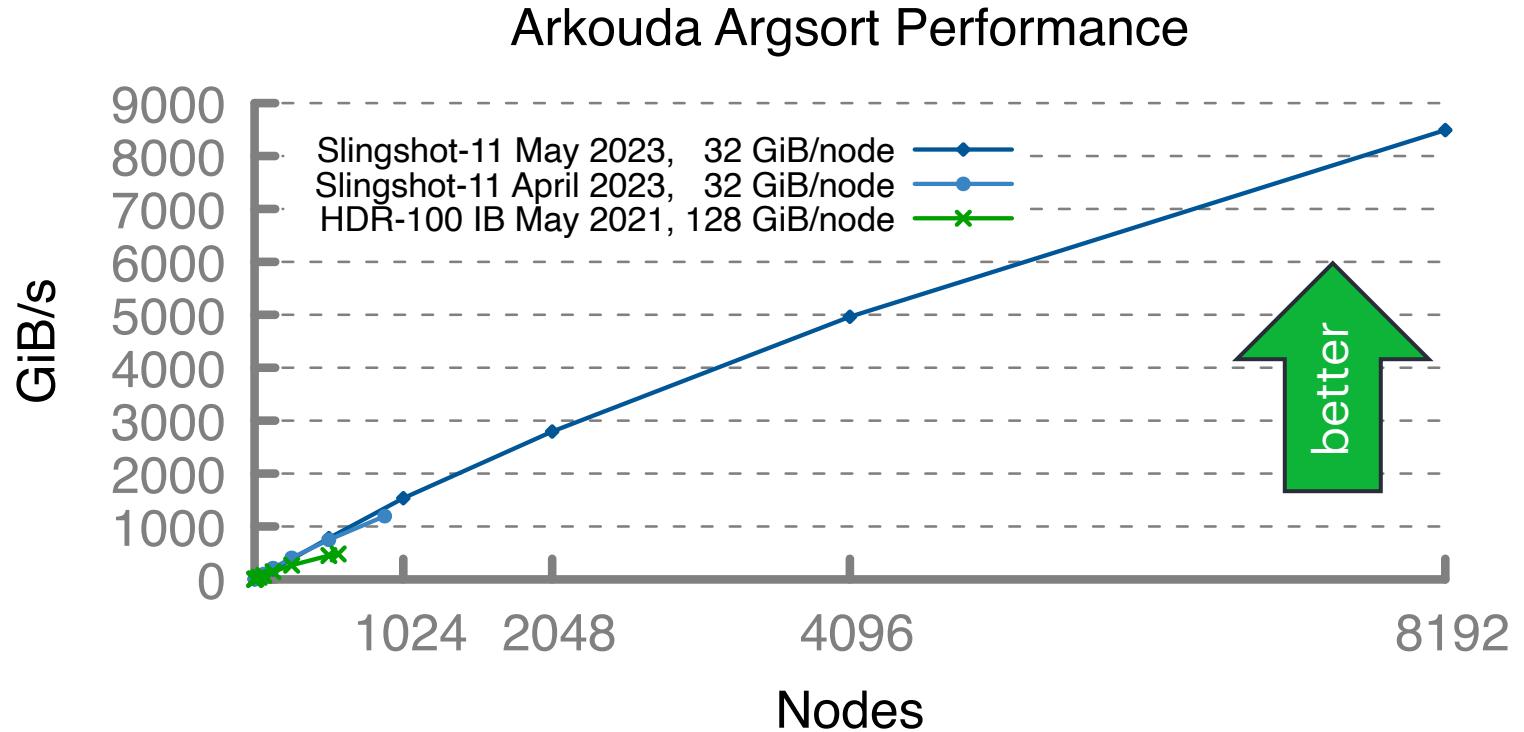
- Slingshot-11 network (200 Gb/s)
- 8192 compute nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)

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



For More Information on Arkouda

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

The screenshot shows the Arkouda website homepage. At the top, there's a navigation bar with the Arkouda logo, GitHub, documentation, and Gitter links. Below the header, a main banner features the text "Massive-scale data science, from the comfort of your laptop". It includes two comparison points: "Arkouda Ready for supercomputers" and "NumPy Industry standard". A code snippet in the center shows how to use Arkouda to perform operations like generating large arrays, adding them, and sorting the results.

