Hewlett Packard Enterprise

Parallel Programmability from Laptops to HPCs with Chapel and Arkouda

Brad Chamberlain
UW CSE PLSE
January 28, 2025

A Bit About Me

What is Chapel?

Chapel: A modern parallel programming language

- Portable & scalable
- Open-source & collaborative

Goals:

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

Productive Parallel Programming: One Definition

Imagine a programming language for parallel computing that is as…

…**readable and writeable** as Python

…yet also as…

…**fast** as Fortran / C / C++

…**scalable** as MPI / SHMEM

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

…**portable** as C

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

This is our motivation for Chapel

Six Key Characteristics of Chapel

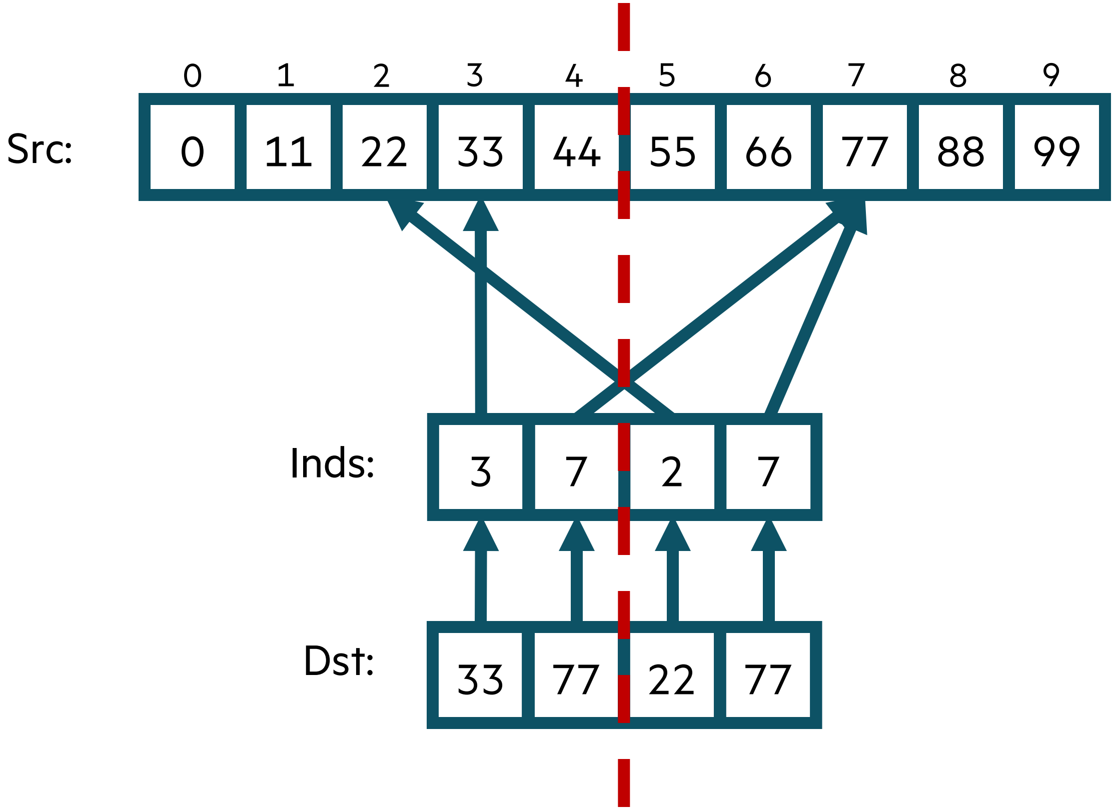
1. **portable:** runs on laptops, clusters, the cloud, supercomputers
2. **open-source:** to reduce barriers to adoption and leverage community contributions
3. **compiled:** to generate the best performance possible
4. **statically typed:** to avoid simple errors after hours of execution
5. **interoperable:**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

Outline

- Chapel Goals and Characteristics
- A Brief Introduction to Chapel
- Applications of Chapel
- Global-view vs. SPMD Programming
- Chapel Parallelism and Locality Features
- Sample Compiler Optimizations
- Programming GPUs in Chapel
- Wrap-up

A Brief Introduction to Chapel
(via Bale IndexGather)

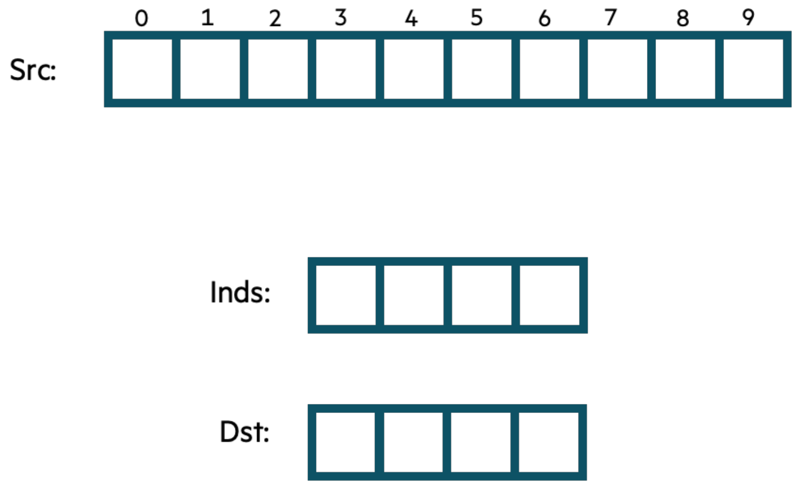
Bale IndexGather (IG): In Pictures



Bale IG in Chapel: Array Declarations

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



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

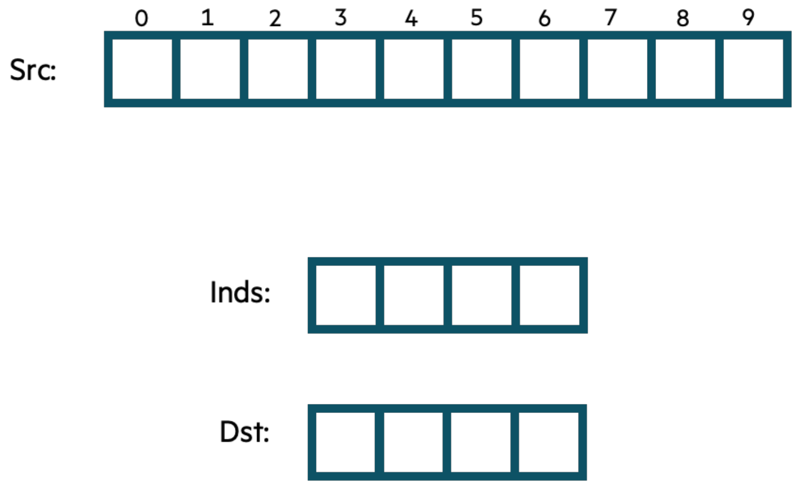
```
$
```



Bale IG in Chapel: Compiling

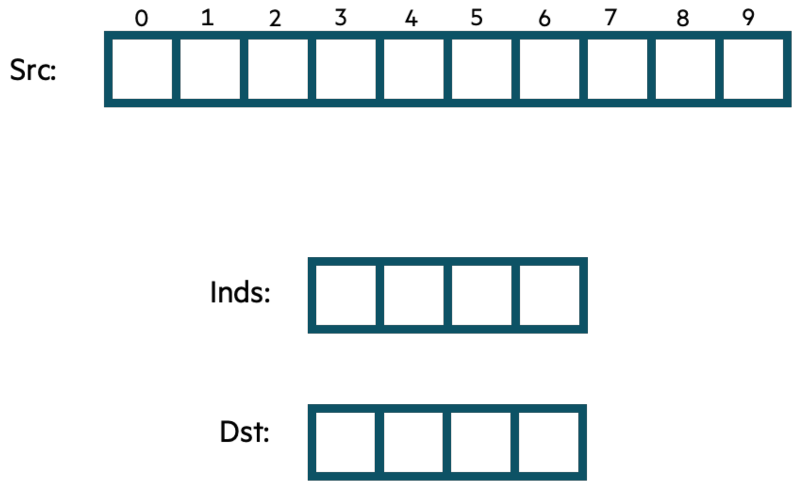
```
config const n = 10,
             m = 4;



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

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



Bale IG in Chapel: Executing

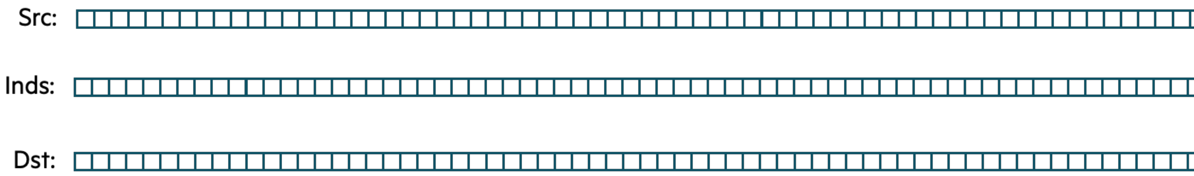
```
config const n = 10,
             m = 4;



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




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

Bale IG in Chapel: Executing, Overriding Configs

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



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

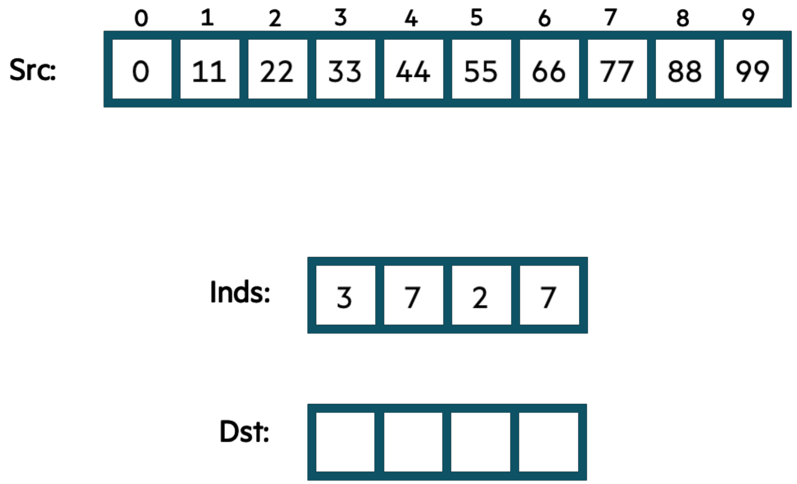



```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000
$
```

Bale IG in Chapel: Array Initialization

```
use Random;

config const n = 10,
             m = 4;




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

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

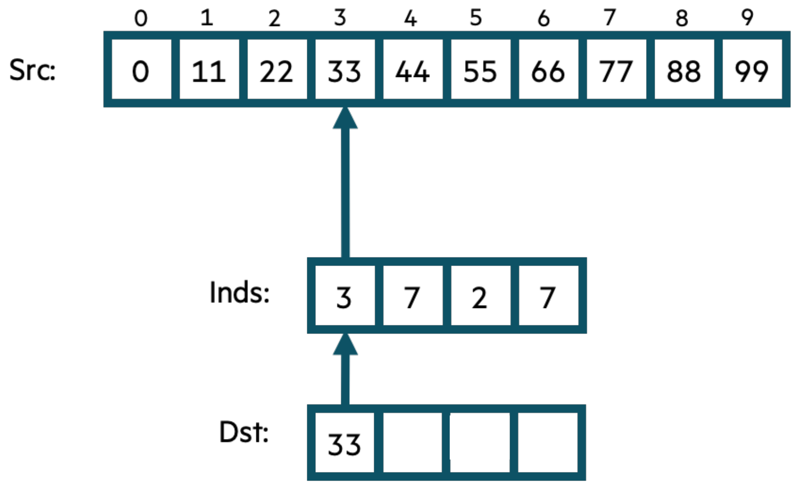
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Serial Version

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



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

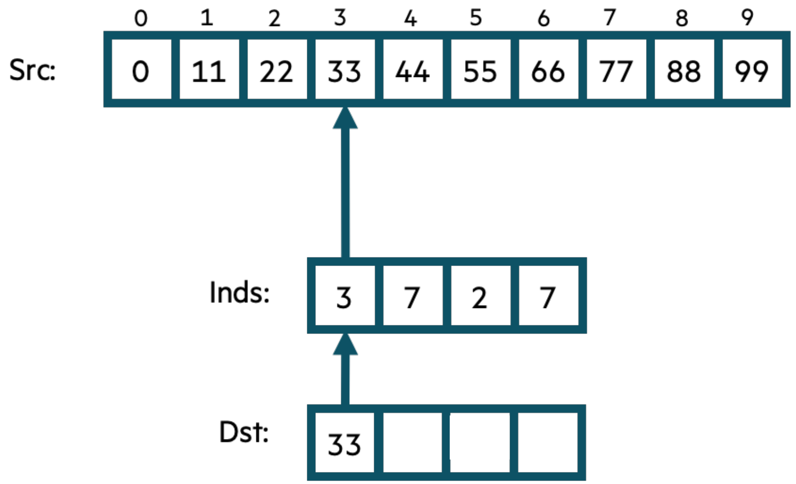
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Serial, Zippered Version

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



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

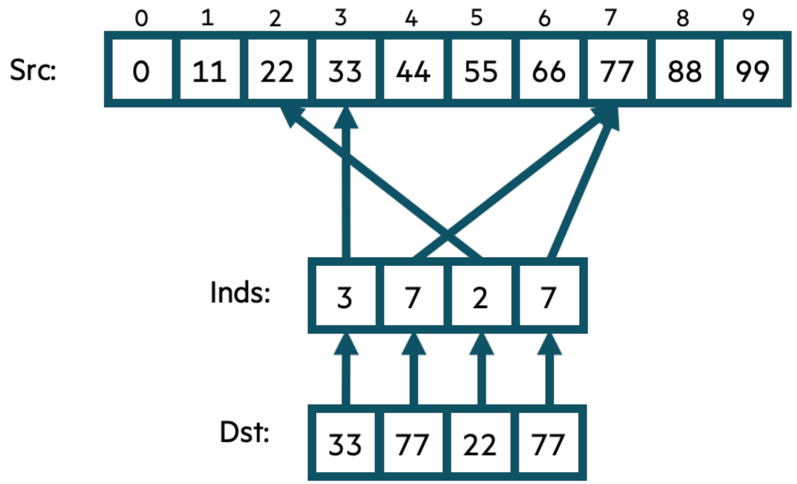
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



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

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



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

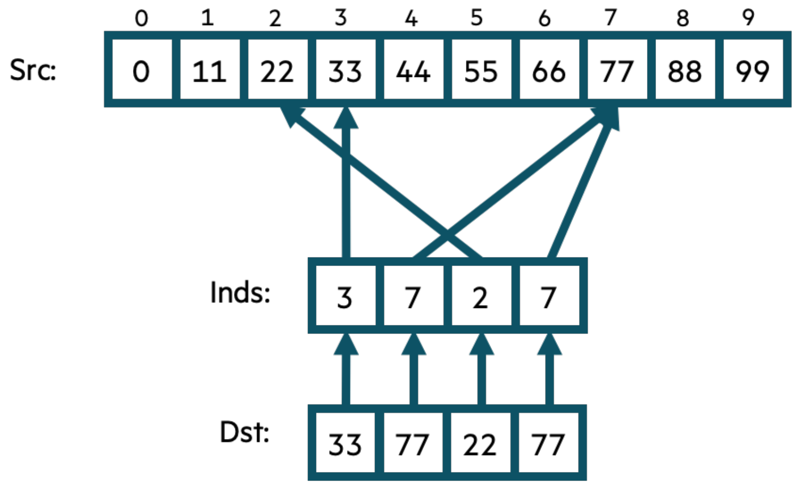
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



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

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



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

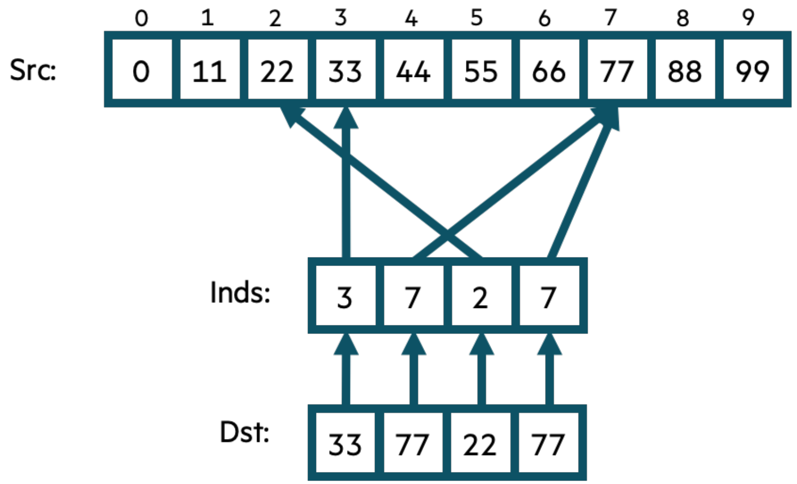
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



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

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



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

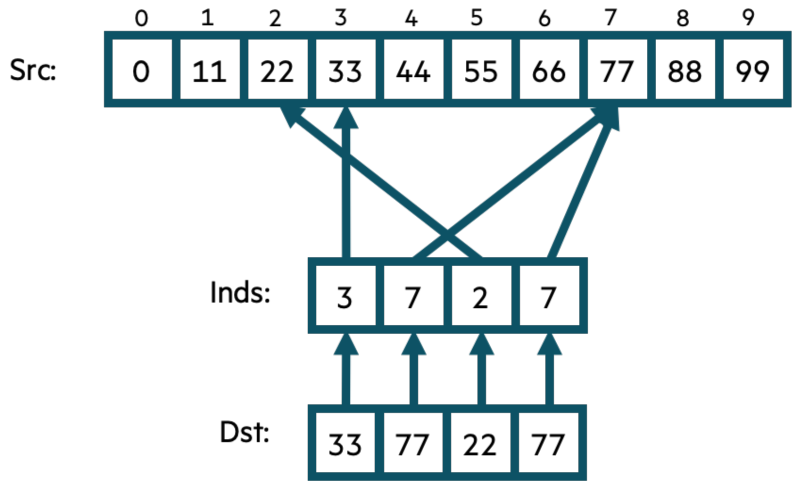
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



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

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



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

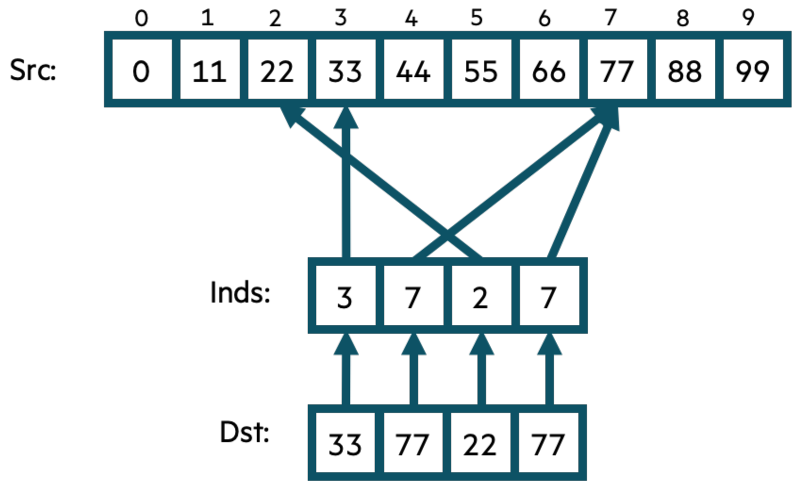
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



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

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



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

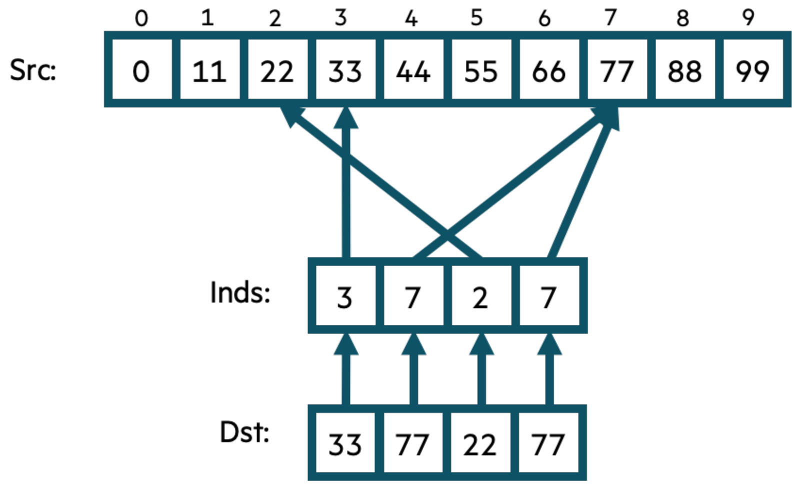
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



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

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



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

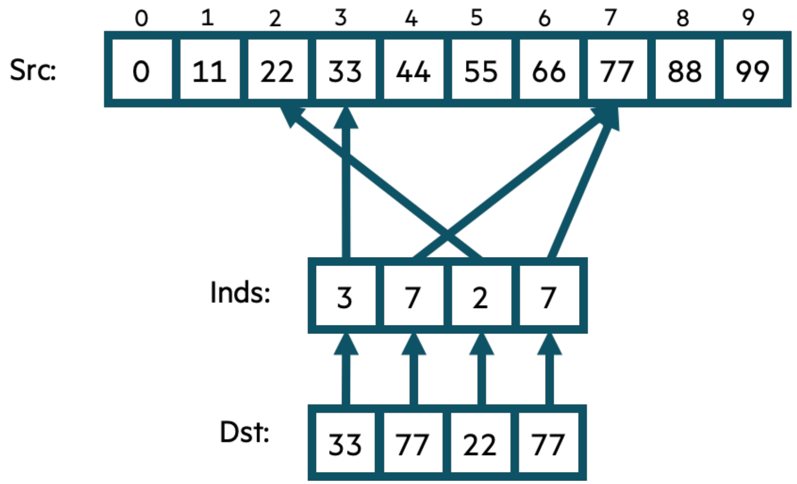
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



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

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

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

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

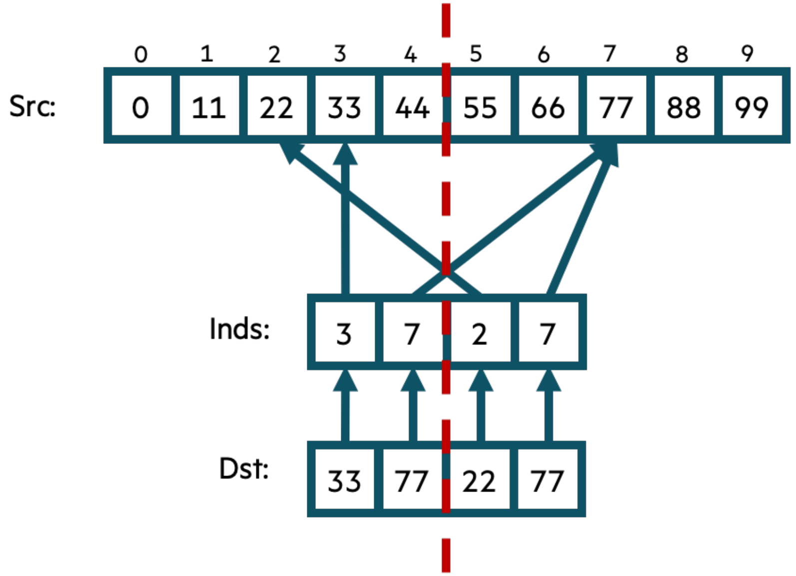
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Distributed Parallel Version

