HPE

Productive Parallel Programming from Laptops to Supercomputers with Chapel

Brad Chamberlain

SeaGL 2025
November 7, 2025

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.

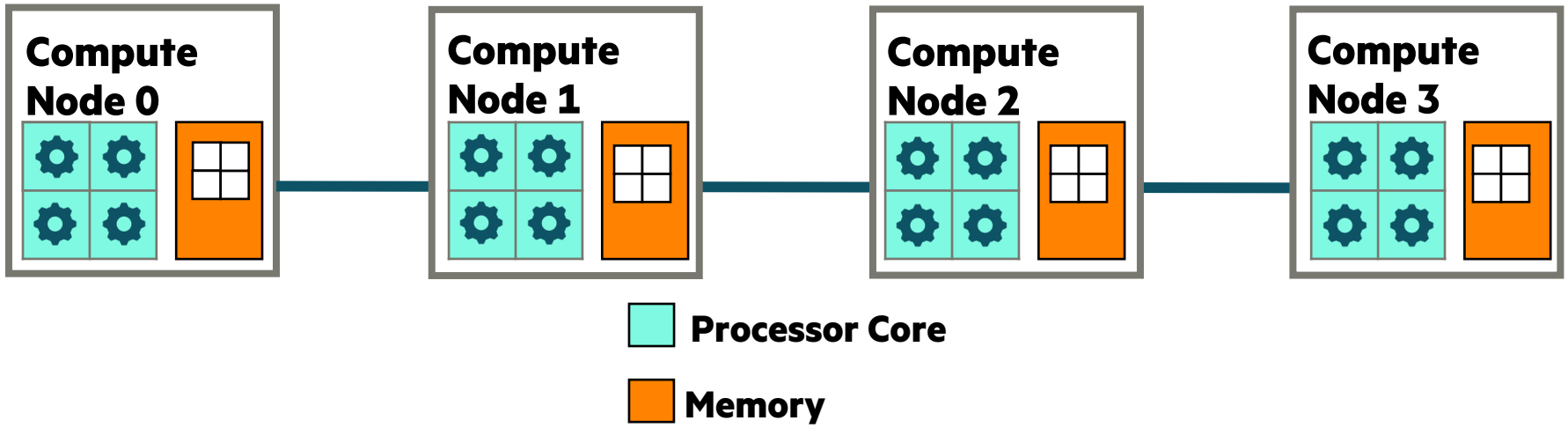
What is [Scalable] Parallel Computing?

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

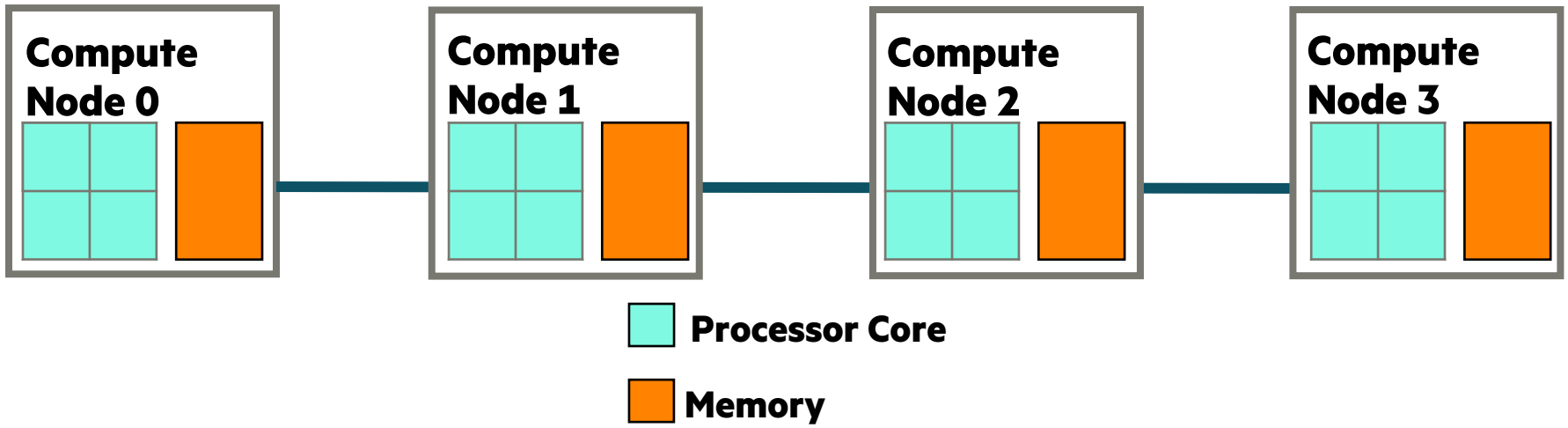
Parallel Computing has become Ubiquitous

Parallel computing, historically:

- supercomputers
- commodity clusters

Additional, ubiquitous parallelism today:

- multicore processors
- cloud computing
- GPUs



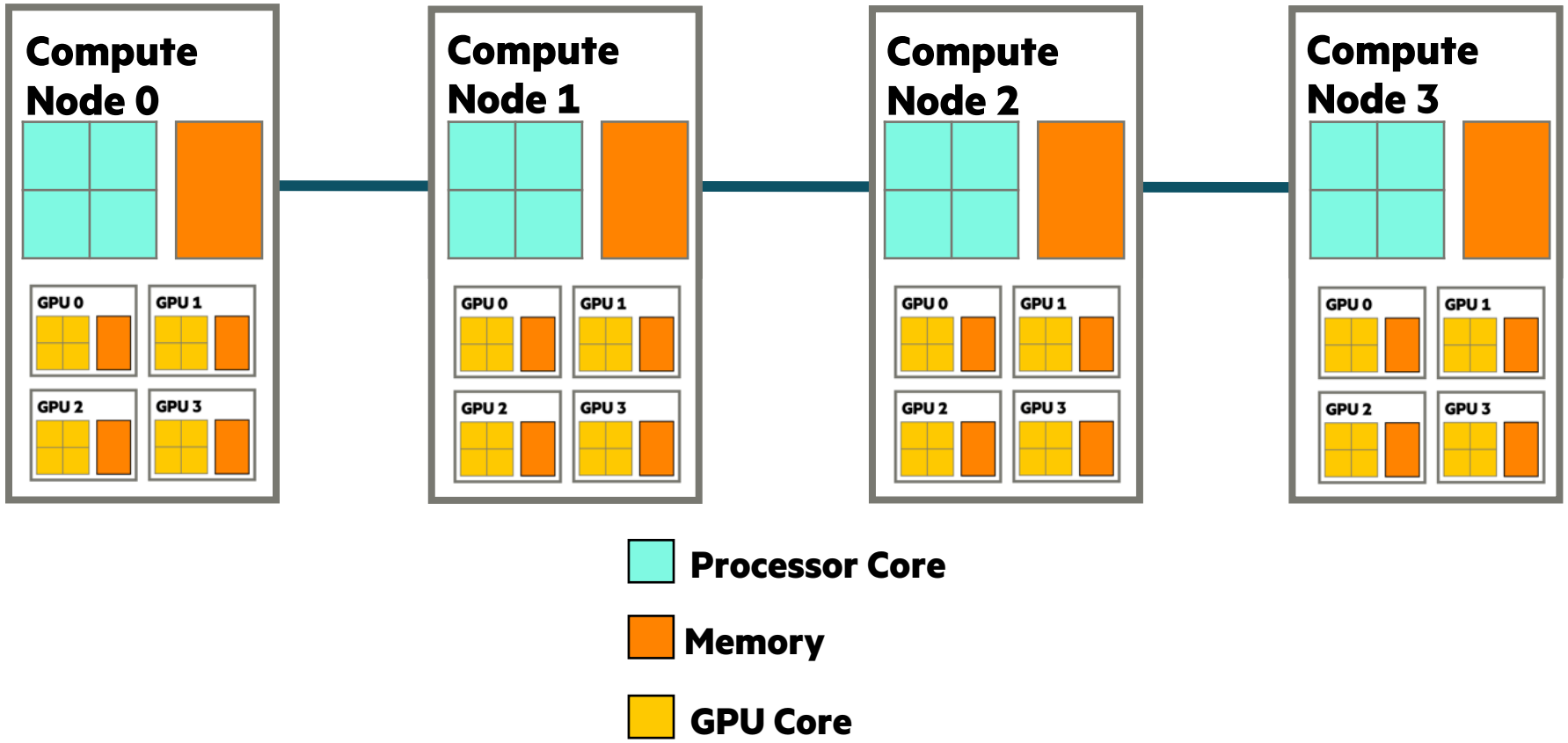
Parallel Computing has become Ubiquitous

Parallel computing, historically:

- supercomputers
- commodity clusters

Additional, ubiquitous parallelism today:

- multicore processors
- cloud computing
- GPUs



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

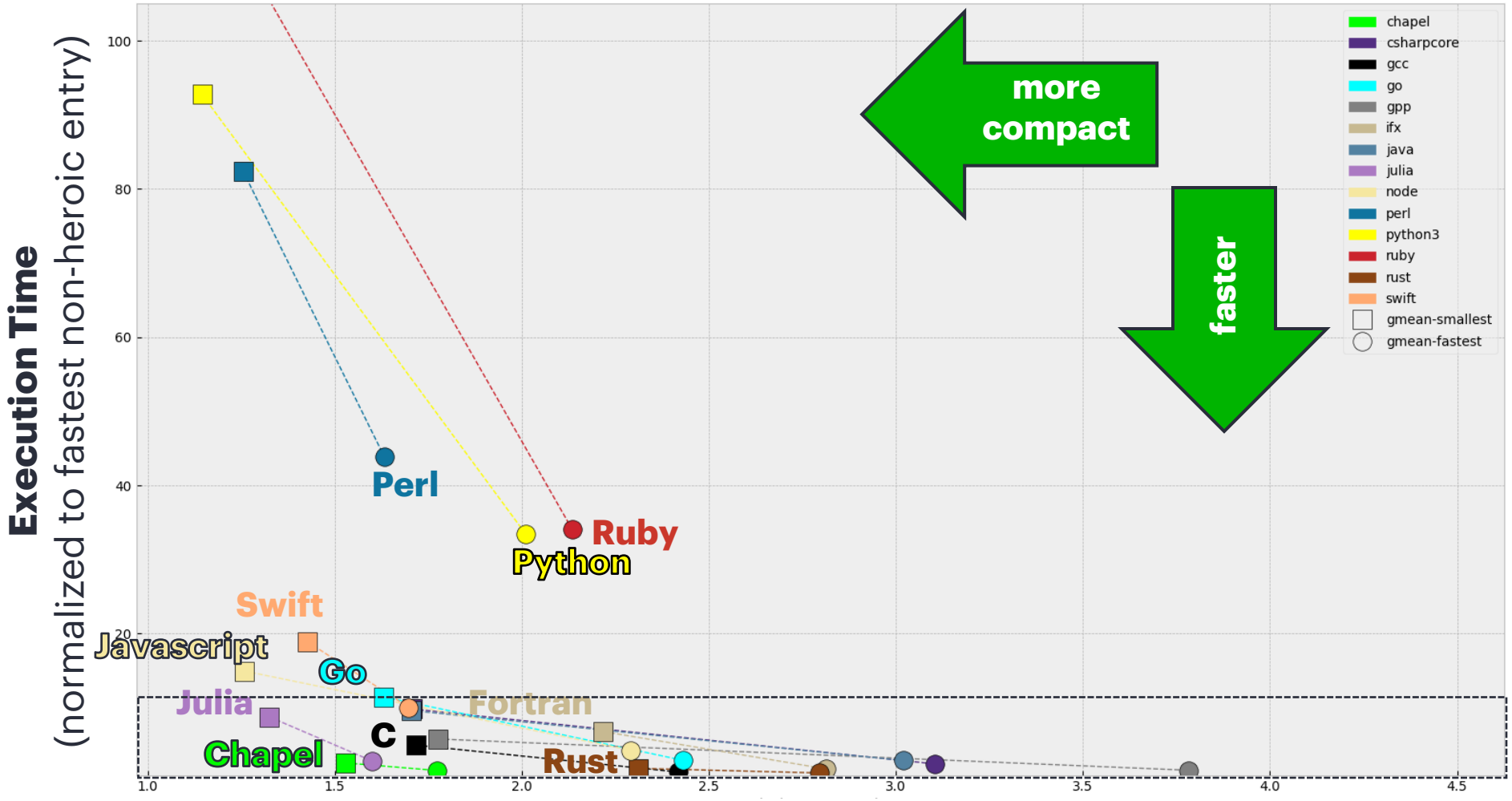
…**GPU-ready** as CUDA / HIP / OpenMP / Kokkos / OpenCL / OpenACC / …

