








Productive Parallel Programming at all scales with Chapel

Brad Chamberlain and Jade Abraham, Advanced Programming Team, HPE

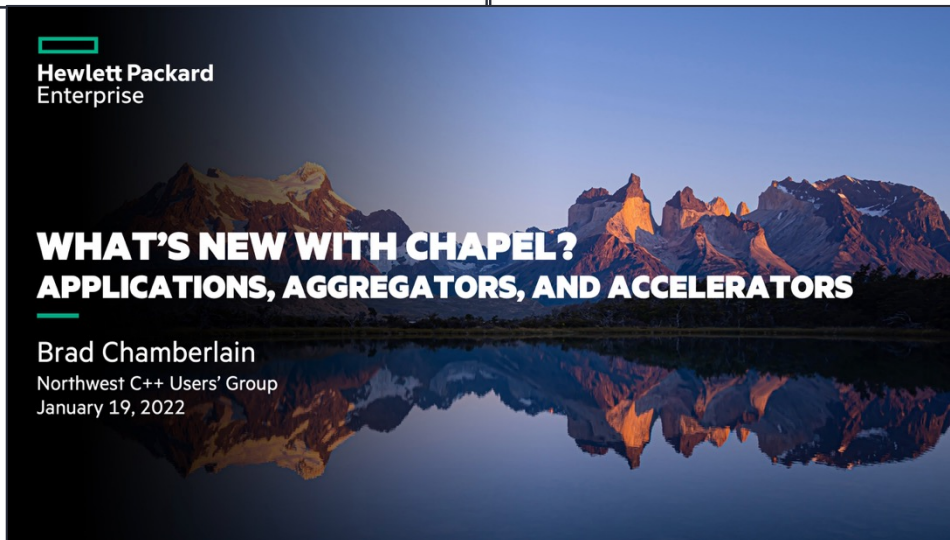
Northwest C++ Users' Group
May 21, 2026

Proud to be a member of the “NWC++ Five-Timers Club”

 <h2>Chapel: An Emerging Parallel Programming Language</h2> <p>Thomas Van Doren, Chapel Team, Cray Inc. Northwest C++ Users' Group April 16th, 2014</p> 	 <h2>Chapel: Productive Parallel Programming at Scale</h2> <p>Brad Chamberlain, Chapel Team, Cray Inc. Northwest C++ Users Group October 19th, 2016</p>  <small>COMPUTE STORE ANALYZE</small>	 <h2>Chapel's Multiresolution Programming Model Mixing High-level Parallel Abstractions with Lower-level Control</h2> <p>Brad Chamberlain, Chapel Team, Cray Inc. Northwest C++ Users Group February 21, 2018</p>   <small>COMPUTE STORE ANALYZE</small>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



**Hewlett Packard
Enterprise**



WHAT'S NEW WITH CHAPEL? APPLICATIONS, AGGREGATORS, AND ACCELERATORS

Brad Chamberlain
Northwest C++ Users' Group
January 19, 2022



Productive Parallel Programming at all scales with Chapel

Brad Chamberlain and Jade Abraham, Advanced Programming Team, HPE
Northwest C++ Users' Group
May 21, 2026



Motivation



30 Years Ago vs. Now: Top HPC Systems in the Top500

Top 5 systems in the Top500, November 1995:

- **Cores:** 80–3,680
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D

Rank	System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)	Power (kW)
1	Numerical Wind Tunnel, Fujitsu National Aerospace Laboratory of Japan Japan	140	170.00	235.79	
2	XP/S140, Intel Sandia National Laboratories United States	3,680	143.40	184.00	
3	XP/S-MP 150, Intel DOE/SC/Oak Ridge National Laboratory United States	3,072	127.10	154.00	
4	T3D MC1024-8, Cray/HPE Government United States	1,024	100.50	153.60	
5	VPP500/80, Fujitsu National Lab. for High Energy Physics Japan	80	98.90	128.00	

Top 5 systems in the Top 500, November 2025:

- **Cores:** 2,073,600–11,340,616
- **Rmax:** 561.2–1809 PFlop/s
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	El Capitan - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,340,000	1,809.00	2,821.10	29,685
2	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	Aurora - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
4	JUPITER Booster - BullSequana XH3000, GH Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Enterprise Linux, EVIDEN EuroHPC/FZJ Germany	4,801,344	1,000.00	1,226.28	15,794
5	Eagle - Microsoft NdV5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	



30 Years Ago vs. Now: Top HPC Systems in the Top500

Top 5 systems in the Top500, November 1995:

- **Cores:** 80–3,680
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D

Cores: ~563x–141,750x
Rmax: ~3,300,000x–18,300,000x

Top 5 systems in the Top 500, November 2025:

- **Cores:** 2,073,600–11,340,616
- **Rmax:** 561.2–1809 PFlop/s
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure

HPC HW has become far more capable...

Rank	System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)	Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	Numerical Wind Tunnel, Fujitsu National Aerospace Laboratory of Japan Japan	140	170.00	235.79	1	El Capitan - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/SC/Oak Ridge National Laboratory United States	11,340,000	1,809.00	2,821.10	29,685
2	XP/S140, Intel Sandia National Laboratories United States				2	Summit, AMD Optimized 3rd Gen EPYC, AMD Instinct MI250X, HPE National Aerospace Laboratory of Japan	9,066,176	1,353.00	2,055.72	24,607
3	XP/S-MP 150, Intel DOE/SC/Oak Ridge National Laboratory United States				3	Orion, AMD EPYC 9004, AMD Instinct MI300A, Slingshot-11, TOSS, HPE Compute Blade, Center GPU	9,264,128	1,012.00	1,980.01	38,698
4	T3D MC1024-8, Cray/HPE Government United States				4	Frontier, AMD EPYC 9004, AMD Instinct MI300A, Slingshot-11, TOSS, HPE 8000, GH Superchip chip, Quad-Rail NVIDIA Enterprise Linux, EVIDEN	4,801,344	1,000.00	1,226.28	15,794
5	VPP500/80, Fujitsu National Lab. for High Energy Physics Japan	80	98.90	128.00	5	Eagle, AMD EPYC 9004, AMD Instinct MI300A, Slingshot-11, TOSS, HPE NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	

And complex!

- **commodity vector processors**
- **commodity multicore/manycore processors**
- **multi-socket compute nodes**
- **NUMA processor/node architectures**
- **high-radix, low-diameter interconnects**
- **GPU computing**

(Often in ways that hurt programmability)



30 Years Ago vs. Now: Broadly-adopted HPC Programming Notations

Top 5 systems in the Top500, November 1995:

- **Cores:** 80–3,680
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D

**HPC HW has
become far
more capable...**

Top 5 systems in the Top 500, November 2025:

- **Cores:** 2,073,600–11,340,616
- **Rmax:** 561.2–1809 PFlop/s
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure

Broadly-adopted HPC notations, November 1995:

- **Languages:** Fortran, C, C++
- **Inter-node:** MPI, PVM, SHMEM
- **Intra-node:** Pthreads, vendor-specific pragmas & intrinsics
 - OpenMP on the horizon (1997)
- **Scripting:** Perl, sh/csh/tcsh, Tcl/TK

Broadly-adopted HPC notations, November 2025:

- **Languages:** Fortran, C, C++
- **Inter-node:** MPI, SHMEM
- **Intra-node:** Pthreads, OpenMP, Kokkos
- **GPUs:** CUDA, HIP, SYCL, OpenMP, Kokkos, OpenACC, OpenCL, ...
- **Scripting:** Python, bash



30 Years Ago vs. Now: Broadly-adopted HPC Programming Notations

Top 5 systems in the Top500, November 1995:

- **Cores:** 80–3,680
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D

HPC HW has become far more capable...

Top 5 systems in the Top 500, November 2025:

- **Cores:** 2,073,600–11,340,616
- **Rmax:** 561.2–1809 PFlop/s
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure

Broadly-adopted HPC notations, November 1995:

- **Languages:** Fortran, C, C++
- **Inter-node:** MPI, PVM, SHMEM
- **Intra-node:** Pthreads, vendor-specific pragmas & intrinsics
 - OpenMP on the horizon (1997)
- **Scripting:** Perl, sh/csh/tcsh, Tcl/TK

HPC SW notations have largely stayed the same, modulo GPU computing and scripting

Broadly-adopted HPC notations, November 2025:

- **Languages:** Fortran, C, C++
- **Inter-node:** MPI, SHMEM
- **Intra-node:** Pthreads, OpenMP, Kokkos
- **GPUs:** CUDA, HIP, SYCL, OpenMP, Kokkos, OpenACC, OpenCL, ...
- **Scripting:** Python, bash

Notably, HPC has failed to broadly adopt any new programming languages...