```
use BlockDist;

config const n = 10,
             m = 4;

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

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

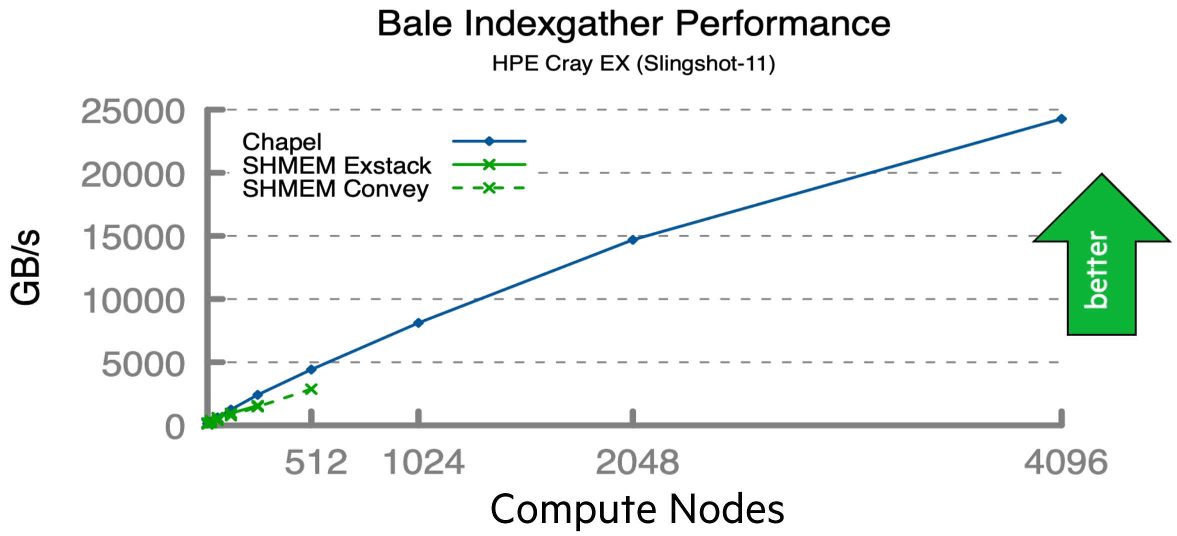
```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4096
$
```

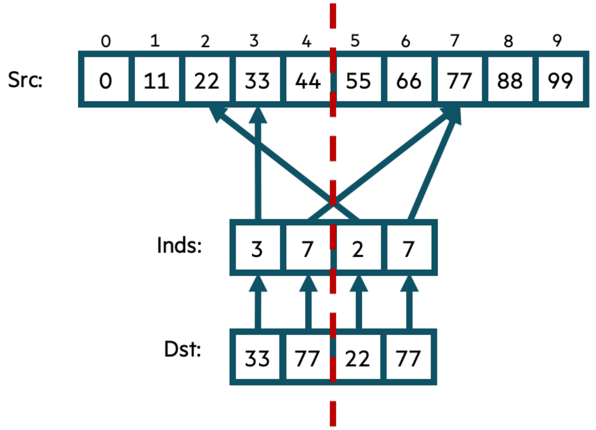


Bale IG in Chapel: Distributed Parallel Version

```
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 --auto-aggregation
$ ./bale-ig -nl 4096
$
```





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

Chapel (Simple / Auto-Aggregated version)

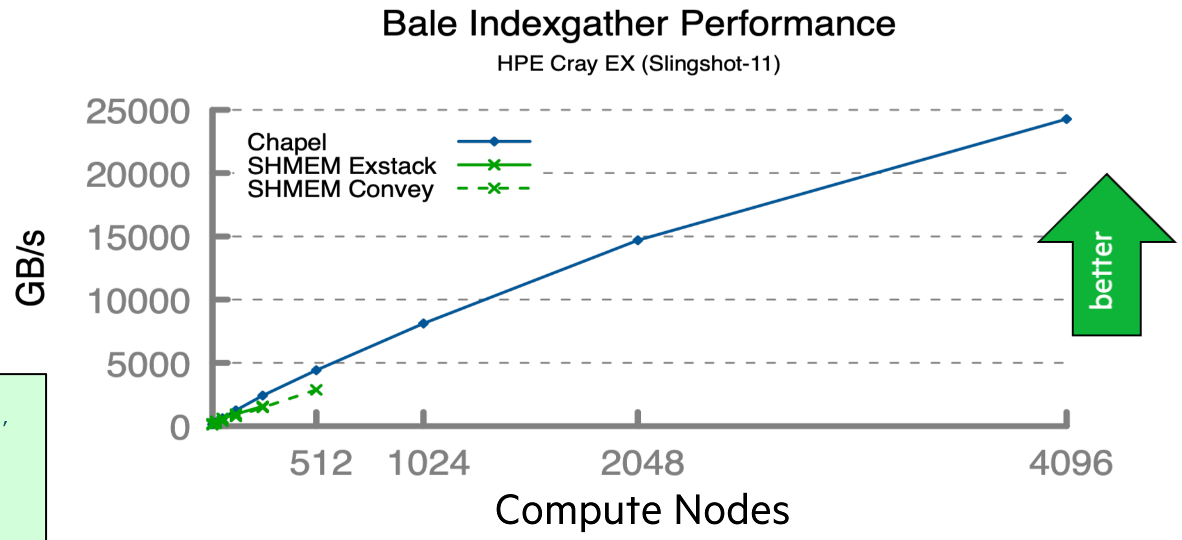
```
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

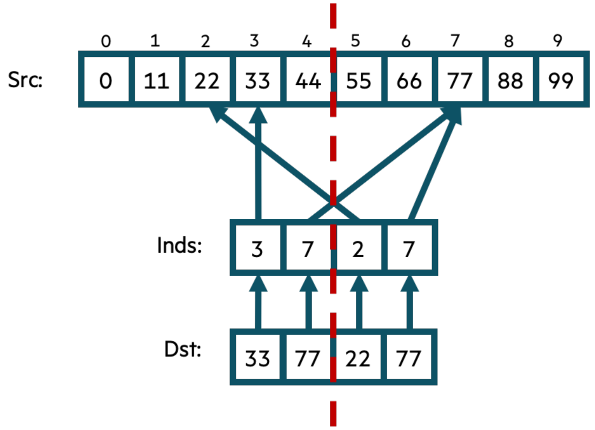
SHMEM (Exstack version)

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

  exstack_exchange(ex);

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

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

SHMEM (Conveyors version)

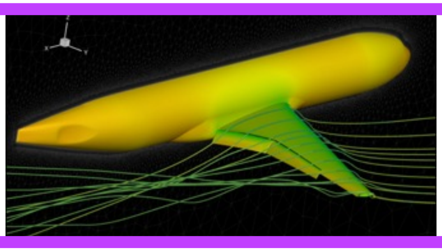
```
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
       more | convey_advance(replies, !more)) {

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

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

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



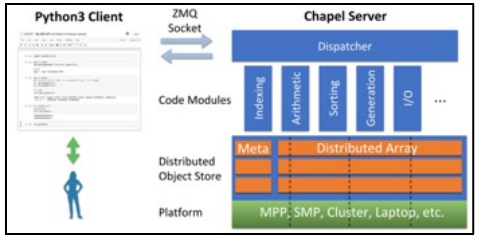


Applications of Chapel

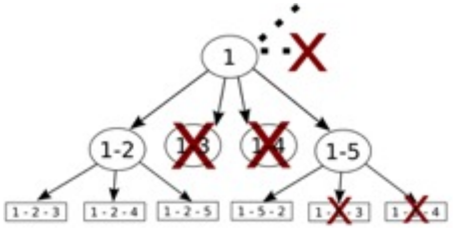
Applications of Chapel



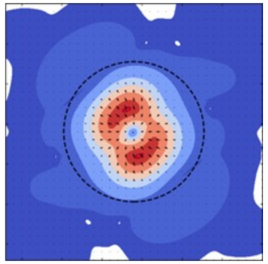
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



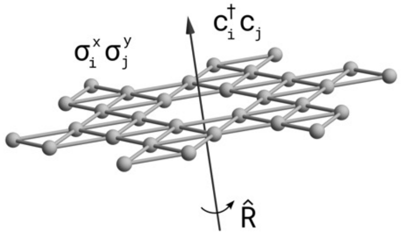
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



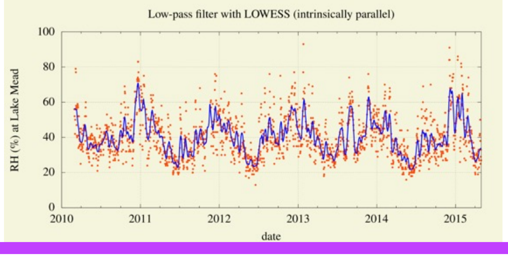
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

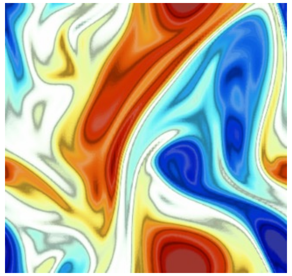


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

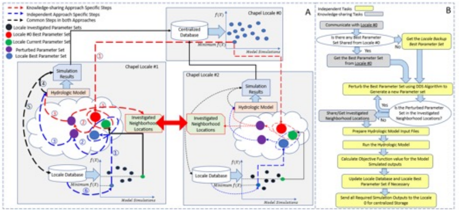


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

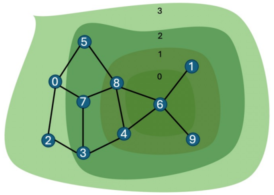
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



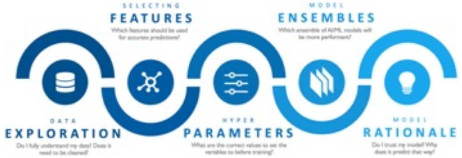
ChapQG: Layered Quasigeostrophic CFD
Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



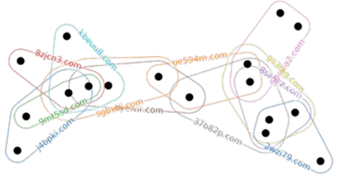
Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



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



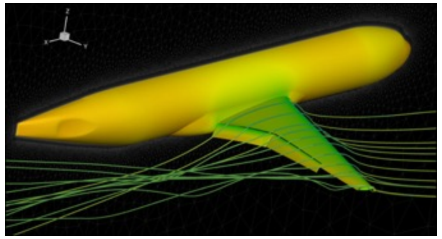
CrayAI HyperParameter Optimization (HPO)
Ben Albrecht et al.
Cray Inc. / HPE



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

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

Productivity Across Diverse Application Scales (code and system size)



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

7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

Posted on September 17, 2024.