…**portable** as C

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

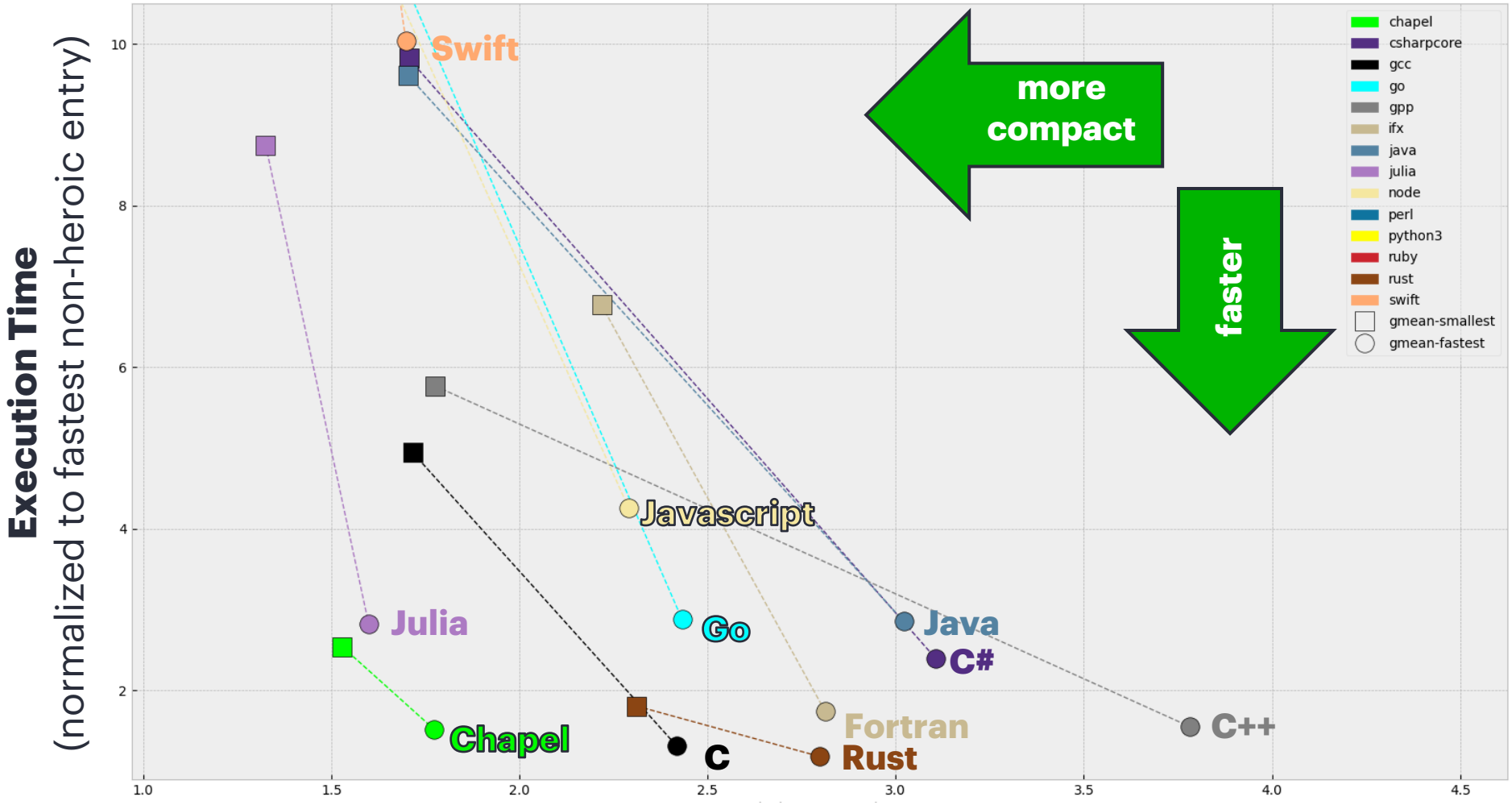
This is our motivation for Chapel

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

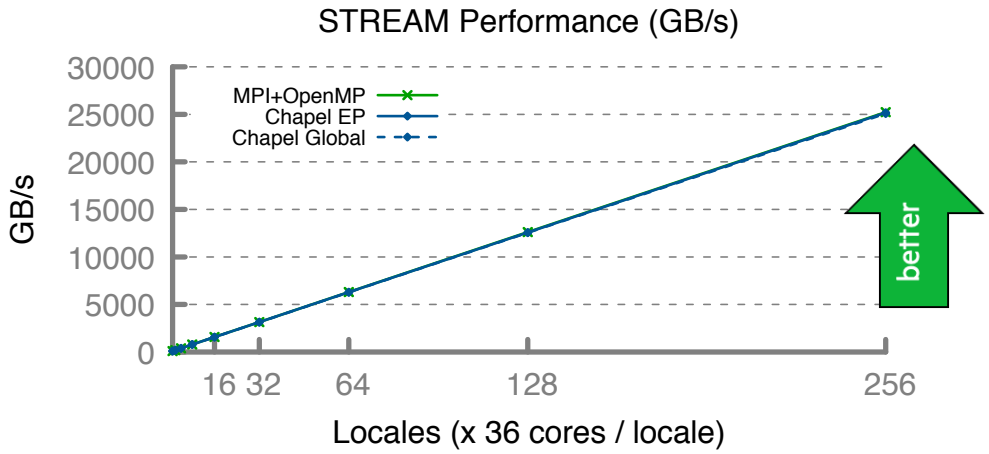
```
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

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

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );
```

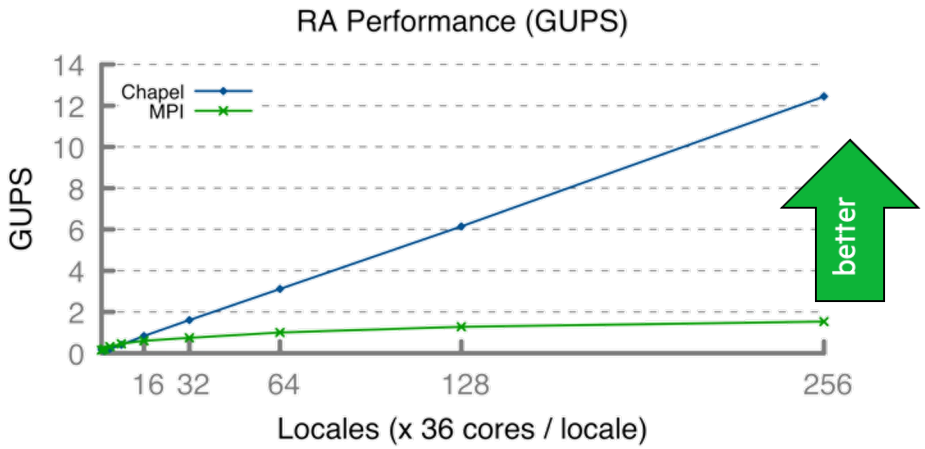
```
use BlockDist;

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

B = 2.0;
C = 1.0;

A = B + alpha * C;
```

HPCC RA: MPI KERNEL

```
/* Perform updates to main table.  The scalar equivalent is:
 *
 *   for (i=0; i<NUPDATE; i++){
 *     Ran = (Ran << 1) ^ ((s64Int) Ran < 0) ? POLY: 0);
 *     Table[Ran & (TABSIZE-1)] ^= Ran;
 *   }
 */

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
while (i < SendCnt) {
  /* receive messages */
  do {
    MPI_Test(&inreq, &have_done, &status);
    if (have_done) {
      if (status.MPI_TAG == UPDATE_TAG) {
        MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
        bufferBase = 0;
        for (j=0; j < recvUpdates; j ++) {
          inmsg = LocalRecvBuffer[bufferBase+j];
          LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                        tparams.GlobalStartMyProc;
          HPCC_Table[LocalOffset] ^= inmsg;
        }
      } else if (status.MPI_TAG == FINISHED_TAG) {
        NumberReceiving--;
      } else
        MPI_Abort( MPI_COMM_WORLD, -1 );
      MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    }
  } while (have_done && NumberReceiving > 0);
  if (pendingUpdates < maxPendingUpdates) {
    Ran = (Ran << 1) ^ ((s64Int) Ran < ZERO64B ? POLY : ZERO64B);
    GlobalOffset = Ran & (tparams.TableSize-1);
    if ( GlobalOffset < tparams.Top)
      WhichPe = ( GlobalOffset / (tparams.MinLocalTableSize + 1) );
    else
      WhichPe = ( (GlobalOffset - tparams.Remainder) /
                  tparams.MinLocalTableSize );
    if (WhichPe == tparams.MyProc) {
      LocalOffset = (Ran & (tparams.TableSize - 1)) -
                    tparams.GlobalStartMyProc;
      HPCC_Table[LocalOffset] ^= Ran;
    } else {
      HPCC_InsertUpdate(Ran, WhichPe, Buckets);
      pendingUpdates++;
    }
    i++;
  }
  else {
    MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
    if (have_done) {
      outreq = MPI_REQUEST_NULL;
      pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                           &peUpdates);
      MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
                UPDATE_TAG, MPI_COMM_WORLD, &outreq);
      pendingUpdates -= peUpdates;
    }
  }
}
/* send remaining updates in buckets */
while (pendingUpdates > 0) {
  /* receive messages */
  do {
    MPI_Test(&inreq, &have_done, &status);
    if (have_done) {
      if (status.MPI_TAG == UPDATE_TAG) {
        MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
        bufferBase = 0;
        for (j=0; j < recvUpdates; j ++) {
          inmsg = LocalRecvBuffer[bufferBase+j];
          LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                        tparams.GlobalStartMyProc;
          HPCC_Table[LocalOffset] ^= inmsg;
        }
      } else if (status.MPI_TAG == FINISHED_TAG) {
        /* we got a done message. Thanks for playing... */
        NumberReceiving--;
      } else {
        MPI_Abort( MPI_COMM_WORLD, -1 );
      }
      MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    }
  } while (have_done && NumberReceiving > 0);
  MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
  if (have_done) {
    outreq = MPI_REQUEST_NULL;
    pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                         &peUpdates);
    MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
              UPDATE_TAG, MPI_COMM_WORLD, &outreq);
    pendingUpdates -= peUpdates;
  }
}
MPI_Waitall( tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);
```

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





Outline

Background & Motivation

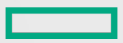
Chapel by Example: Bale Index Gather

Chapel Applications

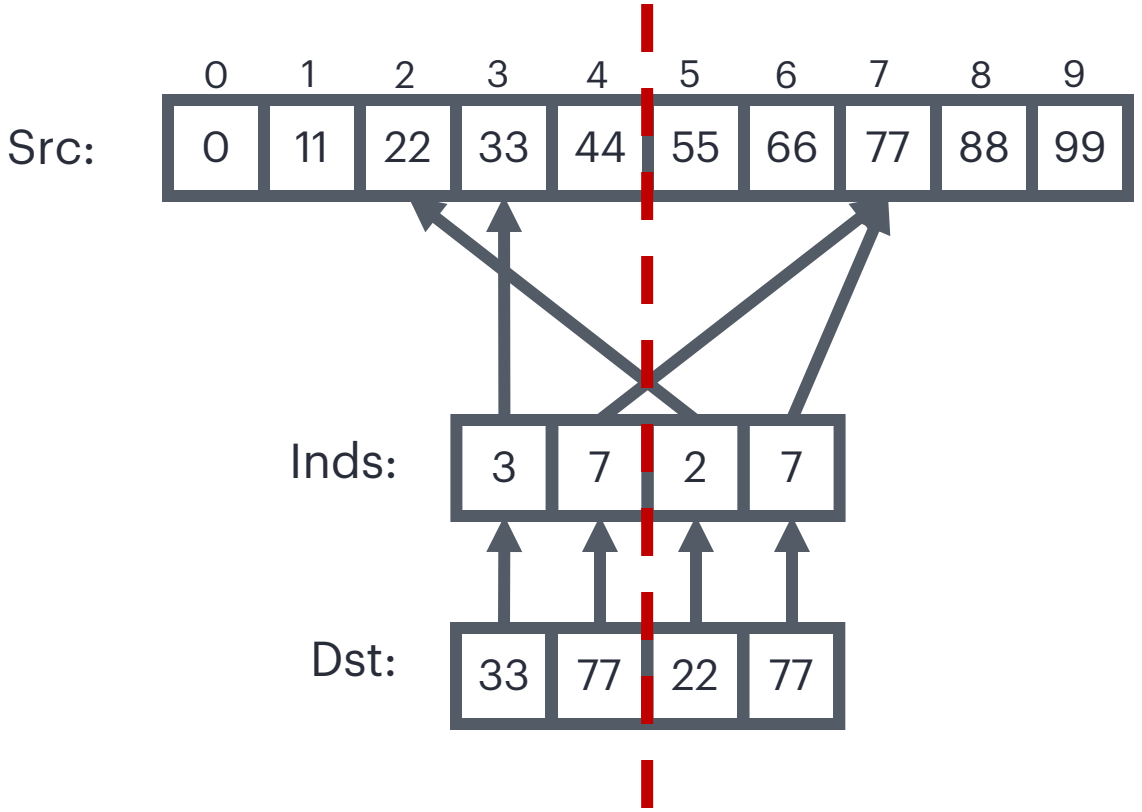
“Low-level” Features for Parallelism & Locality (time permitting)