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 Trookman, Erias

"7 Questions for Chapel Users" Interview series

A good way to learn more about Chapel users' apps and experiences

- <https://chapel-lang.org/blog/series/7-questions-for-chapel-users/>



7 Questions for David Bader: Graph Analytics at Scale with Arkouda and Chapel

Posted on November 6, 2024.

Tags: [User Experiences](#) [Interviews](#) [Graph Analytics](#) [Arkouda](#)
By: [Engin Kayraklıoglu](#), [Brad Chamberlain](#)



7 Questions for Scott Bachman: Analyzing Coral Reefs with Chapel

"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."



Chapel Language Blog

[About](#) [Chapel Website](#) [Featured](#) [Series](#) [Tags](#) [Authors](#) [All Posts](#)



7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

"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 Tiago Carneiro and Guillaume Helbecque: Combinatorial Optimization in Chapel

Posted on July 30, 2025.

Tags: [User Experiences](#) [Interviews](#)
By: [Engin Kayraklıoglu](#), [Brad Chamberlain](#)



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

"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."



7 Questions for Nelson Luís Dias: Atmospheric Turbulence in Chapel

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



7 Questions for Marjan Asgari: Optimizing Hydrological Models with Chapel

Posted on September 15, 2025.

Tags: [User Experiences](#) [Interviews](#) [Earth Sciences](#)
By: [Engin Kayraklıoglu](#), [Brad Chamberlain](#)