Tags: Computational Fluid Dynamics | User Experiences | Interviews

By: Engin Kayraklioglu, Brad Chamberlain

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

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

7 Questions for Scott Bachman: Analyzing Coral Reefs with Chapel

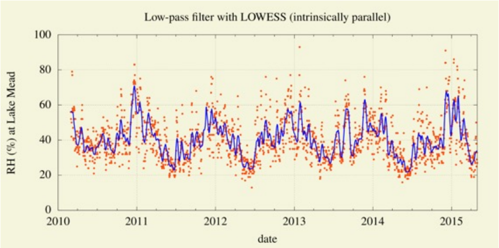
Posted on October 1, 2024.

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

By: Brad Chamberlain, Engin Kayraklioglu

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

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



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

7 Questions for Nelson Luís Dias: Atmospheric Turbulence in Chapel

Posted on October 15, 2024.

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

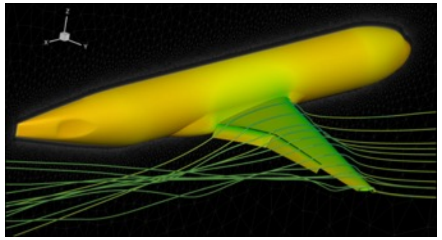
By: Engin Kayraklioglu, Brad Chamberlain

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

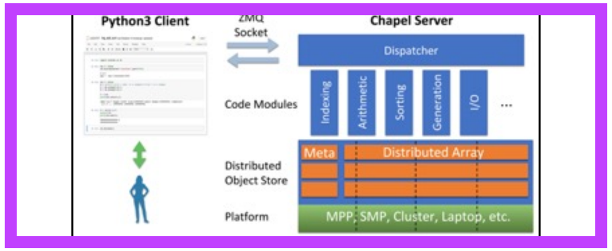
> *"Chapel allows me to use the available CPU and GPU power efficiently without low-level programming of data synchronization, managing threads, etc."*

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

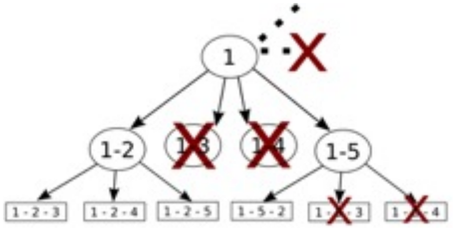
Applications of Chapel



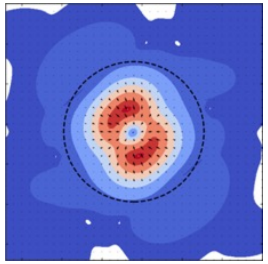
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



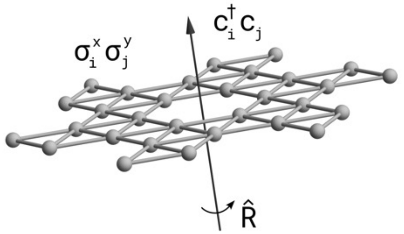
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



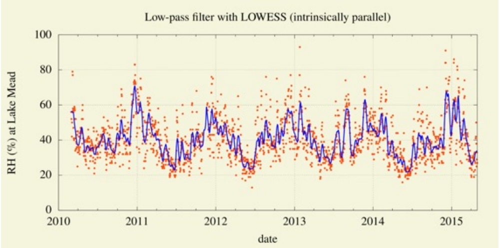
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

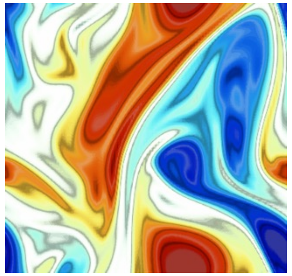


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

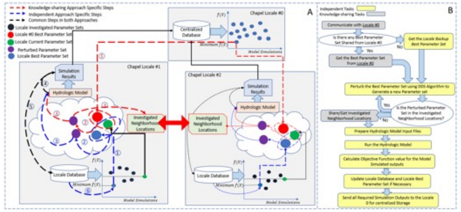


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

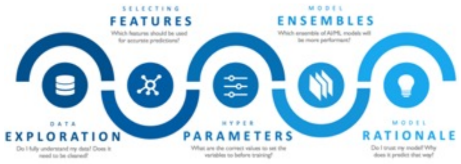
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



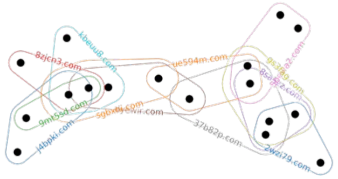
ChapQG: Layered Quasigeostrophic CFD
Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



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



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



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

?

Your Application Here?

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

Data Science In Python at scale?

Motivation: Imagine you've got…

…HPC-scale data science problems to solve

…a bunch of Python programmers

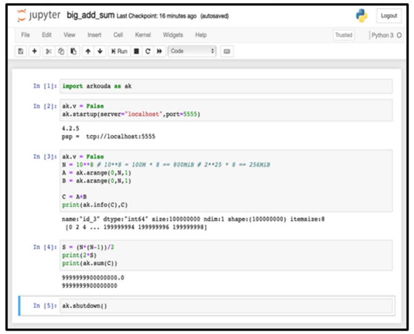
…access to HPC systems

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

What is Arkouda?

Q: “What is Arkouda?”

Arkouda Client
(written in Python)

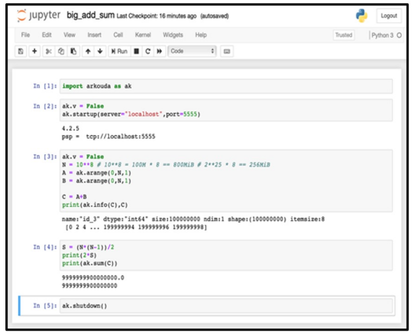


User writes Python code
making familiar NumPy/Pandas calls

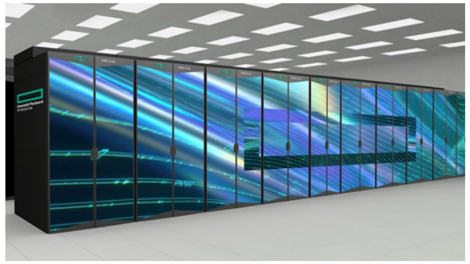
What is Arkouda?

Q: “What is Arkouda?”

Arkouda Client
(written in Python)



Arkouda Server
(written in Chapel)



User writes Python code
making familiar NumPy/Pandas calls

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

Performance and Productivity: Arkouda Argsort

HPE Cray EX

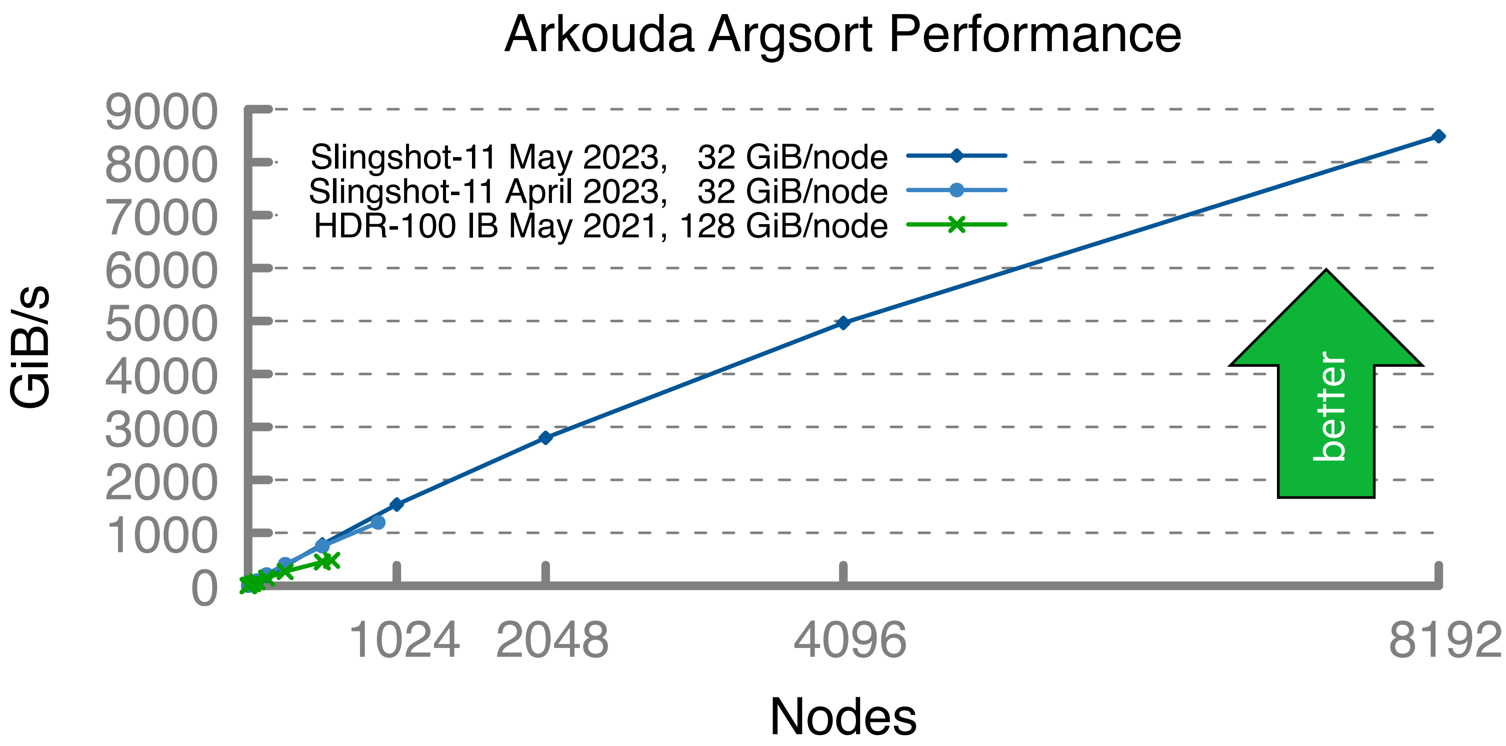
- Slingshot-11 network (200 Gb/s)
- 8192 compute nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)

HPE Cray EX

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

HPE Apollo

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

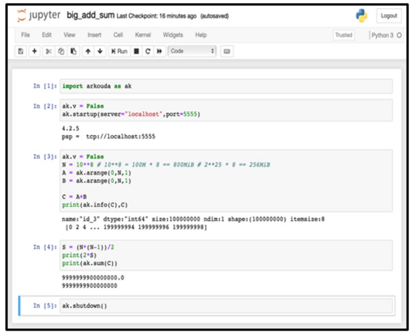


Implemented using ~100 lines of Chapel

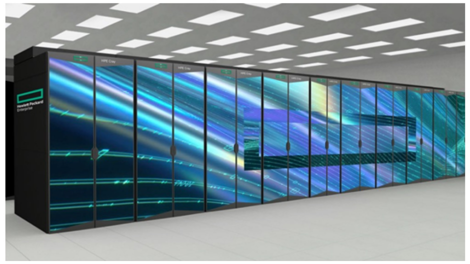
What is Arkouda?

Q: "What is Arkouda?"

Arkouda Client
(written in Python)



Arkouda Server
(written in Chapel)



User writes Python code making familiar NumPy/Pandas calls

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

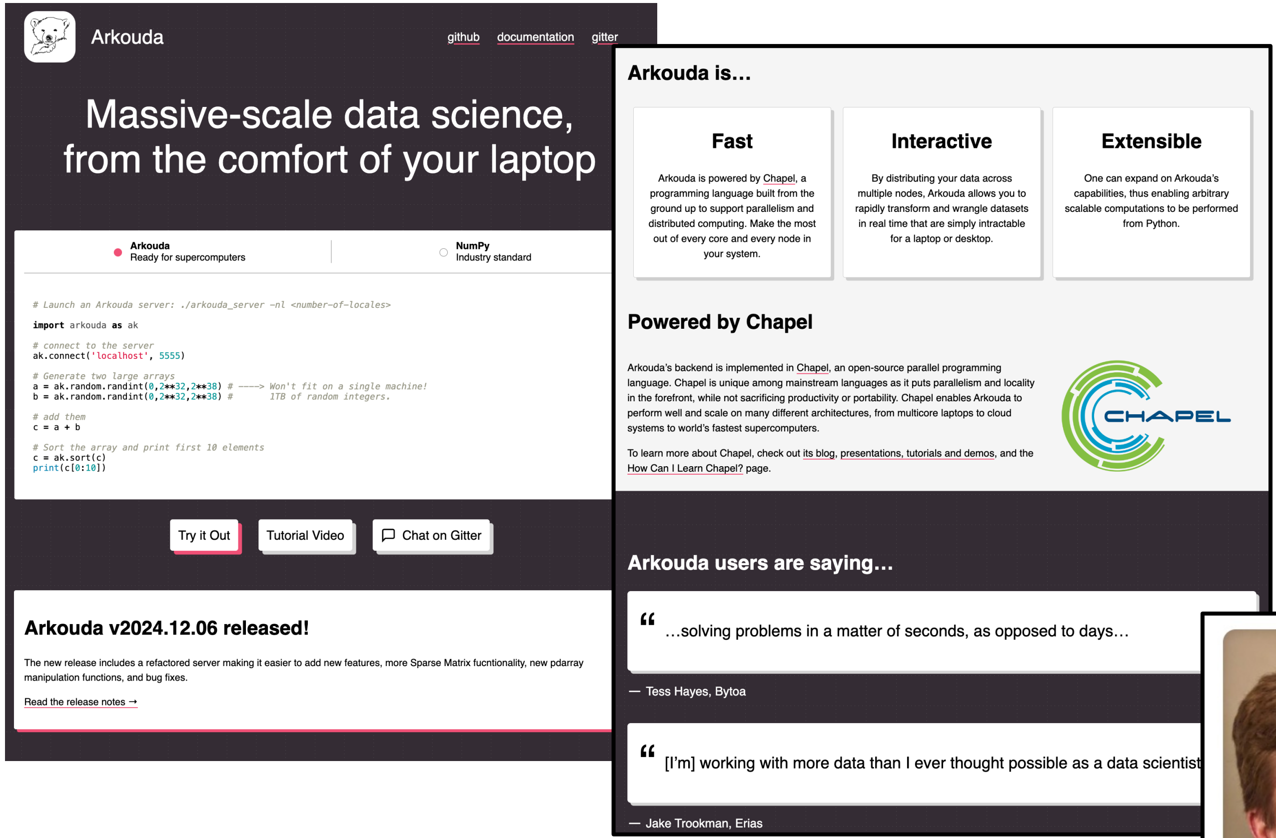
A': "An extensible framework for arbitrary HPC computations"

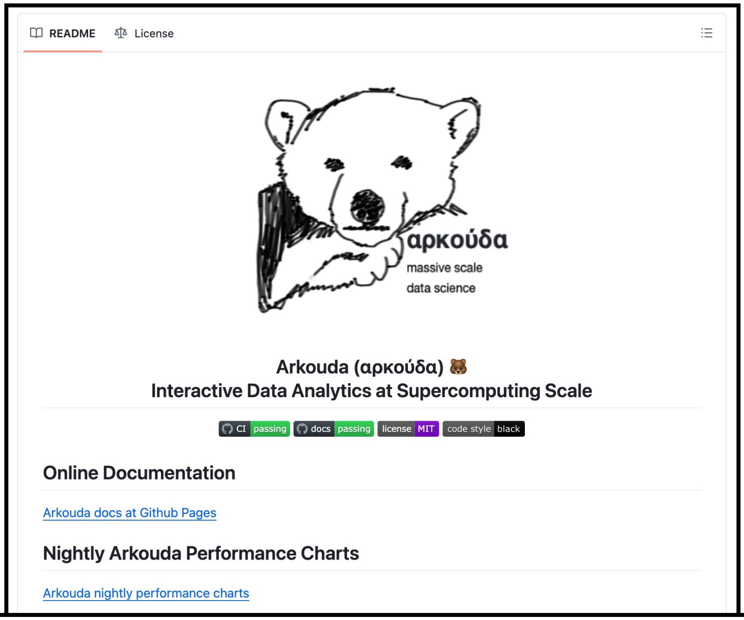
A": "A way to drive HPC systems interactively from Python on a laptop"