30 Years Ago vs. Now: Ubiquity of Parallelism

Top 5 systems in the Top500, November 1995:

- **Cores:** 80–3,680
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D

HPC HW has become far more capable...

Top 5 systems in the Top 500, November 2025:

- **Cores:** 2,073,600–11,340,616
- **Rmax:** 561.2–1809 PFlop/s
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure

Broadly-adopted HPC notations, November 1995:

- **Languages:** Fortran, C, C++
- **Inter-node:** MPI, PVM, SHMEM
- **Intra-node:** Pthreads, vendor-specific pragmas & intrinsics
 - OpenMP on the horizon (1997)
- **Scripting:** Perl, sh/csh/tcsh, Tcl/TK

HPC SW notations have largely stayed the same

Broadly-adopted HPC notations, November 2025:

- **Languages:** Fortran, C, C++
- **Inter-node:** MPI, SHMEM
- **Intra-node:** Pthreads, OpenMP, Kokkos
- **GPUs:** CUDA, HIP, SYCL, OpenMP, Kokkos, OpenACC, OpenCL, ...
- **Scripting:** Python, bash

Availability of parallelism, November 1995

- HPC systems at national labs, large universities, industry

Parallelism has expanded beyond HPC, dramatically!

Availability of parallelism, November 2025:

- HPC systems at national labs, universities, industry
- laptops / desktops with multicore CPUs and GPUS
- cloud computing, AI data centers, ...



What is Chapel?

Chapel: A modern parallel programming language

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

Goals:

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



Chapel's Approach

Top 5 systems in the Top500, November 1995:

- **Cores:** 80–3,680
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D

**Chapel replaces the need to mix multiple notations:
Use a single language for parallelism at all levels,
targeting parallel systems of all types and scales**

**key: express parallelism and locality
independently of hardware mechanisms**

Top 5 systems in the Top 500, November 2025:

- **Cores:** 2,073,600–11,340,616
- **Rmax:** 561.2–1809 PFlop/s
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure

Unified notation for parallelism, HPC or otherwise:

- **Languages:** Fortran, C, C++
 - **Inter-node:** MPI, SHMEM
 - **Intra-node:** Pthreads, OpenMP, Kokkos
 - **GPUs:** CUDA, HIP, SYCL, OpenMP, Kokkos, OpenACC, OpenGL, ...
 - **Scripting:** Python, bash
-

Availability of parallelism, November 2025:

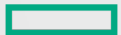
- HPC systems at national labs, universities, industry
 - laptops / desktop with multicore CPUs and GPUS
 - cloud computing, AI data centers, ...
-

Goals of this talk

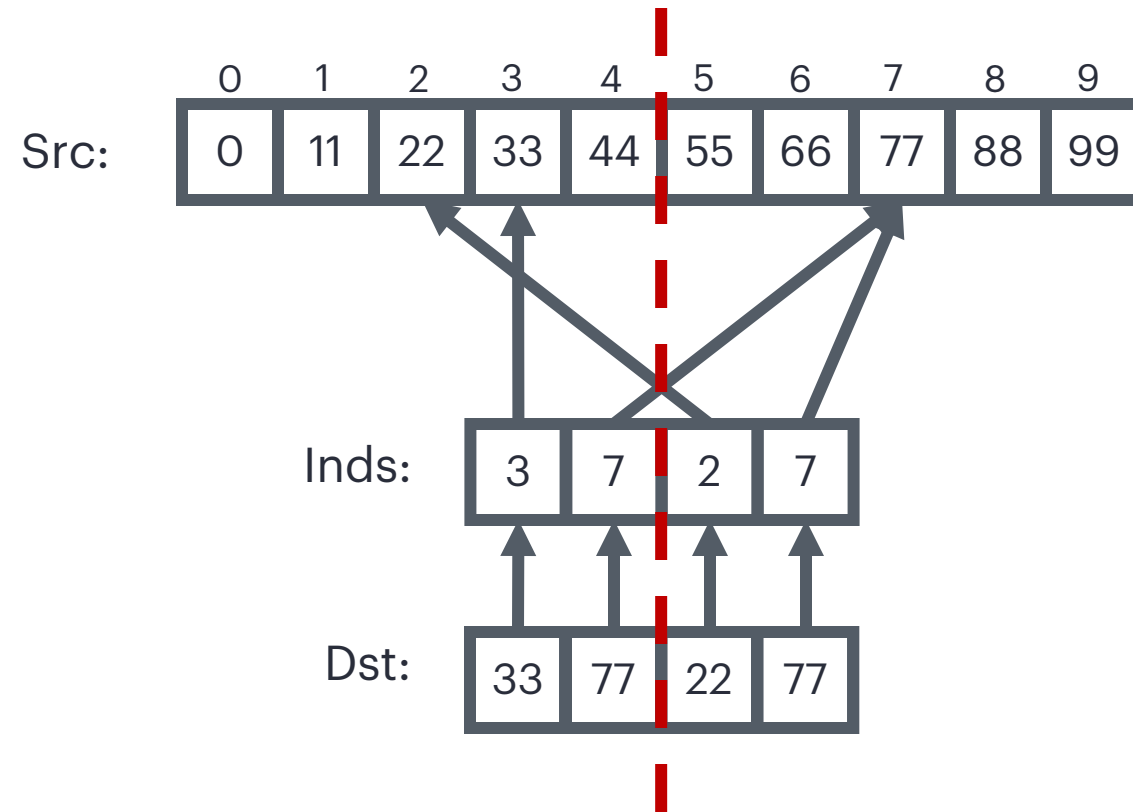
1. Motivate & introduce Chapel for those who've missed our previous NWC++ talks
2. Highlight some key milestones since our last talk in 2022:
 - Performance Results
 - GPU programming
 - User Applications
 - Chapel 2.0
 - HPSF
3. Provide information about how to learn more



Chapel By Example: Bale Index Gather

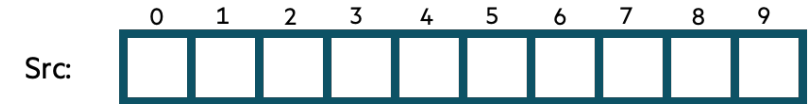


Bale Index Gather (IG): In Pictures



Bale IG in Chapel: Scalar and Array Declarations

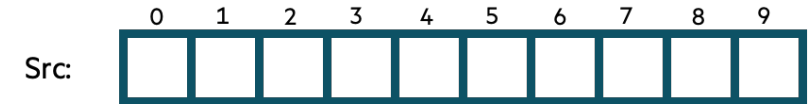
```
config const n = 10,  
            m = 4;  
  
var Src: [0..<n] int,  
    Inds, Dst: [0..<m] int;
```



\$

Bale IG in Chapel: Compiling

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



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

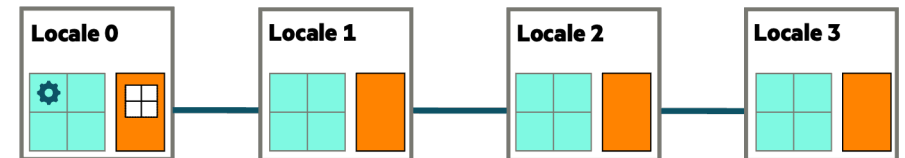
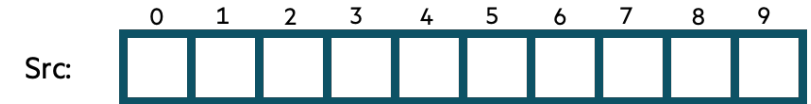
```
$
```



Bale IG in Chapel: Executing

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

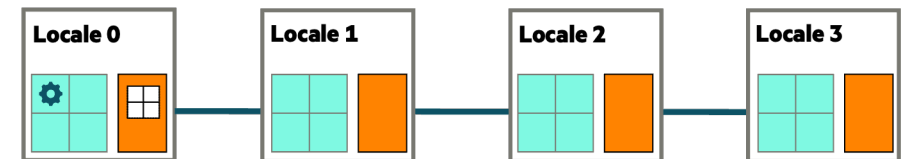
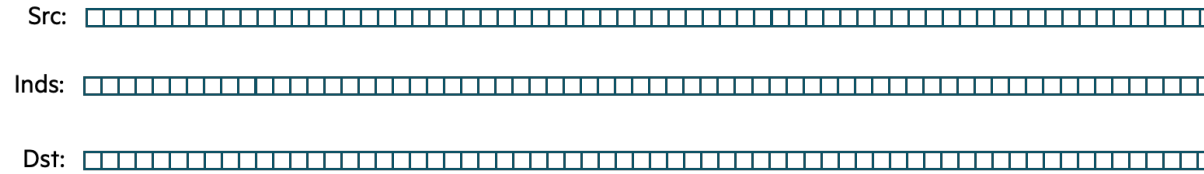
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