Wrap-up

Chapel by Example: Bale Index Gather



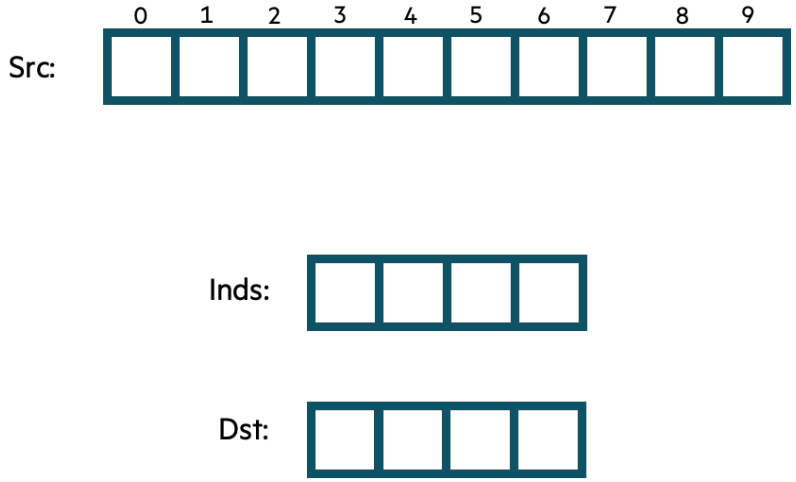
Bale Index Gather (IG): In Pictures



Bale IG in Chapel: Scalar and Array Declarations

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



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

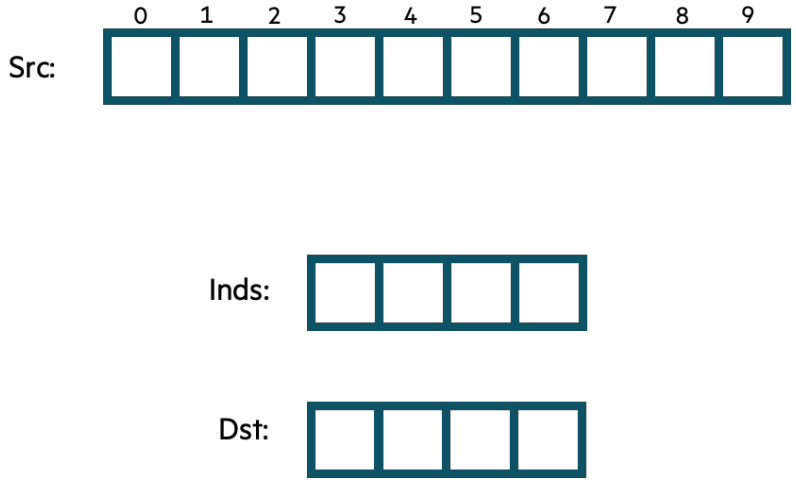
```
$
```



Bale IG in Chapel: Compiling

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



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

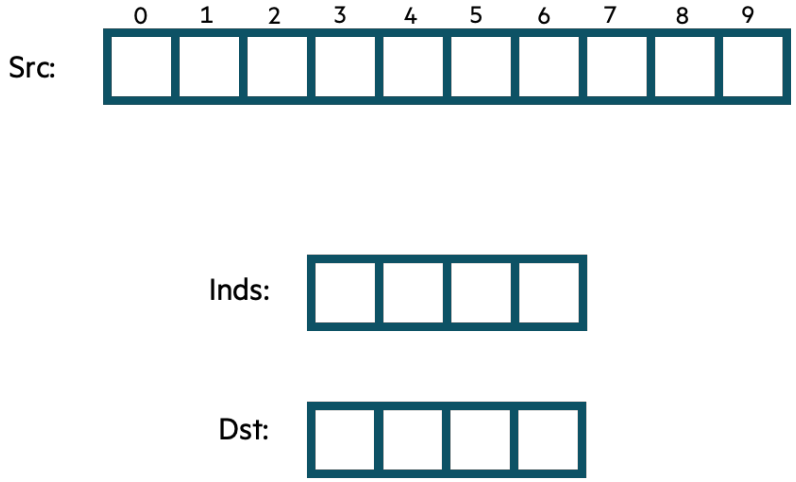
```
$ chpl bale-ig.chpl
$
```

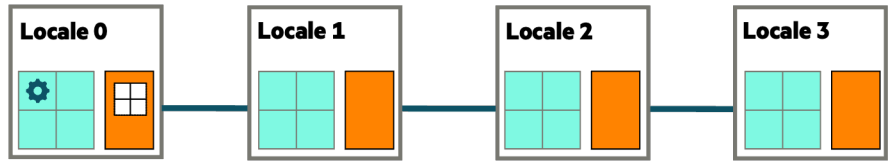


Bale IG in Chapel: Executing

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



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

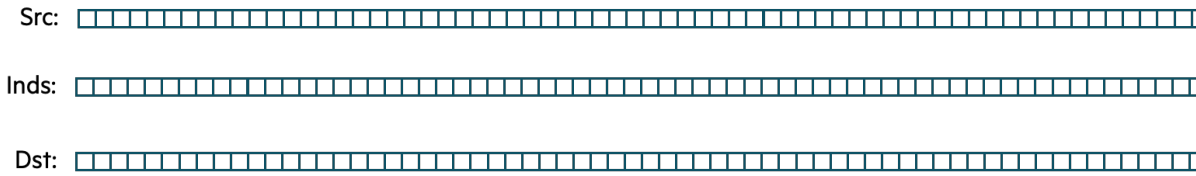
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



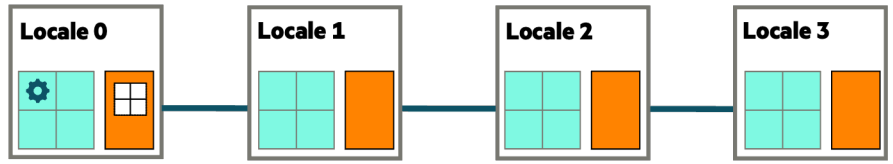


Bale IG in Chapel: Executing, Overriding Configs

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



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

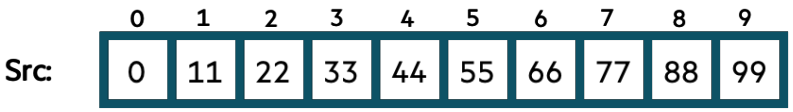


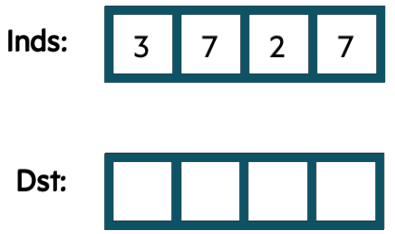

```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000
$
```



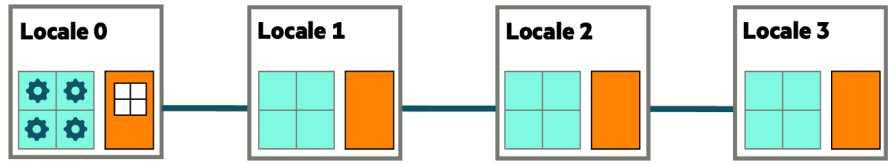
Bale IG in Chapel: Array Initialization

```
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);
```



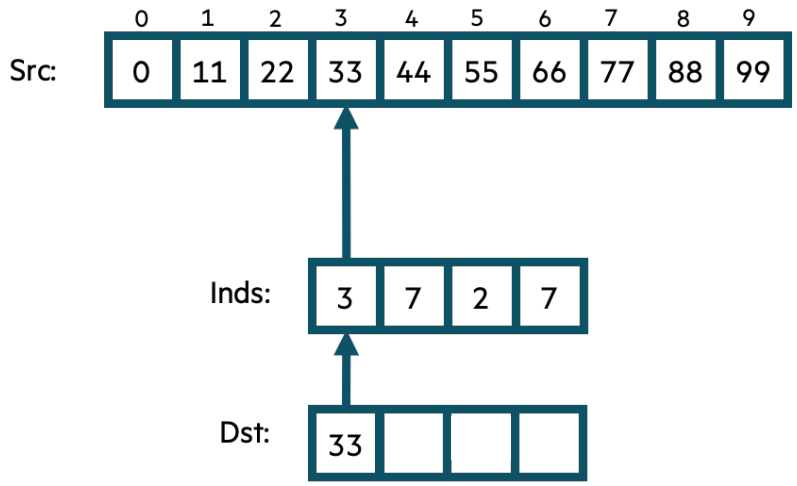



```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

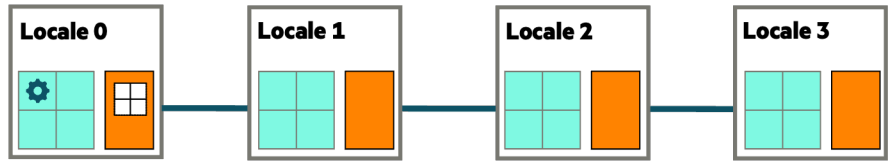


Bale IG in Chapel: Serial Version

```
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]];
```



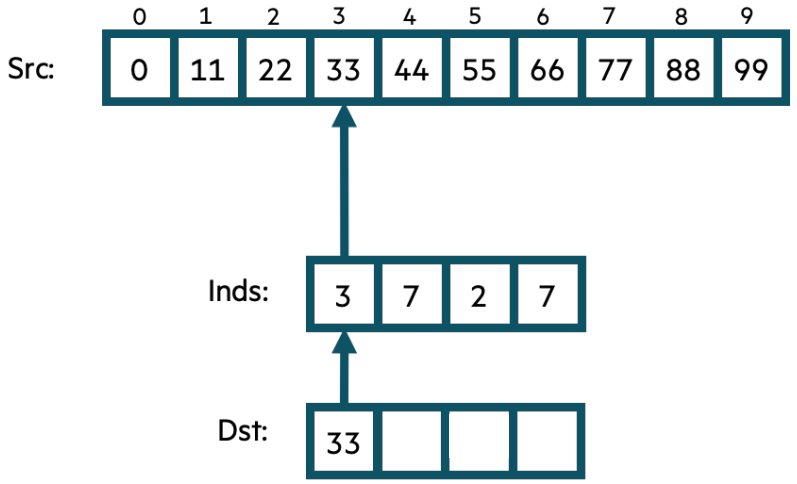

```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

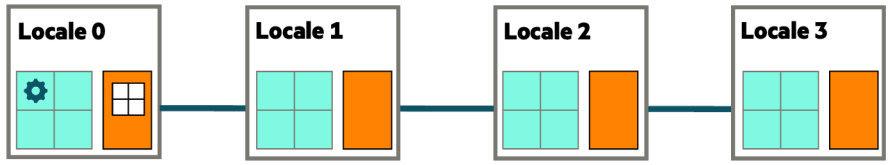


Bale IG in Chapel: Serial, Zippered Version

```
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
$
```

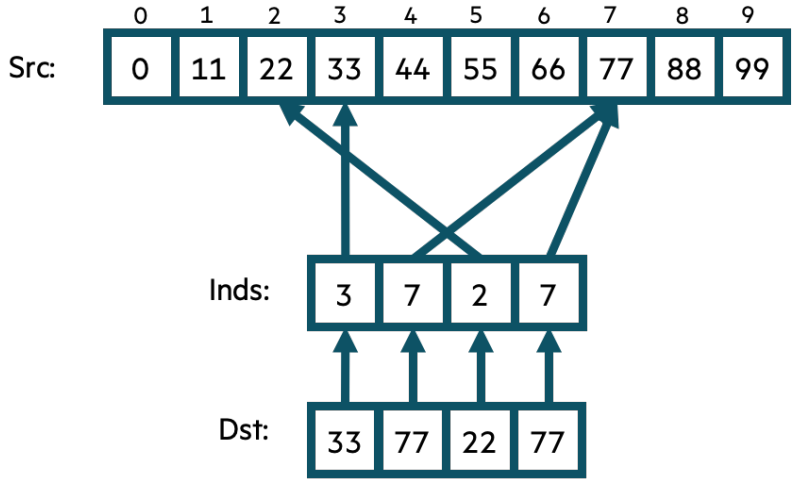


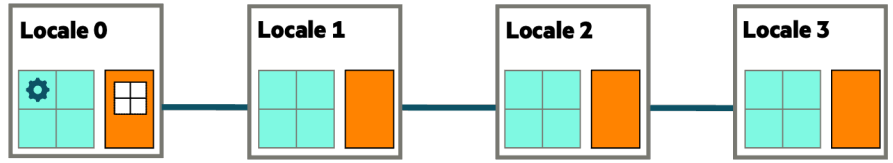


Bale IG in Chapel: Parallel, Zippered Version (vectorized)

```
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
$
```

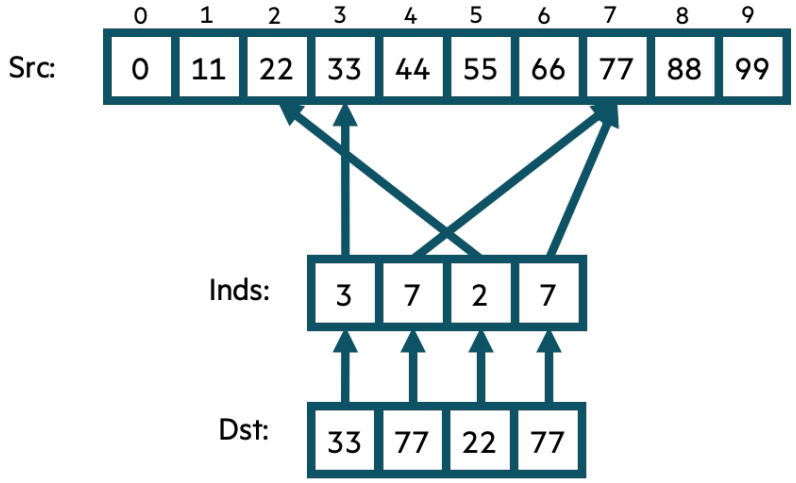


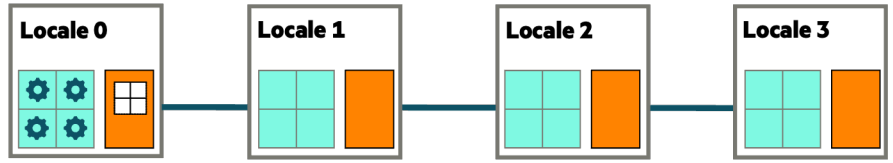


Bale IG in Chapel: Parallel, Zippered Version (multicore)

```
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];
```

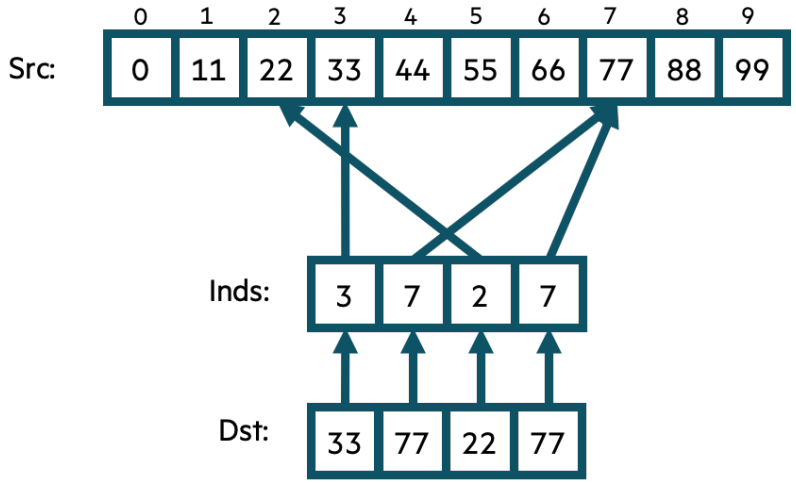
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



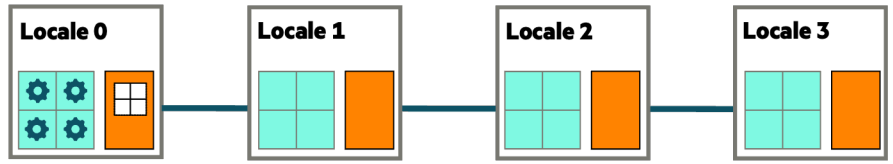


Bale IG in Chapel: Parallel Promoted Version (equivalent to previous version)

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



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

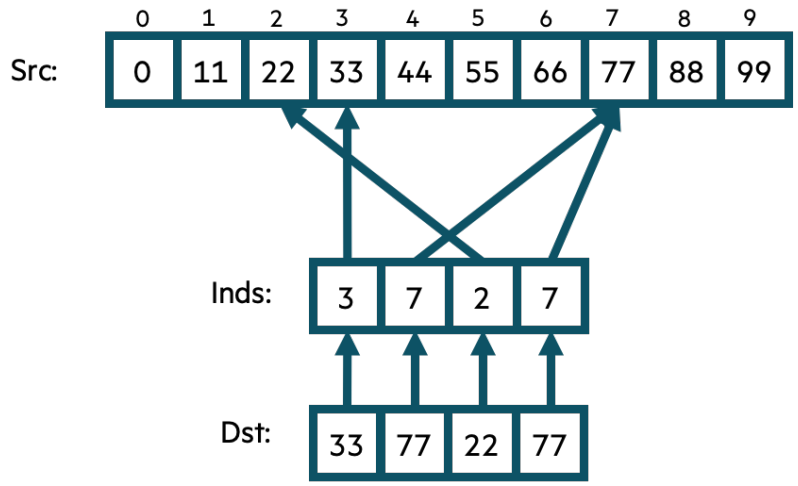



```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

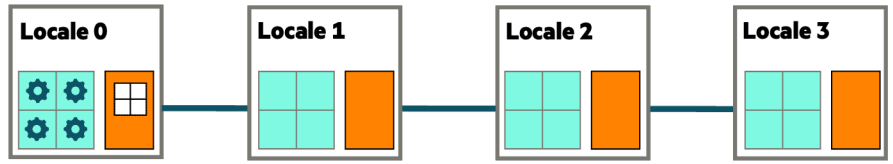


Bale IG in Chapel: Parallel, Zippered Version (Multicore, again)