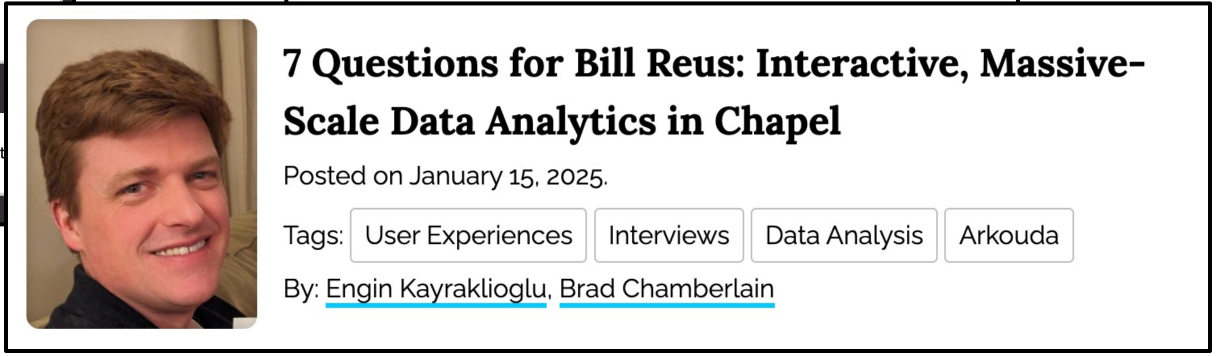
Arkouda Resources

Website: https://arkouda-www.github.io/

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

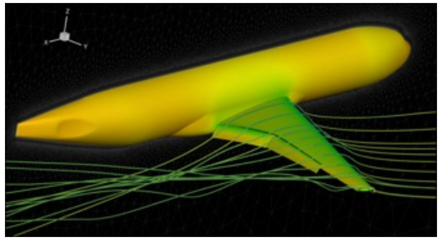






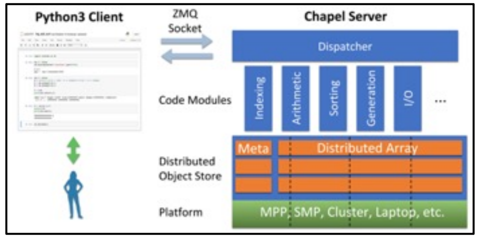
Coming Soon: interview with founding dev, Bill Reus

Applications of Chapel: Links to Users' Talks (slides + video) **& Blog Interviews**



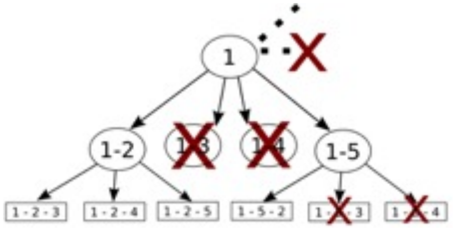
CHAMPS: 3D Unstructured CFD

CHIUW 2021 CHIUW 2022 Blog



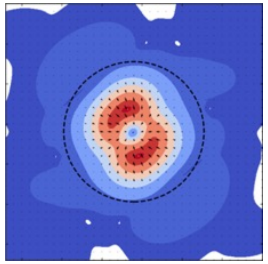
Arkouda: Interactive Data Science at Massive Scale

CHIUW 2020 CHIUW 2023



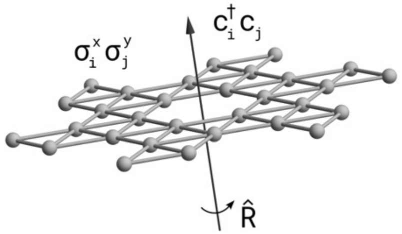
ChOp: Chapel-based Optimization

CHIUW 2021 2023 ChapelCon'24 (I|I)



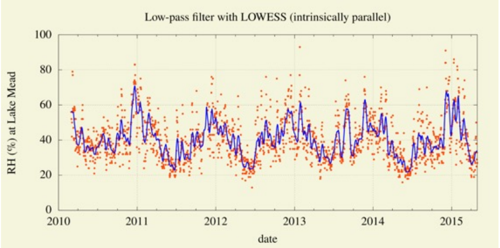
ChplUltra: Simulating Ultralight Dark Matter

CHIUW 2020 CHIUW 2022



Lattice-Symmetries: a Quantum Many-Body Toolbox

CHIUW 2022

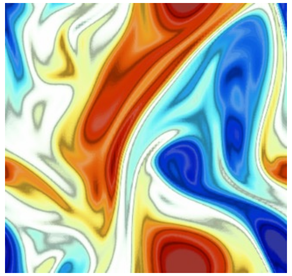


Desk dot chpl: Utilities for Environmental Eng.

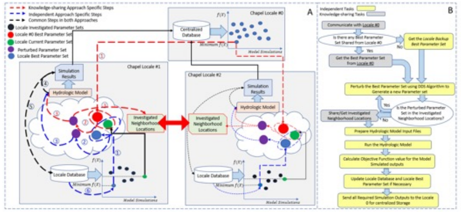
CHIUW 2022 ChapelCon '24 Blog

RapidQ: Mapping Coral Biodiversity

CHIUW 2023 Blog

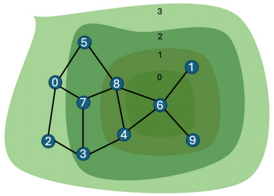


ChapQG: Layered Quasigeostrophic CFD



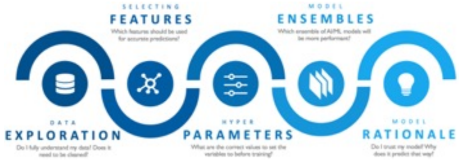
Chapel-based Hydrological Model Calibration

CHIUW 2023



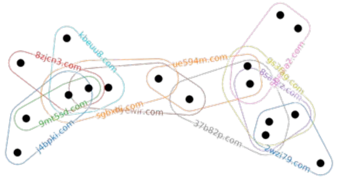
Arachne Graph Analytics

CHIUW 2023 ChapelCon '24 Blog



CrayAI HyperParameter Optimization (HPO)

CHIUW 2021



CHGL: Chapel Hypergraph Library

CHIUW 2020

[NOTE: This slide focuses on presentations and blogs published in Chapel venues, but numerous external publications also exist]

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

Global-view vs. SPMD Programming

A Strained(?) Analogy

HPC Benchmarks Using Conventional Programming Approaches

STREAM TRIAD: C + MPI + OPENMP

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

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

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

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

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

  return errCount;
}

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

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

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

  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
      fprintf( outFile, "Failed to allocate memory (%d).\n", VectorSize );
      fclose( outFile );
    }
    return 1;
  }

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 1.0;
  }
  scalar = 3.0;

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);

  return 0;
}
```

HPCC RA: MPI KERNEL

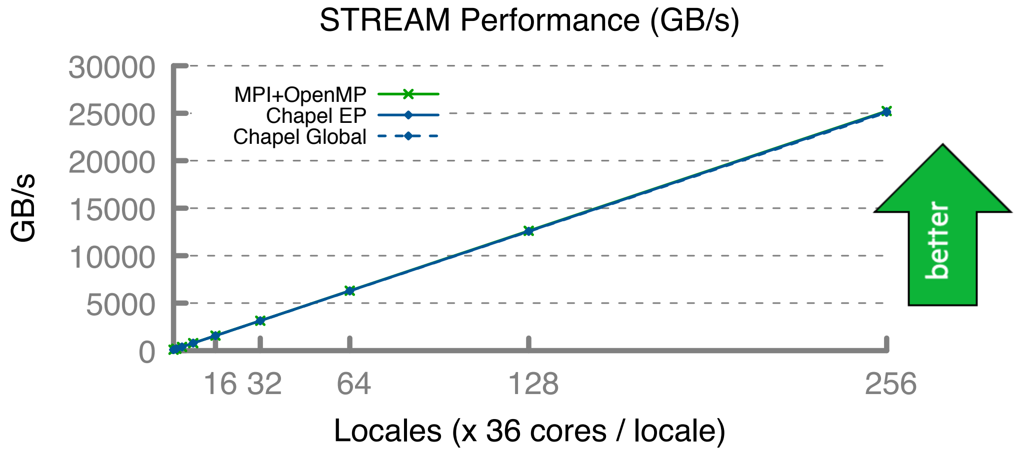
```c
/* Perform updates to main table.  The scalar equivalent is:
 *
 *    for (i=0; i<NUPDATE; i++) {
 *      Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *      Table[Ran & (TABSIZE-1)] ^= Ran;
 *    }
 */

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;
    }
  }
  /* send our done messages */
  for (proc_count = 0 ; proc_count < tparams.NumProcs ; ++proc_count) {
    if (proc_count == tparams.MyProc) { tparams.finish_req[tparams.MyProc] =
                                        MPI_REQUEST_NULL; continue; }
    /* send garbage - who cares, no one will look at it */
    MPI_Isend(&Ran, 0, tparams.dtype64, proc_count, FINISHED_TAG,
              MPI_COMM_WORLD, tparams.finish_req + proc_count);
  }
  /* Finish everyone else up... */
  while (NumberReceiving > 0) {
    MPI_Wait(&inreq, &status);
    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);
  }

  MPI_Waitall( tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);
```

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

STREAM TRIAD: C + MPI + OPENMP

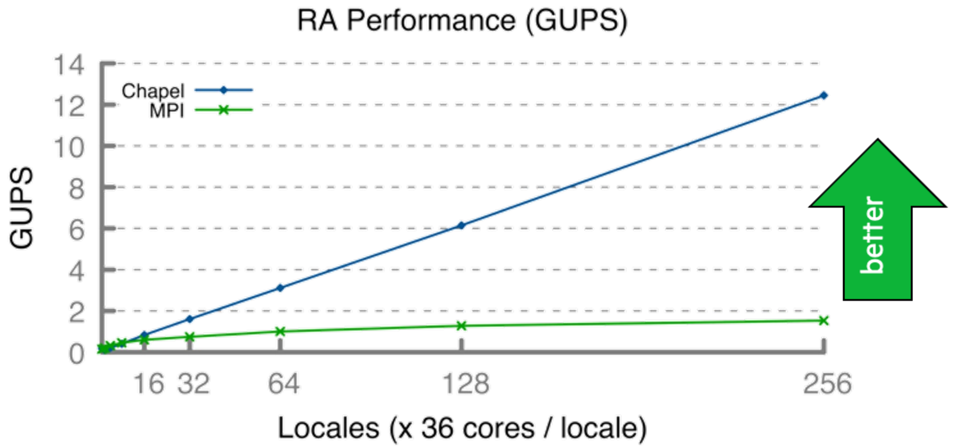
```c
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

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 );
```

```chapel
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

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



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

Chapel (Simple / Auto-Aggregated version)

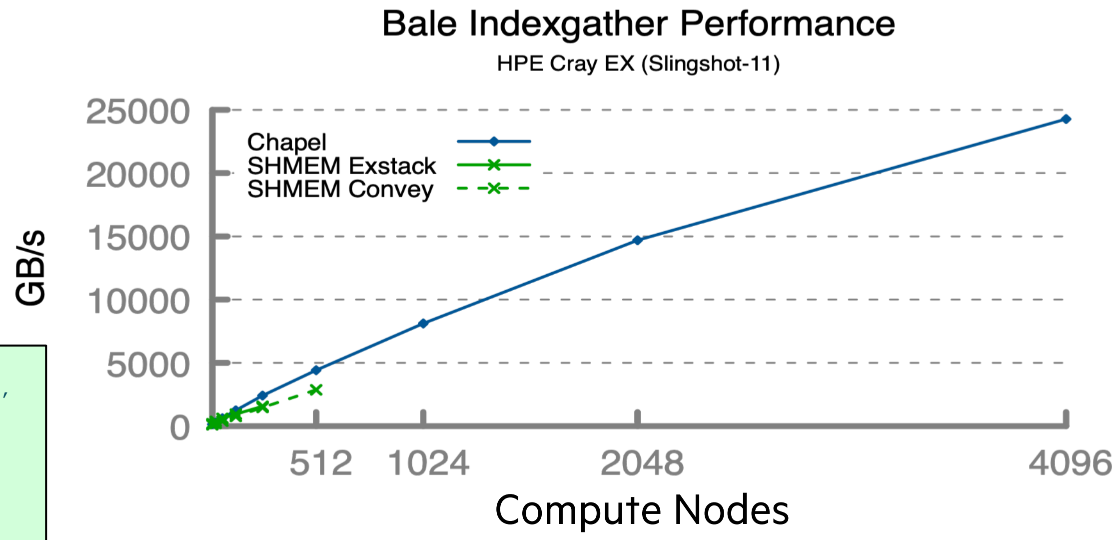
```
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

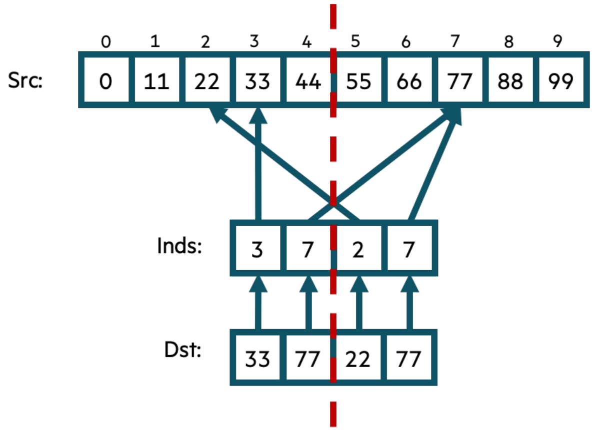
SHMEM (Exstack version)

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

  exstack_exchange(ex);

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

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

SHMEM (Conveyors version)

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

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

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

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



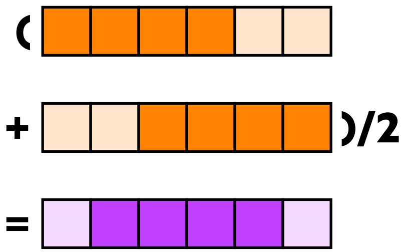


Q: What accounts for the code size disparities between Chapel and SHMEM / MPI?

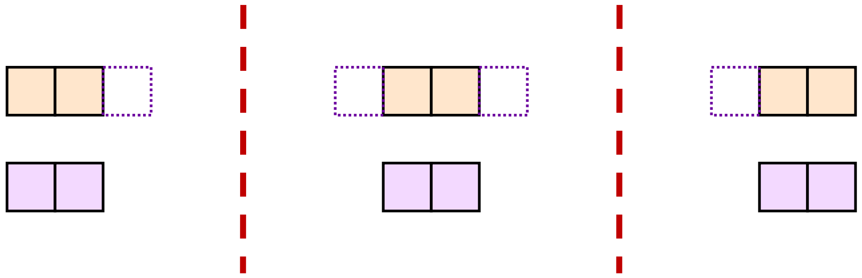
A: Chapel Supports Global-view Programming

Example: “Apply a 3-point stencil to a vector”

Global-View



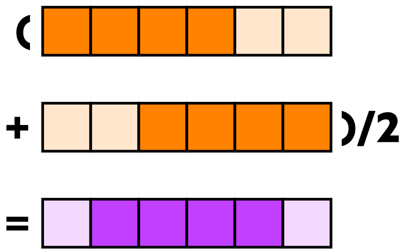
SPMD



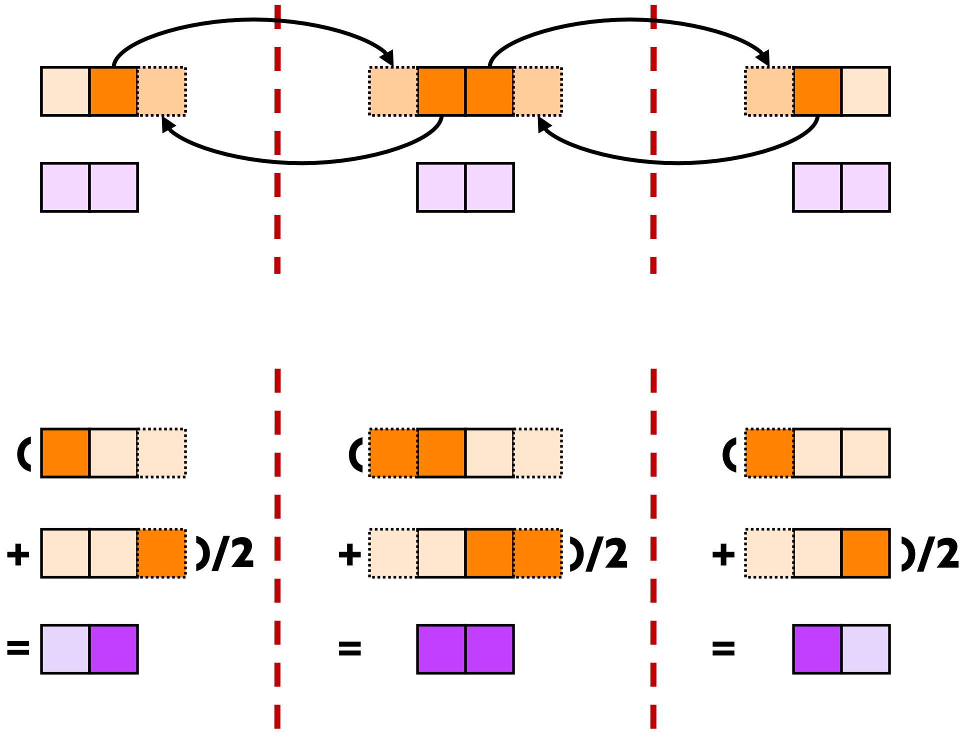
A: Chapel Supports Global-view Programming

Example: "Apply a 3-point stencil to a vector"

Global-View



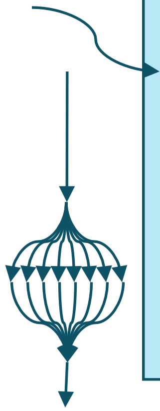
SPMD



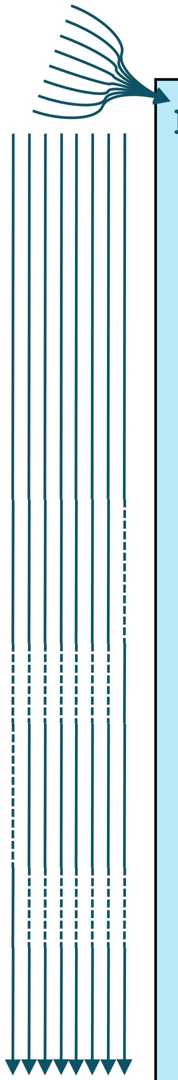
A: Chapel Supports Global-view Programming

Example: "Apply a 3-point stencil to a vector"

Global-View Chapel code