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 -nl 4 --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 -nl 4  
$
```

Src:

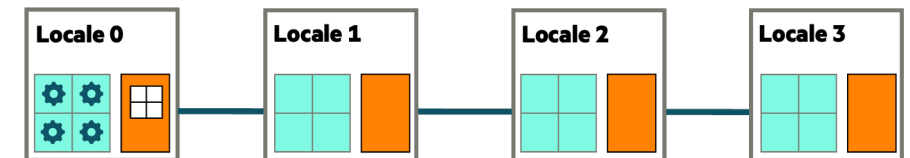
0	1	2	3	4	5	6	7	8	9
0	11	22	33	44	55	66	77	88	99

Inds:

3	7	2	7
---	---	---	---

Dst:

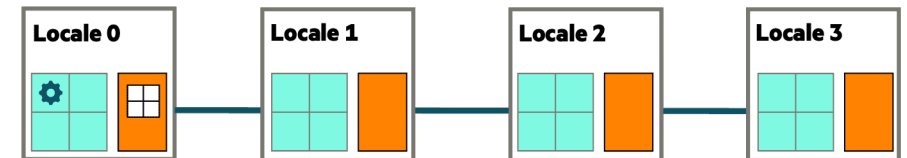
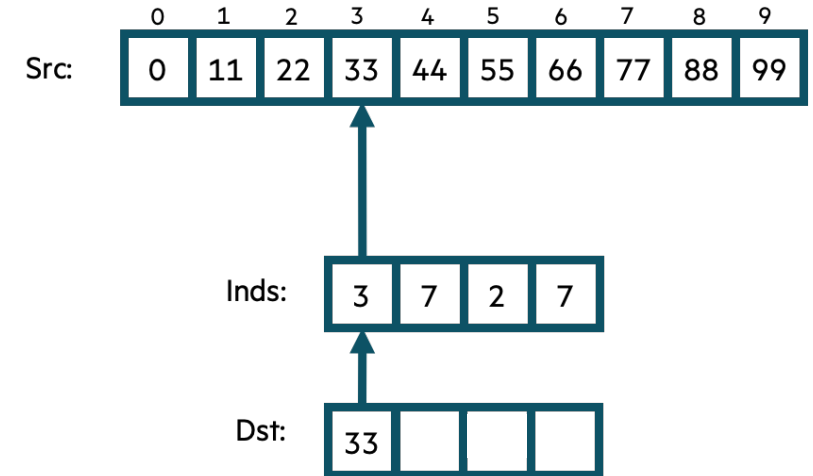
--	--	--	--



Bale IG in Chapel: Serial Version

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

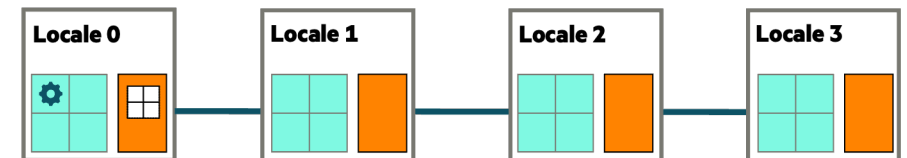
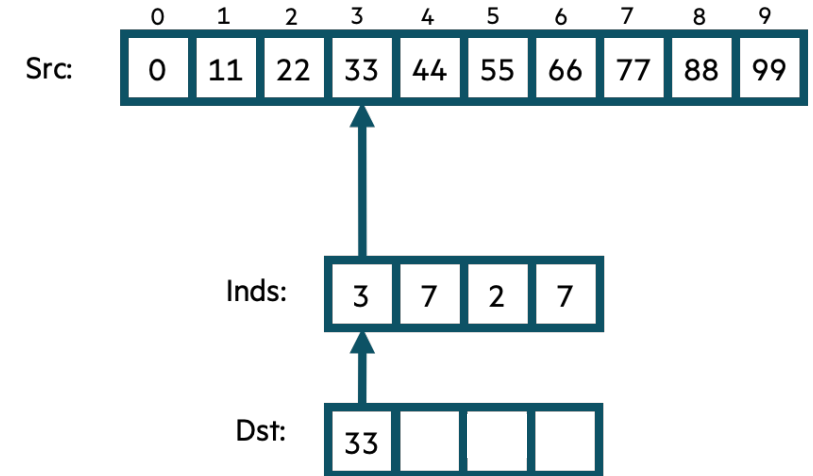
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



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];
```

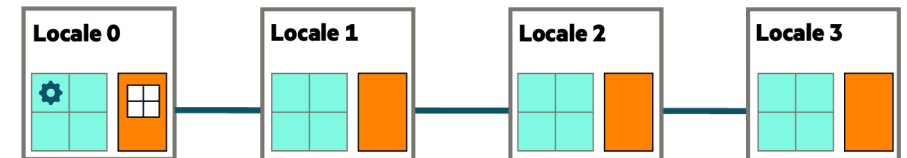
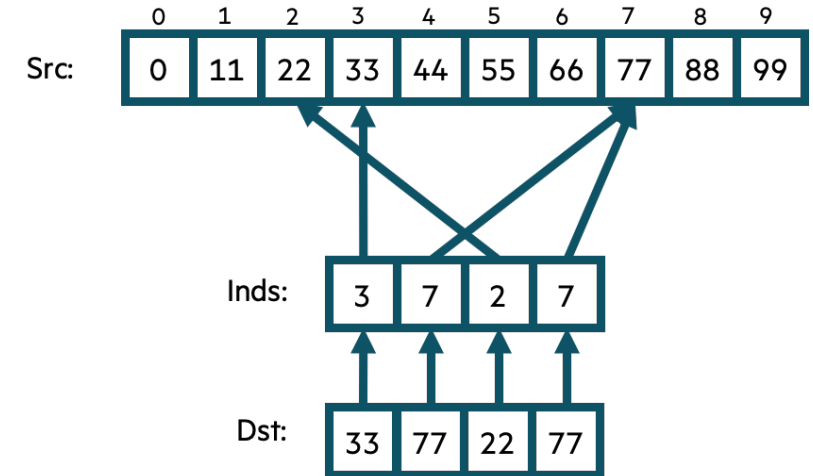
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



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];
```

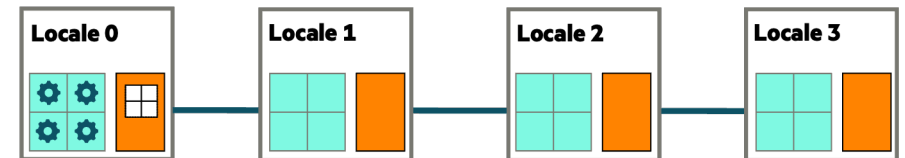
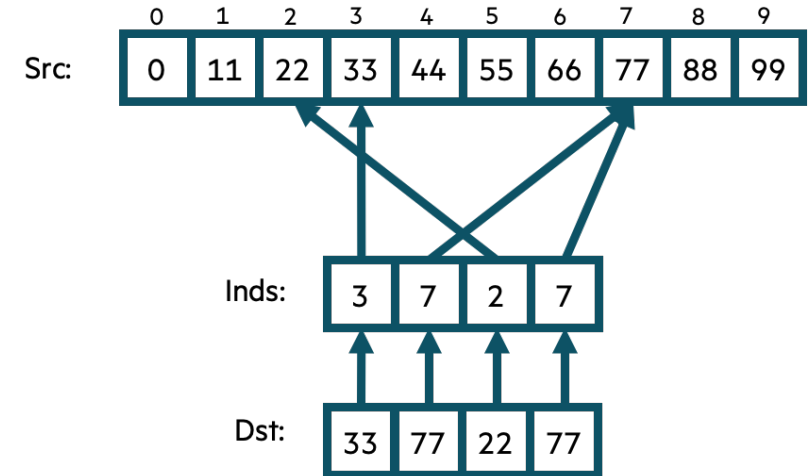
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



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

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

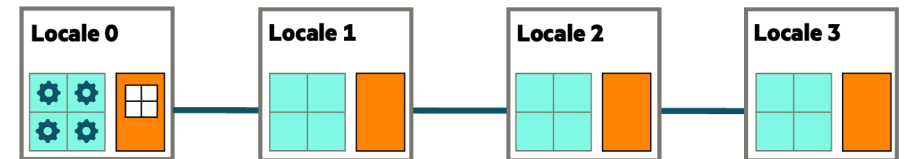
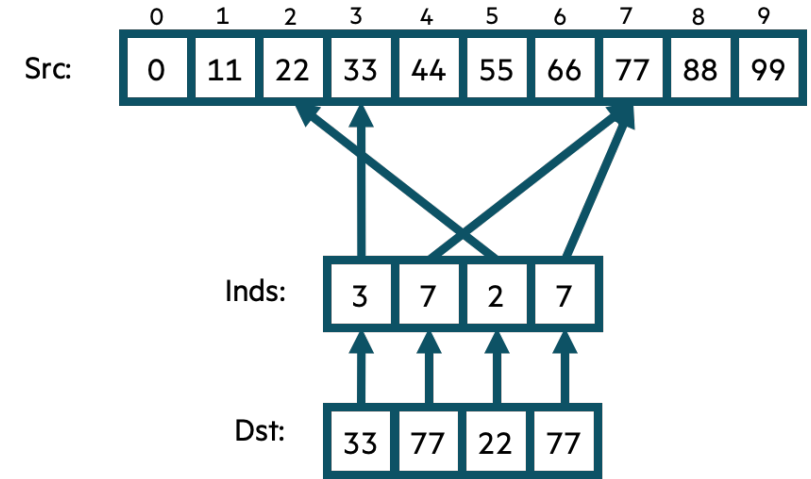
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



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];
```