```
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];
```

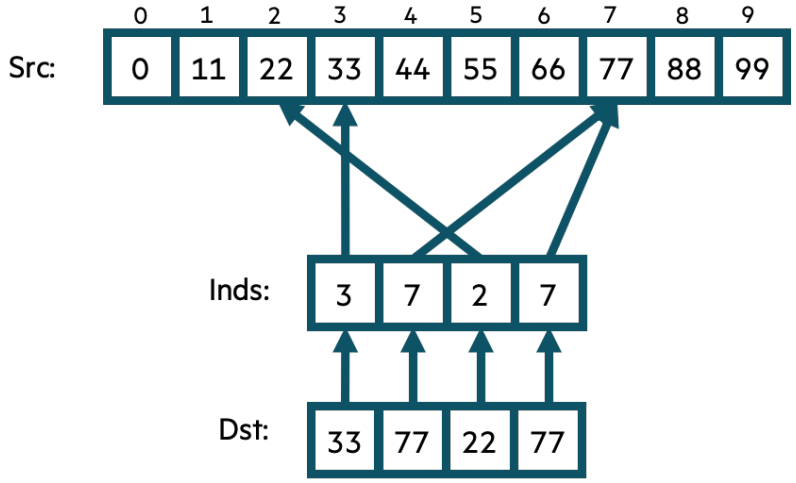



```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

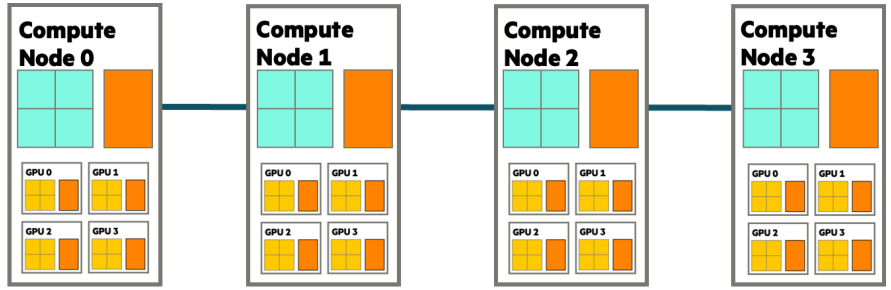


Bale IG in Chapel: Parallel, Zippered Version (GPU)

```
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];
}
```

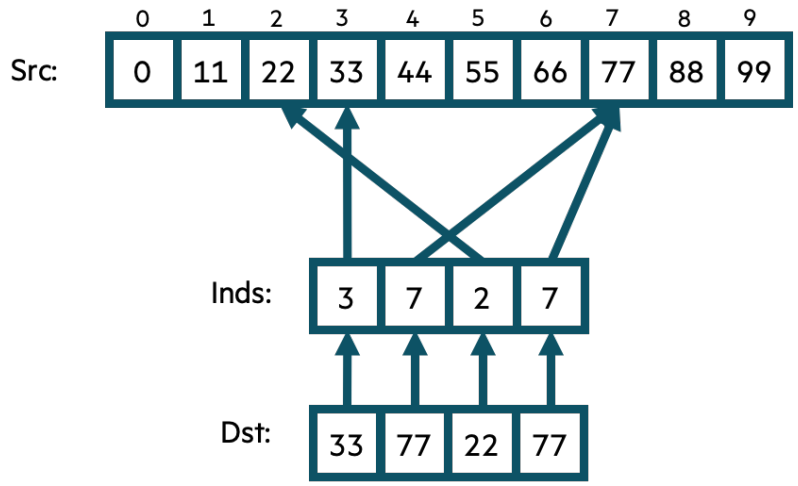



```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

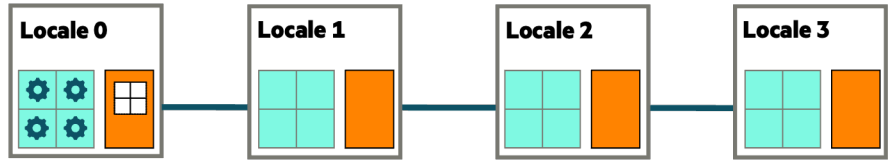


Bale IG in Chapel: Parallel, Zippered Version (Multicore, again)

```
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];
```

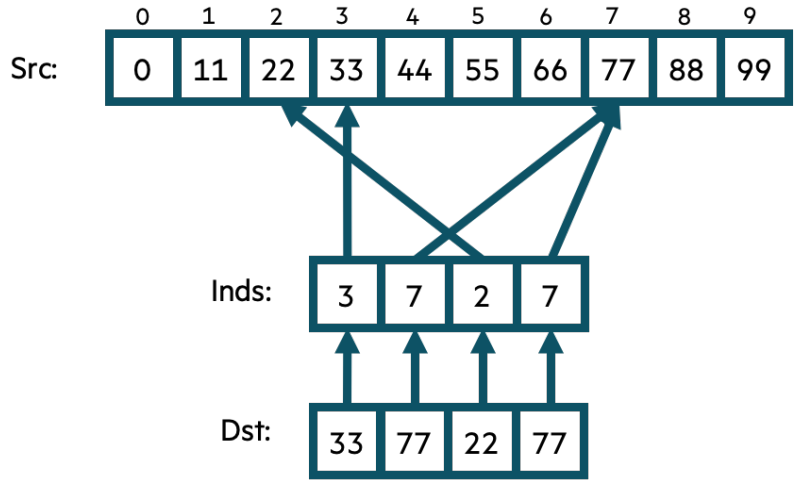



```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

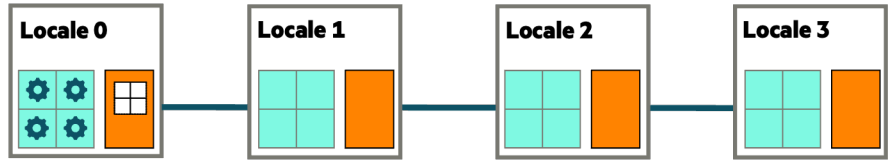


Bale IG in Chapel: Parallel , Zippered Version with Named Domains (Multicore)

```
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];
```

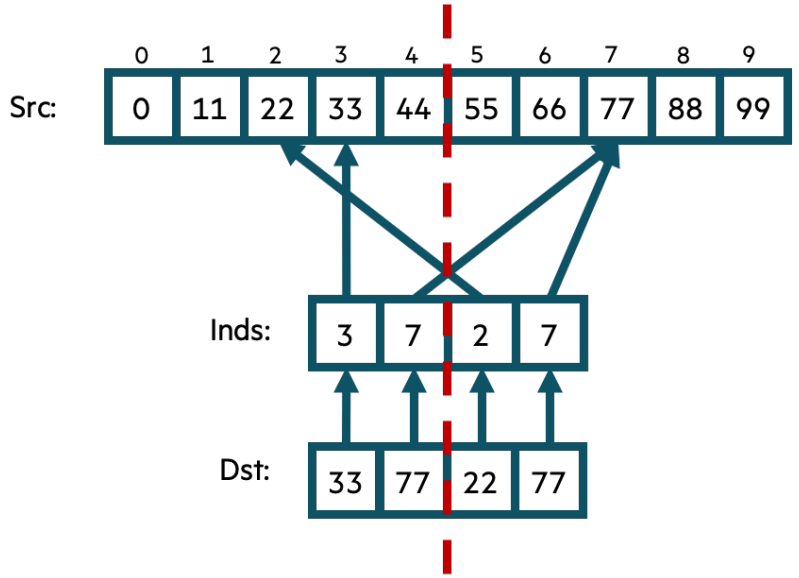



```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

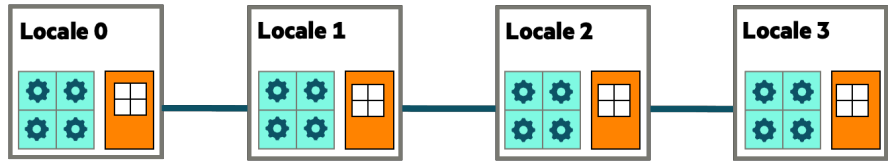


Bale IG in Chapel: Distributed Parallel Version

```
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];
```



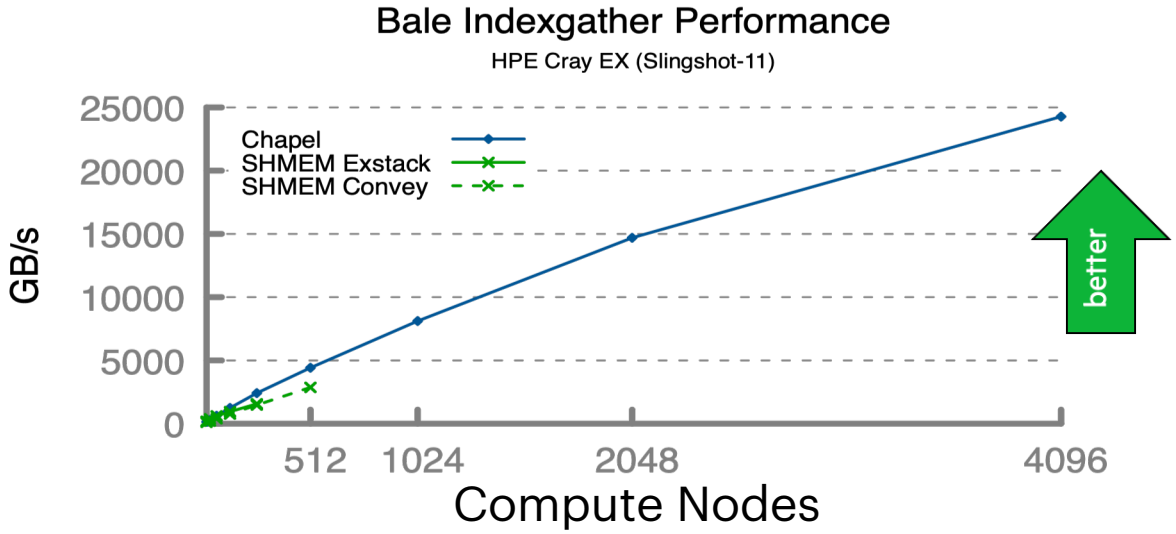

```
$ chpl bale-ig.chpl
$ ./bale-ig --n=… --m=… -nl 4
$
```

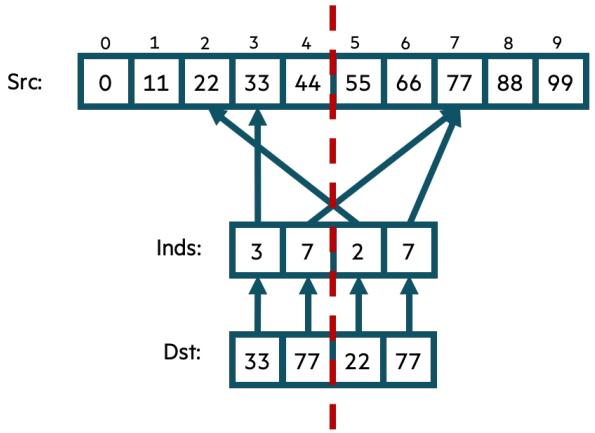


Bale IG in Chapel: Distributed Parallel Version on HPE Cray EX (Slingshot-11)

```
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

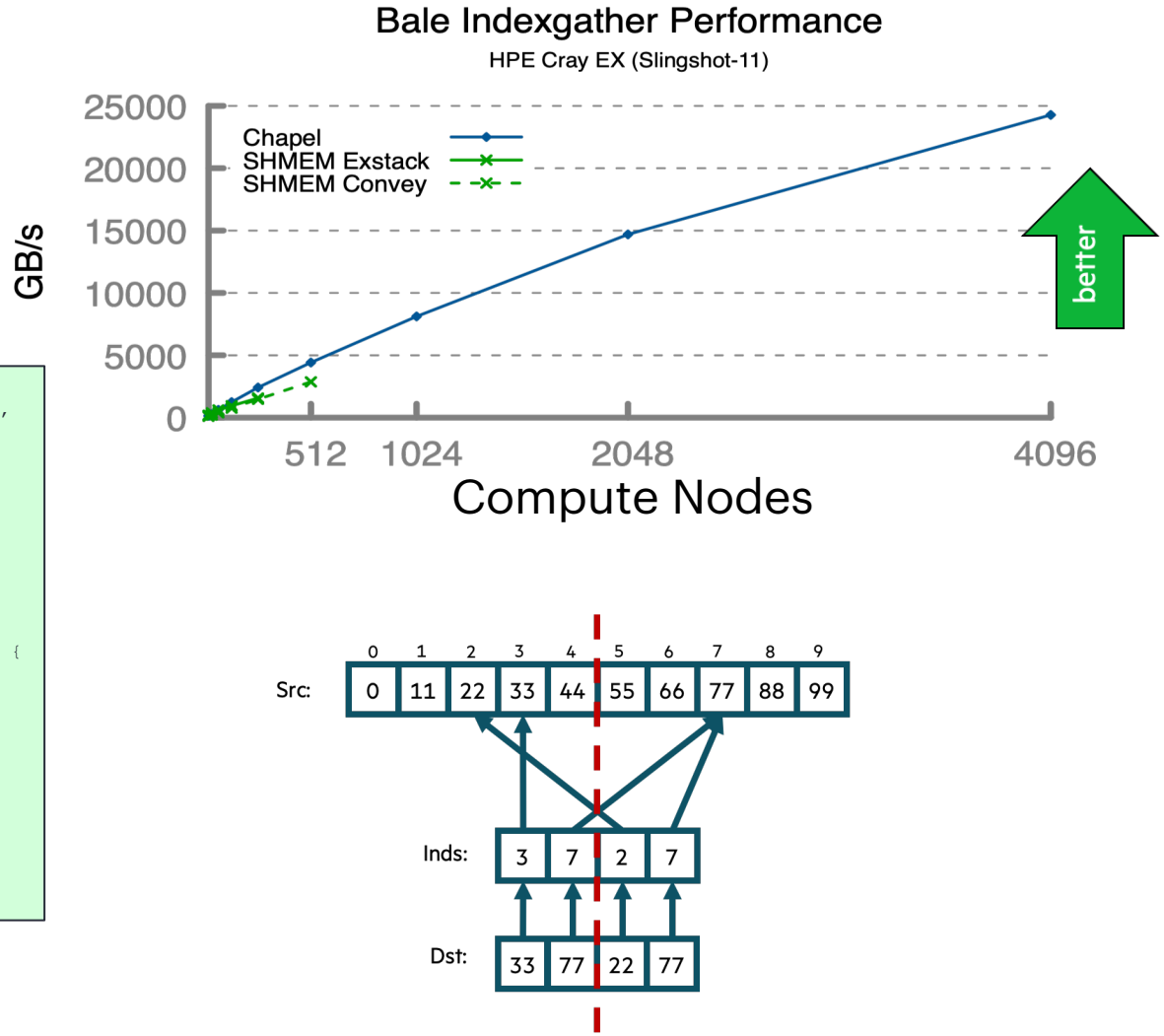
```
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

SHMEM (Exstack version)

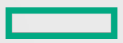
```
i=0;
while( exstack_proceed(ex, (i==l_num_req)) ) {
  i0 = i;
  while(i < l_num_req) {
    l_indx = pckindx[i] >> 16;
    pe  = pckindx[i] & 0xffff;
    if(!exstack_push(ex, &l_indx, pe))
      break;
    i++;
  }

  exstack_exchange(ex);