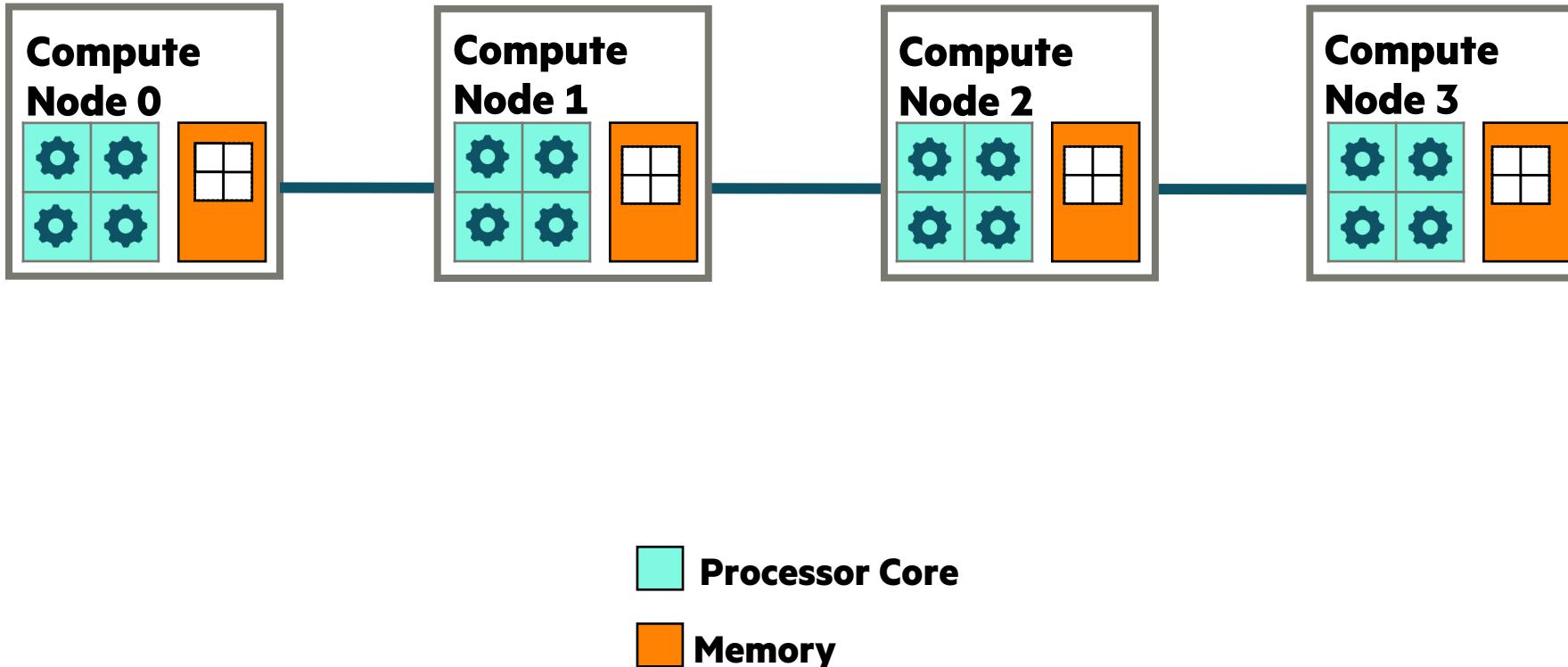


“Low-level” Features for Parallelism and Locality