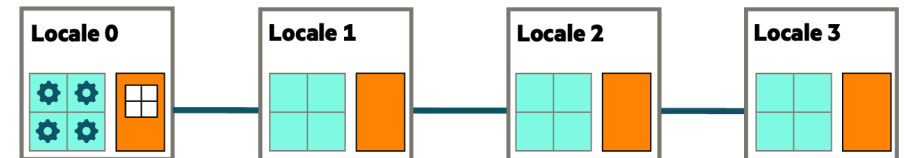
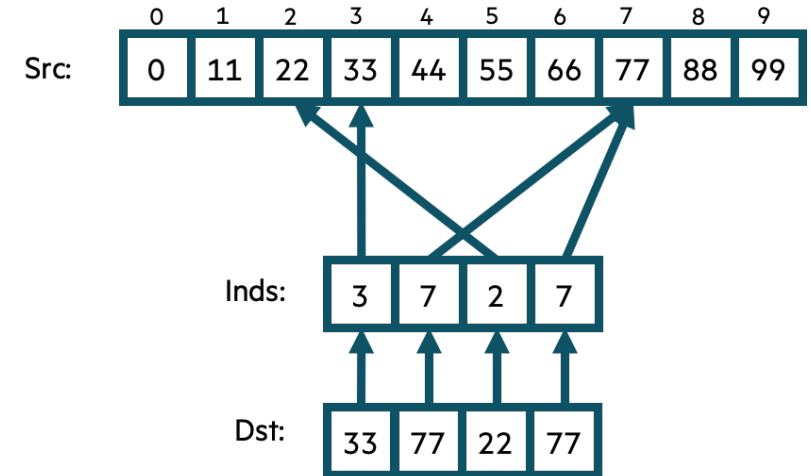
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



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

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

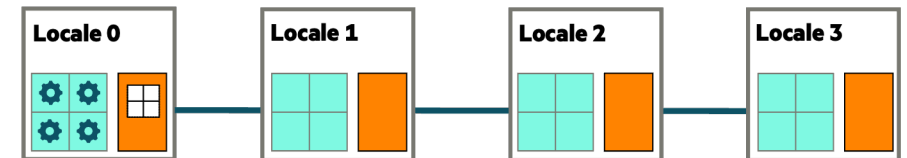
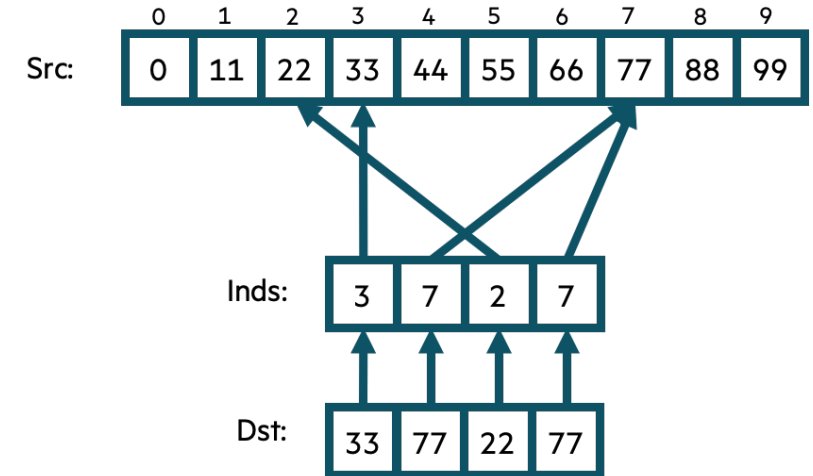
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



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];
```

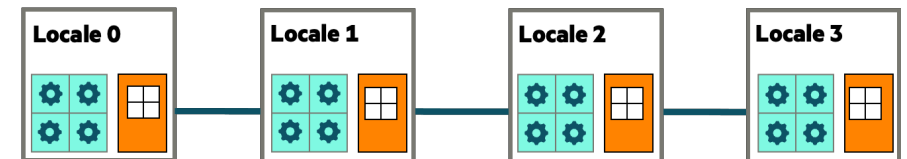
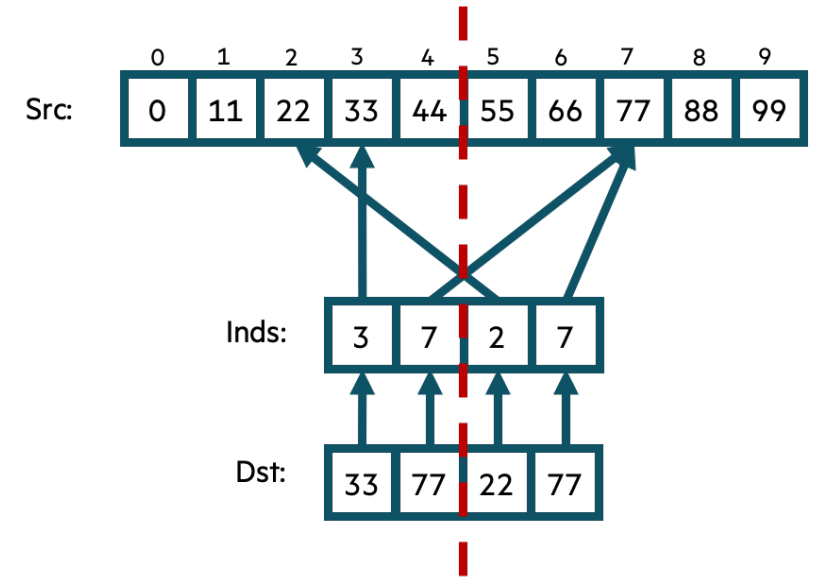
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



Bale IG in Chapel: Distributed Parallel Version

```
use BlockDist;  
  
config const n = 10,  
            m = 4;  
  
const SrcInds = blockDist.createDomain(0..<n),  
       DstInds = blockDist.createDomain(0..<m);  
  
var Src: [SrcInds] int,  
     Inds, Dst: [DstInds] int;  
  
...  
forall (d, i) in zip(Dst, Inds) do  
    d = Src[i];
```

```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



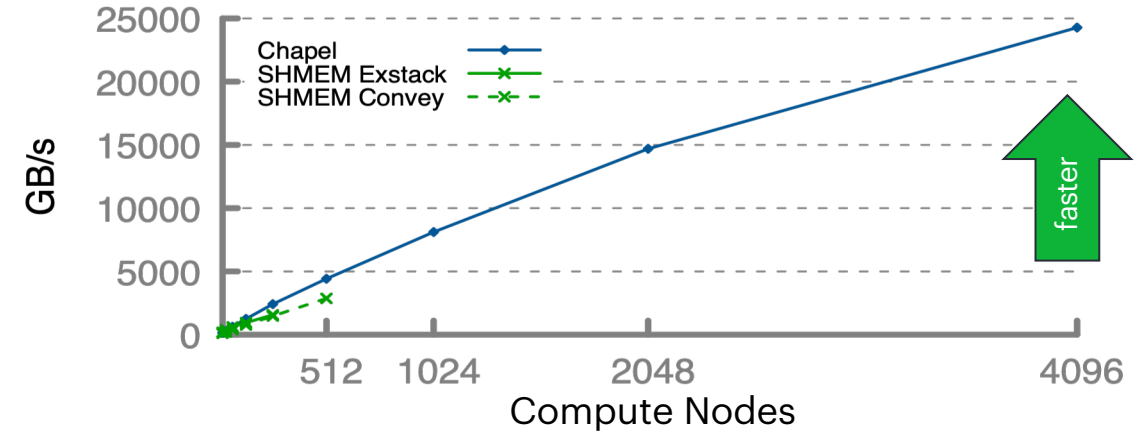
Bale IG in Chapel: Distributed Parallel Version on HPE Cray EX

```
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=...  
$
```

