Hewlett Packard
Enterprise

Chapel: Accessible Parallel Programming from the Desktop to the Supercomputer

Brad Chamberlain

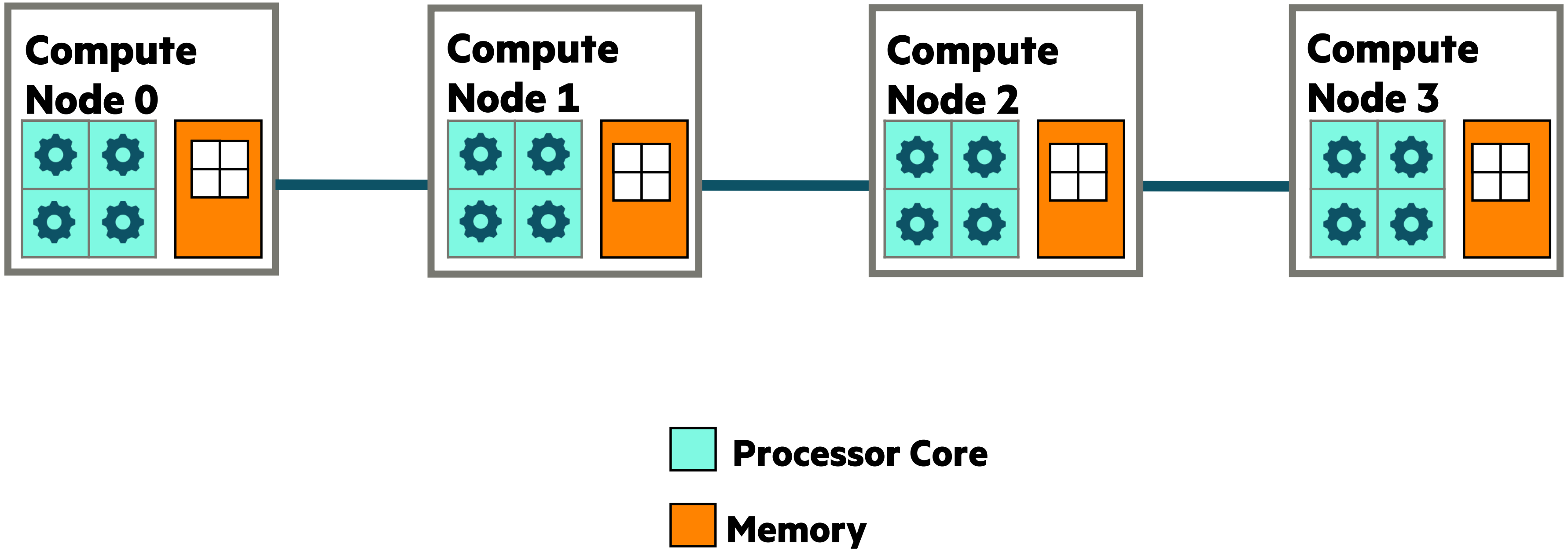
KAUST/KSL seminar
May 13, 2025

A Bit About Me

Parallel Computing in a nutshell

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

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



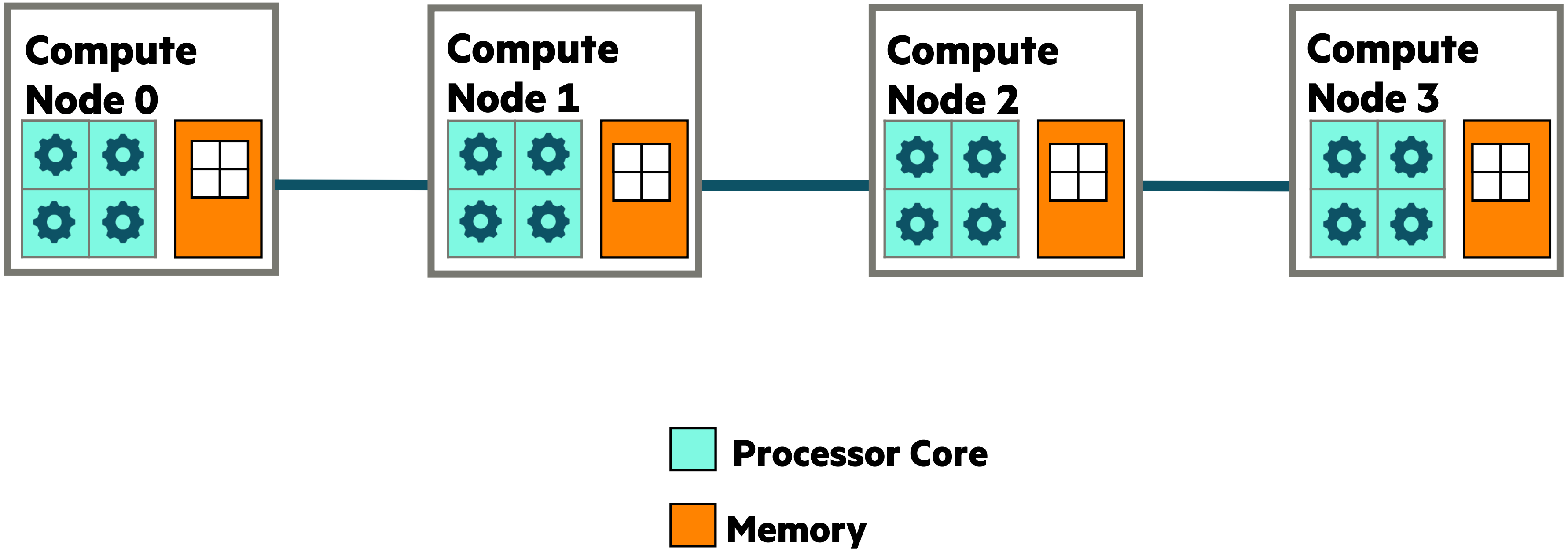
Parallel Computing has become Ubiquitous

Historical parallel computing:

- supercomputers
- commodity clusters

Today, we also have parallelism readily available:

- multicore processors
- GPUs
- cloud computing



What is Chapel?

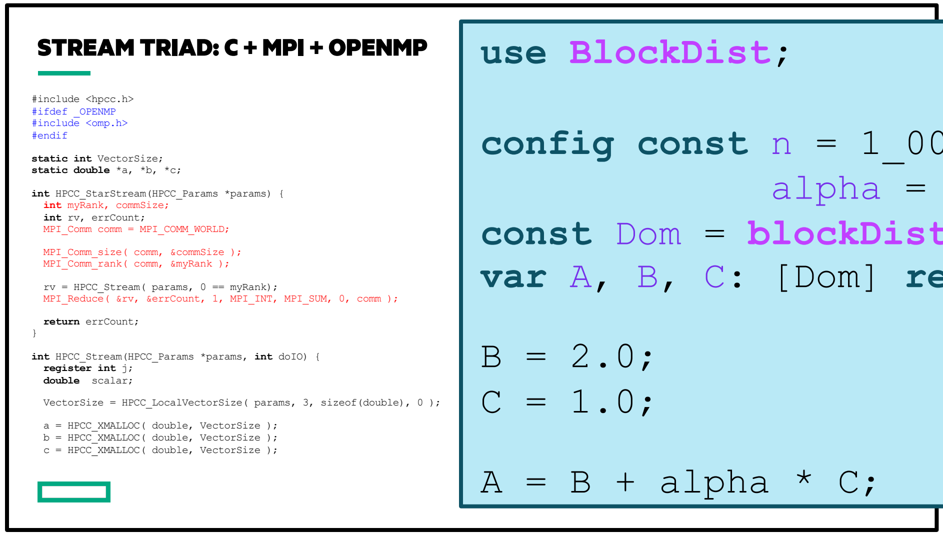
Chapel: A modern parallel programming language

- Portable & scalable
- Open-source & collaborative

Goals:

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

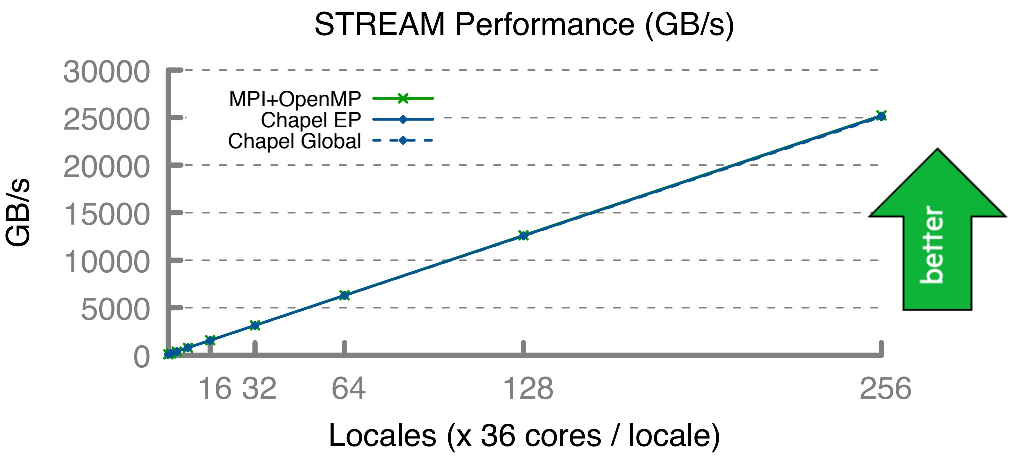
HPCC Stream Triad and RA in C + MPI + OpenMP vs. Chapel

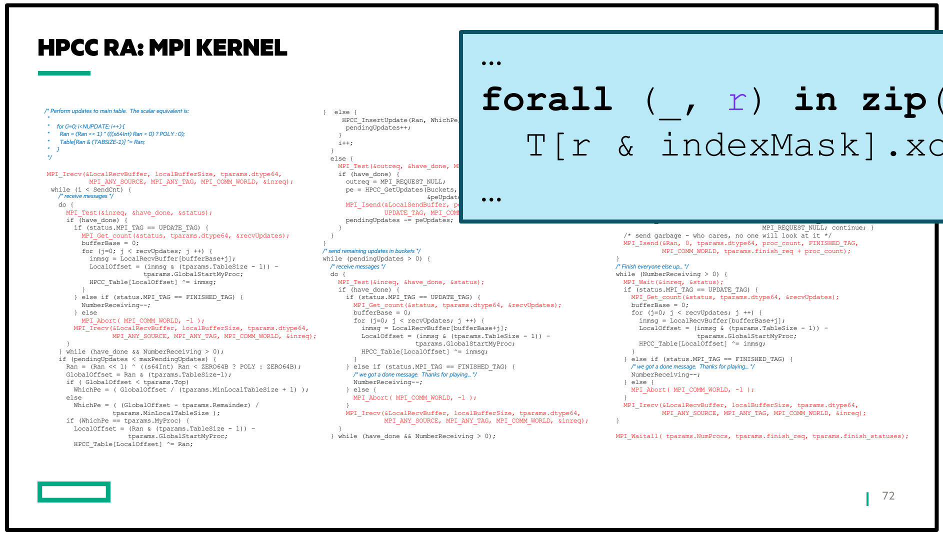


STREAM TRIAD: C + MPI + OPENMP

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

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

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

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

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

  return errCount;
}

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

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

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

```
use BlockDist;

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

B = 2.0;
C = 1.0;

A = B + alpha * C;
```





HPCC RA: MPI KERNEL

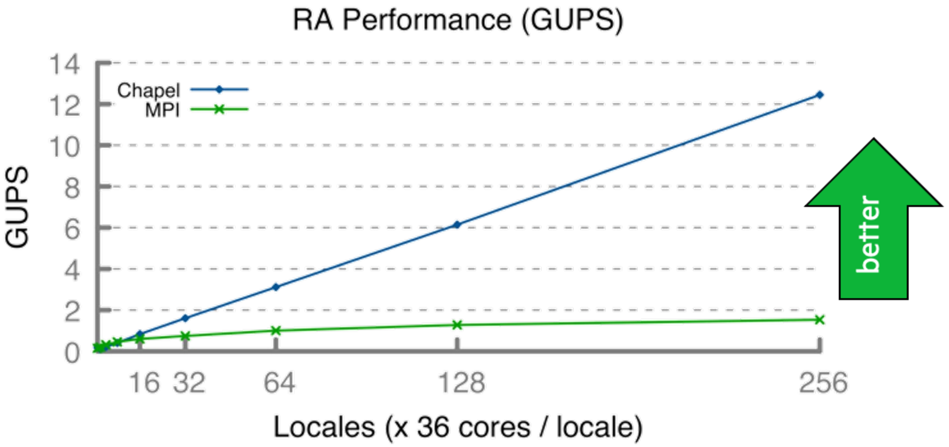
```
/* Perform updates to main table.  The scalar equivalent is:
 *
 *   for (i=0; i<NUPDATE; i++) {
 *     Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *     Table[Ran & (TABSIZE-1)] ^= Ran;
 *   }
 */

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,
     pendingUpdates++;
    }
    i++;
  }
  else {
    MPI_Test(&outreq, &have_done, MP
    if (have_done) {
      outreq = MPI_REQUEST_NULL;
      pe = HPCC_GetUpdates(Buckets,
                           &peUpdate
      MPI_Isend(&LocalSendBuffer, pe
                UPDATE_TAG, MPI_COM
      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_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);
```