```
use BlockDist;

proc main() {
  var n = 1000;
  const D = blockDist.createDomain(1..n);

  forall i in D[2..n-1] do
    B[i] = (A[i-1] + A[i+1])/2;
}
```



SPMD pseudocode (MPI-esque)

```
proc main() {
  var n = 1000;
  var p = numProcs(),
      me = myProc(),
      myN = n/p,
      myLo = 1,
      myHi = myN;
  var A, B: [0..myN+1] real;

  if (me < p-1) {
    send(me+1, A[myN]);
    recv(me+1, A[myN+1]);
  } else
    myHi = myN-1;
  if (me > 0) {
    send(me-1, A[1]);
    recv(me-1, A[0]);
  } else
    myLo = 2;
  forall i in myLo..myHi do
    B[i] = (A[i-1] + A[i+1])/2;
}
```

SPMD Programming in Chapel

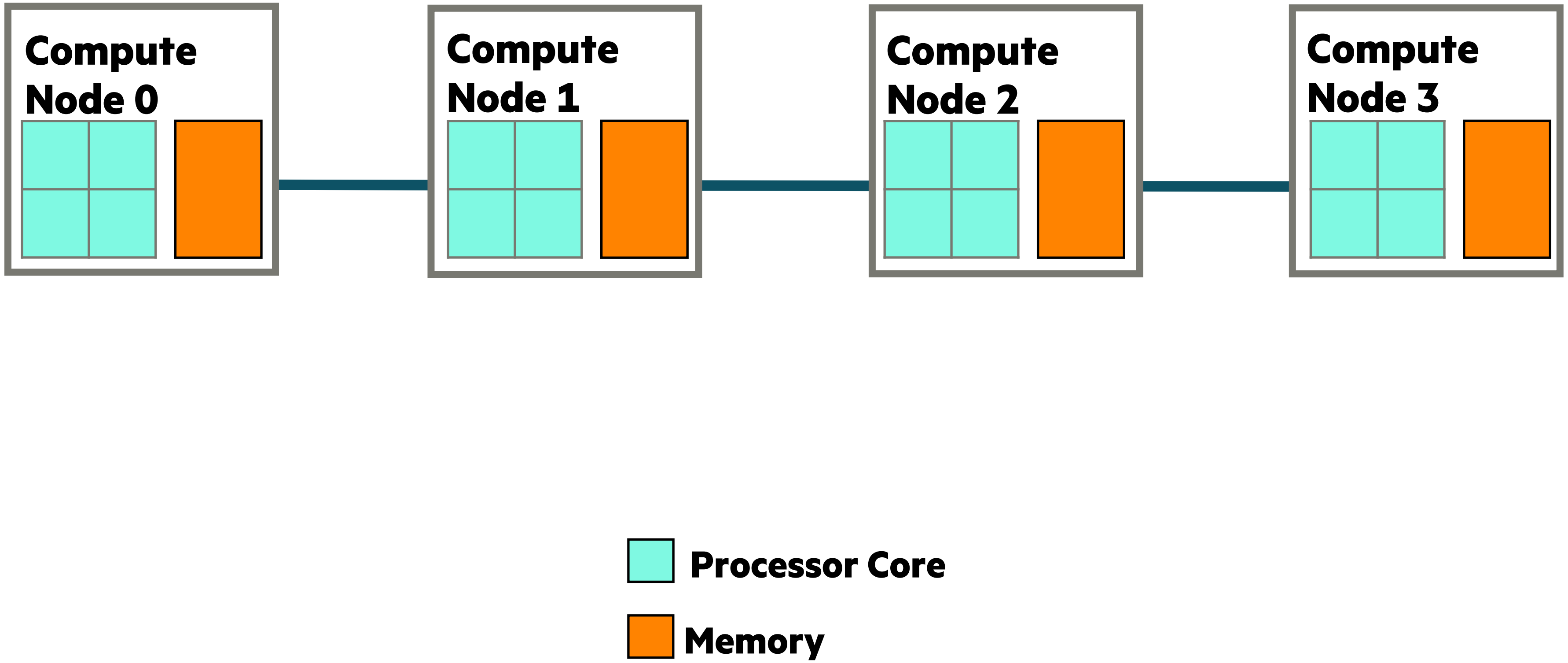
That said, as a general-purpose language, Chapel supports writing SPMD patterns as well:

```
coforall loc in Locales do
  on loc do
    myMain();

proc myMain() {
  // ... write your SPMD computation here ...
}
```

Chapel Features for Parallelism and Locality

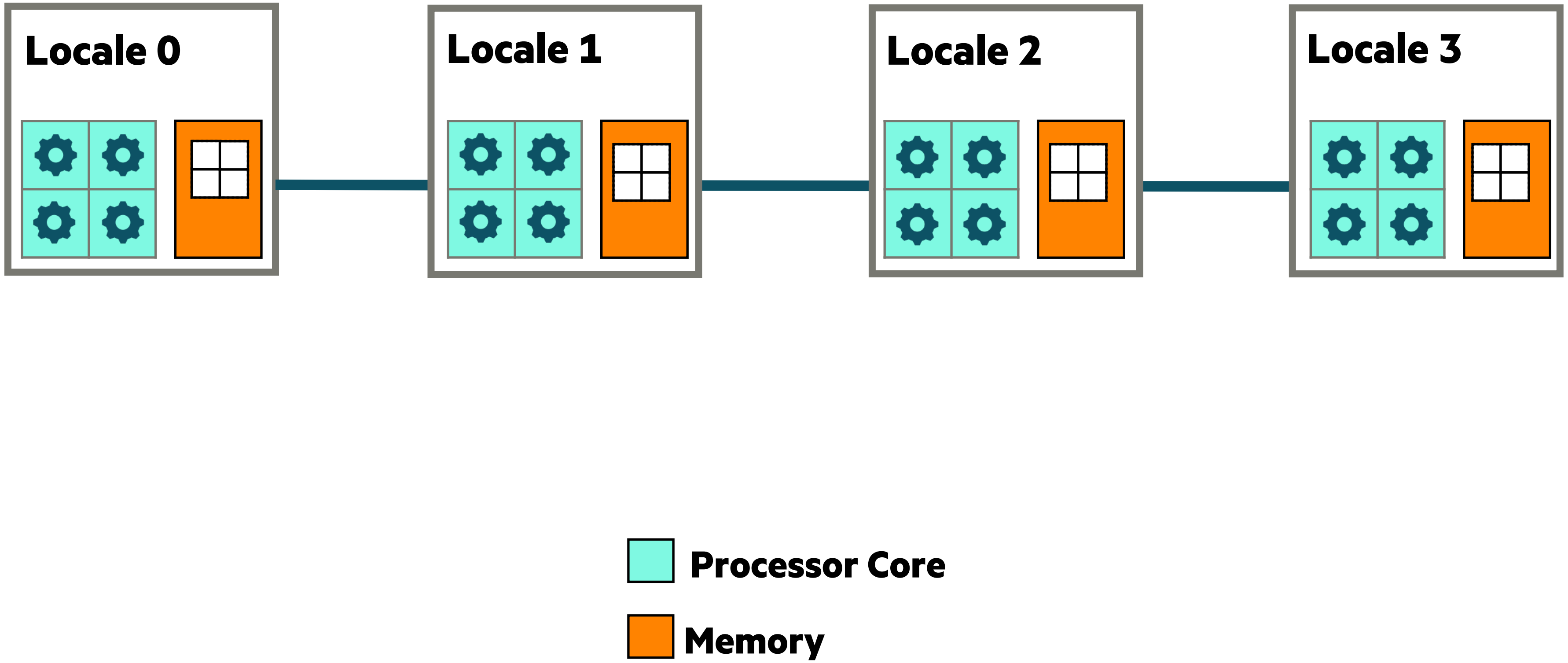
Locales in Chapel

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



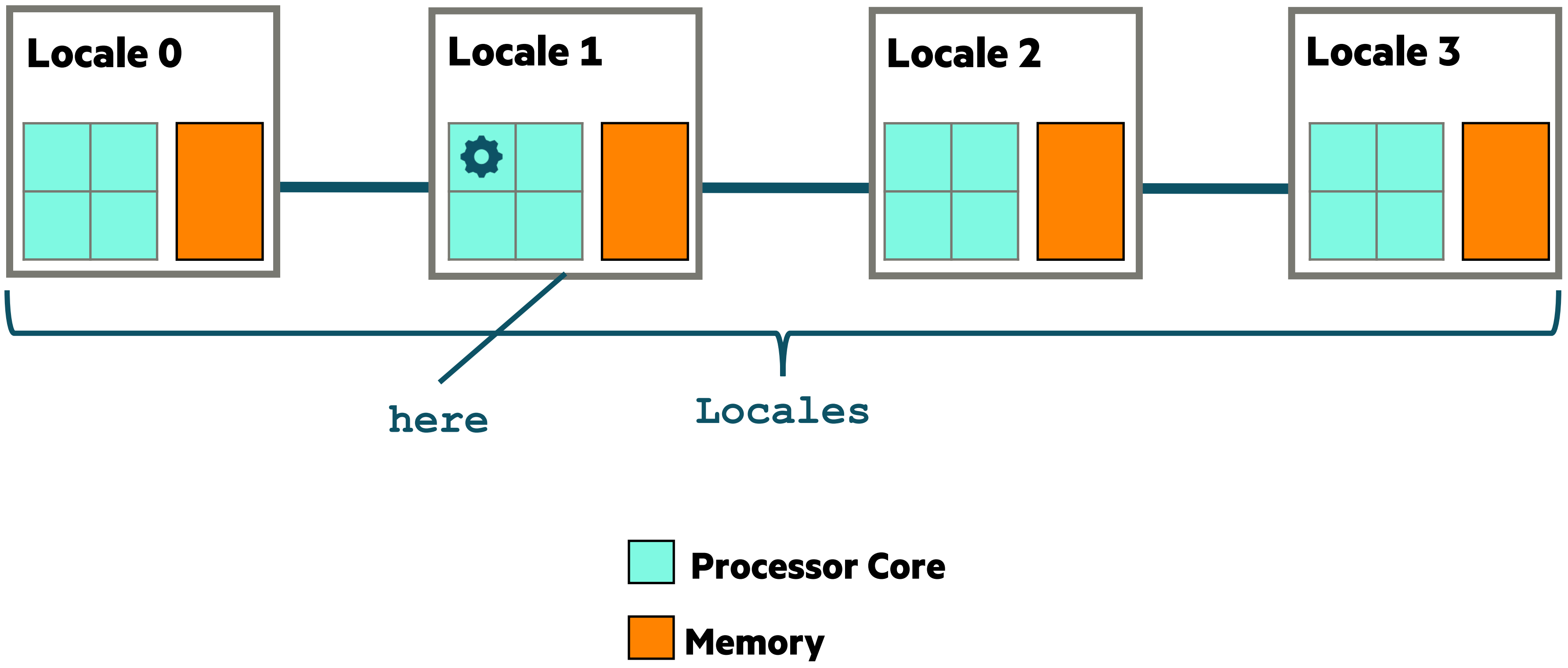
Key Concerns for Scalable Parallel Computing

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



Built-In Locale Variables in Chapel

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



Basic Features for Locality

basics-on.chpl

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

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

on Locales[1] {
  var B: [1..2, 1..2] real;

  B = 2 * A;
}
```

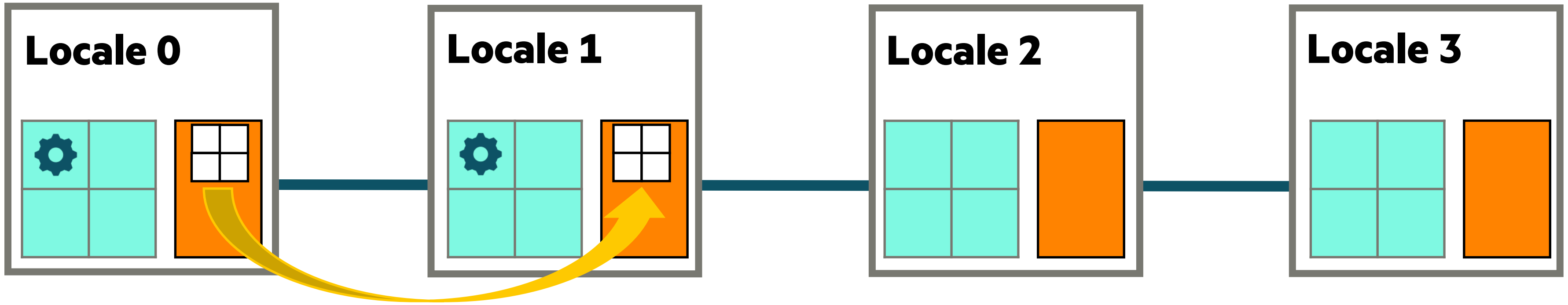
All Chapel programs begin running as a single task on locale 0

Variables are stored using the memory local to the current task

on-clauses move tasks to other locales

remote variables can be accessed directly

This is a serial, but distributed computation



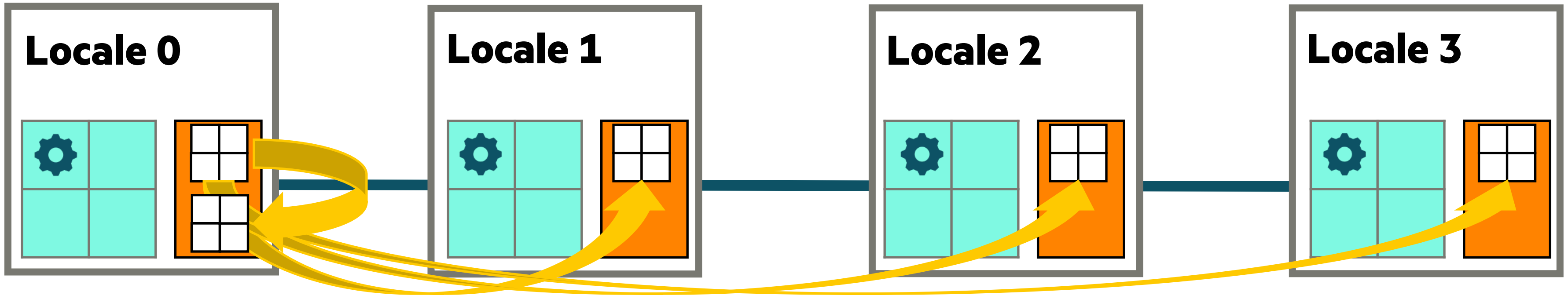
Basic Features for Locality

basics-for.chpl

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

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

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

This loop will serially iterate over the program's locales

This is also a serial, but distributed computation



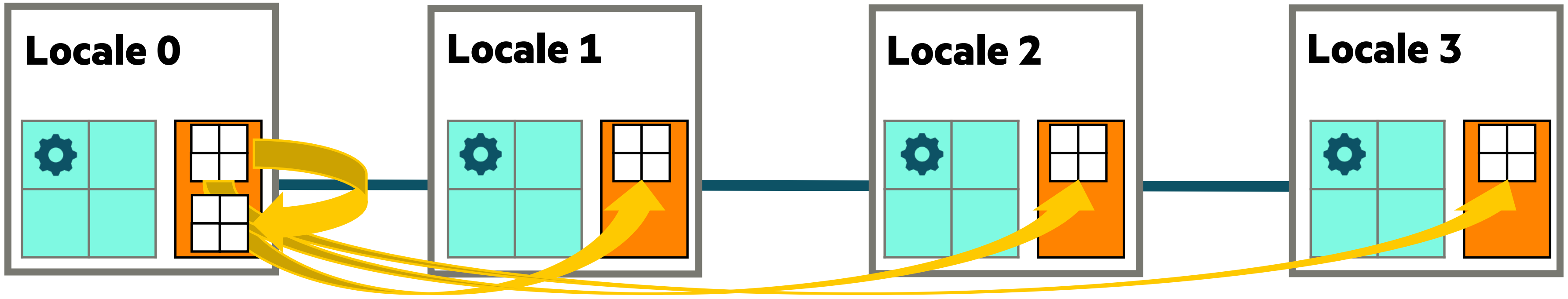
Mixing Locality with Task Parallelism

basics-coforall.chpl

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

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

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

The coforall loop creates a parallel task per iteration

This results in a parallel distributed computation



The Three Ways to Create Parallel Tasks in Chapel

begin: Creates a task to asynchronously execute the statement it prefixes

```
begin writeln("Hello, PLSE!");
writeln("Goodbye!");
```

cobegin: A compound statement in which each child statement is a distinct task

```
cobegin {
  writeln("Hello from task 1");
  writeln("Hello from task 2");
}
writeln("Goodbye!");   // original task waits for child tasks to complete before proceeding
```

coforall: A loop form in which each iteration is a distinct task

```
coforall i in 1..numTasks do
  writeln("Hello from task ", i, " of ", numTasks);
writeln("Goodbye!");   // original task waits for child tasks to complete before proceeding
```