Bale Indexgather Performance

HPE Cray EX (Slingshot-11)



Bale IG in Chapel vs. SHMEM on HPE Cray EX

Chapel

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

SHMEM (Exstack version)

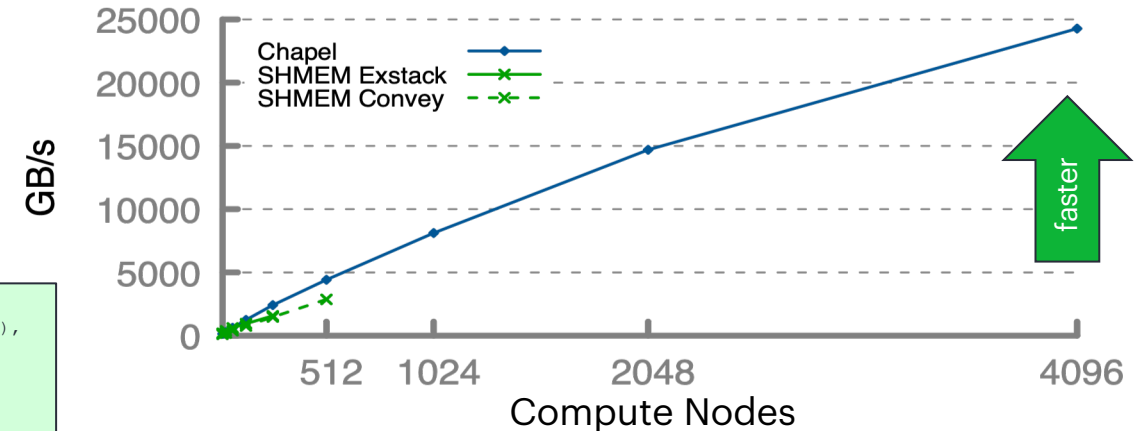
```
i=0;
while( exstack_proceed(ex, (i==l_num_req)) ) {
    i0 = i;
    while(i < l_num_req) {
        l_indx = pckindx[i] >> 16;
        pe = pckindx[i] & 0xffff;
        if(!exstack_push(ex, &l_indx, pe))
            break;
        i++;
    }
    exstack_exchange(ex);
    while(exstack_pop(ex, &idx, &fromth)) {
        idx = ltable[idx];
        exstack_push(ex, &idx, fromth);
    }
    lgp_barrier();
    exstack_exchange(ex);
    for(j=i0; j<i; j++) {
        fromth = pckindx[j] & 0xffff;
        exstack_pop_thread(ex, &idx, (uint64_t)fromth);
        tgt[j] = idx;
    }
    lgp_barrier();
}
```

SHMEM (Conveyors version)

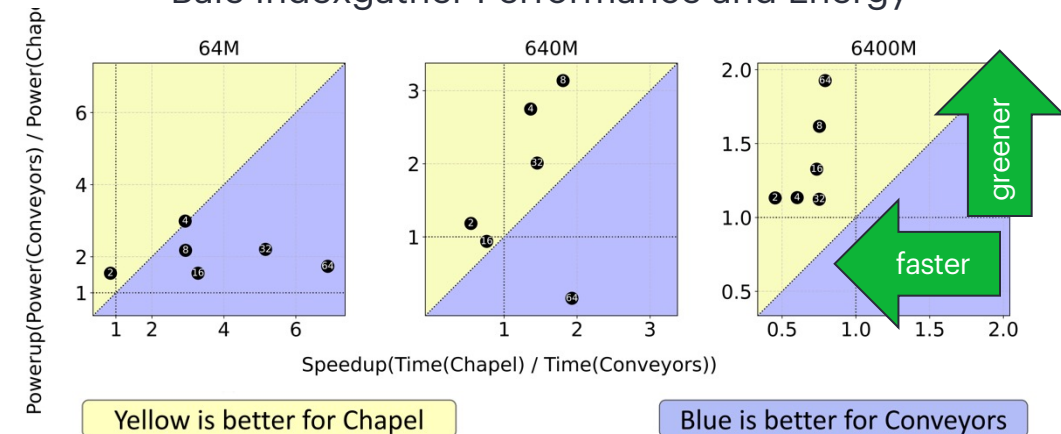
```
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
        more | convey_advance(replies, !more)) {
    for (; i < l_num_req; i++) {
        pkg.idx = i;
        pkg.val = pckindx[i] >> 16;
        pe = pckindx[i] & 0xffff;
        if (!convey_push(requests, &pkg, pe))
            break;
    }
    while (convey_pull(requests, ptr, &from) == convey_OK) {
        pkg.idx = ptr->idx;
        pkg.val = ltable[ptr->val];
        if (!convey_push(replies, &pkg, from)) {
            convey_unpull(requests);
            break;
        }
    }
    while (convey_pull(replies, ptr, NULL) == convey_OK)
        tgt[ptr->idx] = ptr->val;
}
```

Bale Indexgather Performance

HPE Cray EX (Slingshot-11)



Bale Indexgather Performance and Energy



Summary of this section

Chapel supports language features for:

- **data parallelism:**

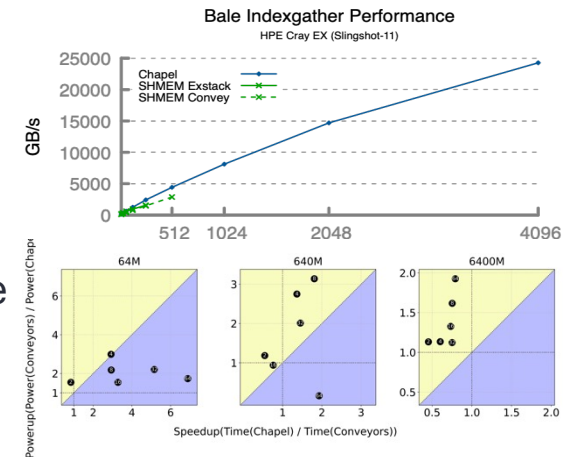
- **foreach:** parallelism without creating tasks (e.g., vectorization)
- **forall:** parallelism using an iterand-defined number of tasks
- **[i in Iter]:** equivalent to 'forall' if *Iter* supports parallel iteration, serial otherwise
- **whole-array operations / promotion:** call scalar operators & procedures with arrays
- **parallel zippered iteration:** loop over multiple iterands simultaneously
- **distributed arrays:** declared using a global problem size, mapped to multiple nodes

```
foreach (d, i) in  
forall (d, i) in  
[i in 0..Dst = Src[Inds];  
zip(Dst, Inds)
```

```
SrcInds = blockDist.createDomain(0..Src: [SrcInds] int
```

Chapel's features support:

- **great performance and scalability:** shown here using 4k processors, 500,000+ cores
- **energy-efficiency:** due in part to running a process per node or socket rather than core

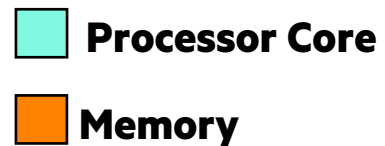
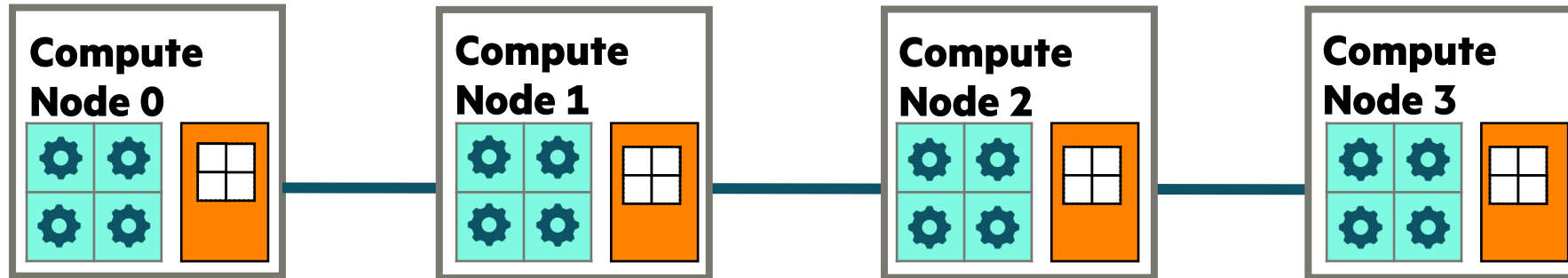