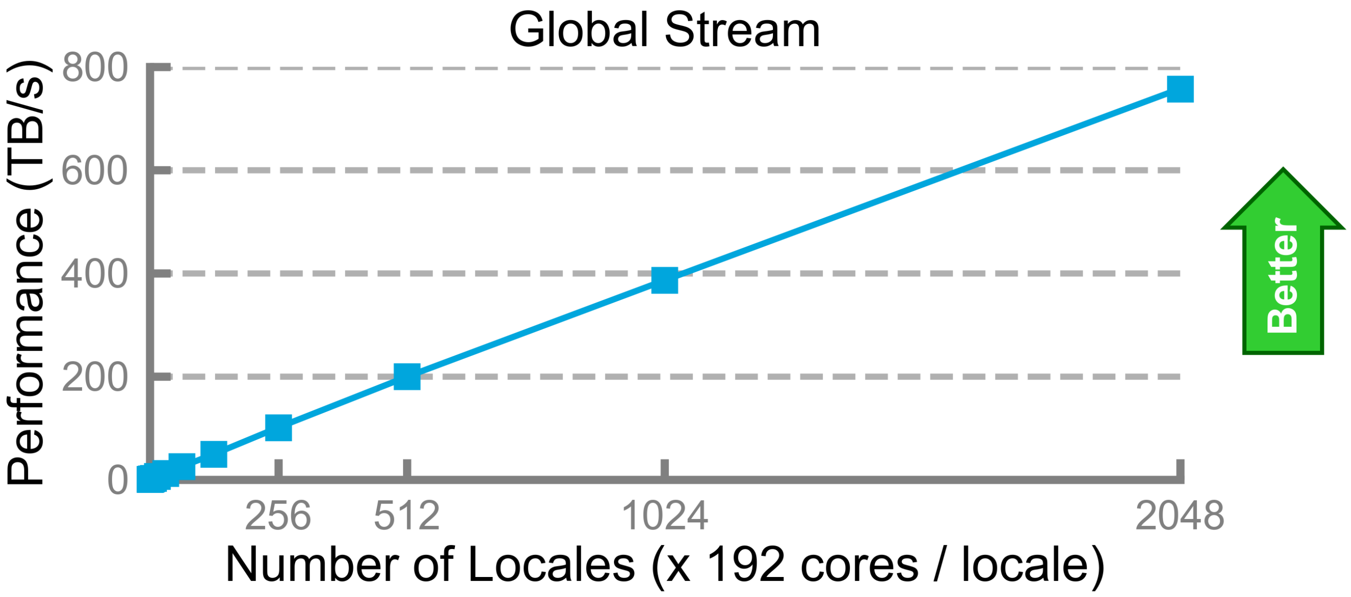
72

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



HPCC Stream Triad in Chapel on Shaheen (Initial Results)

```
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;
```



Accessible Parallel Programming: A Possible 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]

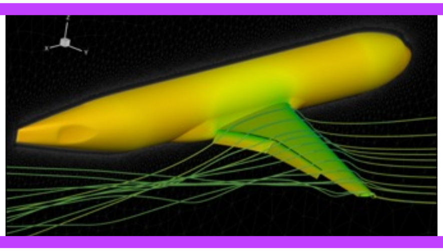
This is our motivation for Chapel

Outline

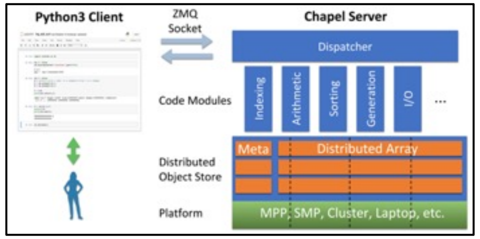
- Why Chapel?
- Applications of Chapel
- Global-view vs. SPMD Programming
- A Brief Introduction to Chapel, by Example (time permitting)
- Wrap-up

Applications of Chapel

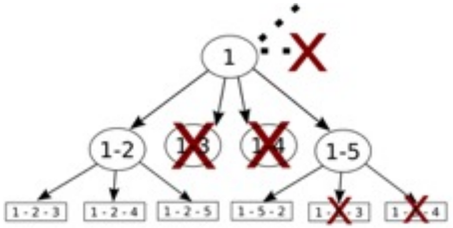
Applications of Chapel



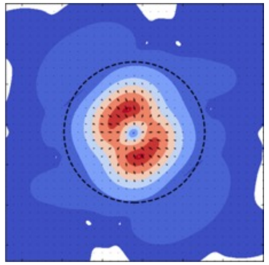
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



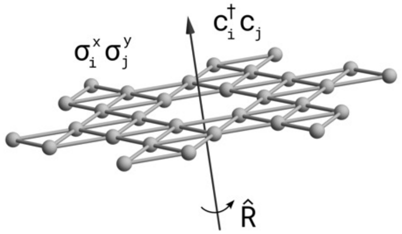
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



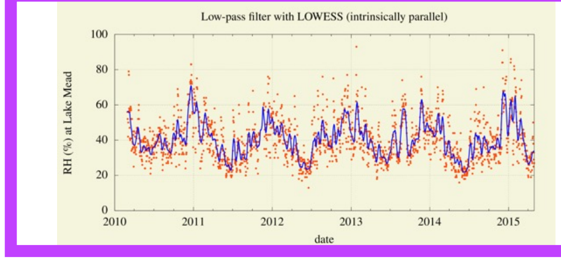
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

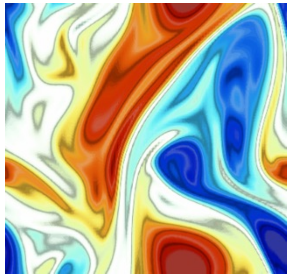


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

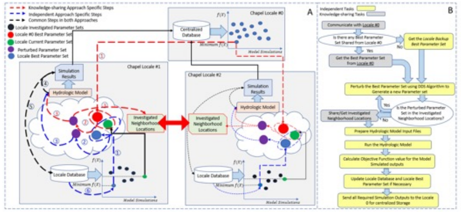


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

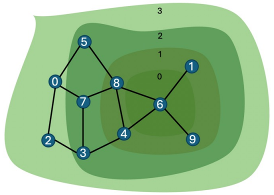
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



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

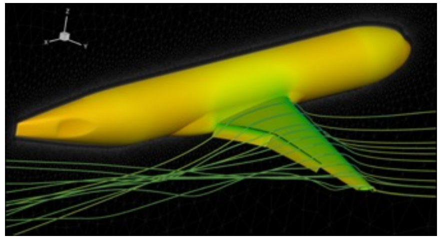
Modeling Ocean Carbon Dioxide Removal
Scott Bachman Brandon Neth, et al.
[C]Worthy

?

Your Application Here?
Your name here
KAUST

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

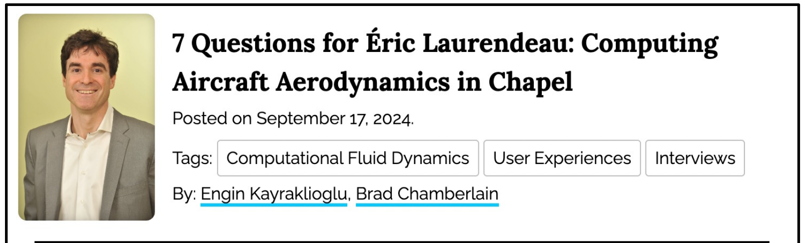
Productivity Across Diverse Application Scales (code and system size)



Computation: Aircraft simulation / CFD

Code size: 100,000+ lines

Systems: Desktops, HPC systems



7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

Posted on September 17, 2024.

Tags: Computational Fluid Dynamics | User Experiences | Interviews

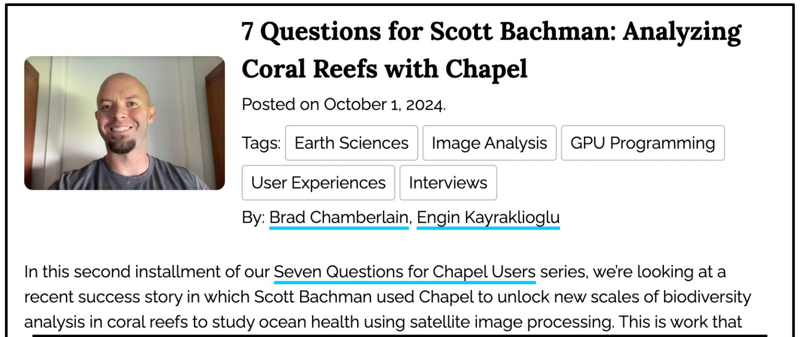
By: Engin Kayraklioglu, Brad Chamberlain

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

Computation: Coral reef image analysis

Code size: ~300 lines

Systems: Desktops, HPC systems w/ GPUs



7 Questions for Scott Bachman: Analyzing Coral Reefs with Chapel

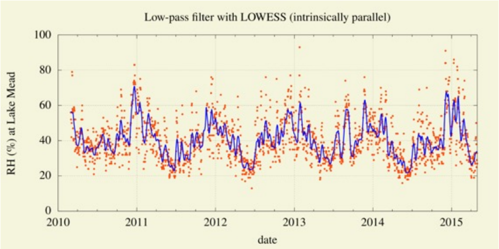
Posted on October 1, 2024.

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

By: Brad Chamberlain, Engin Kayraklioglu

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

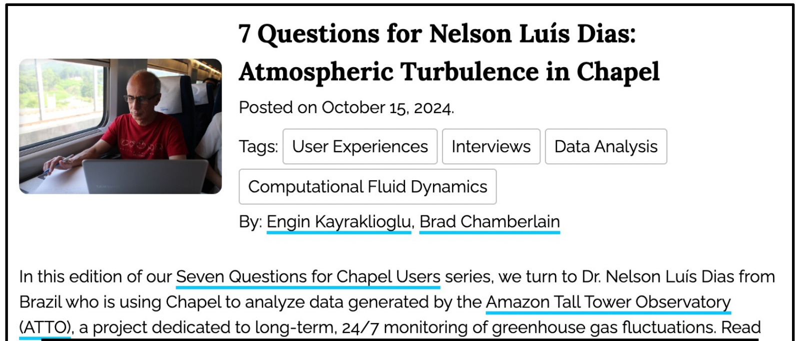
> *"With the coral reef program, I was able to speed it up by a factor of 10,000. Some of that was algorithmic, but Chapel had the features that allowed me to do it."*



Computation: Atmospheric data analysis

Code size: 5000+ lines

Systems: Desktops, sometimes w/ GPUs



7 Questions for Nelson Luís Dias: Atmospheric Turbulence in Chapel

Posted on October 15, 2024.

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

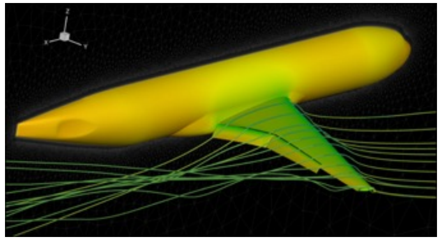
By: Engin Kayraklioglu, Brad Chamberlain

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

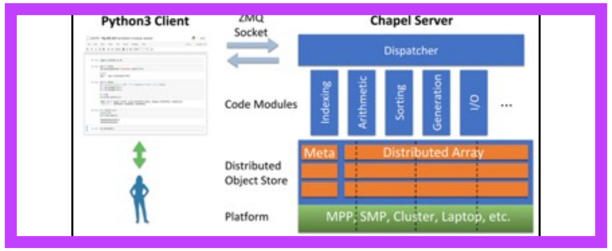
> *"Chapel allows me to use the available CPU and GPU power efficiently without low-level programming of data synchronization, managing threads, etc."*

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

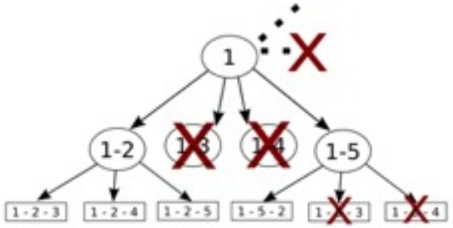
Applications of Chapel



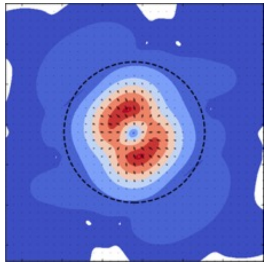
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



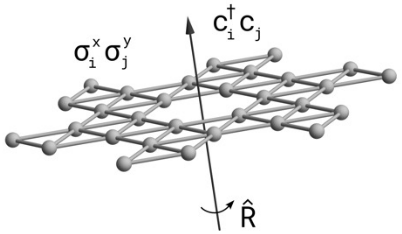
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



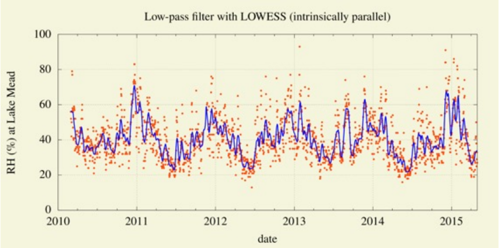
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

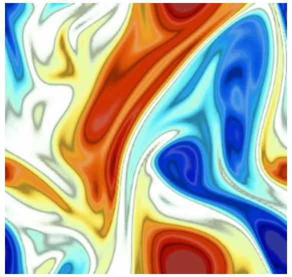


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

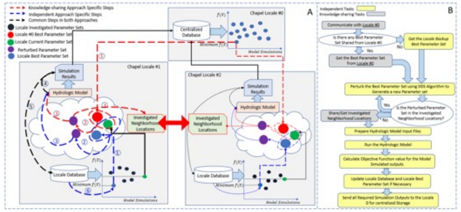


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

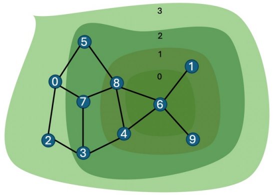
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



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

?