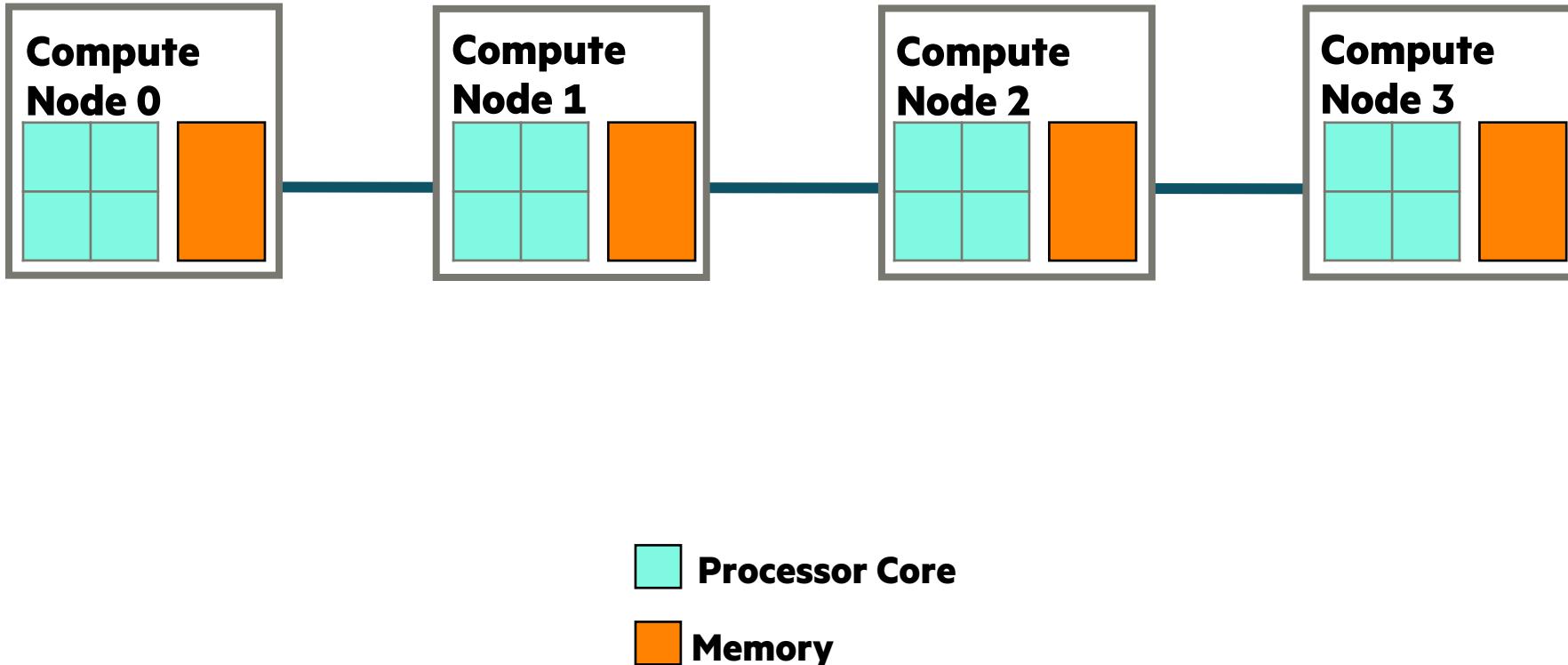
Key Concerns for Scalable Parallel Computing