“Low-level” Chapel Features and GPU support



Key Concerns for Scalable Parallel Computing

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

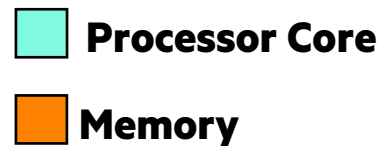
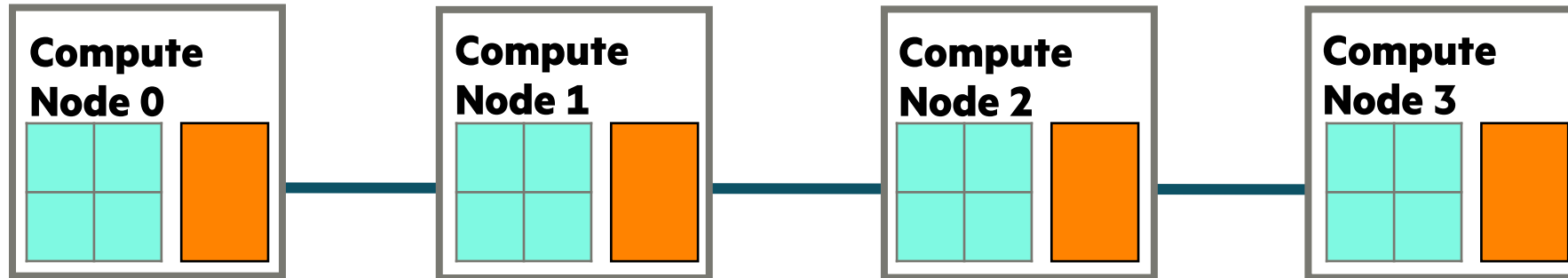


Locales in Chapel

In Chapel, a *locale* refers to a compute resource with...

- processors, so it can run tasks
- memory, so it can store variables

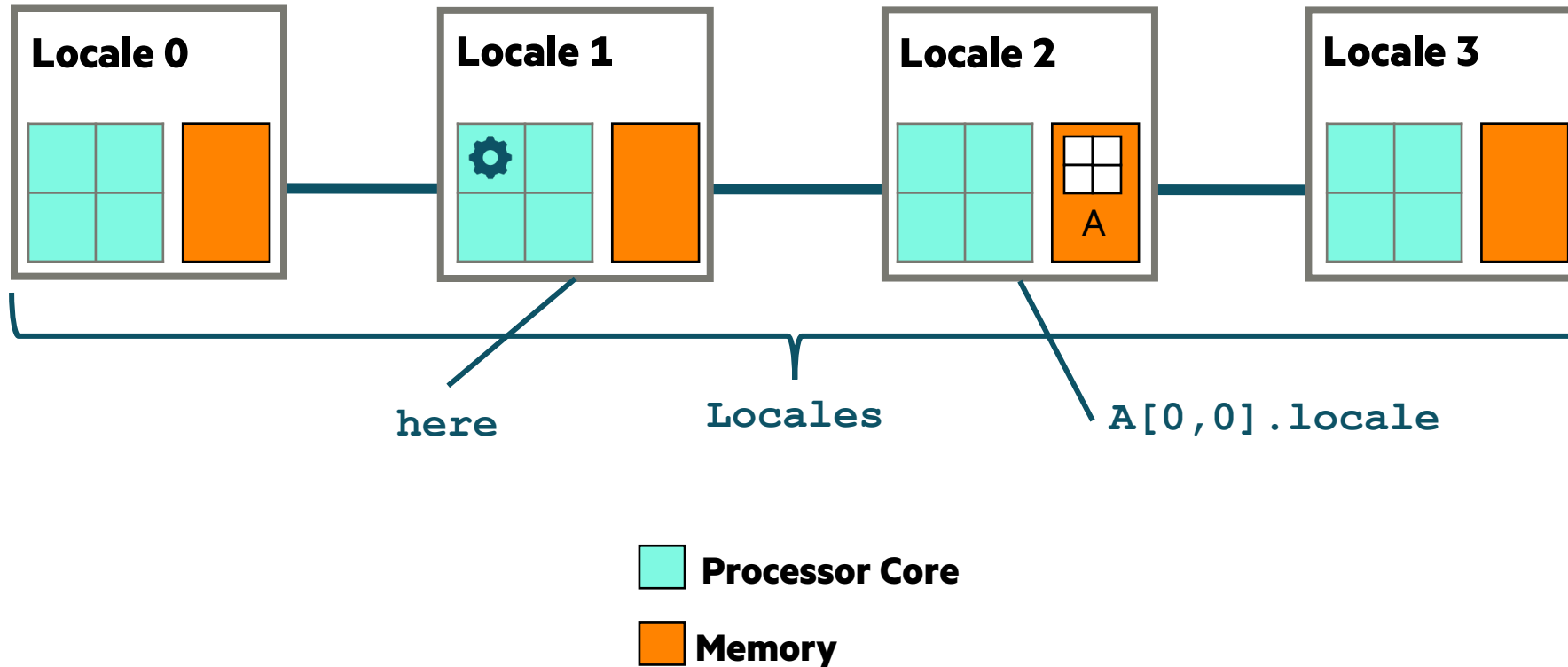
For now, think of each compute node as being a locale



Reasoning About Locales in Chapel

Built-in features for reasoning about locality within Chapel:

- **Locales:** An array of locale values representing the system resources on which the program is running
- **here:** The locale on which the current task is executing
- **expr.locale:** The locale on which a given variable is stored



Basic Features for Locality

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

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

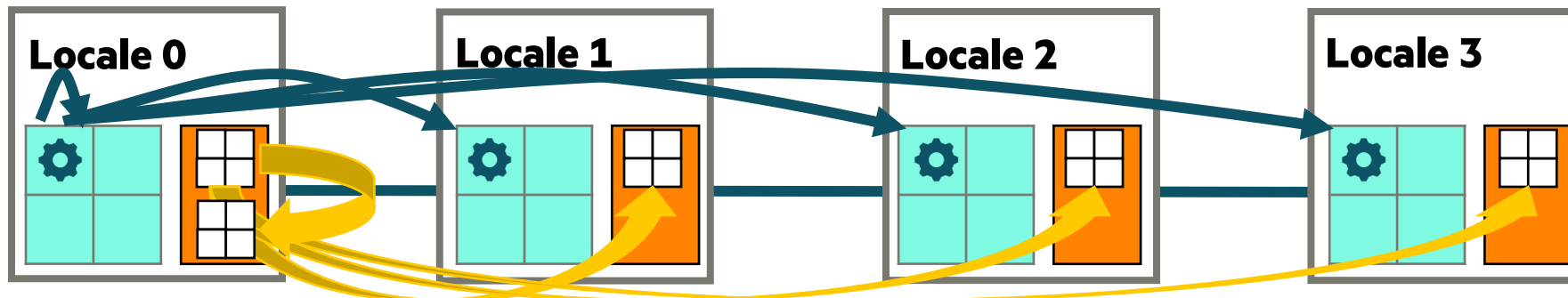
Variables are stored using the memory local to the current task