Your Application Here?
Your name here
KAUST

(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

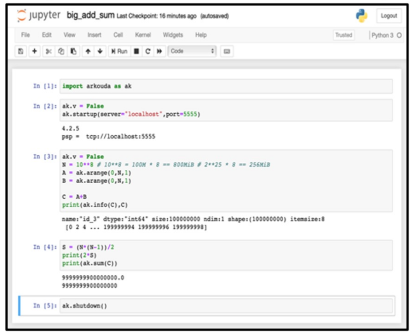
…access to HPC systems

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

What is Arkouda?

Q: “What is Arkouda?”

Arkouda Client
(written in Python)


jupyter big_add_sum Last Checkpoint: 16 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [1]: import arkouda as ak

In [2]: ak.v = False
        ak.startup(server="localhost",port=5555)

        4.2.5
        psp =  tcp://localhost:5555

In [3]: ak.v = False
        N = 10**8 # 10**8 = 100M * 8 == 800MiB # 2**25 * 8 == 256MiB
        A = ak.arange(0,N,1)
        B = ak.arange(0,N,1)

        C = A+B
        print(ak.info(C),C)

        name:"id_3" dtype:"int64" size:100000000 ndim:1 shape:(100000000) itemsize:8
         [0 2 4 ... 199999994 199999996 199999998]

In [4]: S = (N*(N-1))/2
        print(2*S)
        print(ak.sum(C))

        9999999900000000.0
        9999999900000000

In [5]: ak.shutdown()
```

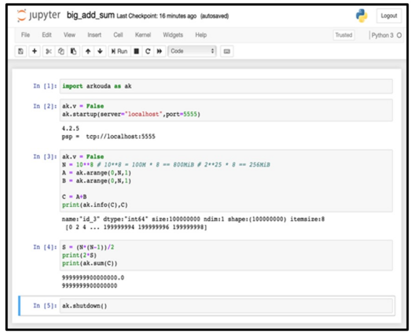
↑↓

User writes Python code
making familiar NumPy/Pandas calls

What is Arkouda?

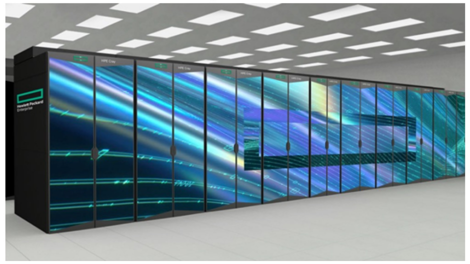
Q: “What is Arkouda?”

Arkouda Client
(written in Python)


jupyter big_add_sum Last Checkpoint: 16 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [1]: import arkouda as ak

In [2]: ak.v = False
        ak.startup(server="localhost",port=5555)

        4.2.5
        psp =  tcp://localhost:5555

In [3]: ak.v = False
        N = 10**8 # 10**8 = 100M * 8 == 800MiB # 2**25 * 8 == 256MiB
        A = ak.arange(0,N,1)
        B = ak.arange(0,N,1)

        C = A+B
        print(ak.info(C),C)

        name:"id_3" dtype:"int64" size:100000000 ndim:1 shape:(100000000) itemsize:8
         [0 2 4 ... 199999994 199999996 199999998]

In [4]: S = (N*(N-1))/2
        print(2*S)
        print(ak.sum(C))

        9999999900000000.0
        9999999900000000

In [5]: ak.shutdown()
```

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 ◆—◆

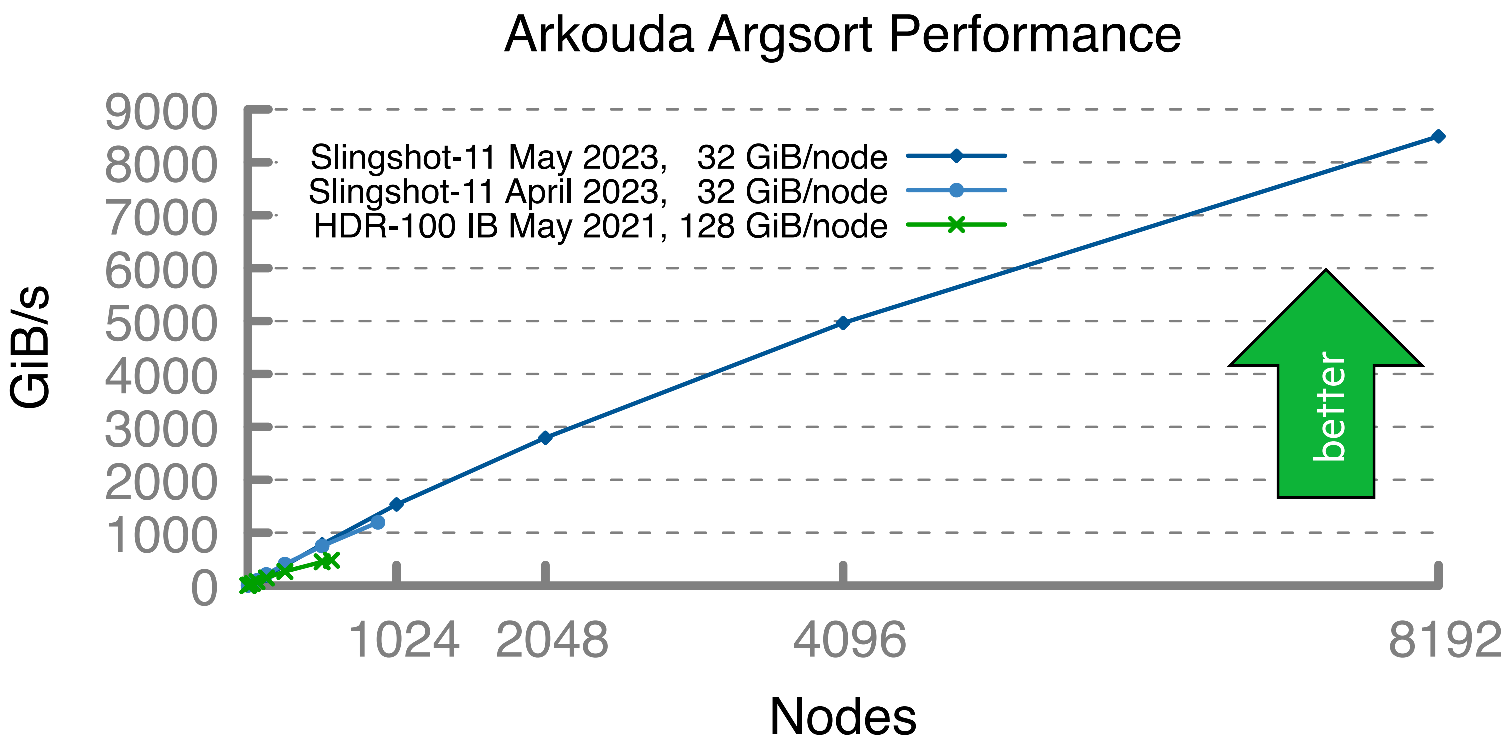
- Slingshot-11 network (200 Gb/s)
- 8192 compute nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)

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)

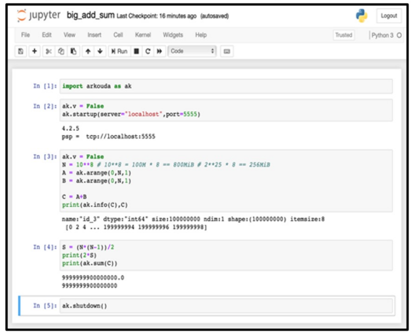


Implemented using ~100 lines of Chapel

What is Arkouda?

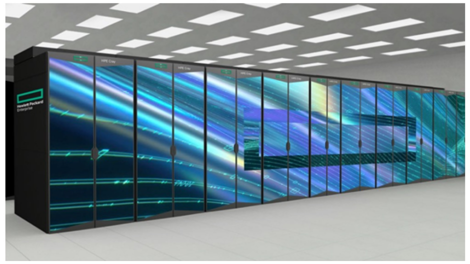
Q: “What is Arkouda?”

Arkouda Client
(written in Python)



```
In [1]: import arkouda as ak

In [2]: ak.v = False
        ak.startup(server="localhost",port=5555)

        4.2.5
        psp =  tcp://localhost:5555

In [3]: ak.v = False
        N = 10**8 # 10**8 = 100M * 8 == 800MiB # 2**25 * 8 == 256MiB
        A = ak.arange(0,N,1)
        B = ak.arange(0,N,1)

        C = A+B
        print(ak.info(C),C)

        name:"id_3" dtype:"int64" size:100000000 ndim:1 shape:(100000000) itemsize:8
         [0 2 4 ... 199999994 199999996 199999998]

In [4]: S = (N*(N-1))/2
        print(2*S)
        print(ak.sum(C))

        9999999900000000.0
        9999999900000000

In [5]: ak.shutdown()
```

Arkouda Server
(written in Chapel)



User writes Python code
making familiar NumPy/Pandas calls

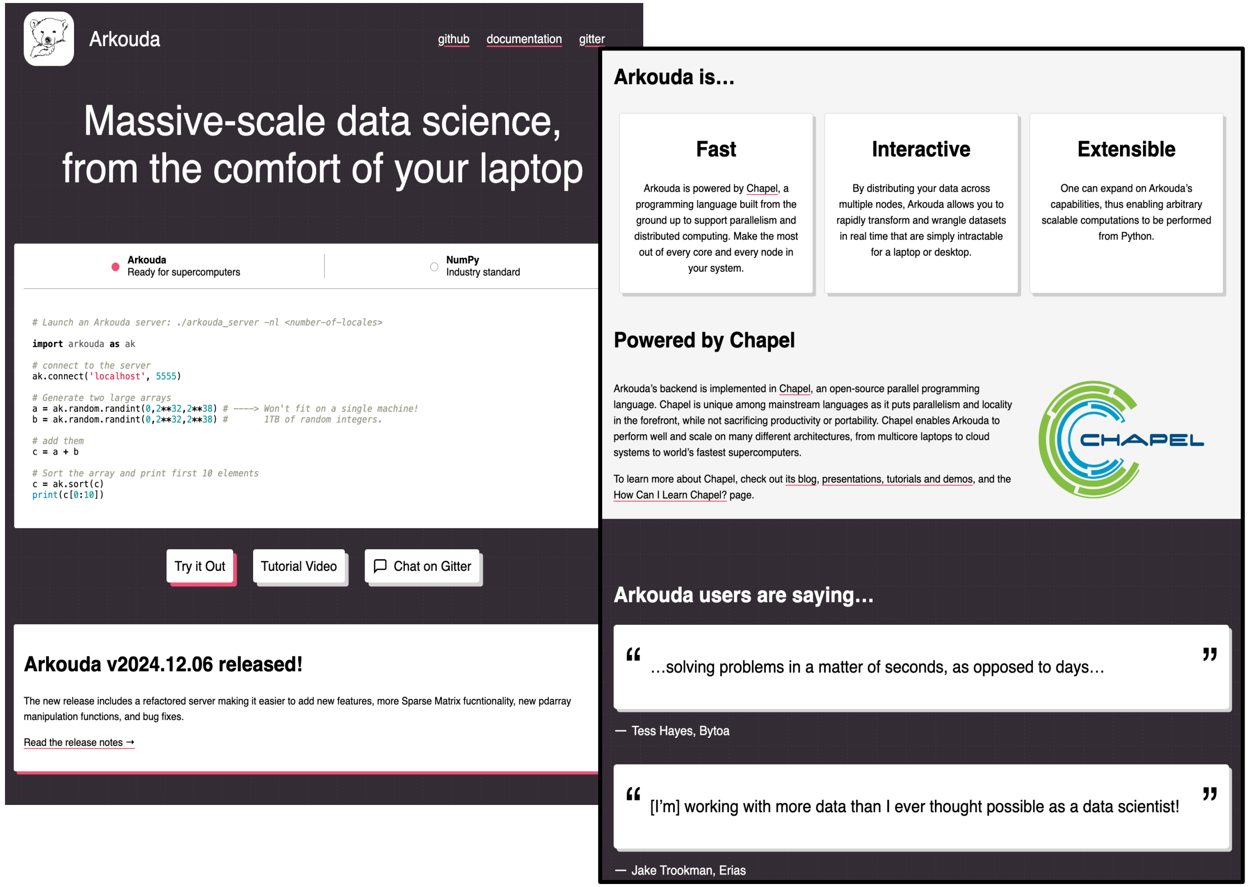
A: “A scalable version of NumPy / Pandas for data scientists”

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



Massive-scale data science, from the comfort of your laptop

```
# Launch an Arkouda server: ./arkouda_server -nl <number-of-locales>

import arkouda as ak

# connect to the server
ak.connect('localhost', 5555)

# Generate two large arrays
a = ak.random.randint(0,2**32,2**38) # ----> Won't fit on a single machine!
b = ak.random.randint(0,2**32,2**38) #       1TB of random integers.

# add them
c = a + b

# Sort the array and print first 10 elements
c = ak.sort(c)
print(c[0:10])
```

Arkouda v2024.12.06 released!

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

Read the release notes →

Arkouda is...

Fast

Arkouda is powered by Chapel, a programming language built from the ground up to support parallelism and distributed computing. Make the most out of every core and every node in your system.