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



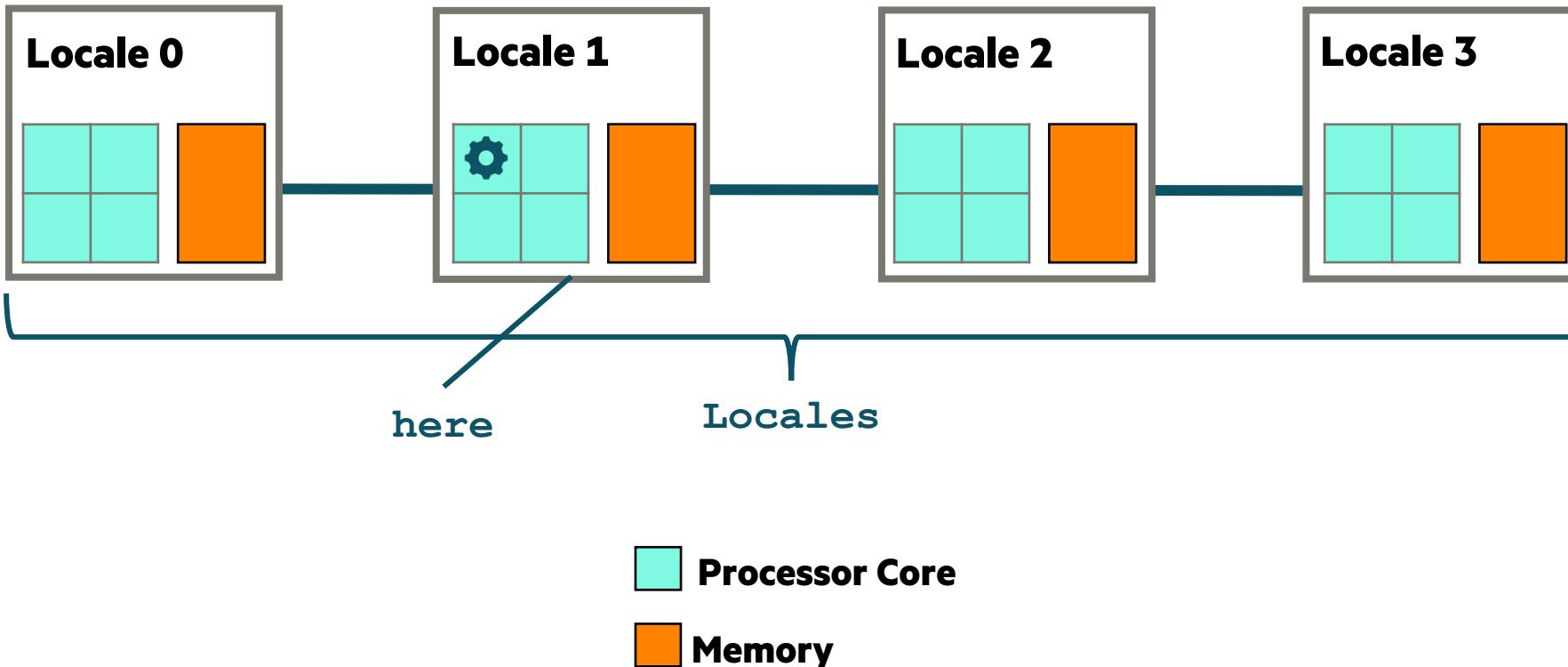
Locales in Chapel

- 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



Built-In Locale Variables in Chapel

- Two built-in variables for referring to locales within Chapel:
 - **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

```
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

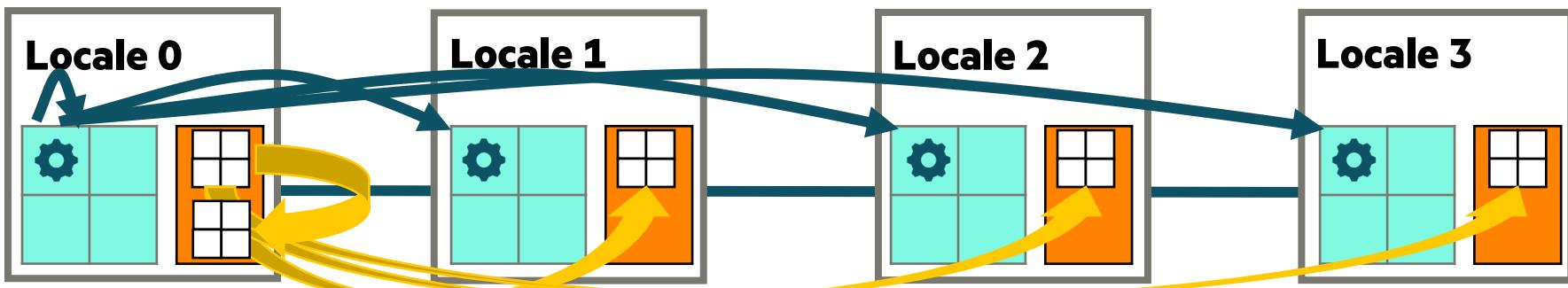
Variables are stored using the memory local to the current task

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

coforall.chpl

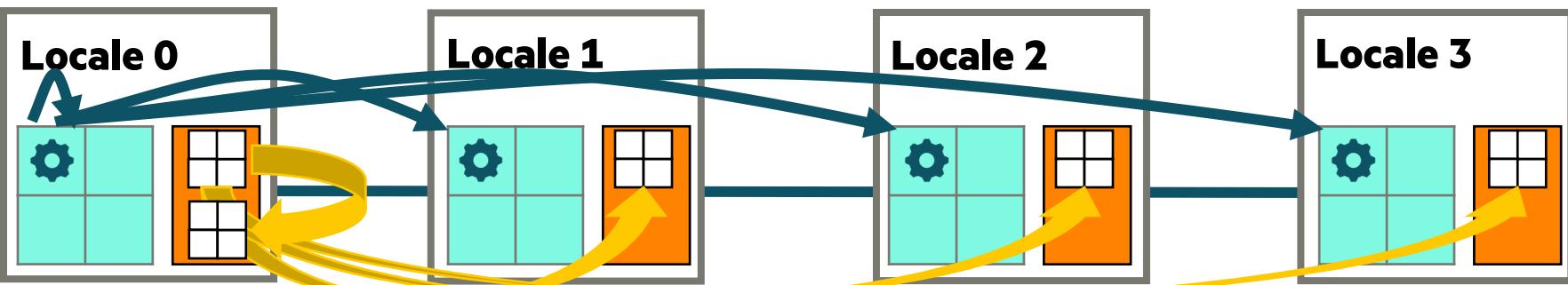
```
writeln("Hello from locale ", here.id);

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 is a distributed parallel computation