This loop will serially iterate over the program's locales

on-clauses move tasks to target locales

remote variables can be accessed directly

This is a distributed, yet serial, computation

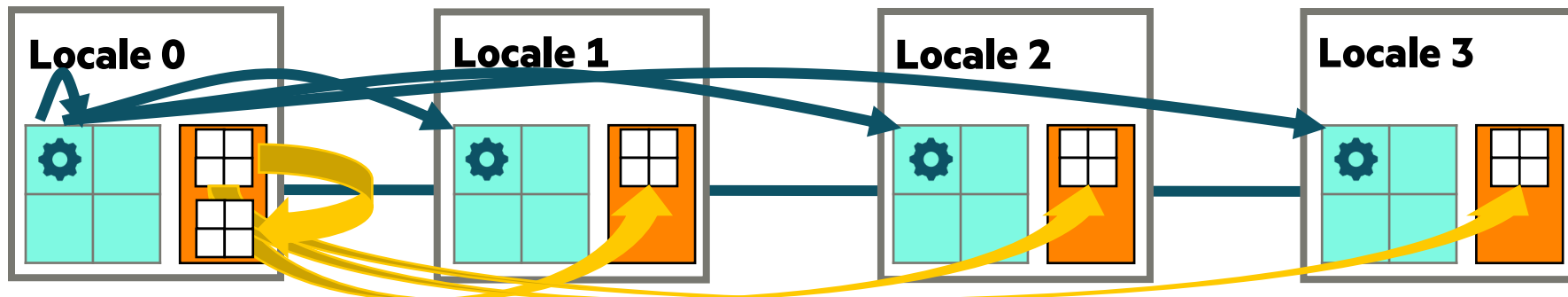


Mixing Locality with Task Parallelism

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

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

This is a distributed parallel computation



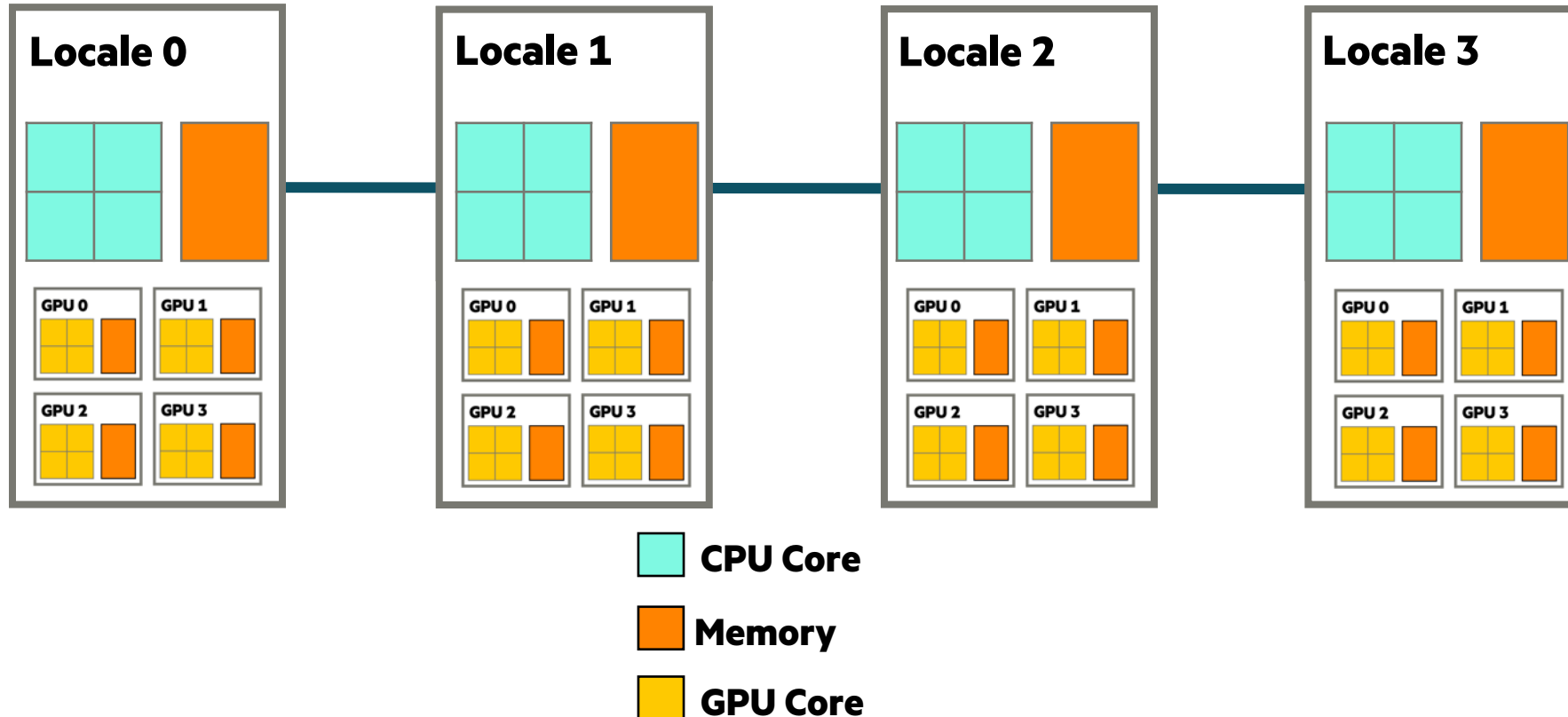
Chapel with GPUs

In Chapel, we represent GPUs as *sub-locales*

- Each top-level locale may have an array of locales called 'gpus'
- We can then target them using Chapel's traditional features for parallelism + locality

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

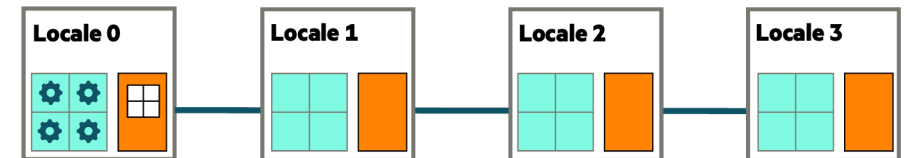
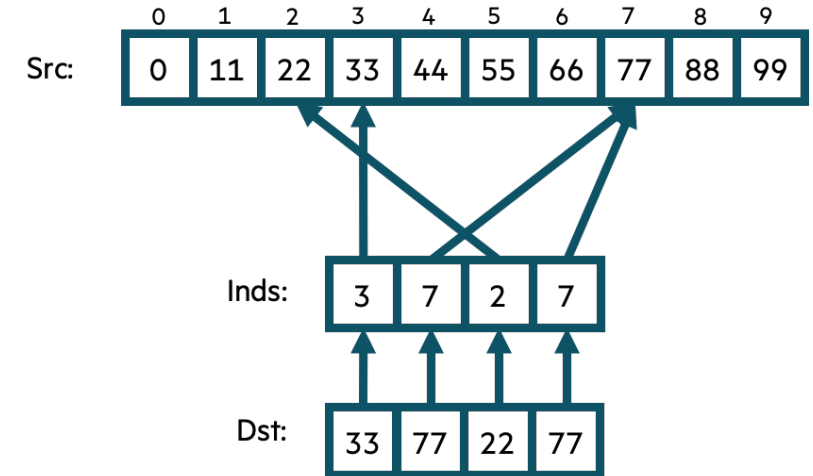
```
coforall gpu in here.gpus do on gpu { ... }
```



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];
```

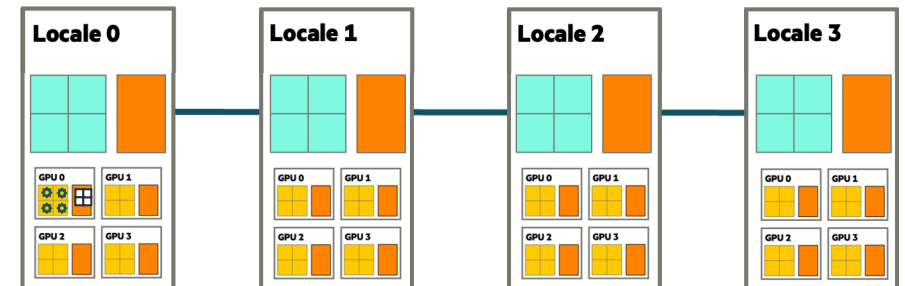
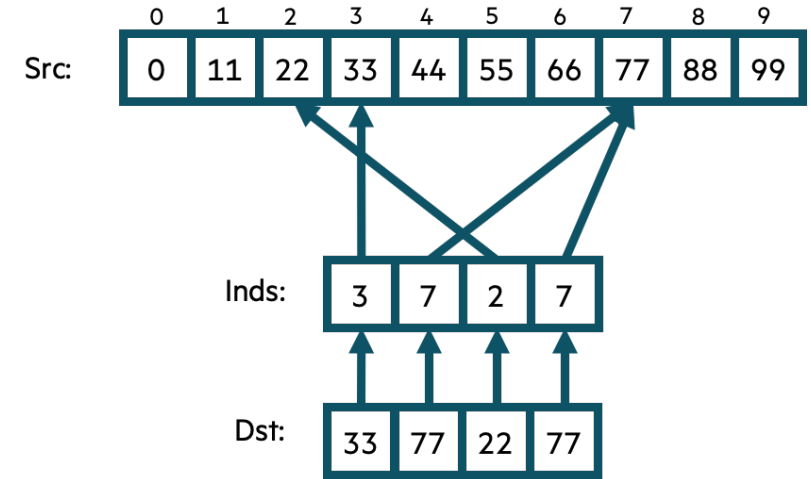
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



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

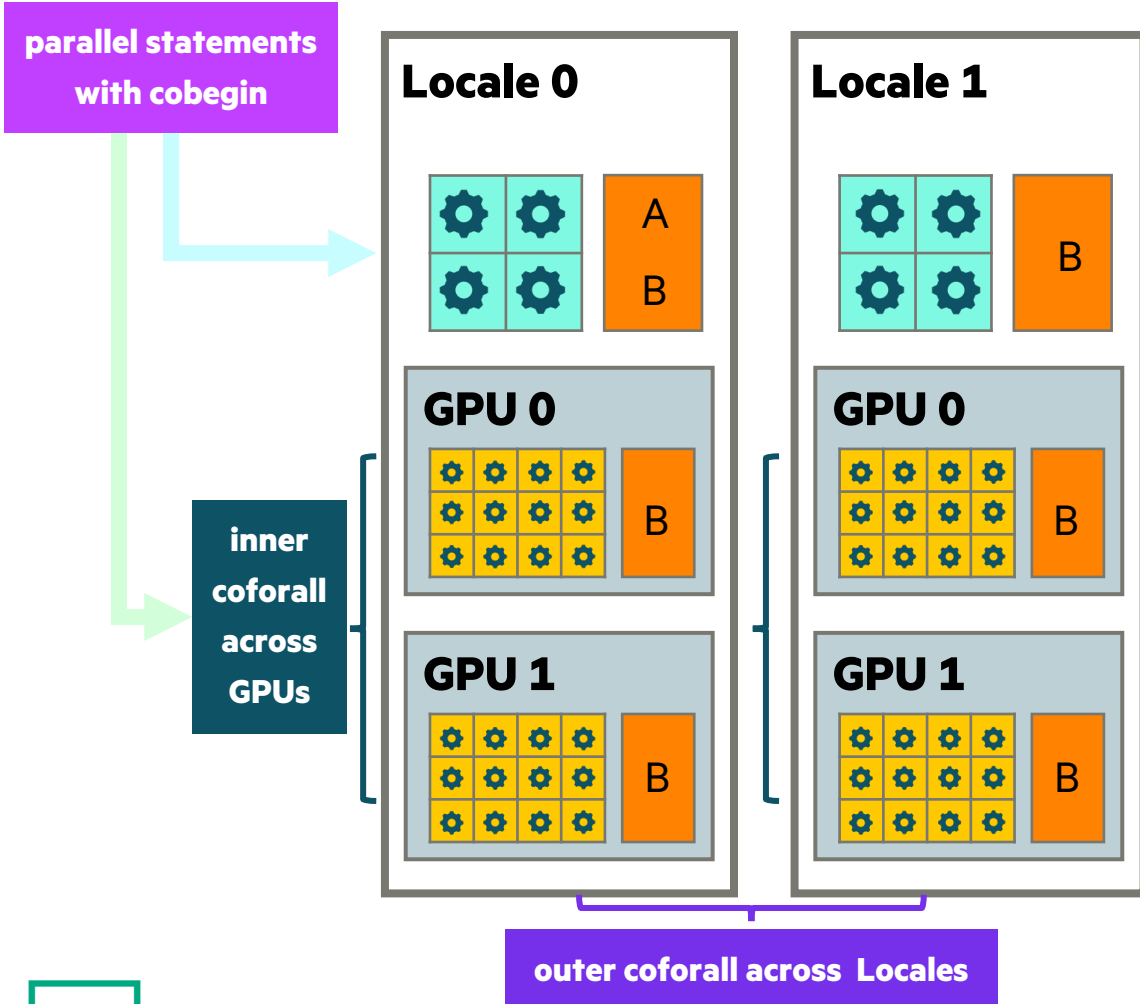
```
config const n = 10,  
            m = 4;  
  