Interactive

By distributing your data across multiple nodes, Arkouda allows you to rapidly transform and wrangle datasets in real time that are simply intractable for a laptop or desktop.

Extensible

One can expand on Arkouda's capabilities, thus enabling arbitrary scalable computations to be performed from Python.

Powered by Chapel

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

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

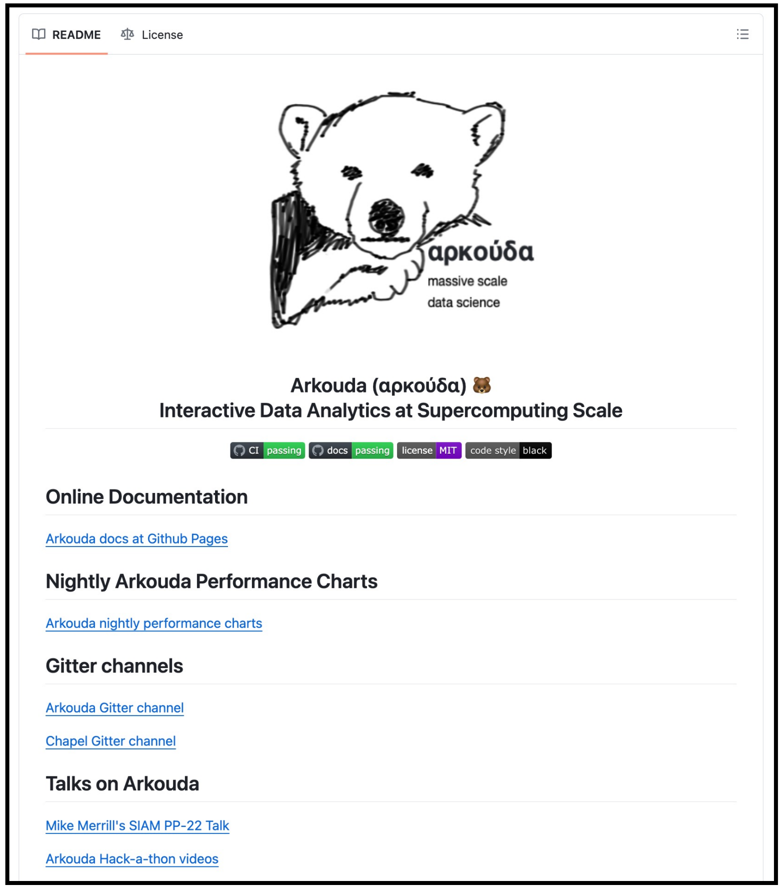
Arkouda users are saying...

> "…solving problems in a matter of seconds, as opposed to days…"

— Tess Hayes, Bytoa

> "[I'm] working with more data than I ever thought possible as a data scientist!"

— Jake Trookman, Erias



Arkouda (αρκούδα) 🐻
Interactive Data Analytics at Supercomputing Scale

Online Documentation

Arkouda docs at Github Pages

Nightly Arkouda Performance Charts

Arkouda nightly performance charts

Gitter channels

Arkouda Gitter channel

Chapel Gitter channel

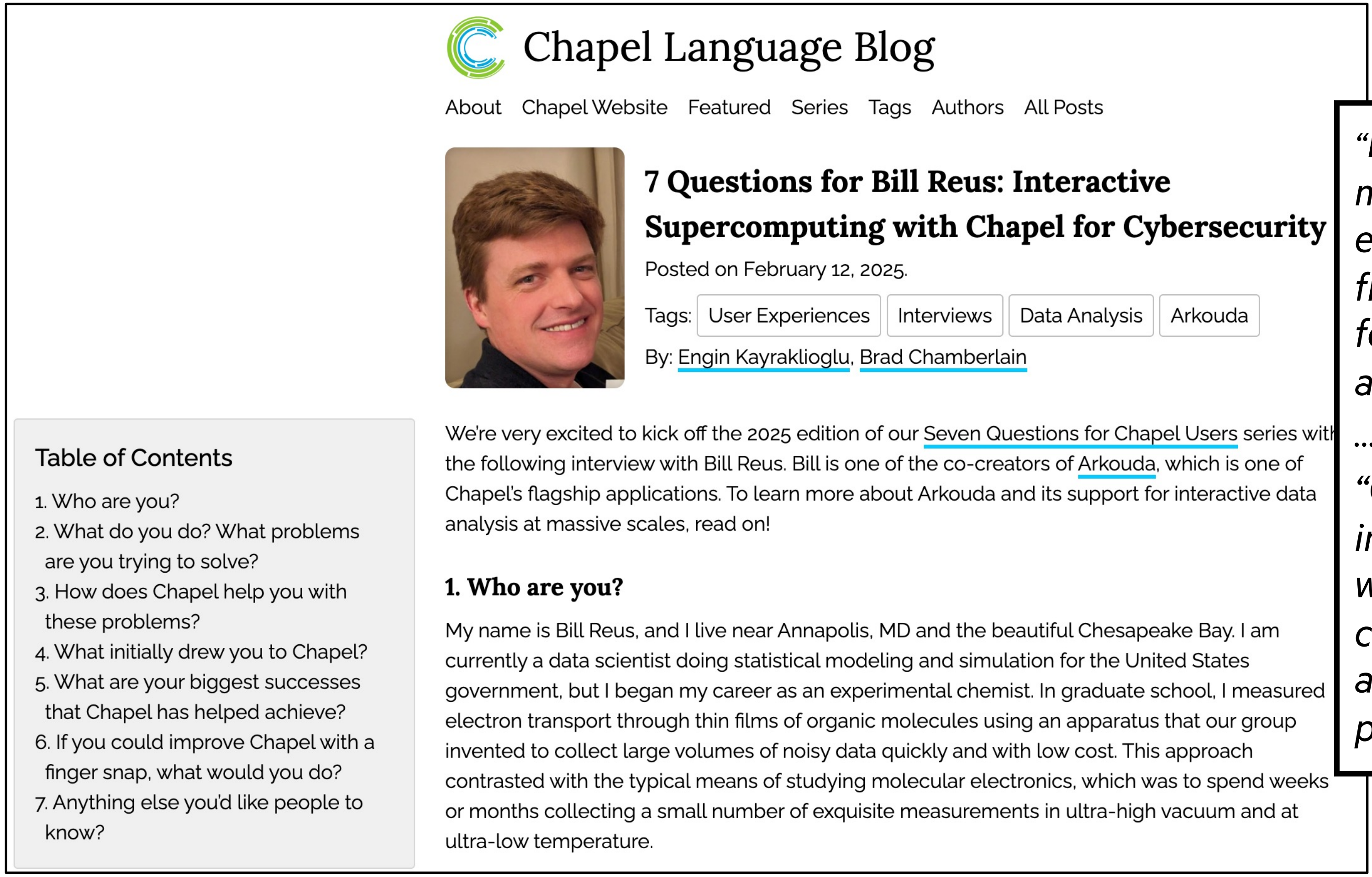
Talks on Arkouda

Mike Merrill's SIAM PP-22 Talk

Arkouda Hack-a-thon videos

Arkouda Interview

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



Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts

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

Posted on February 12, 2025.

Tags: User Experiences | Interviews | Data Analysis | Arkouda

By: Engin Kayraklioglu, Brad Chamberlain

Table of Contents

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

1. Who are you?

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

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

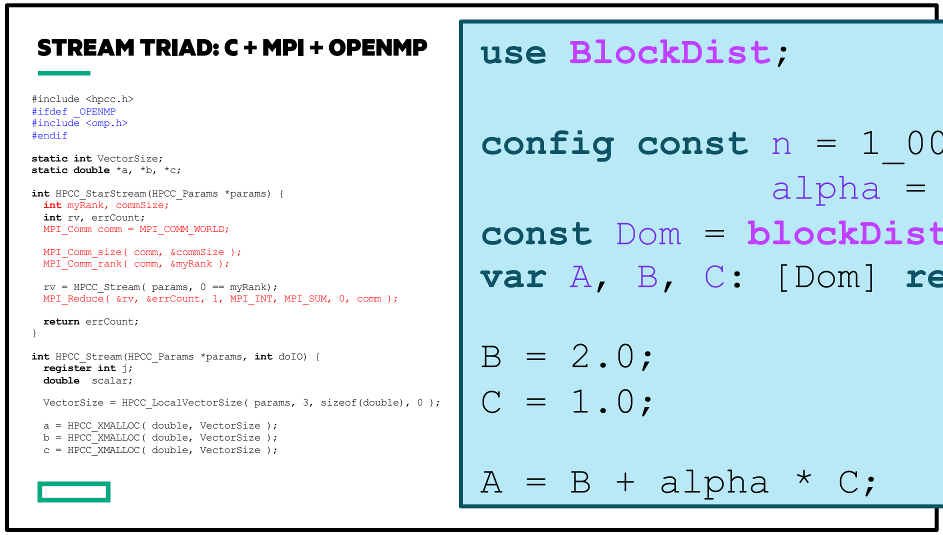
…

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

Global-view vs. SPMD Programming

jump to wrap-up

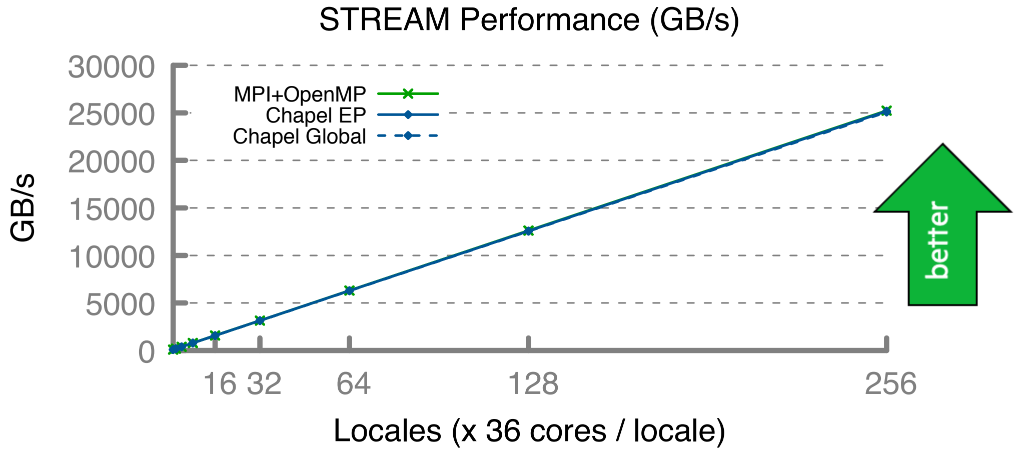
HPCC Stream Triad and RA in C + MPI + OpenMP vs. Chapel

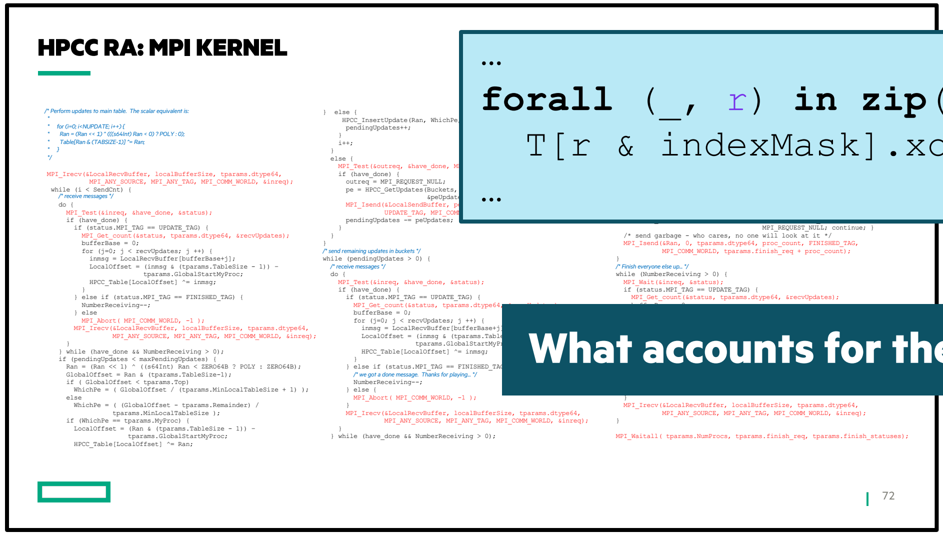


STREAM TRIAD: C + MPI + OPENMP

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

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

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

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

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

  return errCount;
}

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

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

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

```
use BlockDist;

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

B = 2.0;
C = 1.0;

A = B + alpha * C;
```





HPCC RA: MPI KERNEL

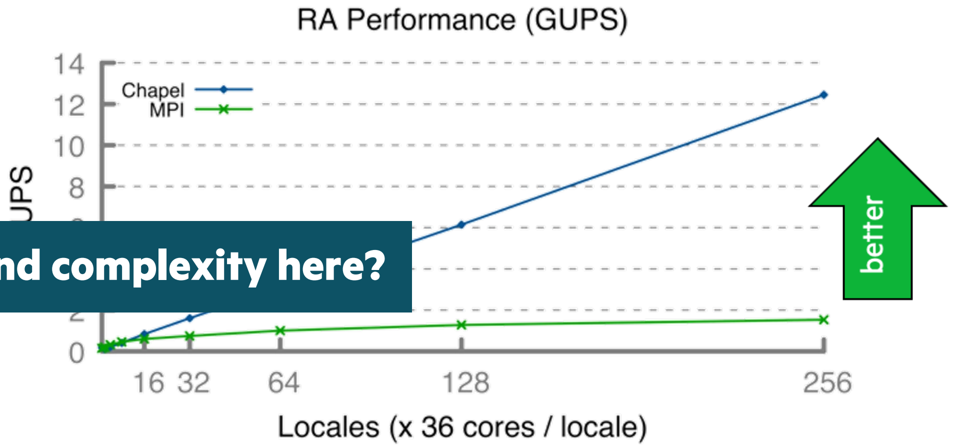
```
/* Perform updates to main table.  The scalar equivalent is:
 *
 *   for (i=0; i<NUPDATE; i++) {
 *     Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *     Table[Ran & (TABSIZE-1)] ^= Ran;
 *   }
 */

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,
      pendingUpdates++;
    }
    i++;
  }
  else {
    MPI_Test(&outreq, &have_done, MP
    if (have_done) {
      outreq = MPI_REQUEST_NULL;
      pe = HPCC_GetUpdates(Buckets,
                           &peUpdate
      MPI_Isend(&LocalSendBuffer, pe
                UPDATE_TAG, MPI_COMM
      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,
        bufferBase = 0;
        for (j=0; j < recvUpdates; j ++) {
          inmsg = LocalRecvBuffer[bufferBase+j]
          LocalOffset = (inmsg & (tparams.Table
                        tparams.GlobalStartMyPr
          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.dt
                MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
    }
  } while (have_done && NumberReceiving > 0);
```

```
                                  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);
  }
  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);
```

72

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

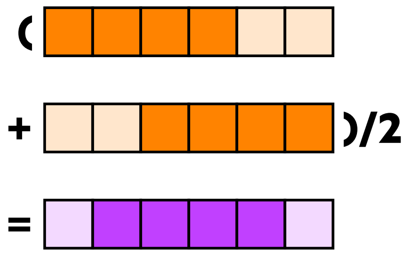


What accounts for the huge difference in code size and complexity here?