Wait, what about ‘forall’ and ‘foreach’?

forall: Invokes a parallel iterator, itself written in terms of ‘coforall’, ‘cobegin’, and/or ‘begin’

```
forall i in 1..n do
  writeln("Hello from iteration ", i, " of ", n);
writeln("Goodbye!");
```

// notionally, the parallel iterator for a range looks something like this:

```
proc range.these(…) {
  const numTasks = computeNumTasks();
  coforall i in 0..<numTasks {
    const chunk = computeMyChunk(lo, hi, stride, numTasks);
    for j in chunk do
      yield j;
  }
}
```

foreach: Doesn’t introduce any tasks, just hints to the compiler that the loop may / should be parallelized

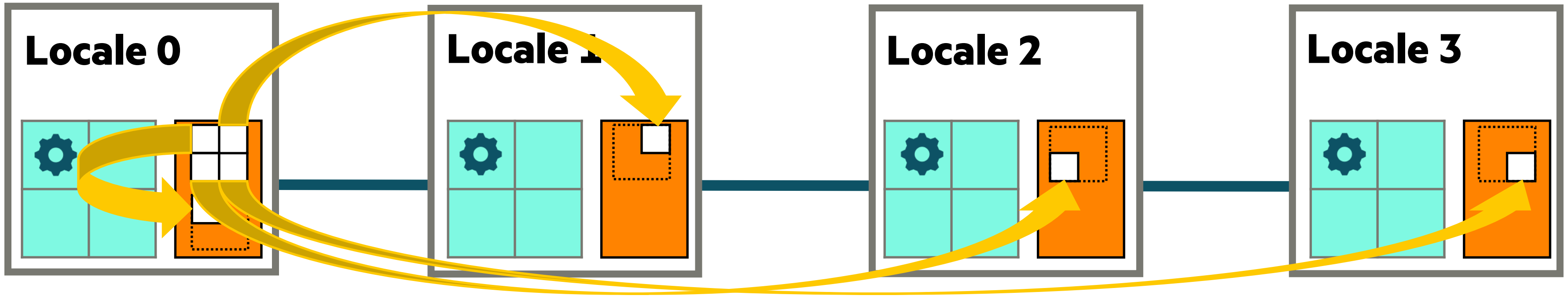
Array-based Parallelism and Locality

basics-distarr.chpl

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

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

use BlockDist;

var D = blockDist.createDomain({1..2, 1..2});
var B: [D] real;
B = A;
```

Chapel also supports distributed domains (index sets) and arrays

They also result in parallel distributed computation



Other Chapel Features

- Chapel is a big language
  - everything you'd expect from a modern, productive language
  - plus, additional features supporting parallelism, locality, and scalable performance

- As a result, there are many features you aren't seeing much of today:

  Serial Features:
  - **Modules:** for namespacing and code organization
  - **Procedures and iterators:** with overloading, generics/polymorphism, rich argument passing, …
  - **OOP:** Value- and Reference-based objects, generic types, inheritance, fields, methods, mix-ins, …
  - …

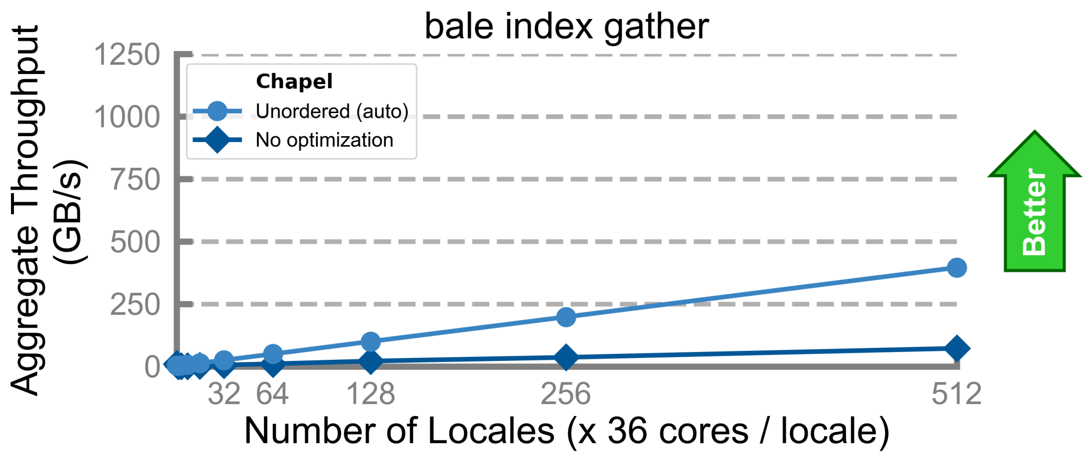
  Parallel Features:
  - **Rich array support:** multidimensional arrays, sparse arrays, slicing, rank change, reindexing, …
  - **Implicit forms of parallelism:** whole-array operations, promotion of scalar routines, reductions, scans
  - **Intra-task synchronization:** atomic and synchronization (full-empty) variables
  - …

Sample Compiler Optimizations (Bale IG Revisited)

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 -nl 512
$
```



Cray XC (Aries)

Bale IG in Chapel: Distributed Parallel Version (rewrite using parallel iterator)

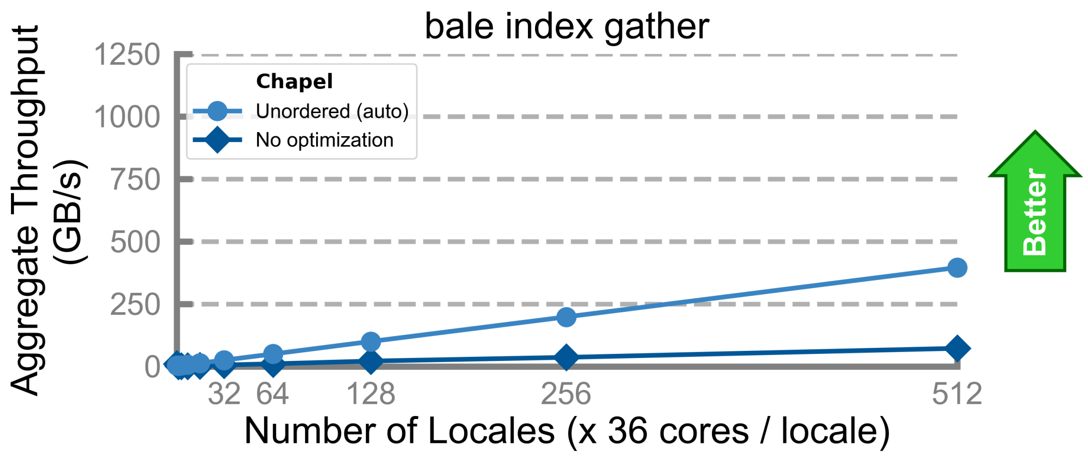
```
use BlockDist;

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];
```

Gets lowered roughly to…

```
$ chp
$ ./b
$
```

```
coforall loc in Dst.targetLocales do on loc do
  coforall tid in 0..<here.maxTaskPar do
    foreach idx in myInds(loc, tid, …) do
      Dst[idx] = Src[Inds[idx]];
```

- Create a task per compute node
- Create a task per core on that node
- Compute that task's gathers



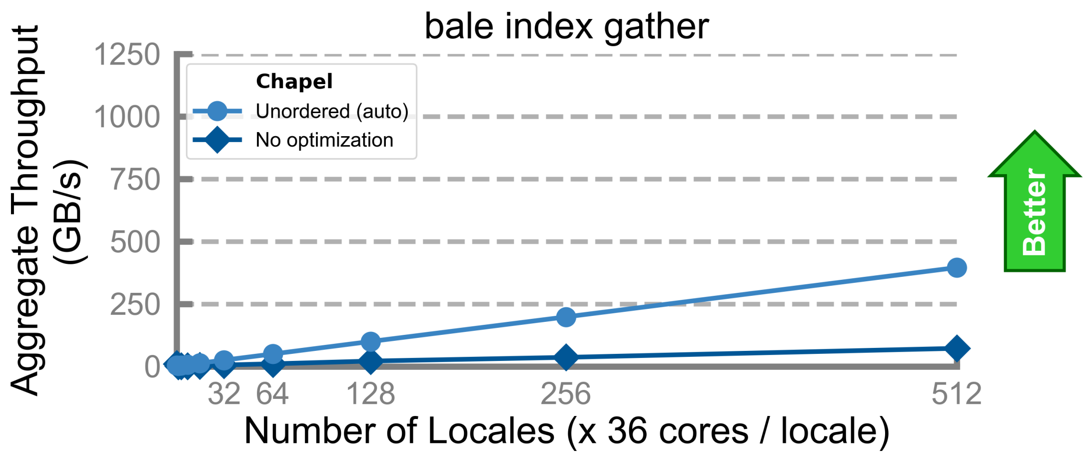
Cray XC (Aries)

Bale IG in Chapel: Distributed Parallel Version (optimized using async copies)

```
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];
```

```
$ chp
$ ./b
$
```

```
coforall loc in Dst.targetLocales do on loc do
  coforall tid in 0..<here.maxTaskPar do
    foreach idx in myInds(loc, tid, …) do
      Dst[idx] = Src[Inds[idx]];
```



Cray XC (Aries)

The user told us this loop was parallel, so why perform these high-latency ops serially?

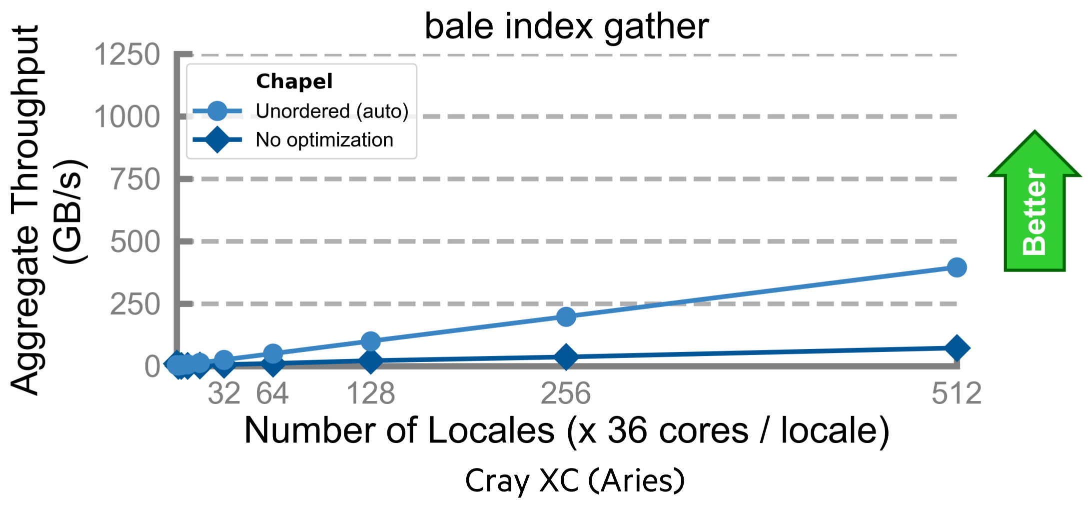
So, the Chapel compiler rewrites the inner loop to perform them asynchronously

```
foreach idx in myInds(loc, tid, …) do
  asyncCopy(Dst[idx], Src[Inds[idx]]);
asyncCopyTaskFence();
```

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 -nl 512
$
```

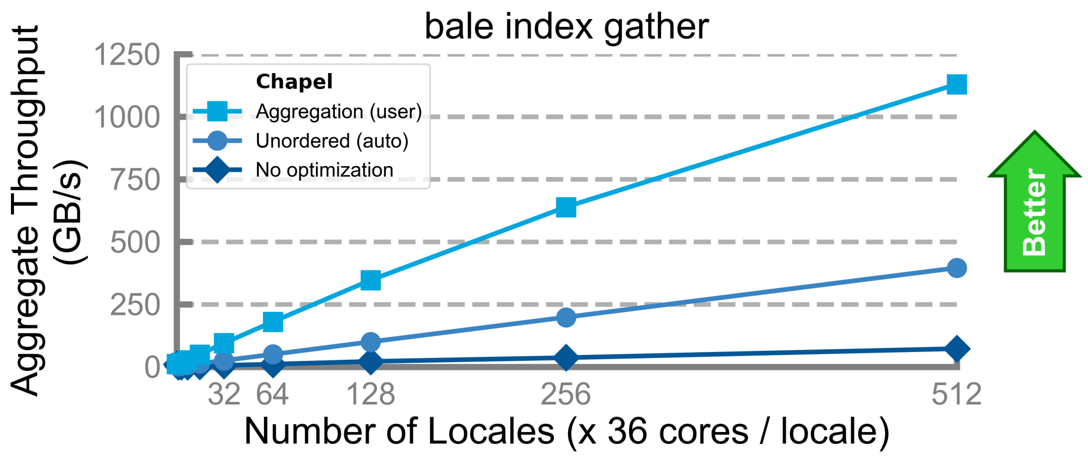


So far, all communications are being done in a fine-grained manner, an element at a time

Bale IG in Chapel: Distributed, Explicitly Aggregated Version

```
use BlockDist, CopyAggregation;

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) with
 (var agg = new SrcAggregator(int)) do
  agg.copy(d, Src[i]);
```

```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 512
$
```

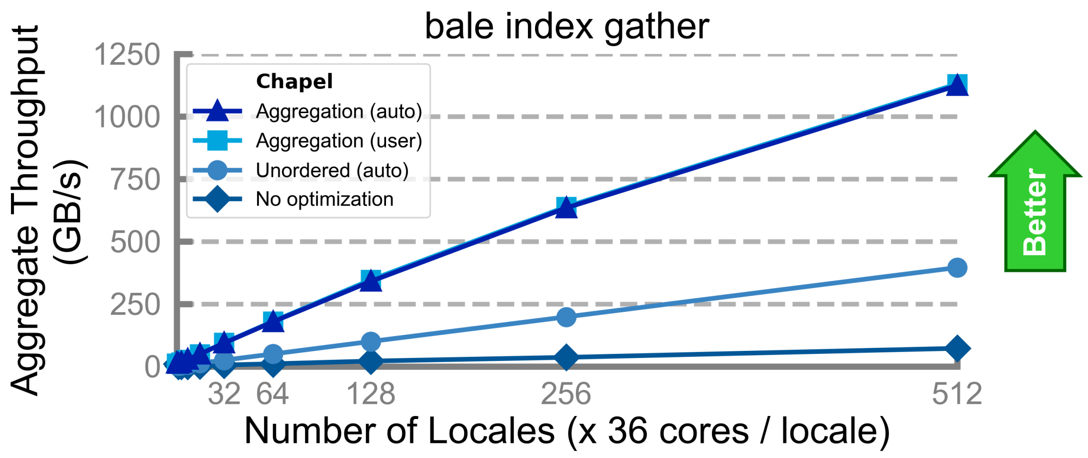


Cray XC (Aries)

Bale IG in Chapel: Distributed, Auto-Aggregated 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 --auto-aggregation
$ ./bale-ig -nl 512
$
```



Cray XC (Aries)

Bale IG in Chapel vs. SHMEM on Cray XC

Chapel (Simple / Auto-Aggregated version)

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

Chapel (Explicitly Aggregated version)

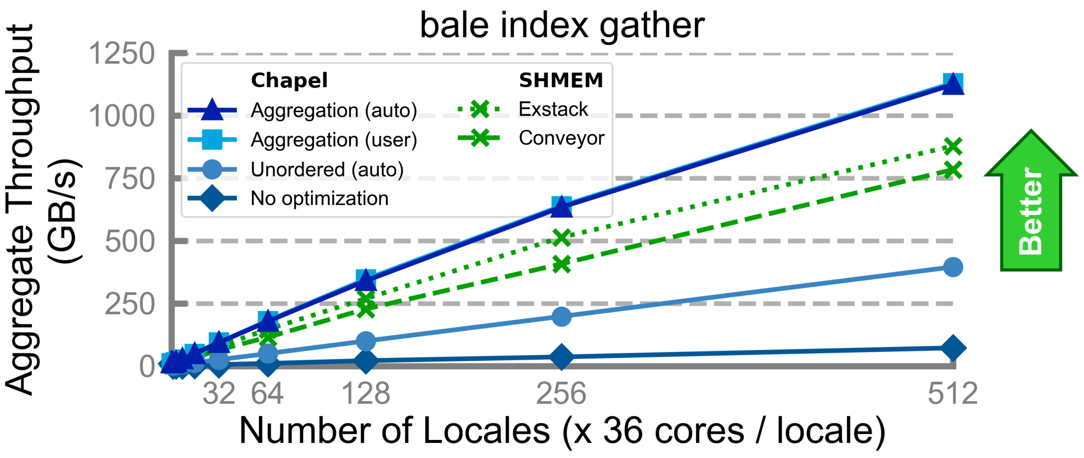
```
forall (d, i) in zip(Dst, Inds) with
 (var agg = new SrcAggregator(int)) do
  agg.copy(d, Src[i]);
```