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

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

SHMEM (Conveyors version)

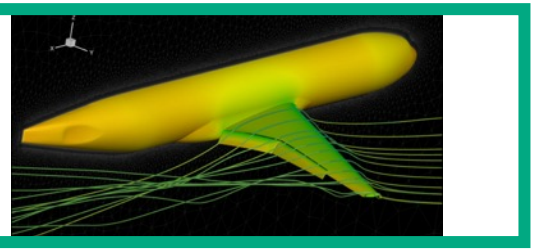
```
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
       more | convey_advance(replies, !more)) {

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

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

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



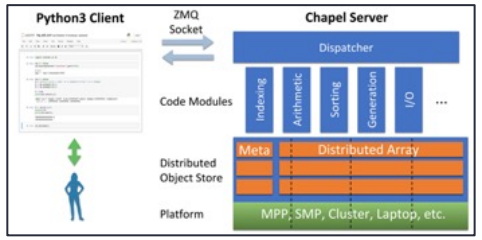
Chapel Applications



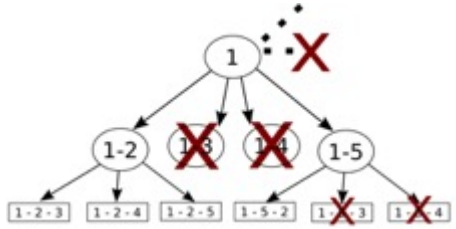
Applications of Chapel



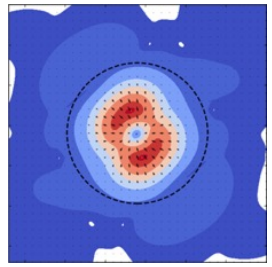
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



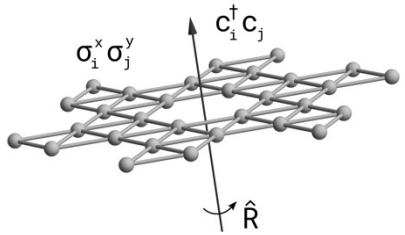
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



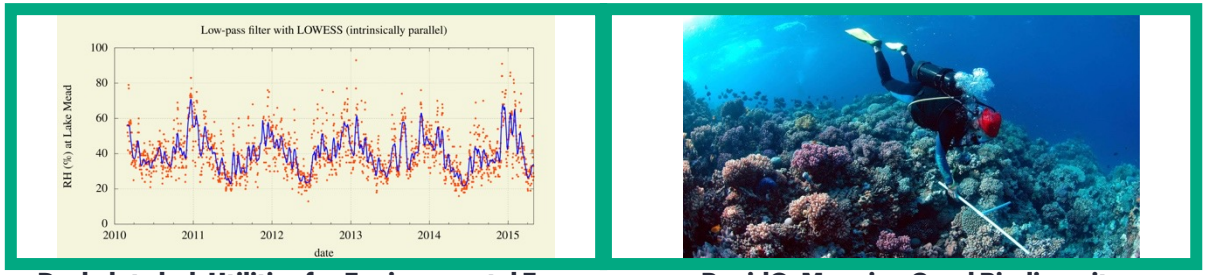
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

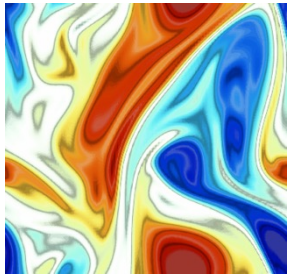


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

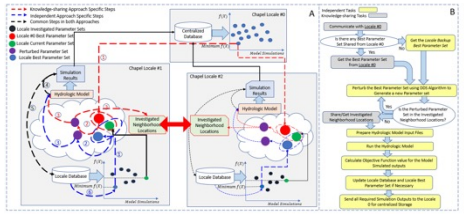


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

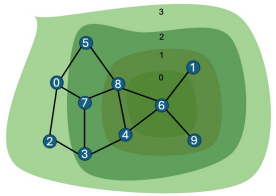
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



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

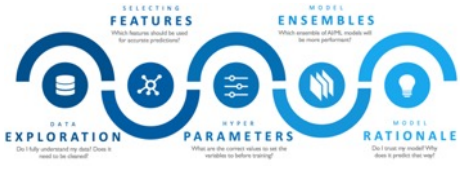


Chapel-based Hydrological Model Calibration
Marjan Asgari, et al.
University of Guelph



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

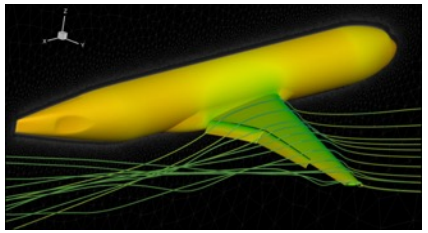
[images provided by their respective teams and used with permission]

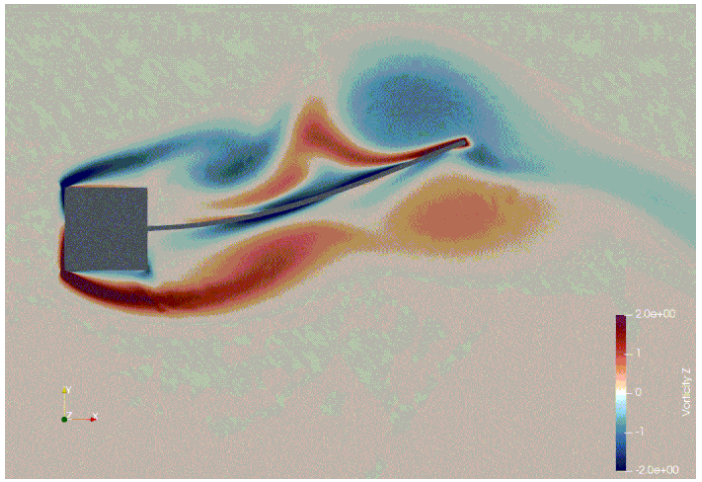
CHAMPS Summary

What is it?

— 3D unstructured CFD framework for airplane simulation

— ~100+k lines of Chapel written since 2019

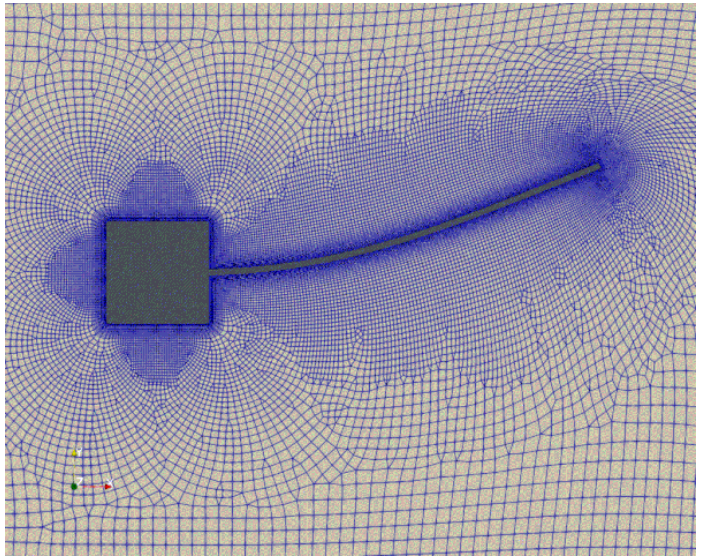




Who wrote it?

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

POLYTECHNIQUE MONTRÉAL



Why Chapel?

— performance and scalability competitive with MPI + C++

— students found it far more productive to use

— enabled them to compete with more established CFD centers

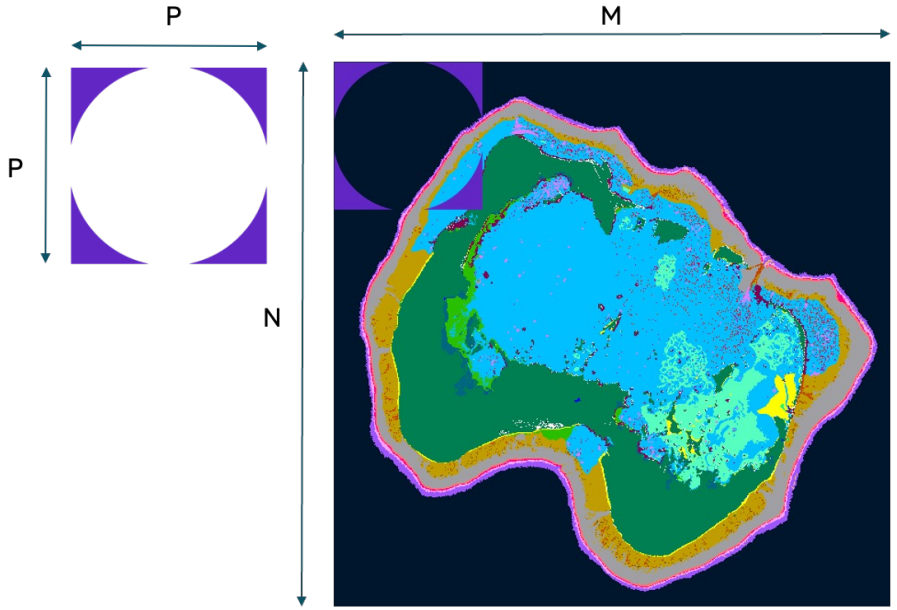
[images provided by the CHAMPS team and used with permission]

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

NCAR Operated by UCAR

[C]Worthy

CORAL REEF ALLIANCE

Why Chapel?

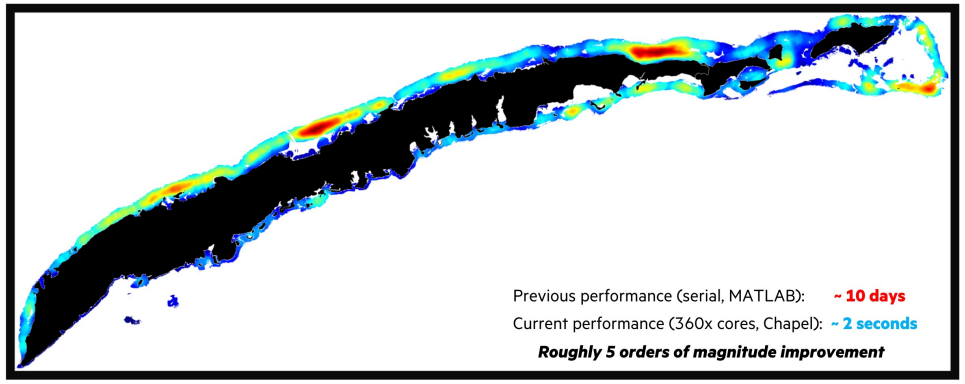
— easy transition from Matlab, which they had been using

— massive performance improvement:

previous ~10-day run finished in ~2 seconds using 360 cores

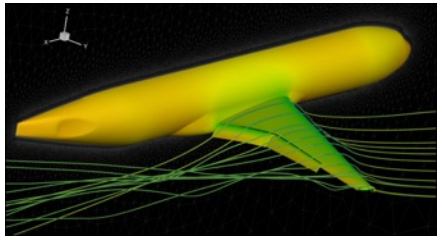
— enabled unanticipated algorithmic improvements

- 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



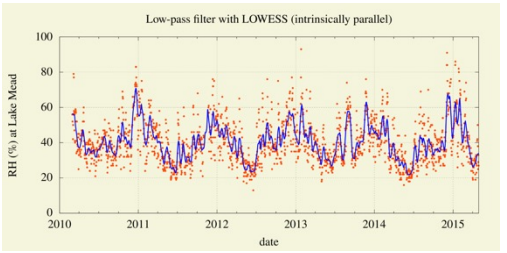
[images provided by Scott Bachman from his CHIUW 2023 talk and used with permission]

Diversity in Chapel Application Scales (Code Size and Systems)



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

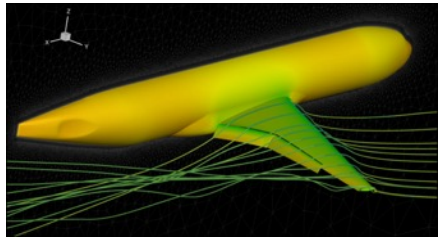
Computation: Coral reef image analysis
Code size: ~320 lines
Systems: Desktops, HPC systems w/ GPUs



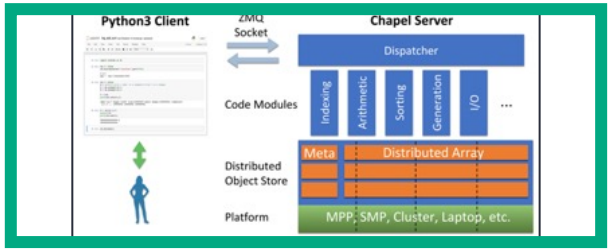
Computation: Atmospheric data analysis
Code size: 5000+ lines
Systems: Desktops, sometimes w/ GPUs

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

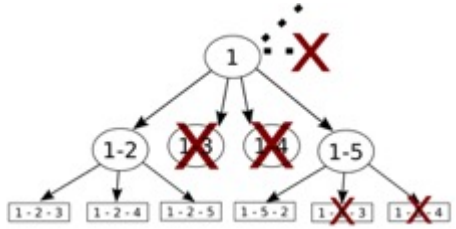
Applications of Chapel



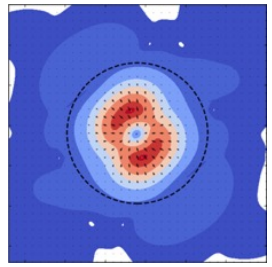
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



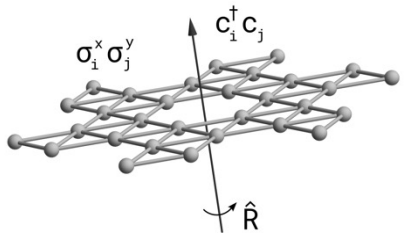
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



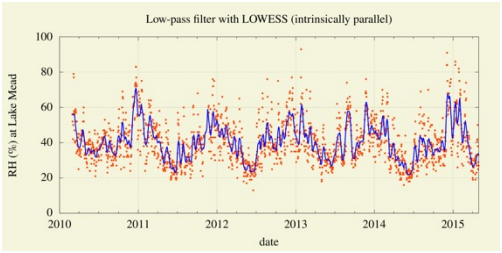
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

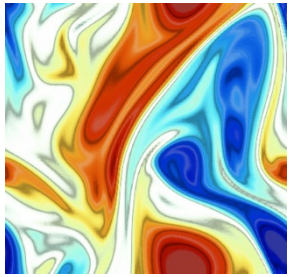


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

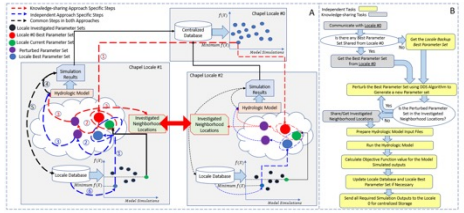


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

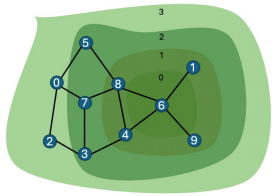
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



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

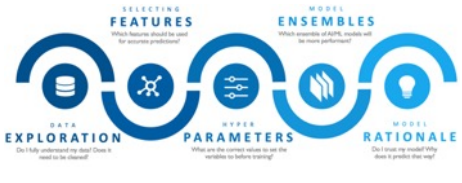


Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



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



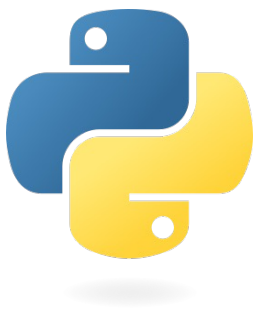
CrayAI HyperParameter Optimization (HPO)
Ben Albrecht et al.
Cray Inc. / HPE

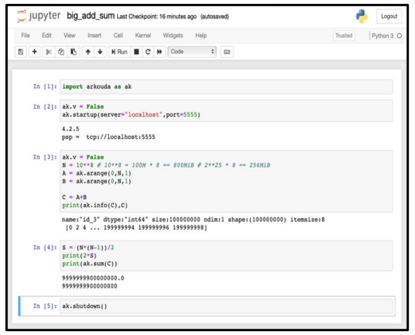
[images provided by their respective teams and used with permission]

What is Arkouda?

Q: “What is Arkouda?”

Arkouda Client
(written in Python)





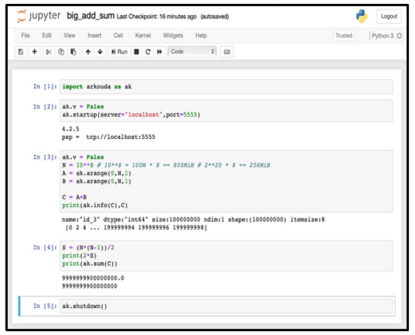
```
import arkouda as ak