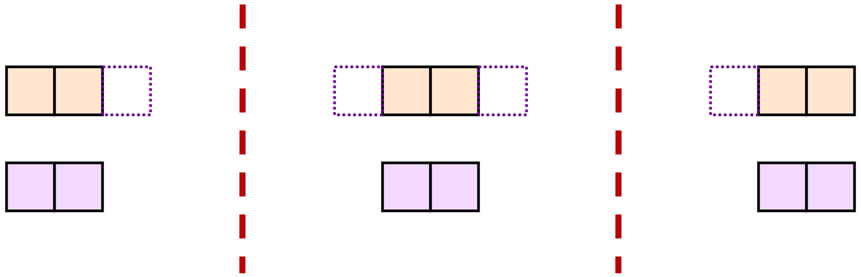
Global-view Programming vs. Single-Program, Multiple Data (SPMD)

Example: “Replace each array’s elements with the average of its neighbors.” (compute a 3-point stencil)

Global-View

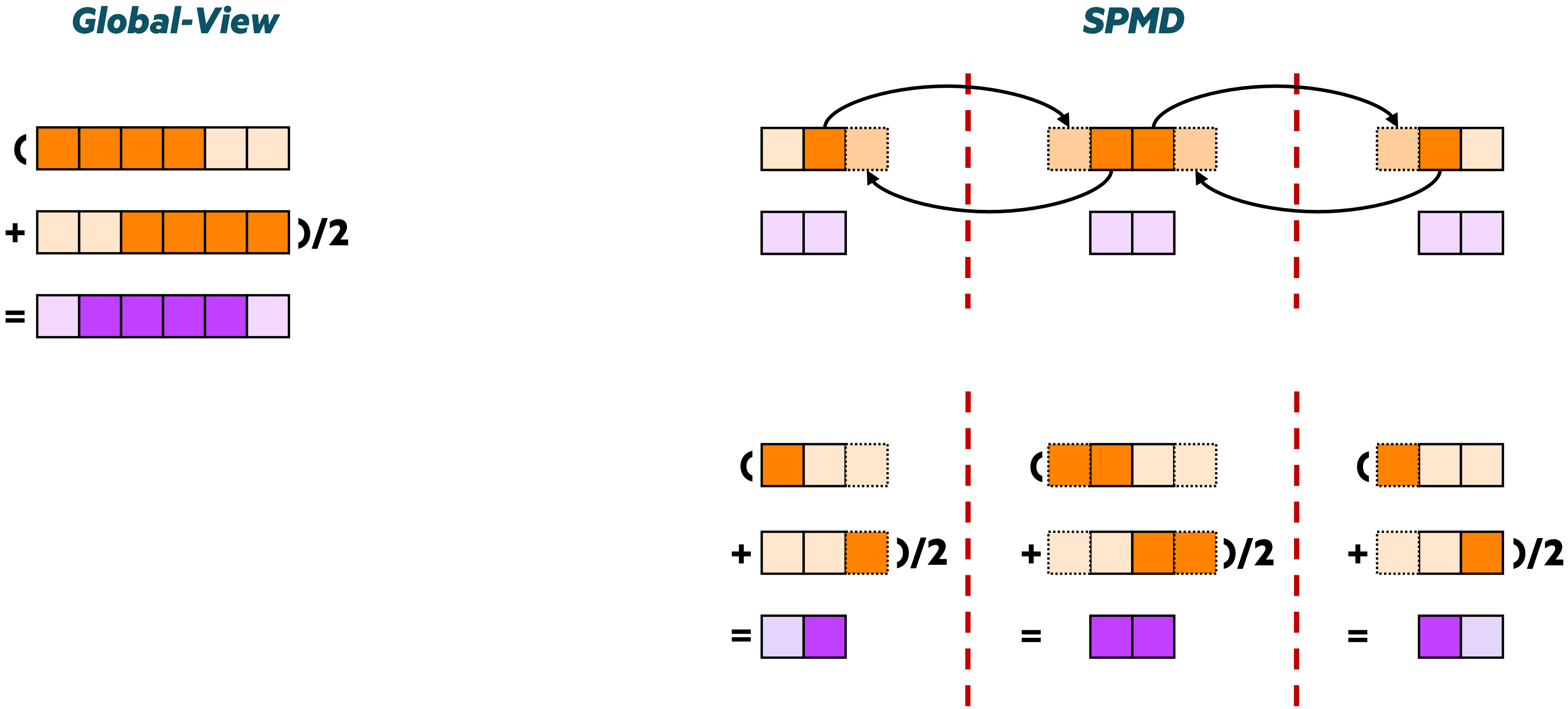


SPMD



Global-view Programming vs. Single-Program, Multiple Data (SPMD)

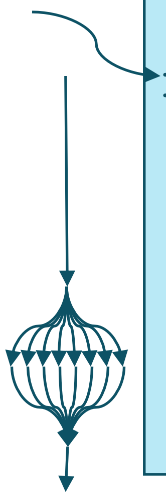
Example: "Replace each array's elements with the average of its neighbors." (compute a 3-point stencil)



Global-view Programming vs. Single-Program, Multiple Data (SPMD)

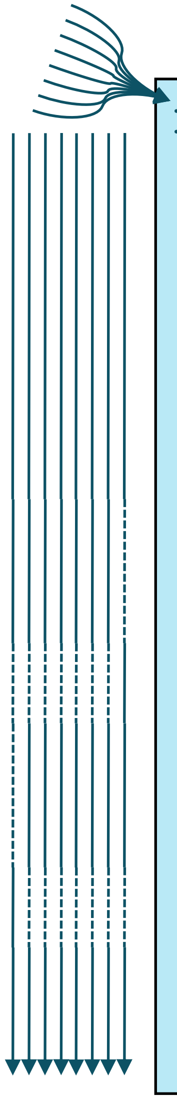
Example: “Apply a 3-point stencil to a vector”

Global-View Chapel code



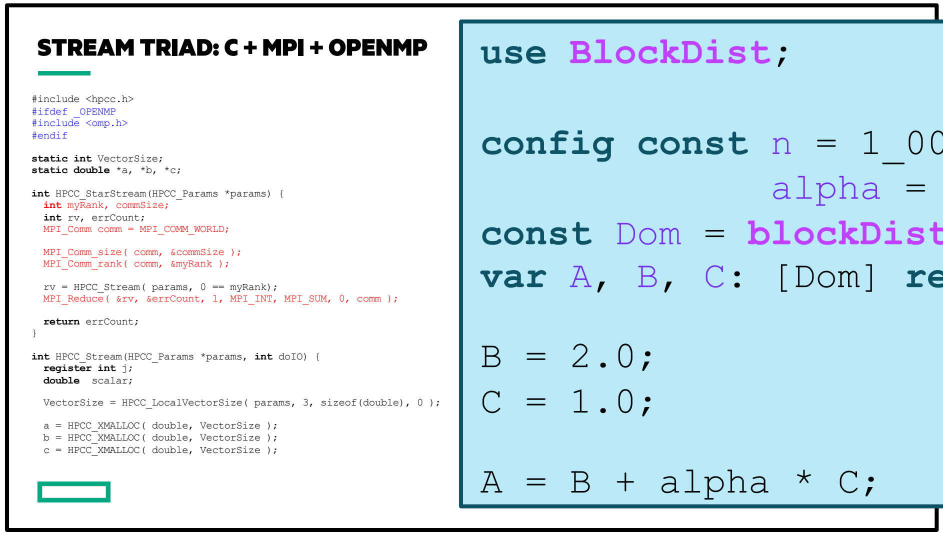
```
use BlockDist;

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

  var A, B: [D] real;

  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;
}
```

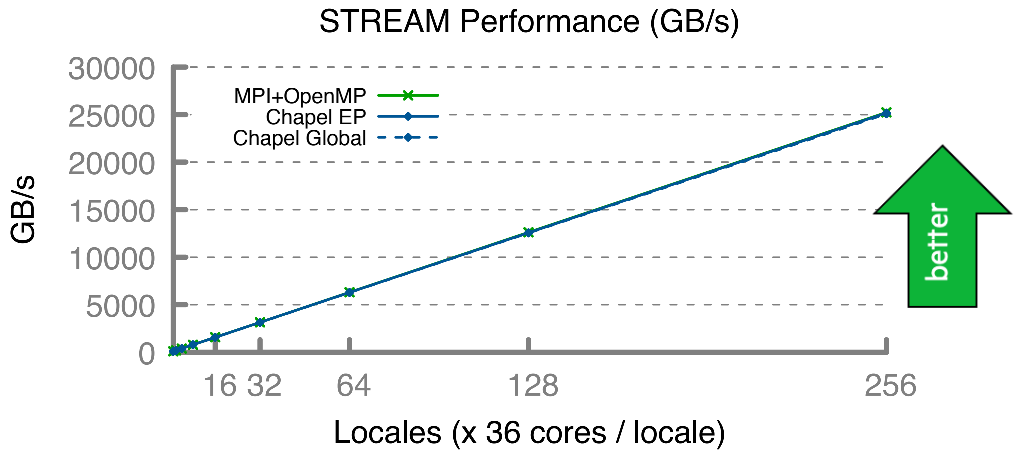
HPCC Stream Triad and RA in C + MPI + OpenMP vs. Chapel

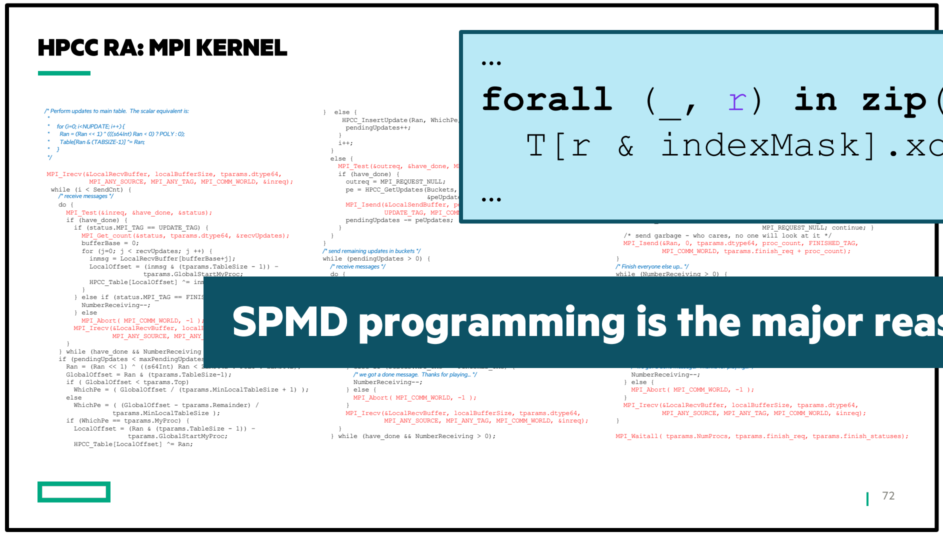


STREAM TRIAD: C + MPI + OPENMP

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

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

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

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

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

  return errCount;
}

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

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

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

```
use BlockDist;

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

B = 2.0;
C = 1.0;

A = B + alpha * C;
```





HPCC RA: MPI KERNEL

```
/* 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] ^= inm
         }
       } else if (status.MPI_TAG == FINIS
         NumberReceiving--;
       } else
         MPI_Abort( MPI_COMM_WORLD, -1 );
       MPI_Irecv(&LocalRecvBuffer, localB
                 MPI_ANY_SOURCE, MPI_ANY_
     }
   } while (have_done && NumberReceiving
   if (pendingUpdates < maxPendingUpdates
     Ran = (Ran << 1) ^ ((s64Int) Ran <
     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,
     pendingUpdates++;
    }
   i++;
  }
  else {
   MPI_Test(&outreq, &have_done, MI
   if (have_done) {
     outreq = MPI_REQUEST_NULL;
     pe = HPCC_GetUpdates(Buckets,
                          &peUpdate
     MPI_Isend(&LocalSendBuffer, pe
               UPDATE_TAG, MPI_COMM
     pendingUpdates -= peUpdates;
    }
  }
}
/* send remaining updates in buckets */
while (pendingUpdates > 0) {
  /* receive messages */
  do {
```