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;
}
```



Cray XC (Aries)

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

Chapel (Simple / Auto-Aggregated version)

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

Chapel (Explicitly Aggregated version)

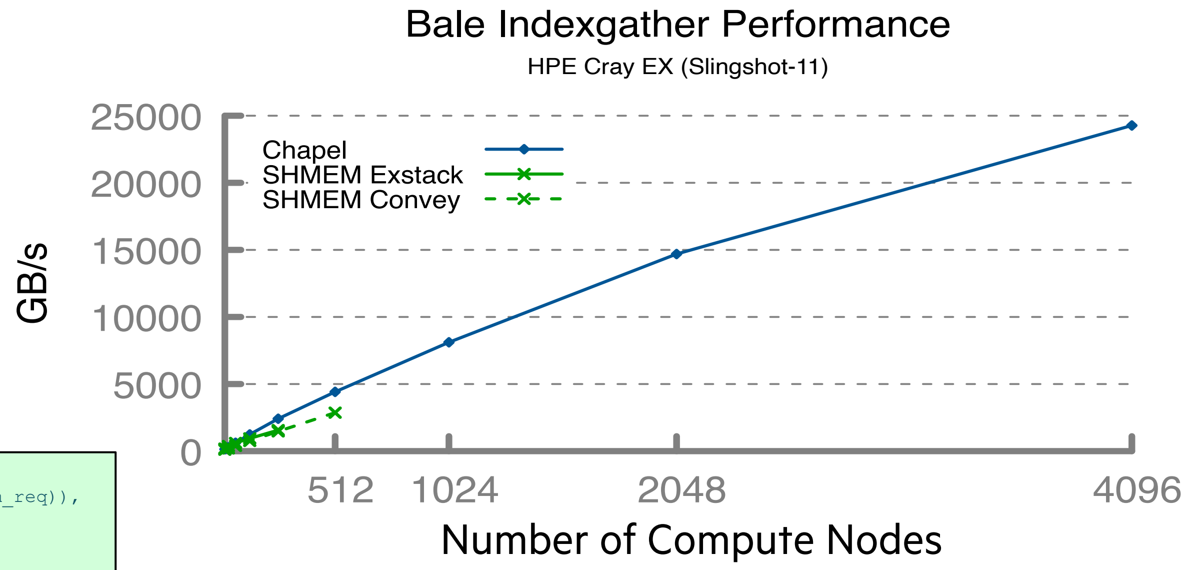
```
forall (d, i) in zip(Dst, Inds) with
 (var agg = new SrcAggregator(int)) do
  agg.copy(d, Src[i]);
```

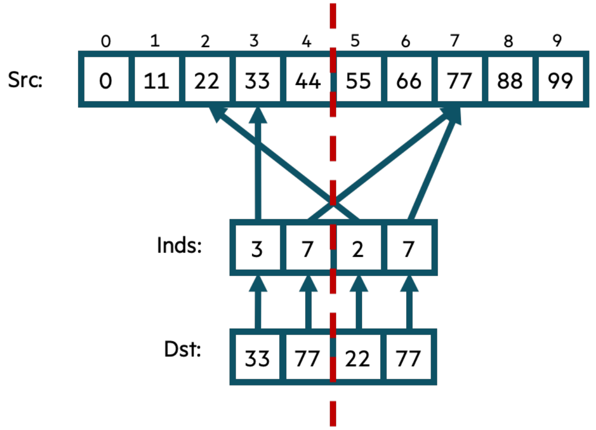
SHMEM (Exstack version)

```
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;
}
```

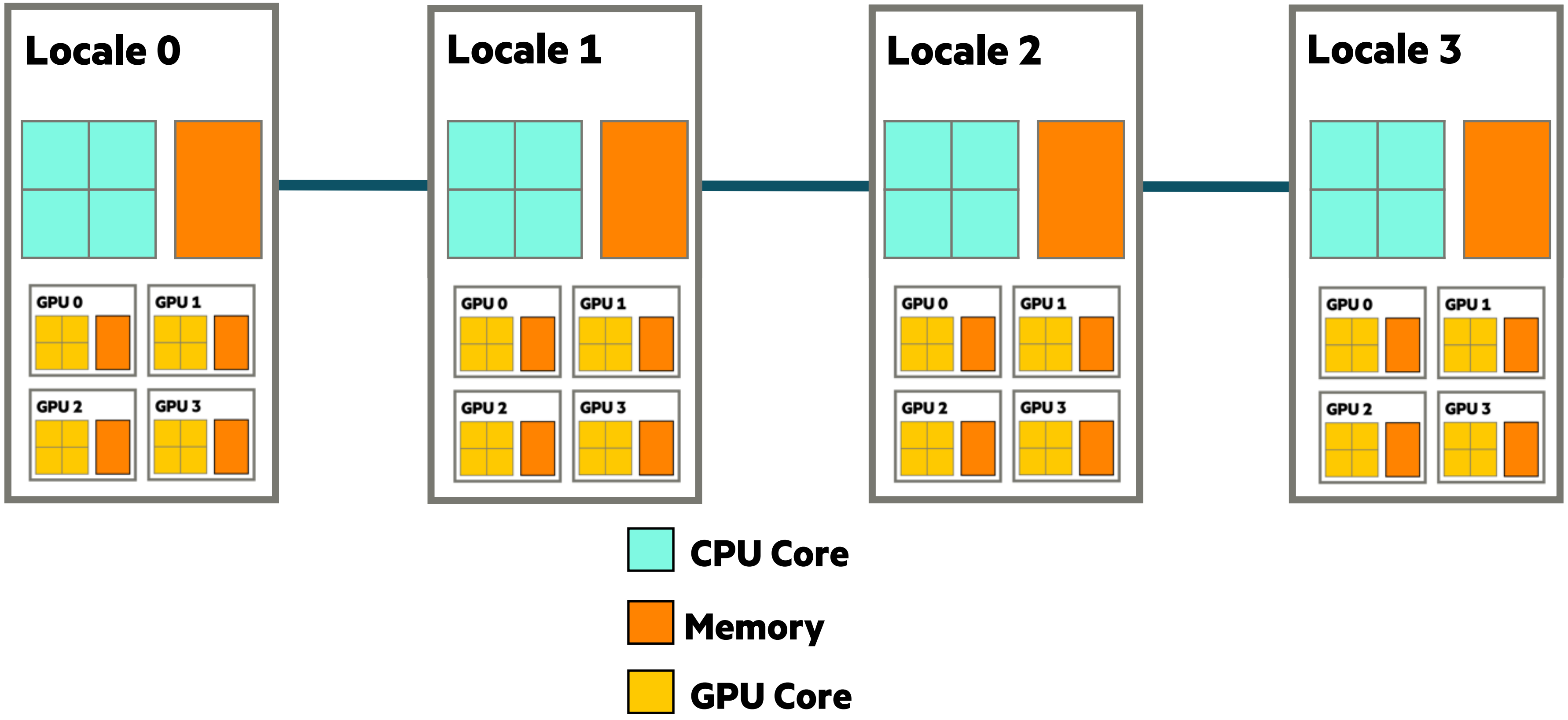




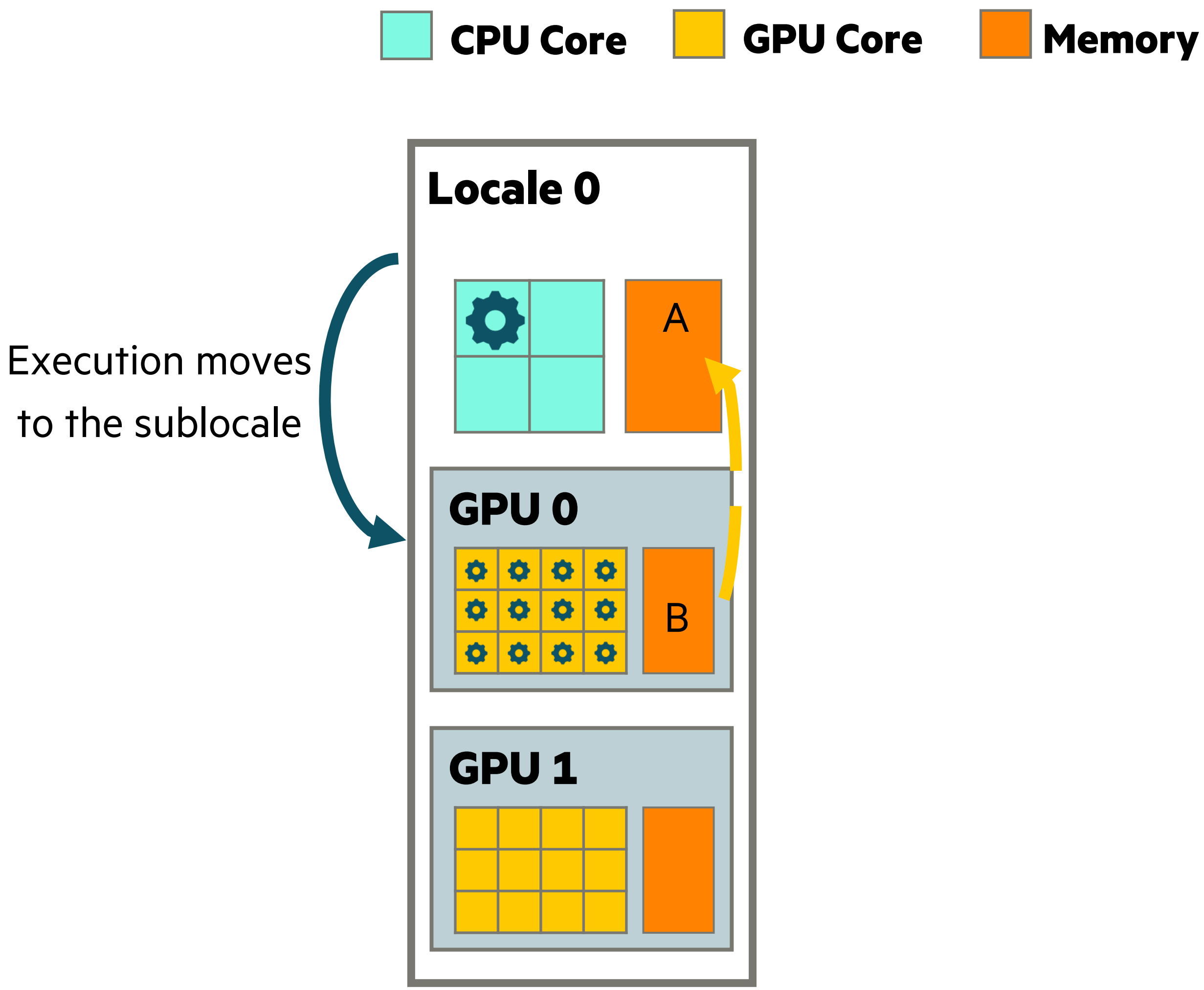
Programming GPUs with Chapel

Representing GPUs in Chapel

- In Chapel, a *locale* refers to a compute resource with processors and memory
  - For now, think of each compute node as being a locale
- Modern systems often involve GPUs as well
  - In Chapel, we represent them as *sub-locales*

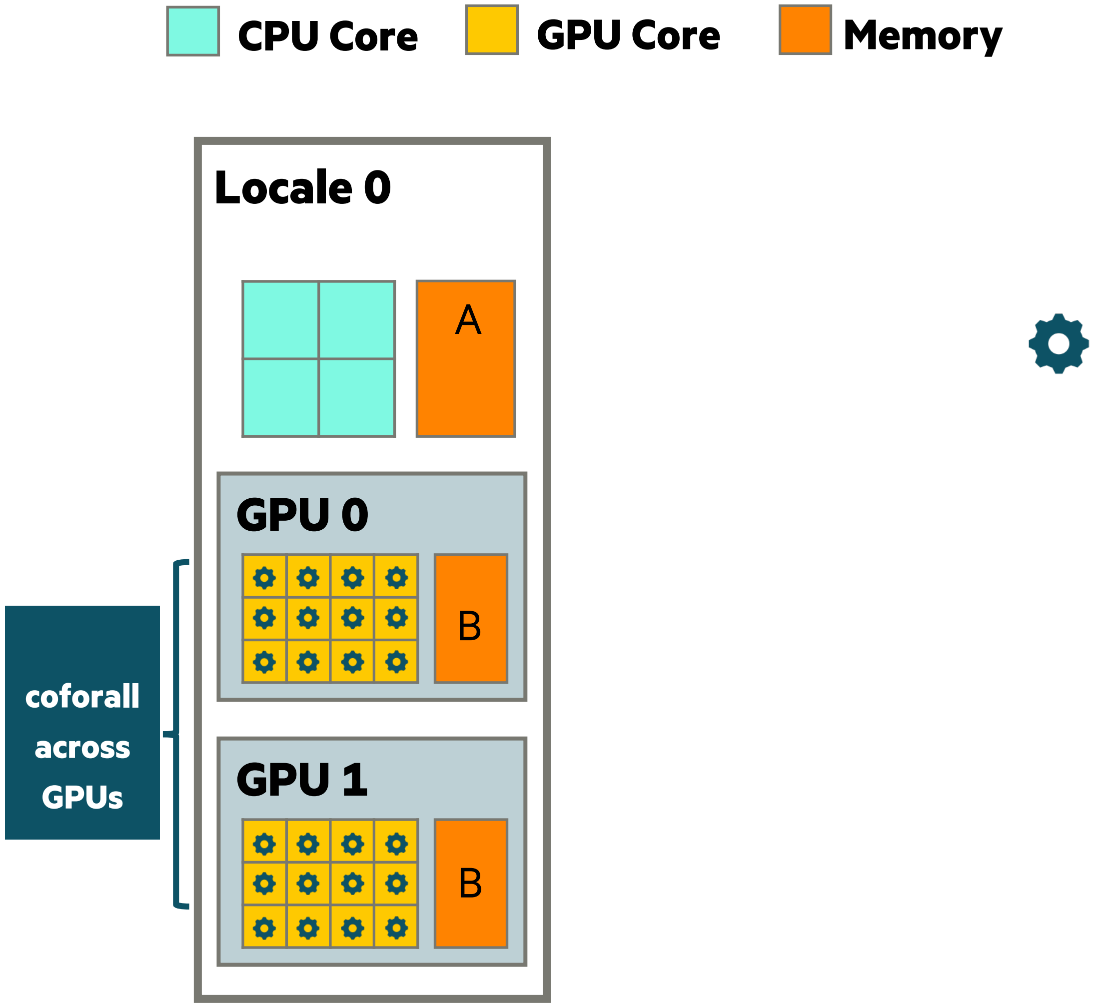


Parallelism and Locality In The Context Of GPUs



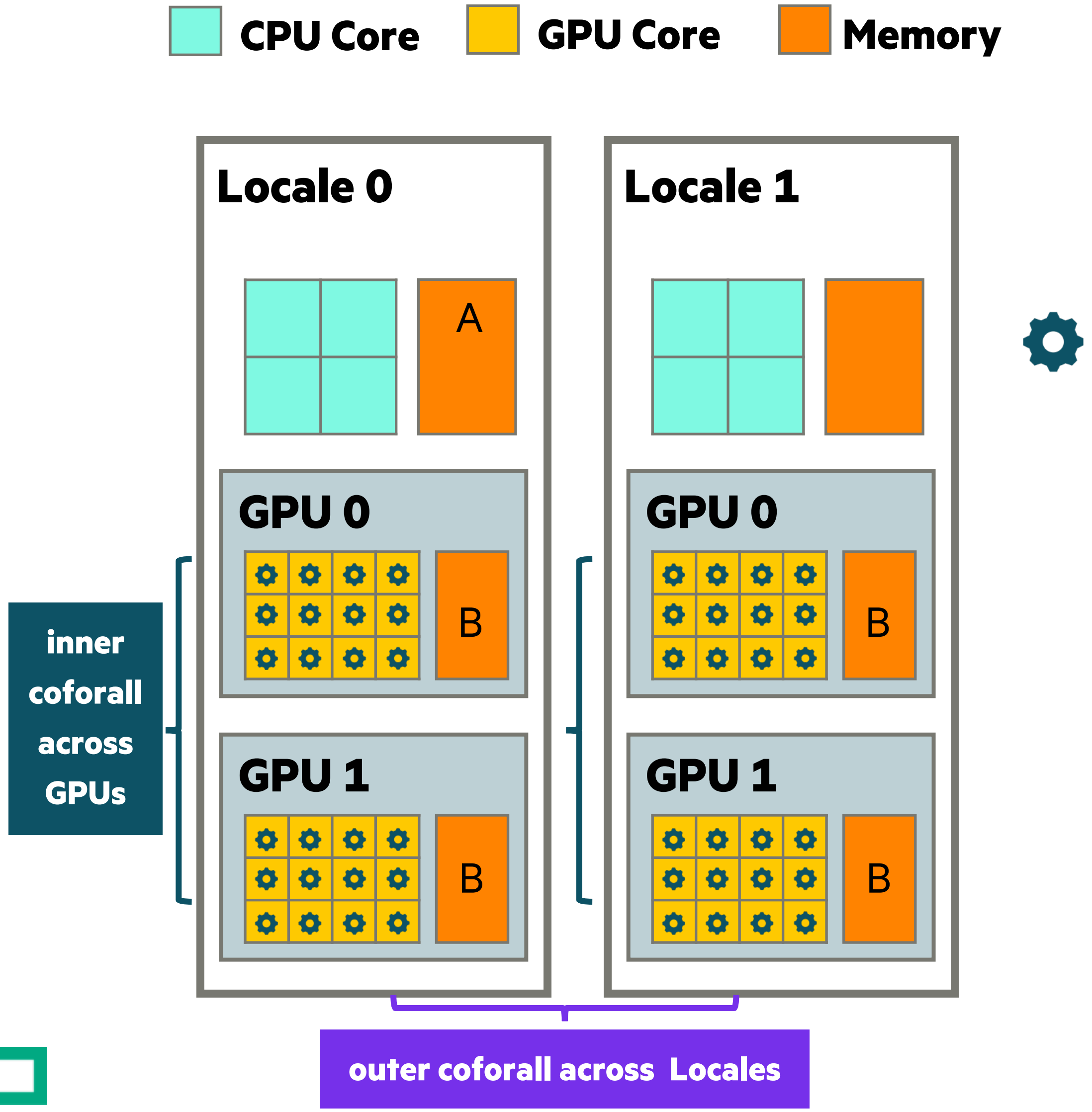