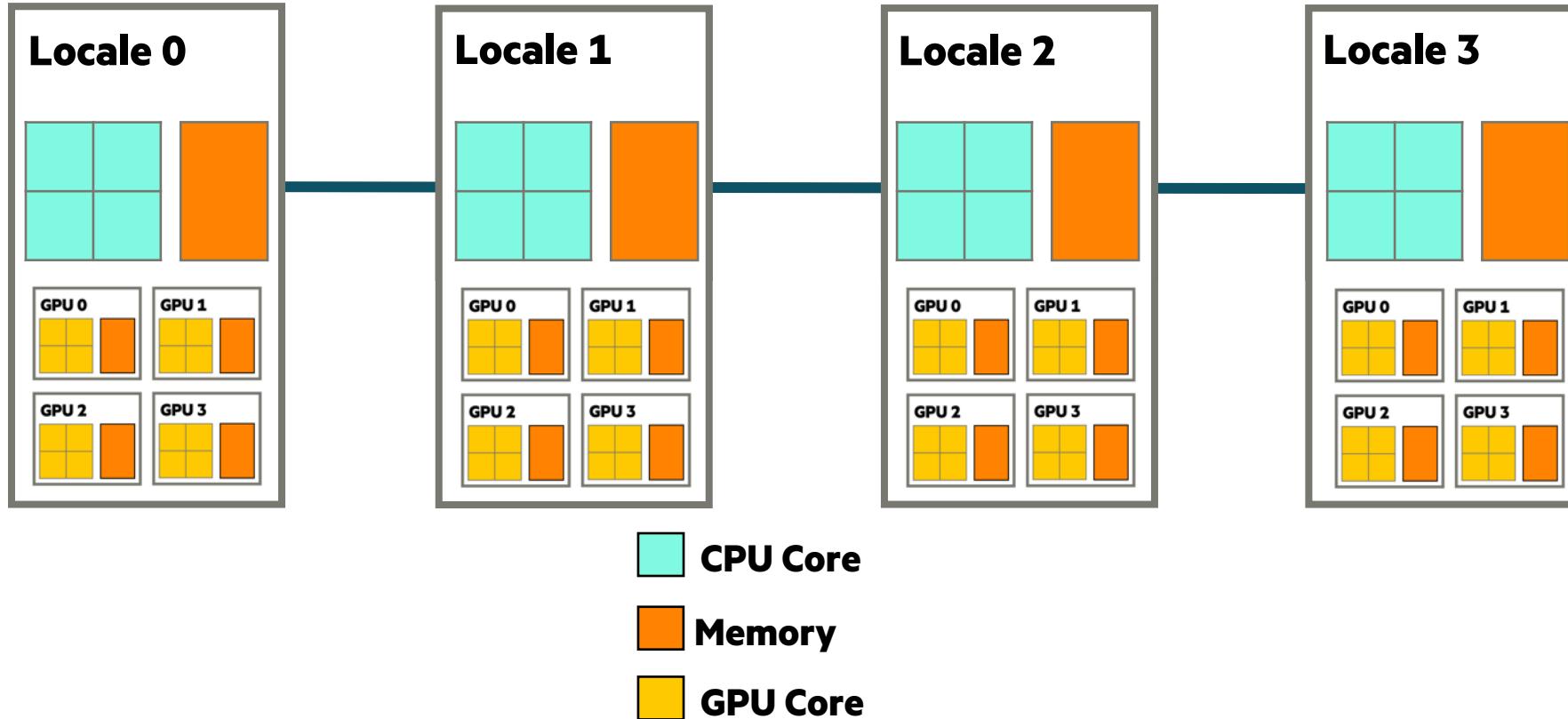


Representing GPUs in Chapel

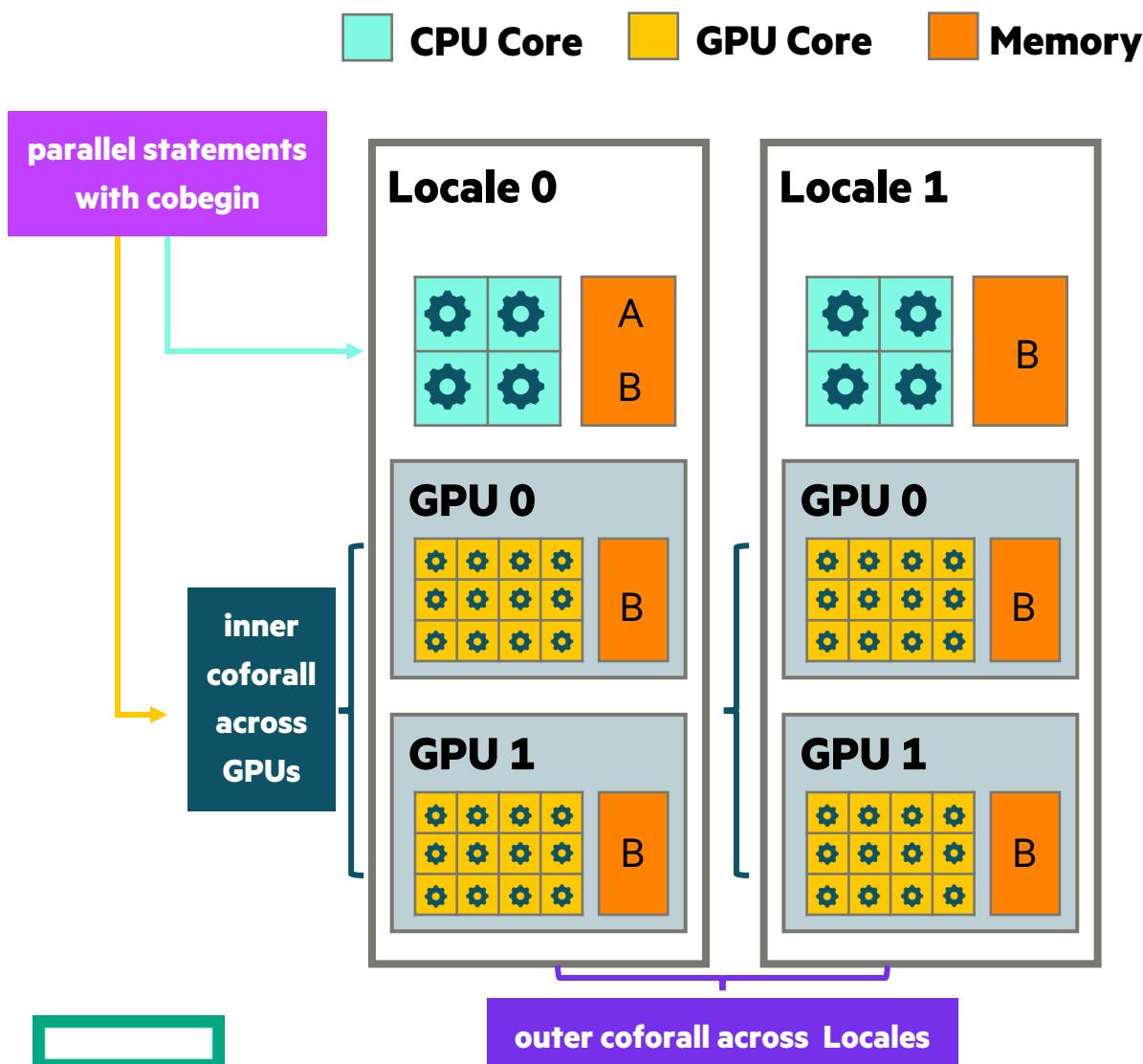
- In Chapel, we represent GPUs as *sub-locales*
 - Each top-level locale may have an array of locales called ‘gpus’
 - We can then target them using Chapel’s traditional features for parallelism + locality

```
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);
```

And much, much more...

This talk has covered just a small subset of Chapel's features... there's much more!

Additional parallel features:

- atomic and sync types for synchronizing between tasks
- additional ways to create tasks and parallel loops
- reduction and scan operations

Traditional language features:

- object-oriented features
- iterators
- generics, polymorphism, overloading
- default arguments, keyword-based argument passing
- modules for namespacing
- interoperability
- etc.