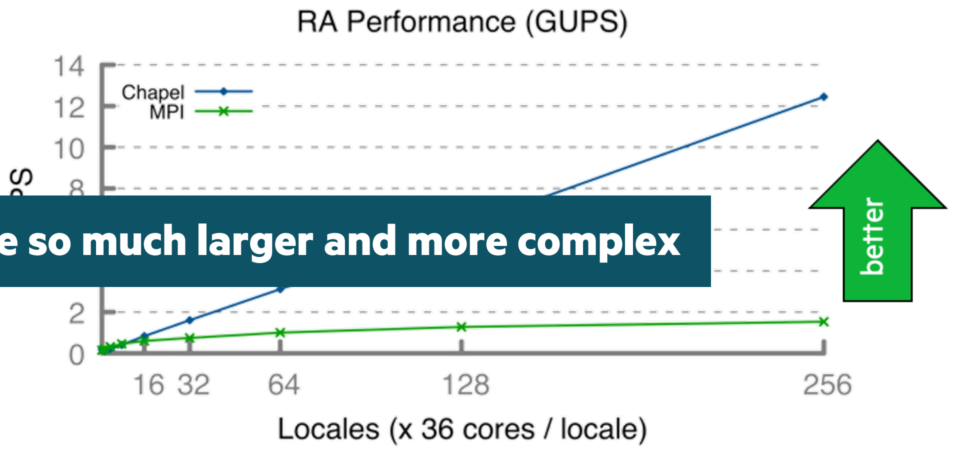
```
      /* we got a done message.  Thanks for playing... */
      NumberReceiving--;
    } else {
      MPI_Abort( MPI_COMM_WORLD, -1 );
    }
    MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
              MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
  }
} while (have_done && NumberReceiving > 0);
```

```
                                      MPI_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) {
```

```
    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);
```

72

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



SPMD programming is the major reason these reference versions are so much larger and more complex

C is a secondary factor

SPMD Programming in Chapel

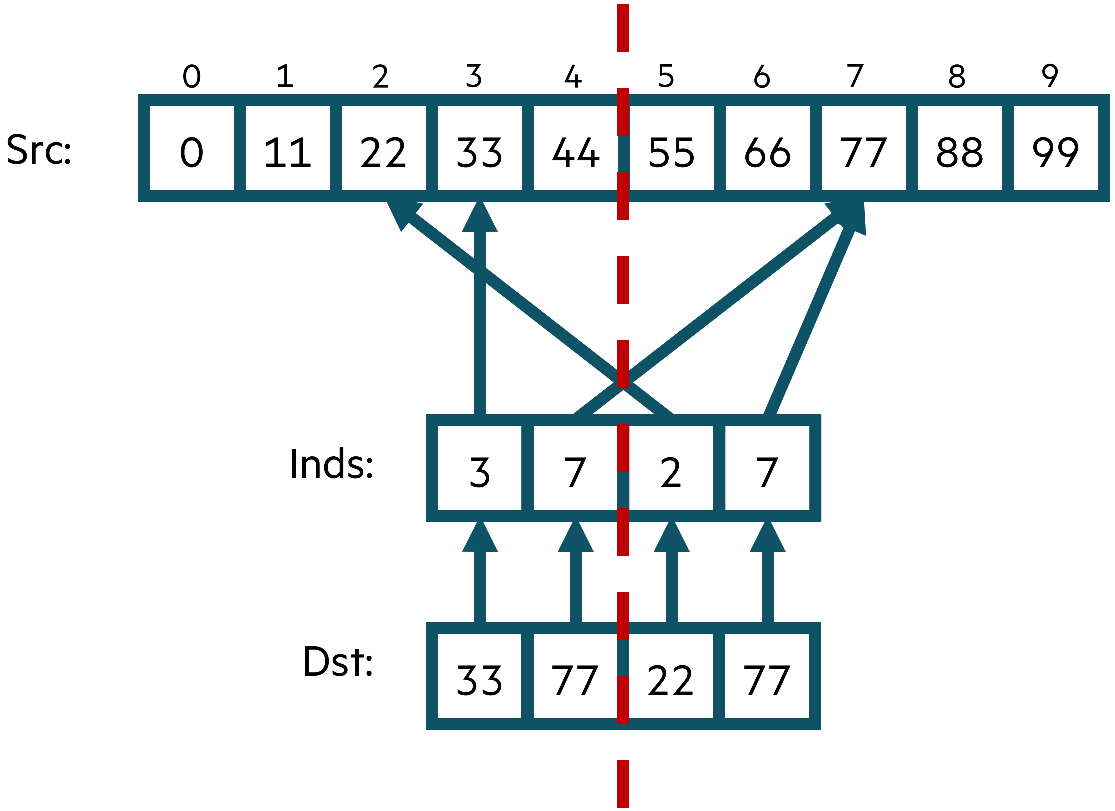
That said, as a general-purpose language, Chapel supports writing SPMD patterns as well:

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

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

A Brief Introduction to Chapel (via Bale IndexGather)

jump to wrap-up

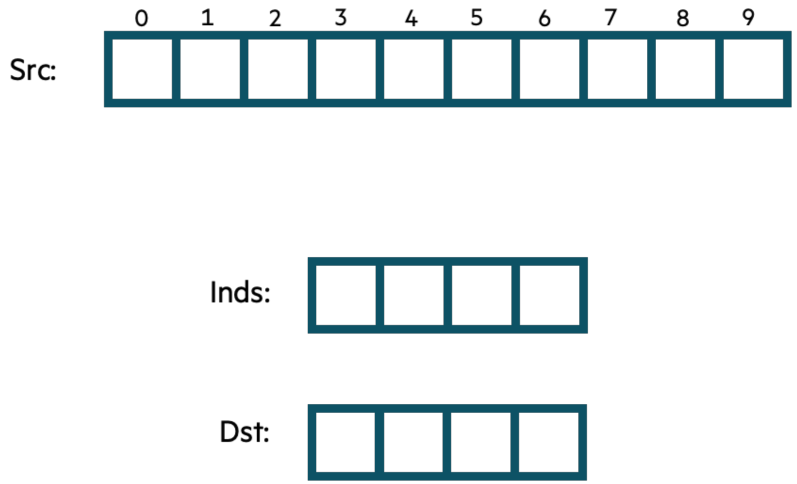
Bale IndexGather (IG): In Pictures



Bale IG in Chapel: Array Declarations

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



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

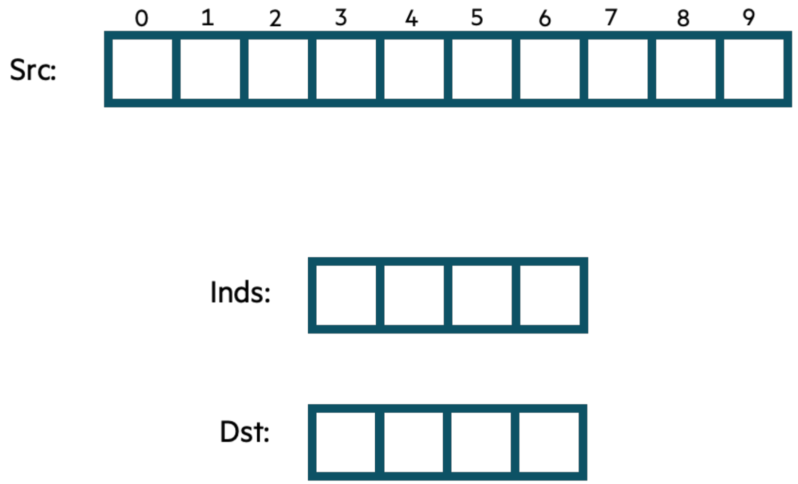
```
$
```



Bale IG in Chapel: Compiling

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




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

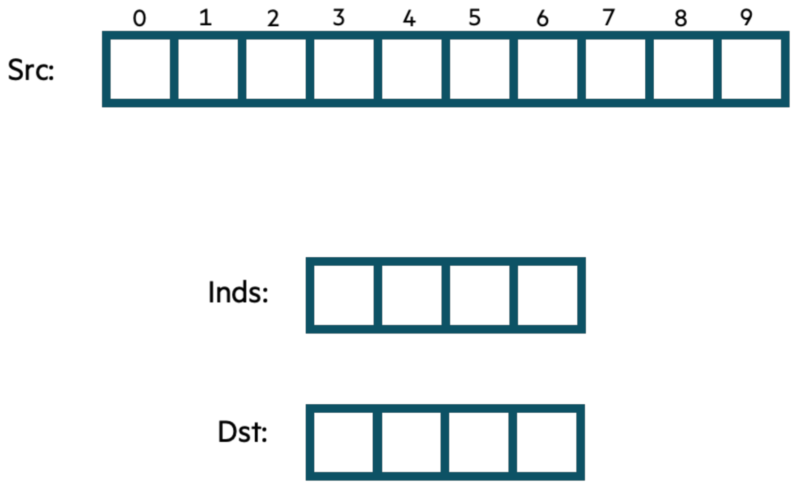
```
$ chpl bale-ig.chpl
$
```



Bale IG in Chapel: Executing

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



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

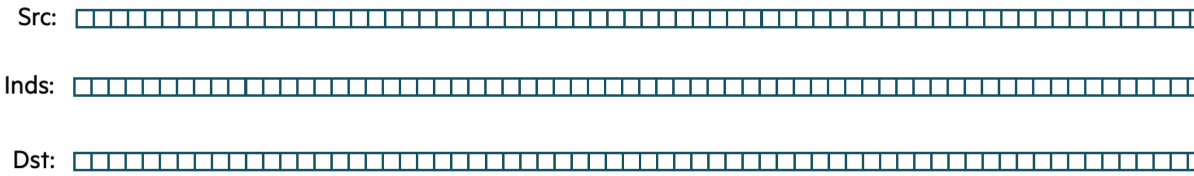
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Executing, Overriding Configs

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



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

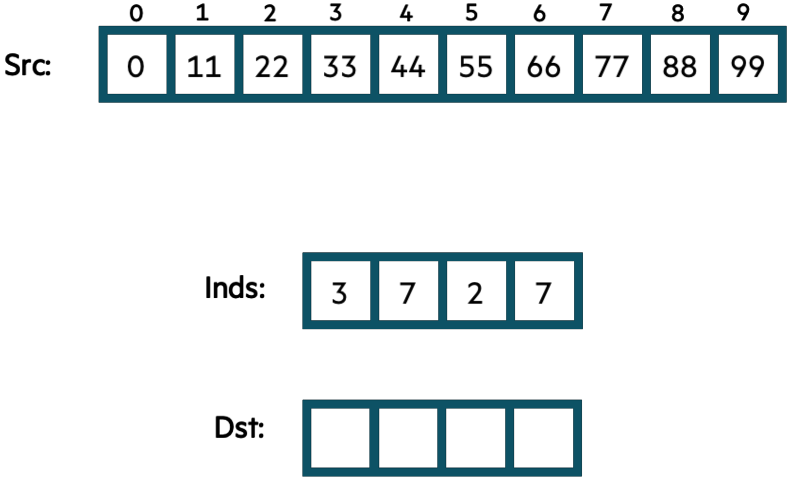
```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000
$
```