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])
```

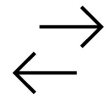
User writes Python code
making familiar NumPy/Pandas calls

What is Arkouda?

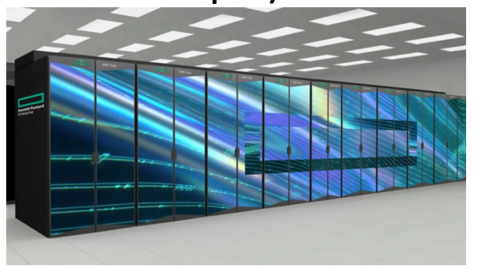
Q: “What is Arkouda?”

Arkouda Client
(written in Python)





Arkouda Server
(written in Chapel)



User writes Python code
making familiar NumPy/Pandas calls

```python
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])
```

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

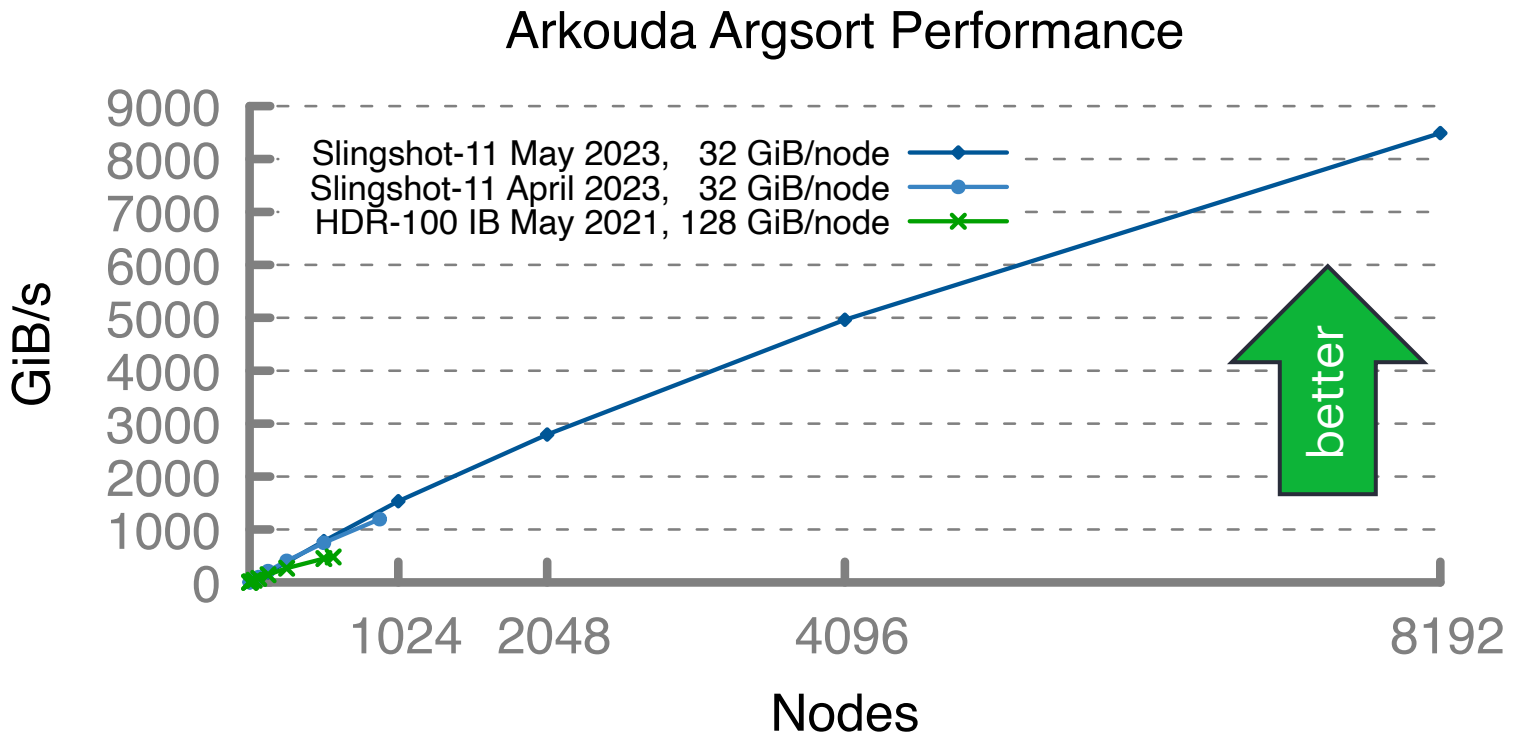
A2: “A framework for driving supercomputers interactively from Python”

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)

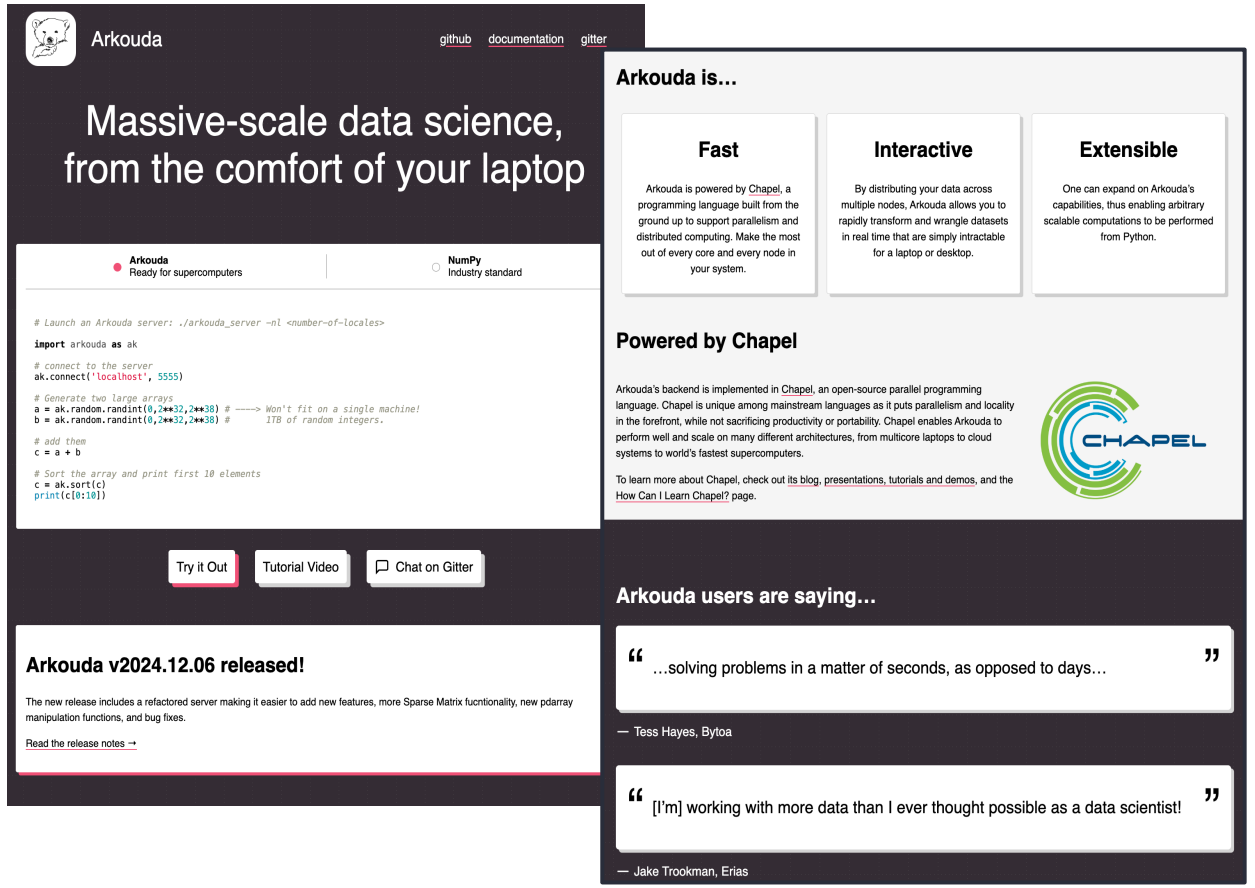
HPE Apollo
- HDR-100 InfiniBand network (100 Gb/s)
- 576 compute nodes
- 72 TiB of 8-byte values
- ~480 GiB/s (~150 seconds)



Implemented using ~100 lines of Chapel

For More Information on Arkouda

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



“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 Kayraklioglu, Brad Chamberlain

7 Questions for Tiago Carneiro and Guillaume Helbecque: Combinatorial Optimization in Chapel

Posted on July 30, 2025.

Tags: User Experiences | Interviews

By: Engin Kayraklioglu, Brad Chamberlain

7 Questions for Marjan Asgari: Optimizing Hydrological Models with Chapel

Posted on September 15, 2025.

Tags: User Experiences | Interviews | Earth Sciences

By: Engin Kayraklioglu, 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.”