```
var A: [1..n, 1..n] real;

  on here.gpus[0] {
    var B: [1..n, 1..n] real;
    B = 2;
    A = B;
  }

writeln(A);
```

Parallelism and Locality In The Context Of GPUs



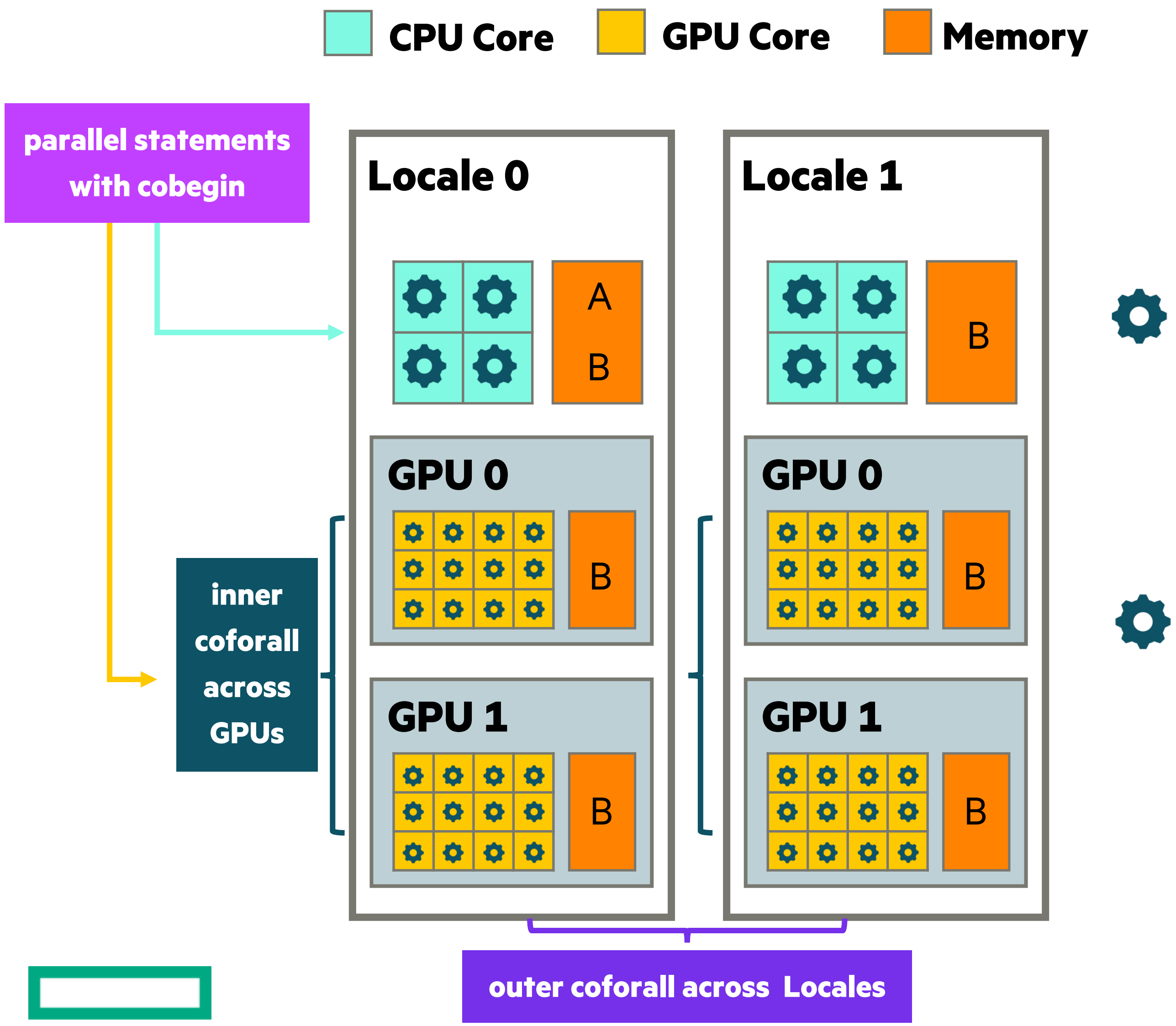

```
var A: [1..n, 1..n] real;

coforall g in here.gpus do on g {
  var B: [1..n, 1..n] real;
  B = 2;
  A = B;
}

writeln(A);
```

Parallelism and Locality In The Context Of GPUs




```
var A: [1..n, 1..n] real;
coforall l in Locales do on l {

  coforall g in here.gpus do on g {
    var B: [1..n, 1..n] real;
    B = 2;
    A = B;
  }

}
writeln(A);
```

Parallelism and Locality In The Context Of GPUs

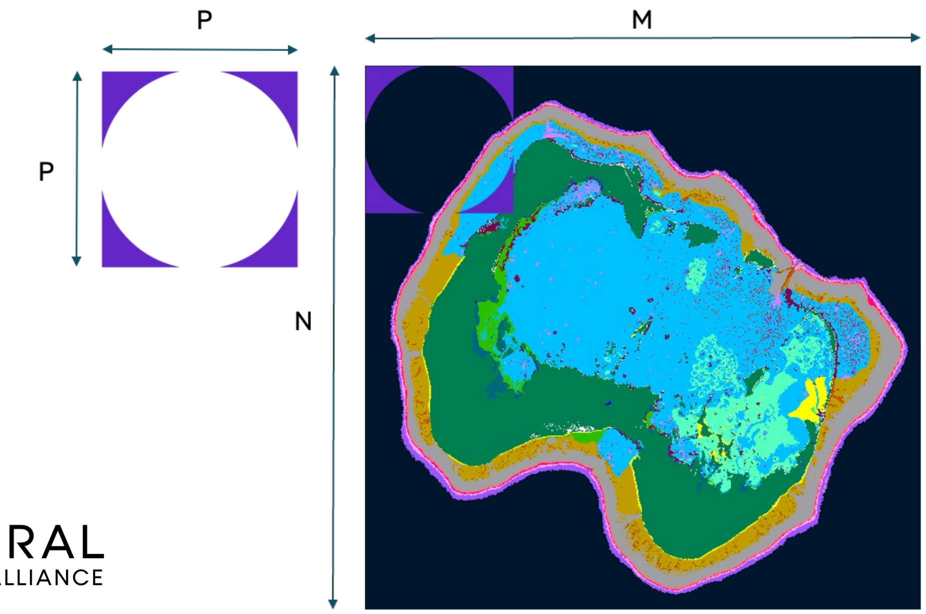



```
var A: [1..n, 1..n] real;
coforall l in Locales do on l {
 cobegin {
  coforall g in here.gpus do on g {
    var B: [1..n, 1..n] real;
    B = 2;
    A = B;
  }
  {
    var B: [1..n, 1..n] real;
    B = 2;
    A = B;
  }
 }
}
writeln(A);
```

RapidQ Coral Biodiversity Summary

What is it?

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

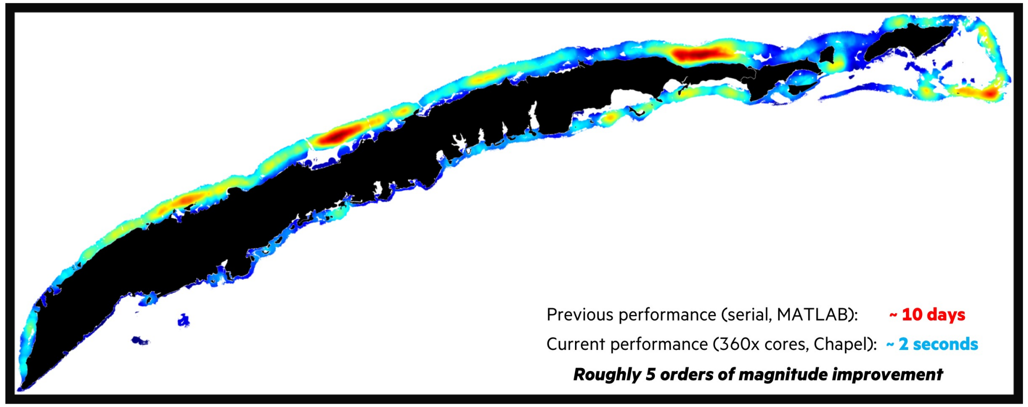


Who wrote it?

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

Why Chapel?

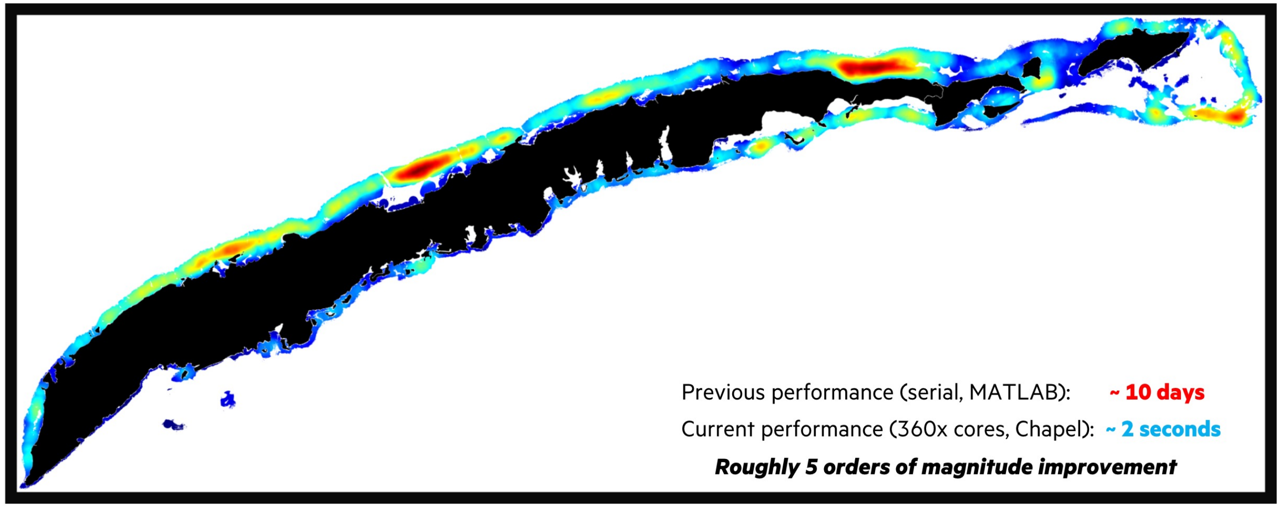
- easy transition from Python, which was being used
- massive performance improvement:
  ~10-day Python run finished in ~2 seconds using 360 cores
- enabled unexpected algorithmic improvements



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

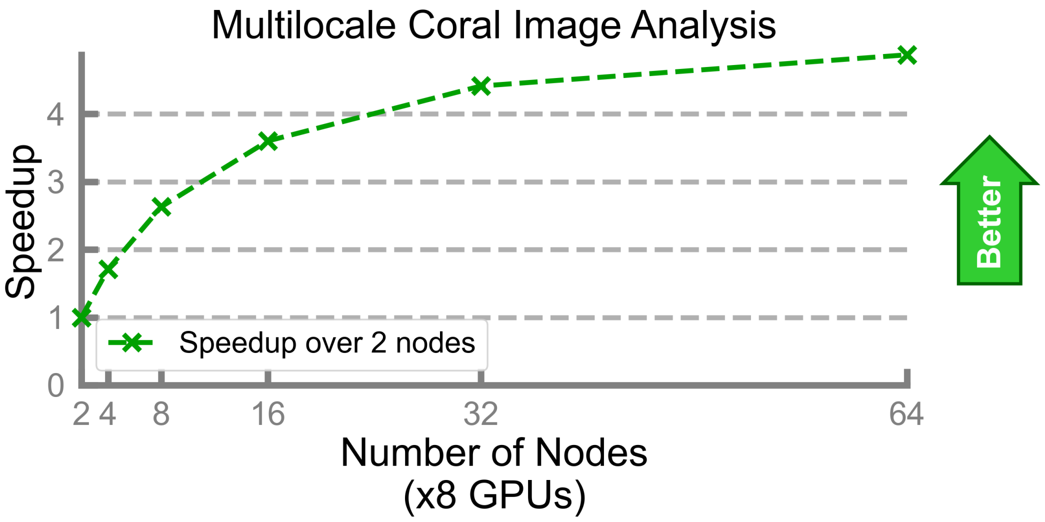
Coral Reef Spectral Biodiversity: Productivity and Performance

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



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

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



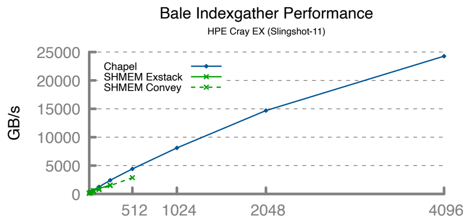
Wrap-up

Chapel Summary

Chapel is unique among programming languages

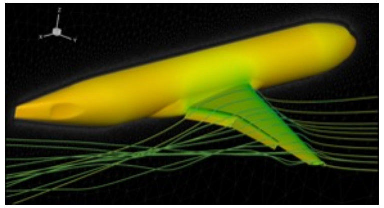
- built-in features for scalable parallel computing make it HPC-ready
- supports clean, concise code relative to conventional approaches
- ports and scales from laptops to supercomputers
- supports GPUs in a vendor-neutral manner

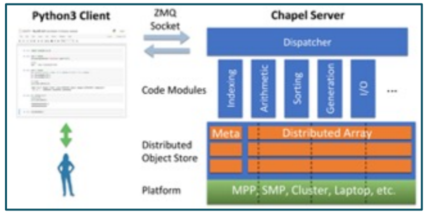
```
use BlockDist;

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];
```

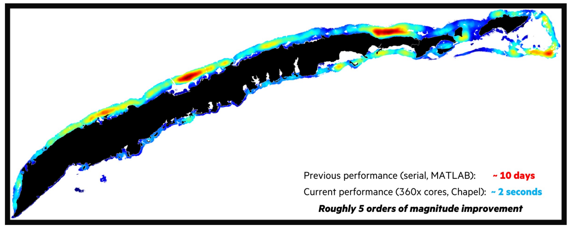


Chapel is being used for productive parallel computing at scale

- users are reaping its benefits in practical, cutting-edge applications
- applicable to domains as diverse as physical simulations and data science
- Arkouda is a particularly unique example of driving HPCs from Python







Takeaways for this PLSE setting

For scalable parallel computing, good language design can…

…**provide built-in abstractions** to simplify the expression of parallel operations

- e.g., parallel loops and iterators, global namespace

…**more clearly represent parallel computations** compared to standard approaches

- e.g., MPI, SHMEM, CUDA, HIP, SYCL, OpenMP, OpenCL, OpenACC, Kokkos, RAJA, …

…permit users to **create new abstractions** supporting performance and/or clean code

- e.g., per-task aggregators

…**enable new optimization opportunities** by expressing parallelism and locality clearly

- e.g., asynchronous operations, auto-aggregation of communication

…**support excellent performance and scalability**

- e.g., to thousands of nodes and over a million cores

The Chapel Team at HPE

Ways to Engage with the Chapel Community

Live/Virtual Events

- ChapelCon (formerly CHIUW), annually
- Office Hours, monthly
- Live Demo Sessions, monthly

Electronic Broadcasts

- Chapel Blog, ~biweekly
- Community Newsletter, quarterly
- Announcement Emails, around big events

Community / User Forums

- Discord
- Discourse
- Email Contact Alias — chapel+qs@discoursemail.com
- GitHub Issues
- Gitter
- Reddit
- Stack Overflow

Social Media

- Bluesky
- Facebook
- LinkedIn
- Mastodon
- X / Twitter
- YouTube

Chapel Website

chapel-lang.org

Thank you

https://chapel-lang.org
@ChapelLanguage