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
$
```



Src:

0	1	2	3	4	5	6	7	8	9
0	11	22	33	44	55	66	77	88	99

Inds:

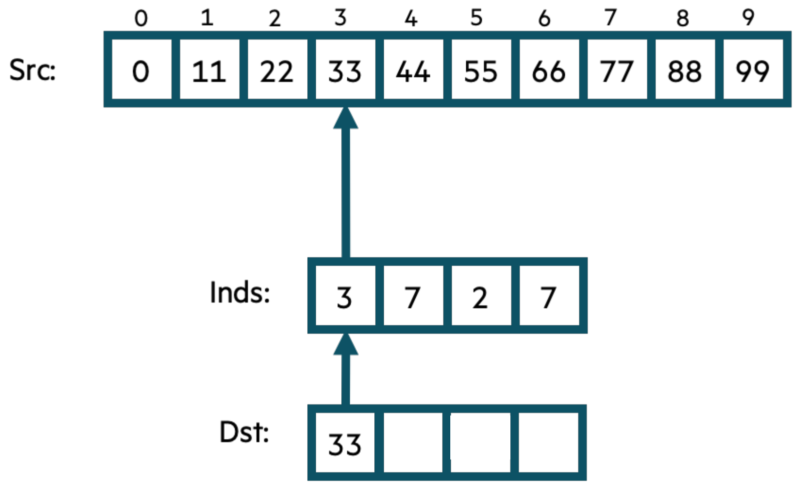
3	7	2	7

Dst:

Bale IG in Chapel: Serial, Zippered Version

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




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

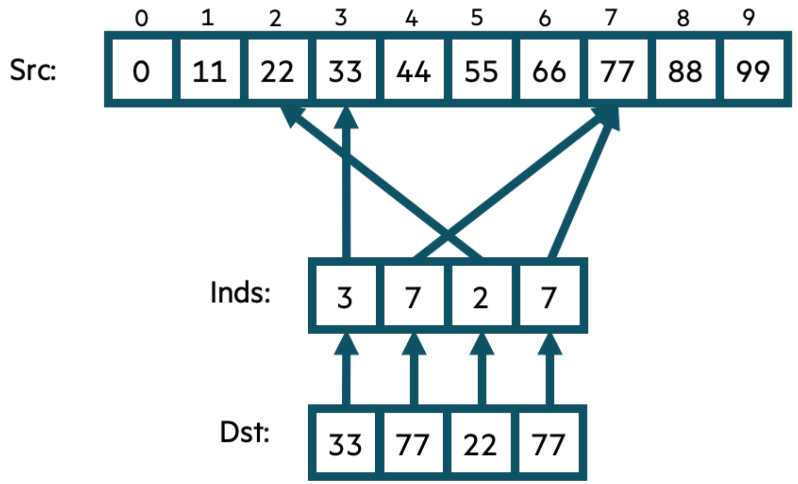
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



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

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




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

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



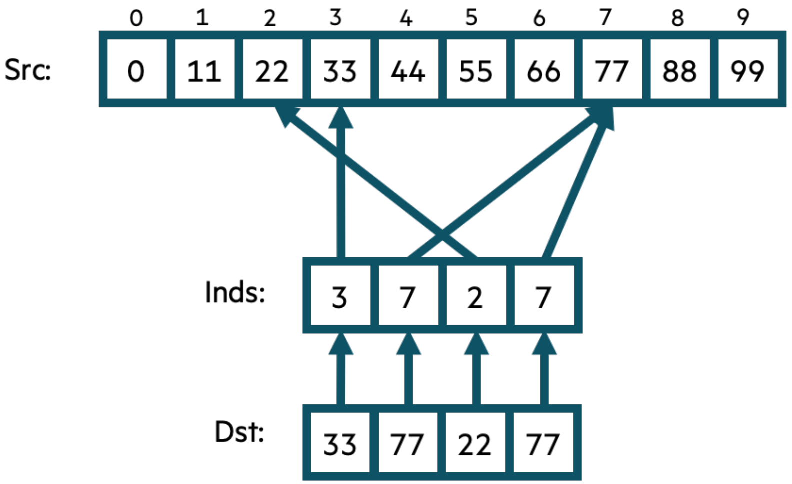
	0	1	2	3	4	5	6	7	8	9
Src:	0	11	22	33	44	55	66	77	88	99

Inds:	3	7	2	7
Dst:	33	77	22	77

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];
}
```

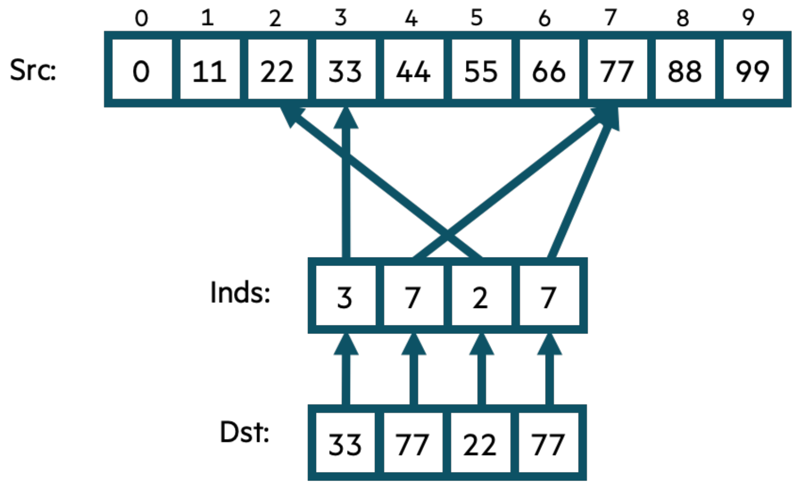
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



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];
```

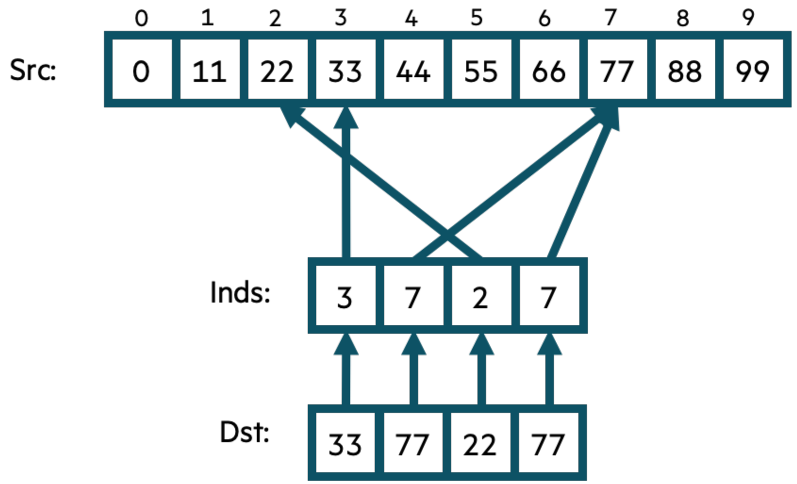
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



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];
```

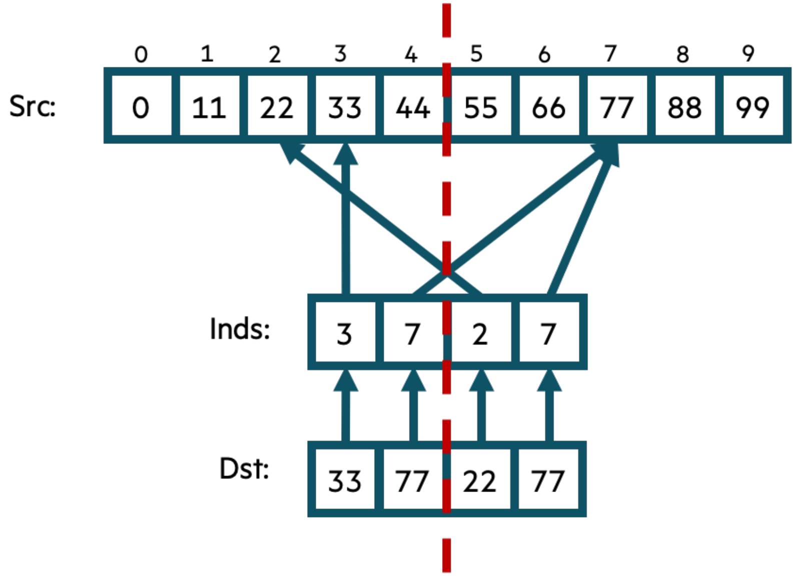
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



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];
```

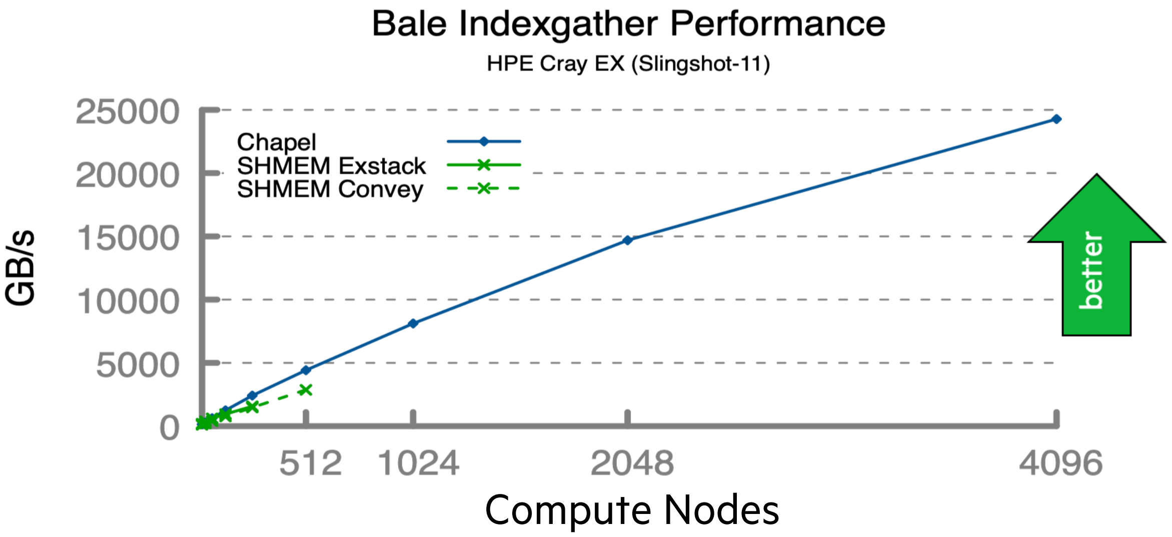
```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4096 --n=… --m=…
$
```

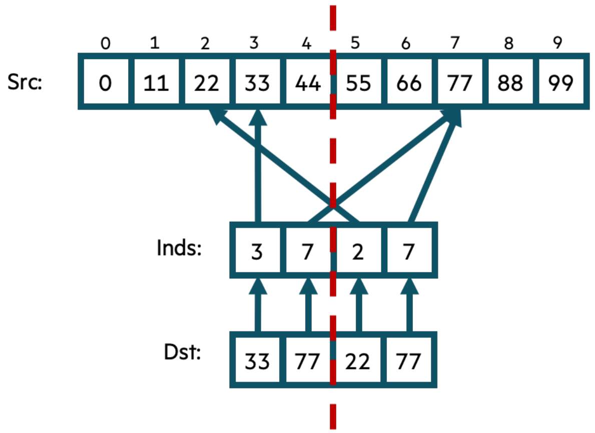


Bale IG in Chapel: Distributed Parallel Version

```
use BlockDist;

config const n = 10,
             m = 4;

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

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

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





	0	1	2	3	4	5	6	7	8	9
Src:	0	11	22	33	44	55	66	77	88	99

Inds:	3	7	2	7
Dst:	33	77	22	77

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

Chapel (Simple / Auto-Aggregated version)

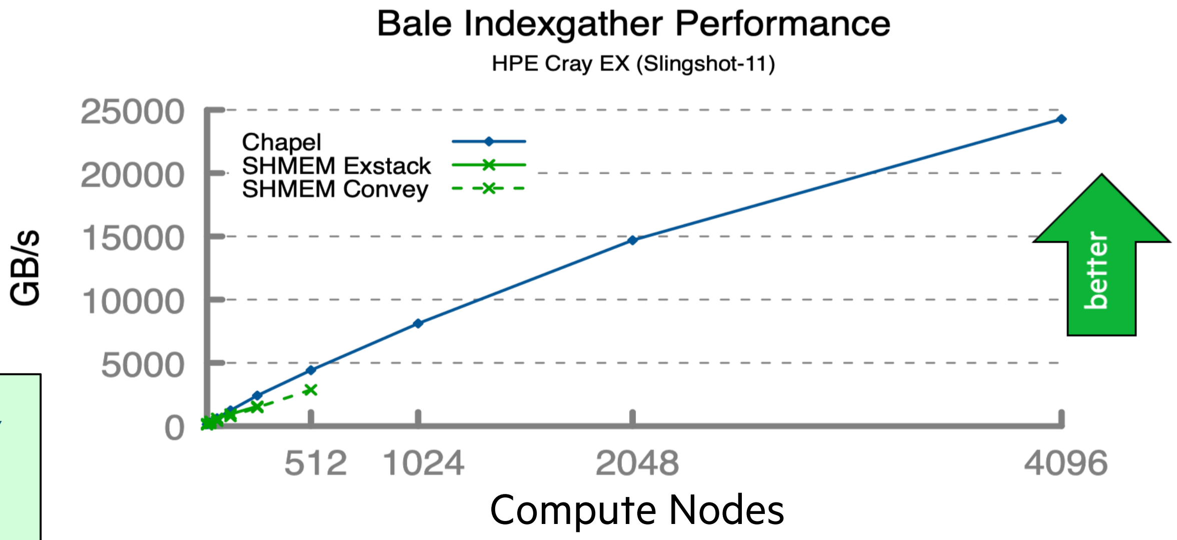
```
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

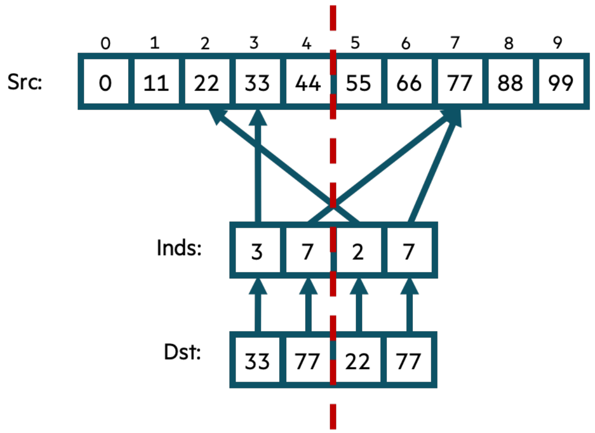
SHMEM (Exstack version)

```
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)