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.”

Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts

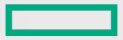
7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

“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 Nelson Luís Dias: Atmospheric Turbulence in Chapel

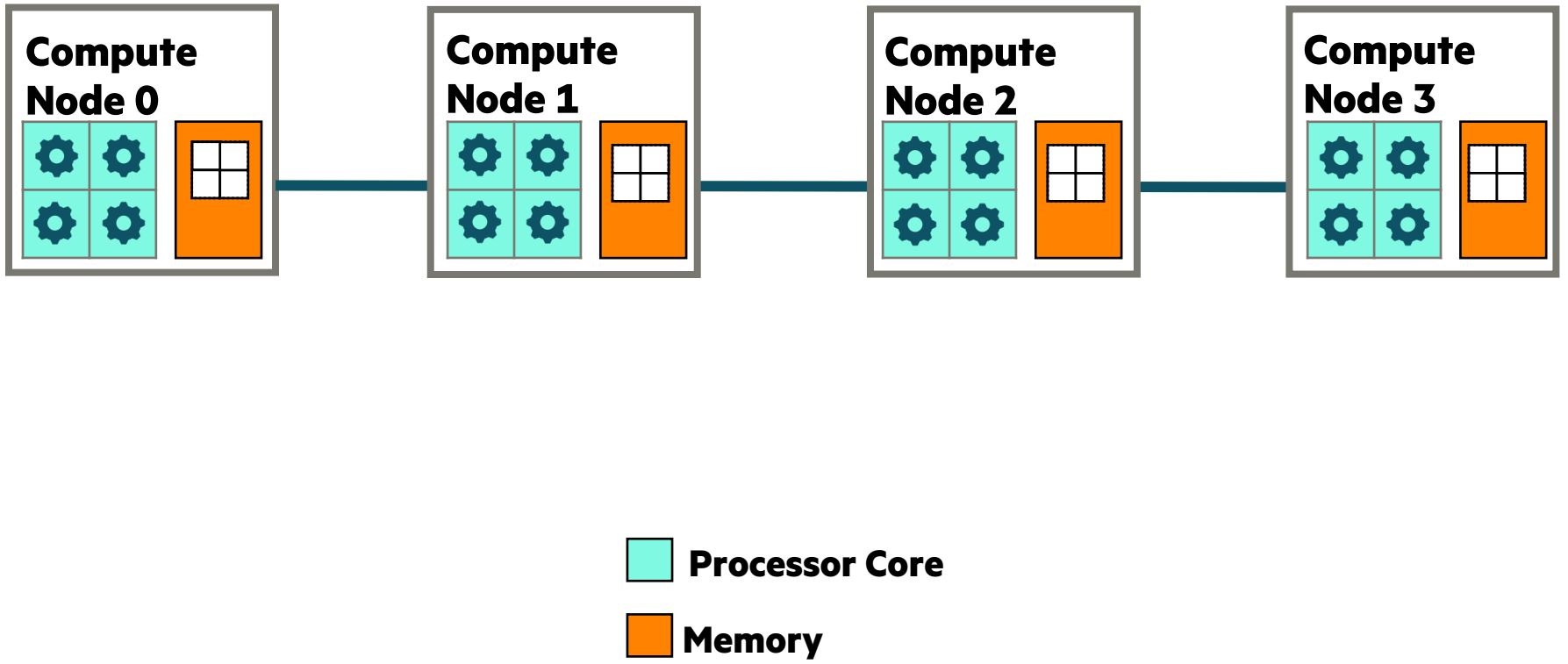
“Chapel allows me to use the available CPU and GPU power efficiently without low-level programming of data synchronization, managing threads, etc.”

“Low-level” Features for Parallelism and Locality



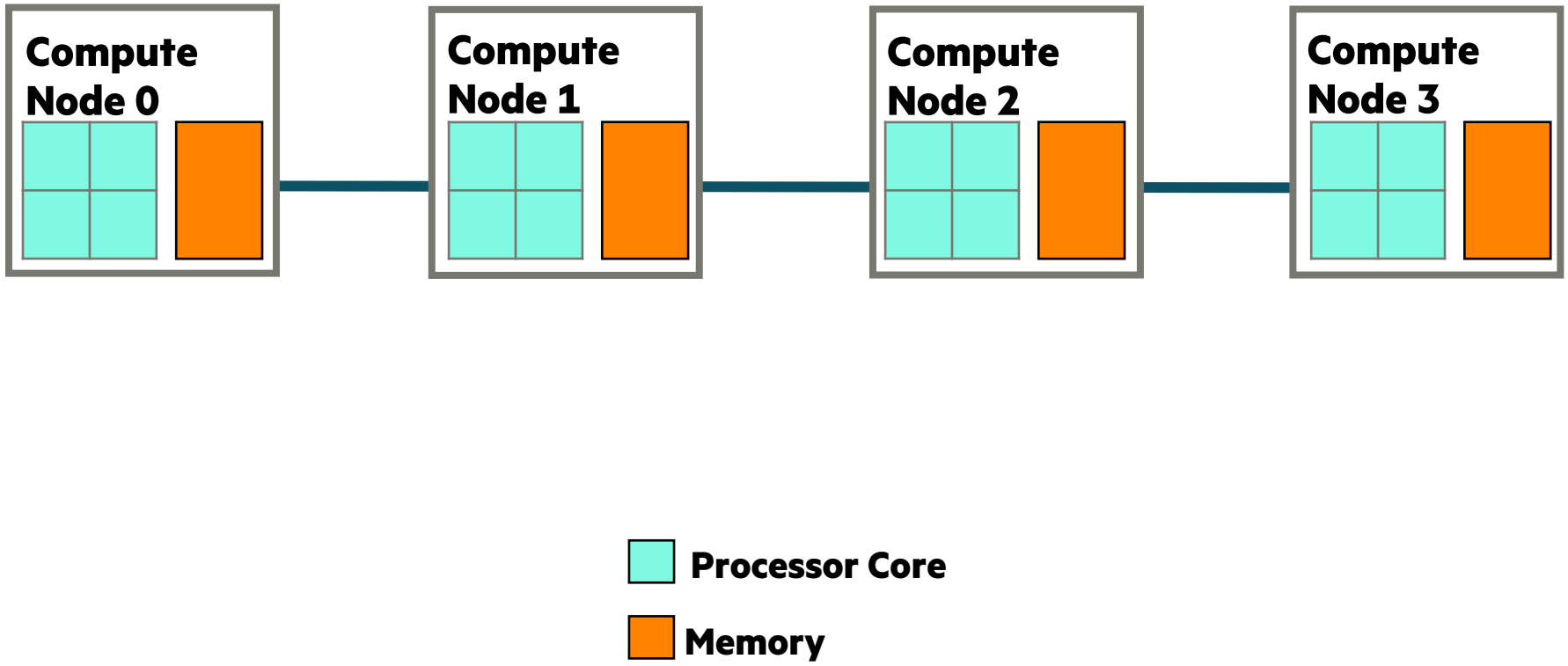
Key Concerns for Scalable Parallel Computing

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



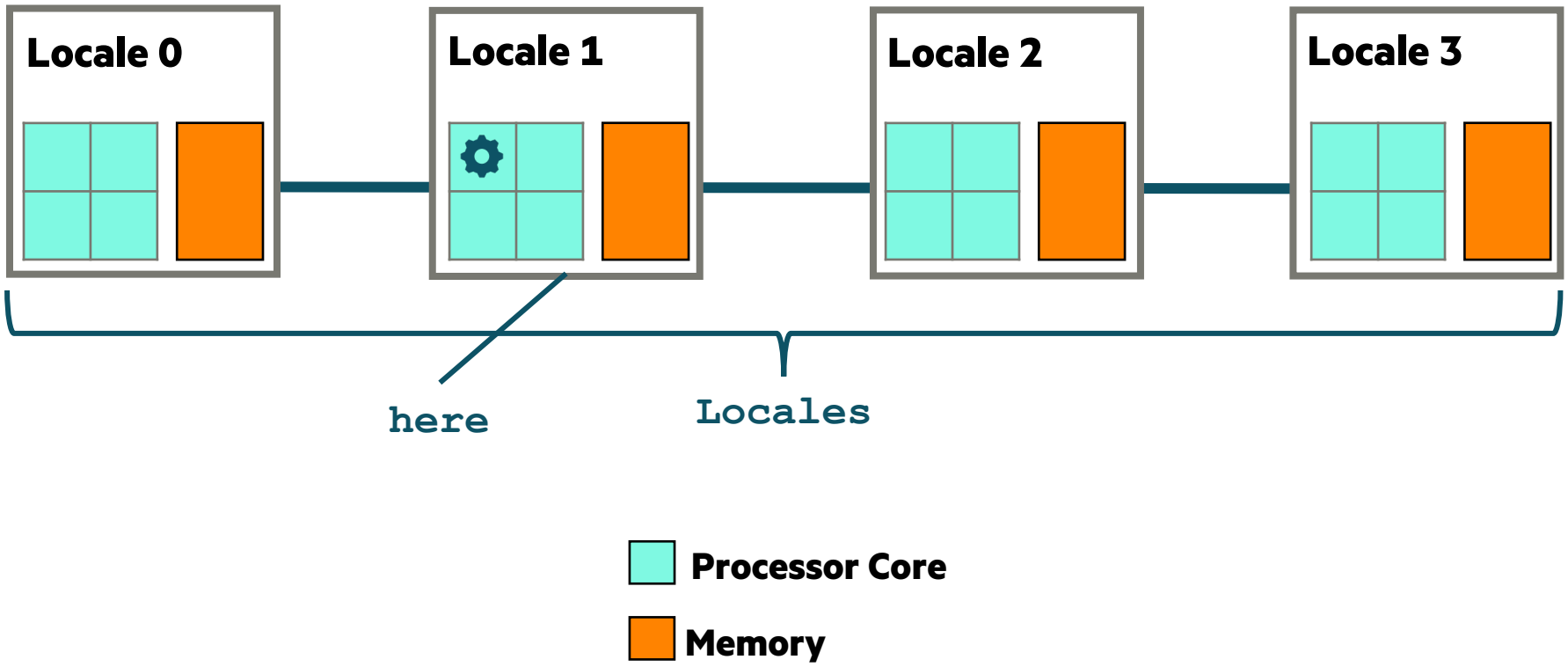
Locales in Chapel

- 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

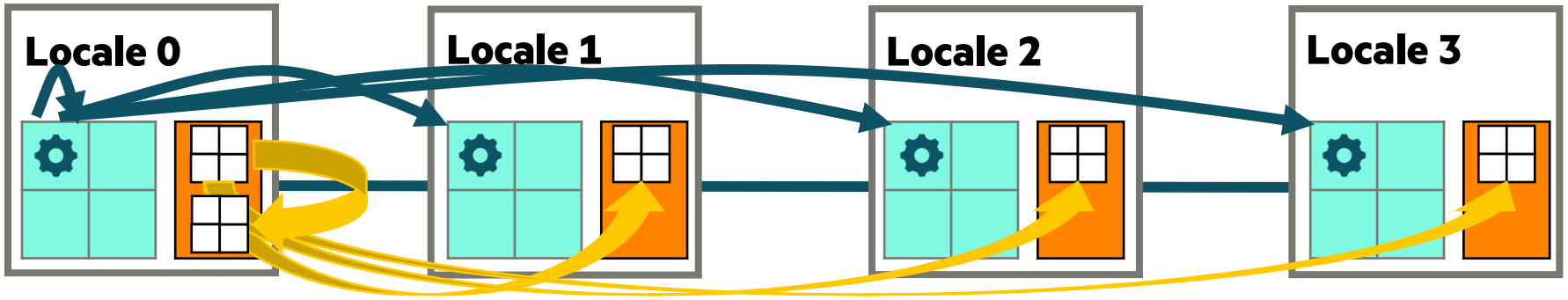
Variables are stored using the memory local to the current task

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



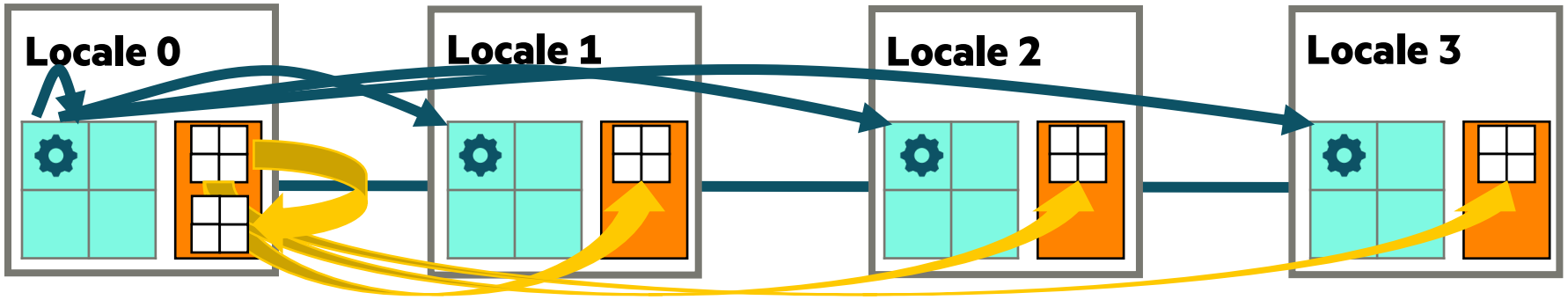
Mixing Locality with Task Parallelism

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

This is a distributed parallel computation

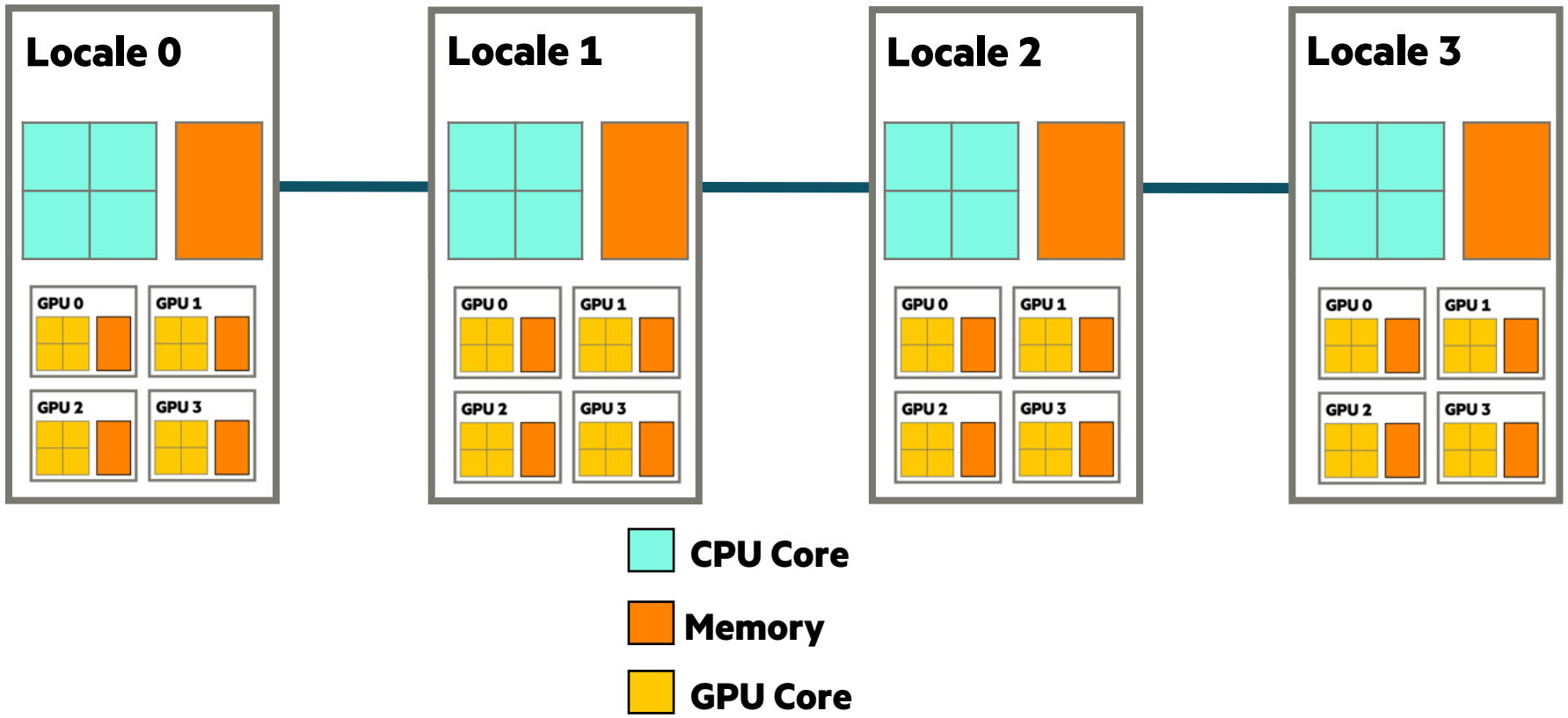


Representing GPUs in Chapel

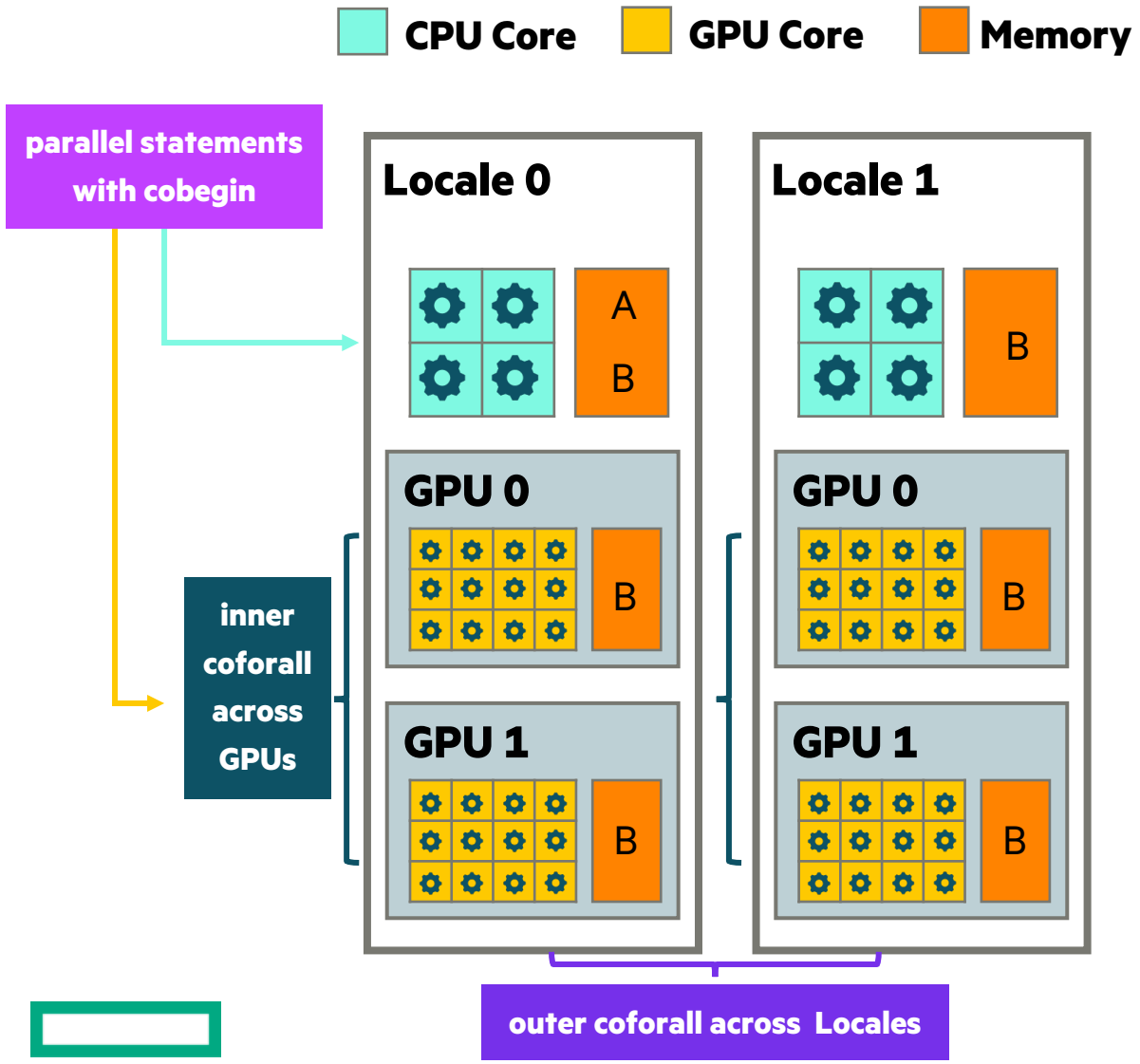
- In Chapel, we represent GPUs as *sub-locales*
  - Each top-level locale may have an array of locales called 'gpus'
  - We can then target them using Chapel's traditional features for parallelism + locality