Wrap-up



Summary

Chapel is unique among programming languages

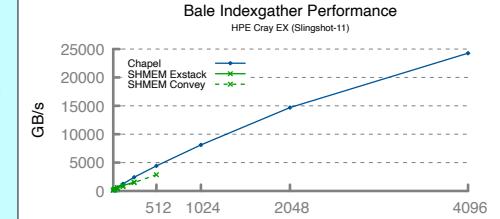
- supports first-class concepts for parallelism and locality
- ports and scales from laptops to supercomputers
- supports clean, concise code relative to conventional approaches
- supports GPUs in a vendor-neutral manner

```
use BlockDist;

config const n = 10,
      m = 4;

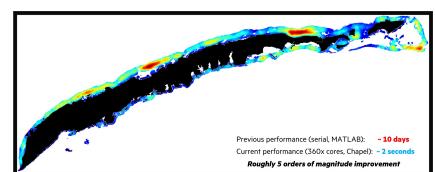
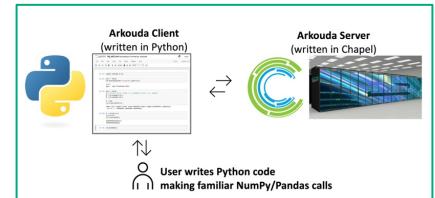
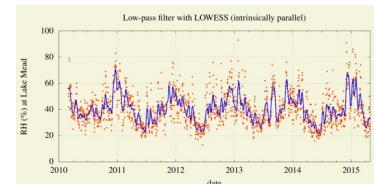
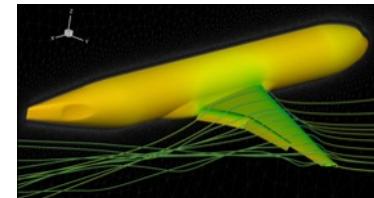
const SrcInds = blockDist.createDomain(0..<n>,
                                       DstInds = blockDist.createDomain(0..<m>);

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 interactive HPC



Three Ways to Get Started with Chapel

Listen to [last week's episode](#) of HPE's "Technology Now" podcast



Read interviews from the aforementioned [7 Questions for Chapel Users](#) series

A screenshot of a webpage titled "7 Questions for Chapel Users" from the Chapel Language Blog. The page features a grid of seven interview snippets, each with a small profile picture of the interviewee, their name, and a brief quote. The interviews cover topics such as graph analysis, scientific computing, and GPU optimization. The layout is clean with a white background and a grid-based structure for the interviews.

Visit the [Get Involved](#) page on our website

A screenshot of the Chapel project's "GET INVOLVED" page. The page has a green header bar with the Chapel logo and navigation links for DOWNLOAD, DOCS, LEARN, RESOURCES, COMMUNITY, GET INVOLVED, and BLOG. Below the header, there's a section titled "Interested in helping the Chapel project grow?" with a list of "Easy Actions" and "Use Chapel". The "Easy Actions" list includes items like "Star the project on GitHub", "Follow us on social media", and "Leave a review for our software". The "Use Chapel" section lists actions like "Download and install Chapel", "Compile and run sample code", and "Reach out to a Chapel developer". The overall design is modern with a white background and a mix of green and black text.

Ways to engage with the Chapel Community

Synchronous Community Events

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

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

Asynchronous Communications

- [Chapel Blog](#), typically ~2 articles per month
- [Community Newsletter](#), quarterly
- [Announcement Emails](#), around big events

Ways to Use Chapel

GET STARTED

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

(from the footer of chapel-lang.org)

Chapel Website

CHAPEL

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.

FAS
Chapel is a compact language generating efficient native code that meets or beats the performance of languages like C++.

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 under the MIT license. Built by a growing community of developers.

CHAMPS

World-class multiphysics simulation

Written by students and post-docs in Eric Laurendeau's lab at Polytechnique Montréal. 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 being overwhelmed by it with everything at once.

- Tess Hayes

WHAT'S NEW?

Technology Now Chapel Featured on 'Technology Now' Podcast

on October 30, 2025

This week's 'Technology Now' podcast features a conversation about Chapel with Brad Chamberlain.

[CONTINUE READING](#)

10 Myths Part 7

By Brad Chamberlain on October 15, 2025

The seventh archival post from the 2012 IEEE TCSC blog series with a current reflection on it.

[CONTINUE READING](#)

ChapelCon '25 Program Released!

on September 23, 2025

ChapelCon '25 is just two weeks away, and the program is live! Check out the webpage for the full schedule.

[CONTINUE READING](#)

v2.6 Announcing Chapel 2.6!

By David Longnecker, Jade Abraham, Lydia Duncan, Daniel Fedorin, Ben Harshbarger, Brad Chamberlain on September 18, 2025

Highlights from the September 2025 release of Chapel 2.6.

[CONTINUE READING](#)

7 Questions for Marjan Asgari: Optimizing Hydrological Models with Chapel

By Engin Kayakoglu, Brad Chamberlain on September 15, 2025

An interview with Marjan Asgari, a hydrologist at the University of Waterloo, who is using Chapel to optimize hydrological models.

[CONTINUE READING](#)

FOLLOW US

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

GET IN TOUCH

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

GET STARTED

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

chapel-lang.org

57



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