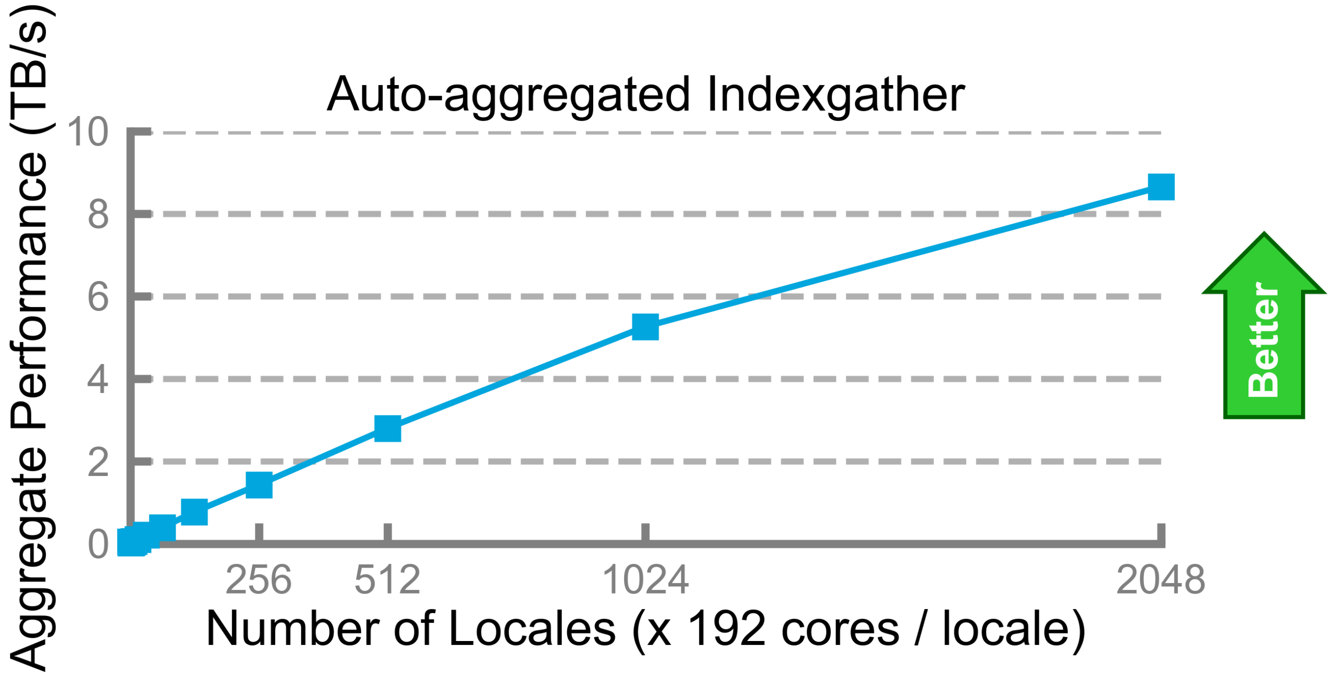
```
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
       more | convey_advance(replies, !more)) {

  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;
}
```





Bale Index Gather in Chapel on Shaheen (Initial Results)

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

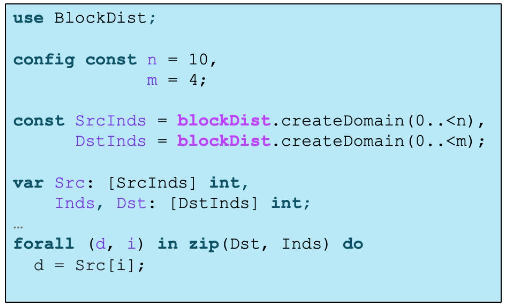


Wrap-up

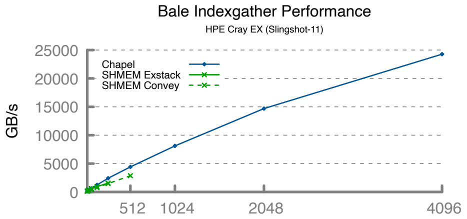
Summary

Chapel is unique among programming languages

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

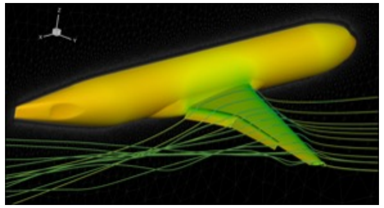


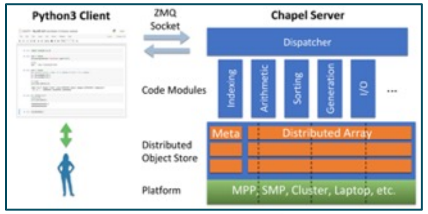
```
use BlockDist;

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];
```

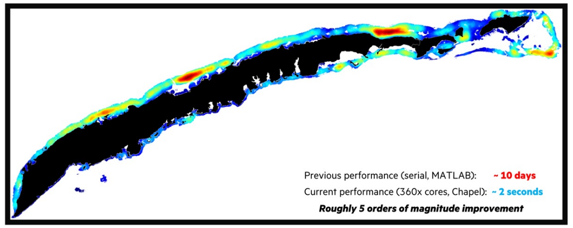


Chapel is being used for productive parallel computing at scale

- 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







Previous performance (serial, MATLAB): ~ 10 days

Current performance (360x cores, Chapel): ~ 2 seconds

Roughly 5 orders of magnitude improvement

But wait, there’s more!

There are lots of things we couldn’t get to (much) today that are worthy of more time

- Chapel features
- GPU support
- VSCode support, with integrated linter
- Arkouda, in more detail
- Chapel performance on Shaheen
- Compiler optimizations
- …

We’d be happy to follow up on any of these topics, or others, as schedules and interest permit

The Advanced Programming Team at HPE

Ways to Engage with the Chapel Community

"Live" Virtual Events

- ChapelCon (formerly CHIUW), annually
- Project Meetings, weekly
- Demo Sessions, monthly (recorded)

Community / User Forums

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

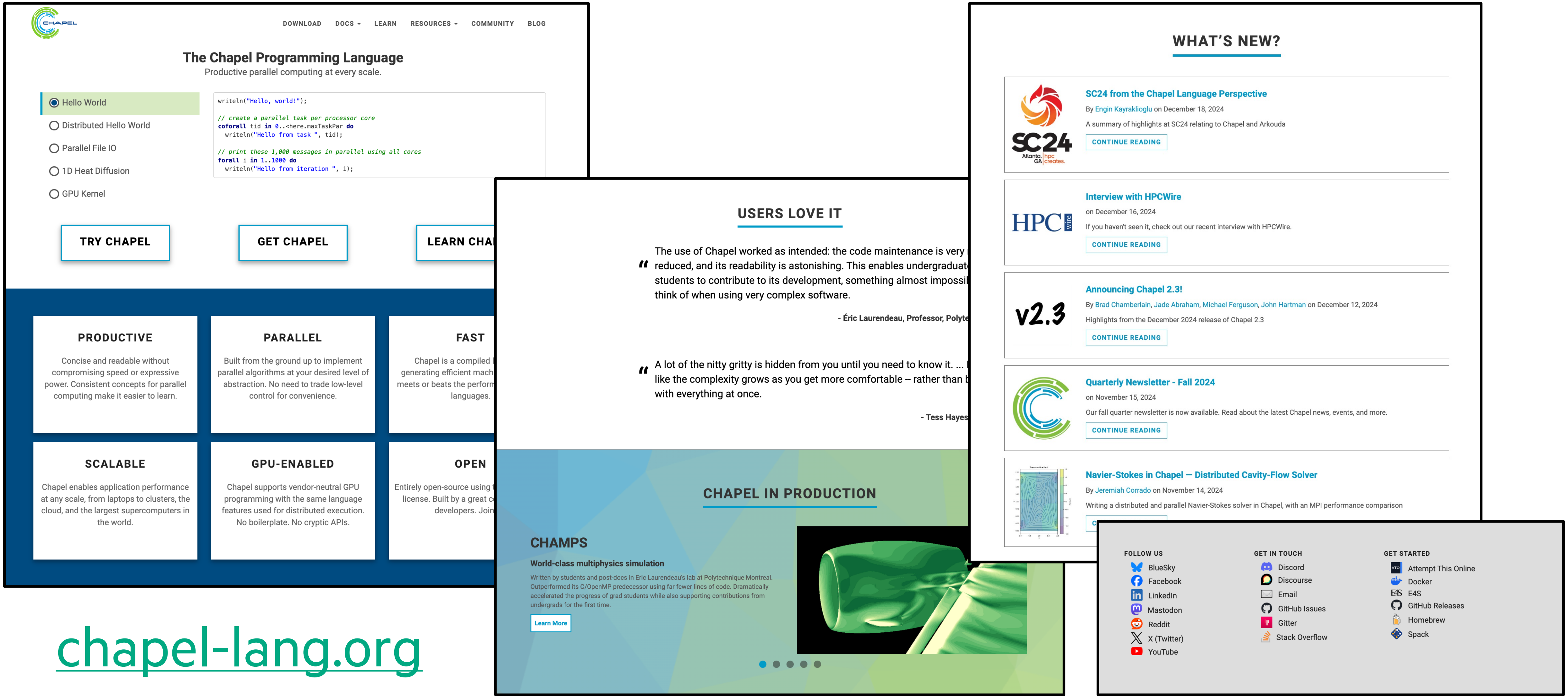
Electronic Communications

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

Social Media

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

Chapel Website



The Chapel Programming Language

Productive parallel computing at every scale.

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

```
writeln("Hello, world!");

// create a parallel task per processor core
coforall tid in 0..<here.maxTaskPar do
  writeln("Hello from task ", tid);

// print these 1,000 messages in parallel using all cores
forall i in 1..1000 do
  writeln("Hello from iteration ", i);
```

PRODUCTIVE

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

PARALLEL

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

FAST

Chapel is a compiled l
generating efficient mach
meets or beats the perform
languages.

SCALABLE

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

GPU-ENABLED

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

OPEN

Entirely open-source using t
license. Built by a great c
developers. Join

USERS LOVE IT

> “ The use of Chapel worked as intended: the code maintenance is very
> reduced, and its readability is astonishing. This enables undergraduate
> students to contribute to its development, something almost impossib
> think of when using very complex software.
>
> - Éric Laurendeau, Professor, Polyte

> “ A lot of the nitty gritty is hidden from you until you need to know it. ... l
> like the complexity grows as you get more comfortable -- rather than b
> with everything at once.
>
> - Tess Hayes

CHAPEL IN PRODUCTION

CHAMPS

World-class multiphysics simulation

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

WHAT’S NEW?

SC24 from the Chapel Language Perspective

By Engin Kayrakliogiu on December 18, 2024

A summary of highlights at SC24 relating to Chapel and Arkouda

Interview with HPCWire

on December 16, 2024

If you haven't seen it, check out our recent interview with HPCWire.

Announcing Chapel 2.3!

By Brad Chamberlain, Jade Abraham, Michael Ferguson, John Hartman on December 12, 2024

Highlights from the December 2024 release of Chapel 2.3

Quarterly Newsletter - Fall 2024

on November 15, 2024

Our fall quarter newsletter is now available. Read about the latest Chapel news, events, and more.

Navier-Stokes in Chapel — Distributed Cavity-Flow Solver

By Jeremiah Corrado on November 14, 2024

Writing a distributed and parallel Navier-Stokes solver in Chapel, with an MPI performance comparison

FOLLOW US

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

GET IN TOUCH

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

GET STARTED

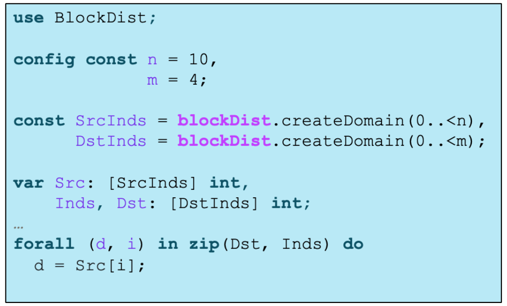
- Attempt This Online
- Docker
- E4S
- GitHub Releases
- Homebrew
- Spack

chapel-lang.org

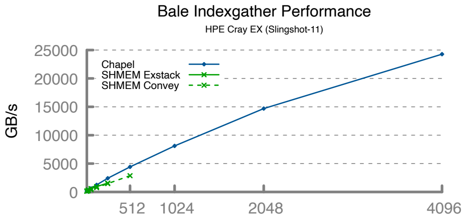
Summary

Chapel is unique among programming languages

- 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

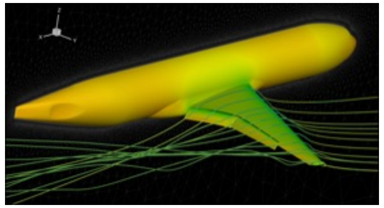


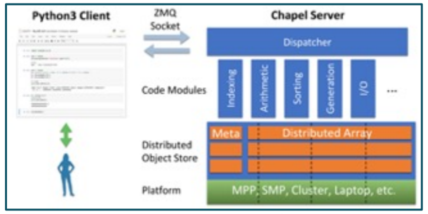
```
use BlockDist;

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];
```

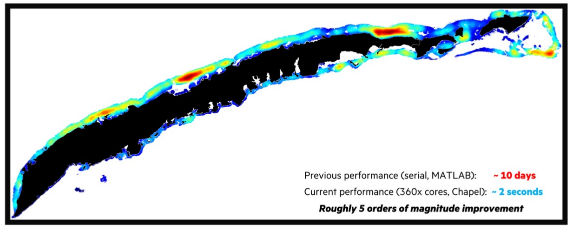


Chapel is being used for productive parallel computing at scale

- 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







Previous performance (serial, MATLAB): **~ 10 days**

Current performance (360x cores, Chapel): **~ 2 seconds**

Roughly 5 orders of magnitude improvement

Thank you

https://chapel-lang.org
@ChapelLanguage