```
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.

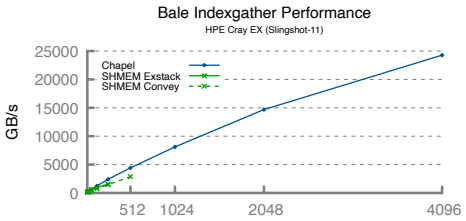
Wrap-up

Summary

Chapel is unique among programming languages

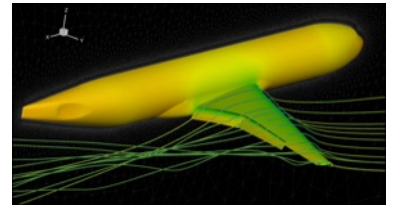
— supports first-class concepts for parallelism and locality
— ports and scales from laptops to supercomputers
— supports clean, concise code relative to conventional approaches
— supports GPUs in a vendor-neutral manner

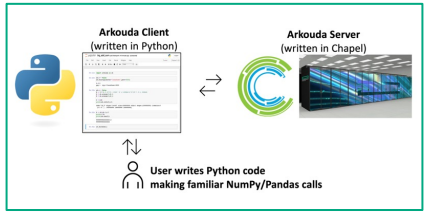
```
use BlockDist;

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];
```

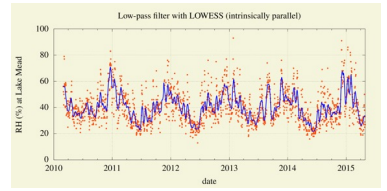


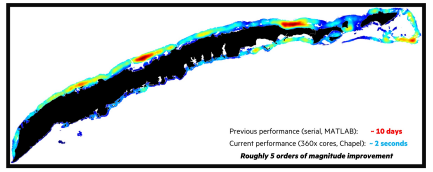
Chapel is being used for productive parallel computing at all scales

— 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





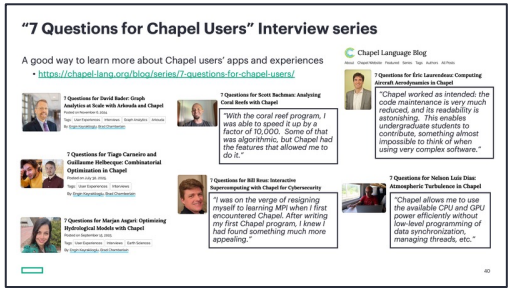




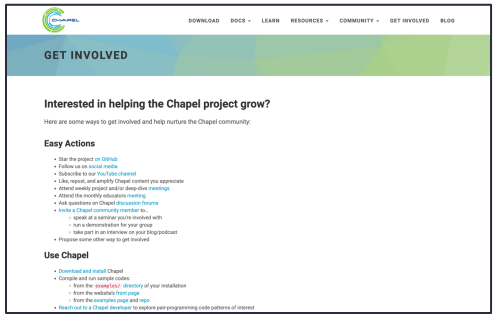
Three Ways to Get Started with Chapel

Listen to last week's episode of HPE's "Technology Now" podcast

Read interviews from the aforementioned *7 Questions for Chapel Users* series



Visit the Get Involved page on our website



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

Asynchrounous Communications

- Chapel Blog, typically ~2 articles per month
- Community Newsletter, quarterly
- Announcement Emails, around big events

Social Media

FOLLOW US

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

Discussion Forums

GET IN TOUCH

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

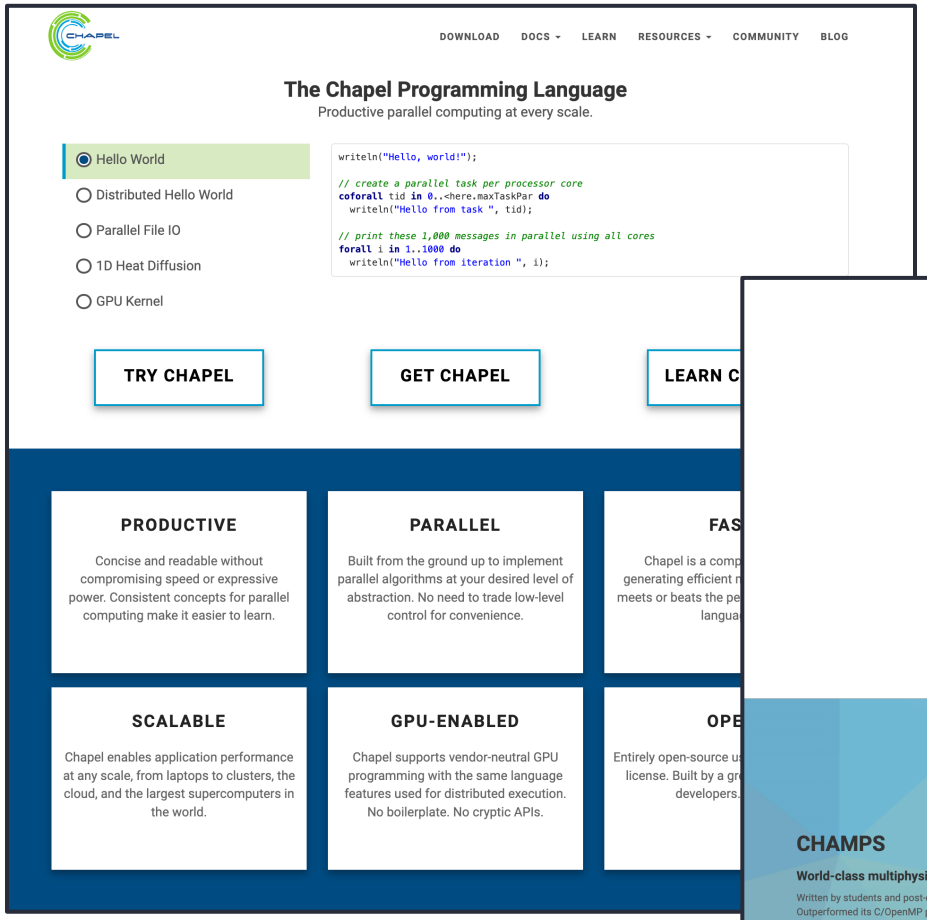
Ways to Use Chapel

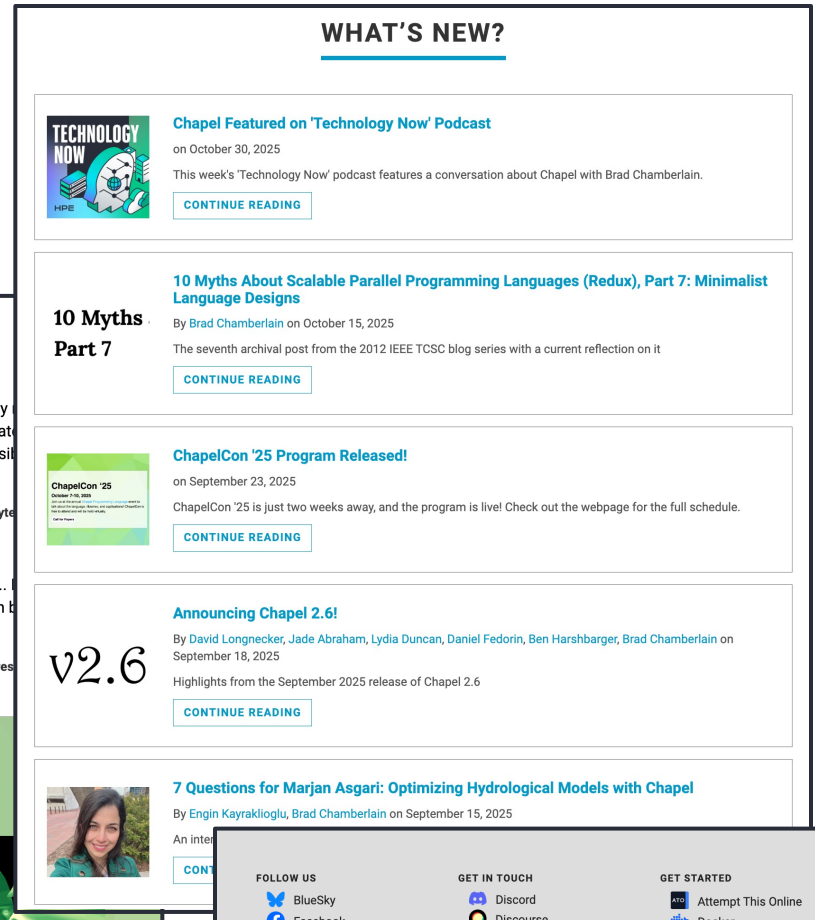
GET STARTED

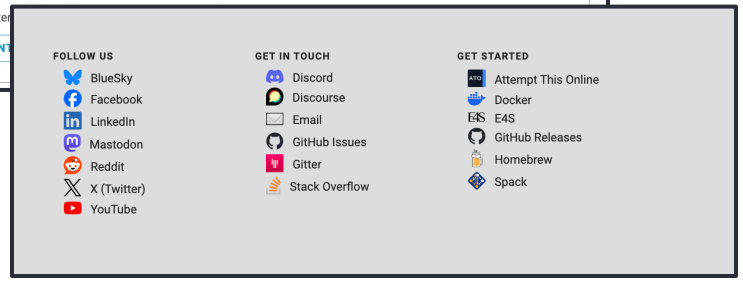
- Attempt This Online
- Docker
- E4S
- GitHub Releases
- Homebrew
- Spack

(from the footer of chapel-lang.org)

Chapel Website







chapel-lang.org

https://chapel-lang.org
@ChapelLanguage

© 2025 Hewlett Packard Enterprise Development LP