on here.gpus[0] {  
  var Src: [0..<n] int,  
      Inds, Dst: [0..<m] int;  
  ...  
  forall (d, i) in zip(Dst, Inds) do  
    d = Src[i];  
}
```

```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



Targeting CPUs and GPUs using Parallelism and Locality

■ CPU Core
 ■ GPU Core
 ■ Memory



```

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

Summary of this section

Chapel supports language features for:

- **task parallelism:** creation of explicit tasks
 - **coforall:** creates a task per iteration
 - **cobegin:** creates a task per child statement

```
coforall loc in  
cobegin
```

- **locality:**

- **locales:** represent locality on the target system
- **on-clauses:** specify where data should be allocated and tasks should be run
- **locality queries:**
 - "Where is this task running?"
 - "Where is this data allocated?"
- **sub-locales:** represent nested, potentially heterogeneous locality (e.g., GPUs)

```
Locales  
on loc
```

```
here  
expr.locale  
here.gpus
```



Chapel Features Not Covered Tonight

Rich base language features:

- **object-oriented programming:** value- and reference-based objects
- **iterators**
- **procedures:** generics, polymorphism, overloading, default arguments, keyword-based argument passing
- **namespacing**
- **interoperability**
- ...

Additional parallel features:

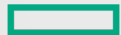
- **begin:** create asynchronous tasks
- **atomic / sync variables:** synchronize between tasks
- **rich array types and operations:** multidimensional, associative, sparse, slicing, reindexing, ...
- **reductions / scans:** apply operations to collections
- **user-defined parallel iterators and distributions**
- ...

Tools: linter, language server, VSCode support, docs-generation, debugger, package manager, ...

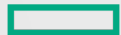


Live Demo of Chapel

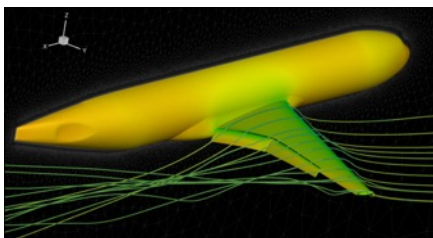
Jade Abraham



Applications of Chapel

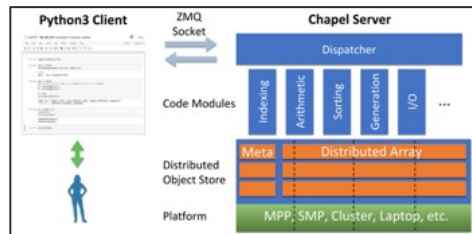


Applications of Chapel



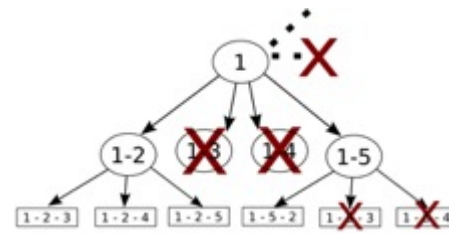
CHAMPS: 3D Unstructured CFD

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



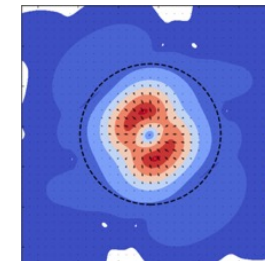
Arkouda: Interactive Data Science at Massive Scale

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



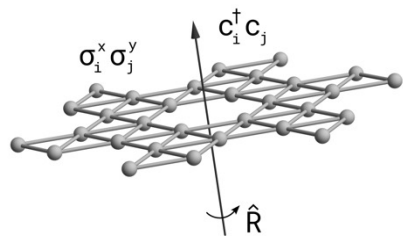
ChOp: Chapel-based Optimization

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



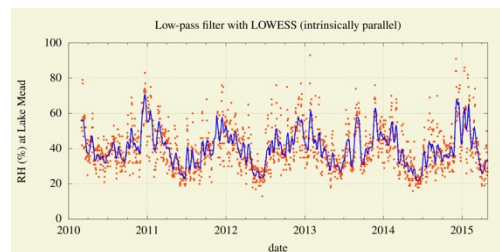
ChplUltra: Simulating Ultralight Dark Matter

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



Lattice-Symmetries: a Quantum Many-Body Toolbox

Tom Westerhout
Radboud University



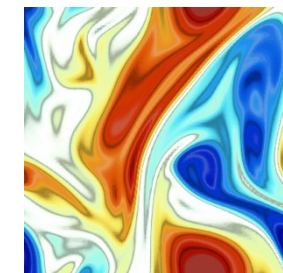
Desk dot chpl: Utilities for Environmental Eng.

Nelson Luis Dias
The Federal University of Paraná, Brazil



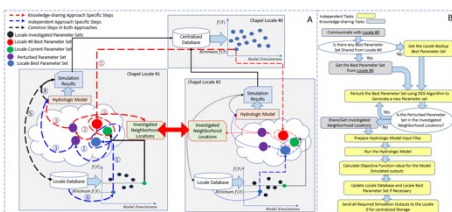
RapidQ: Mapping Coral Biodiversity

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



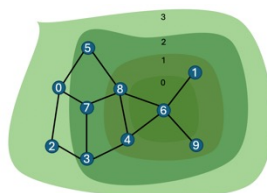
ChapQG: Layered Quasigeostrophic CFD

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



Chapel-based Hydrological Model Calibration

Marjan Asgari et al.
University of Guelph



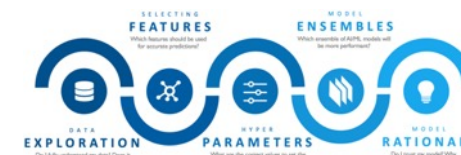
Arachne Graph Analytics

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



Modeling Ocean Carbon Dioxide Removal

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



CrayAI HyperParameter Optimization (HPO)

Ben Albrecht et al.
Cray Inc. / HPE

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

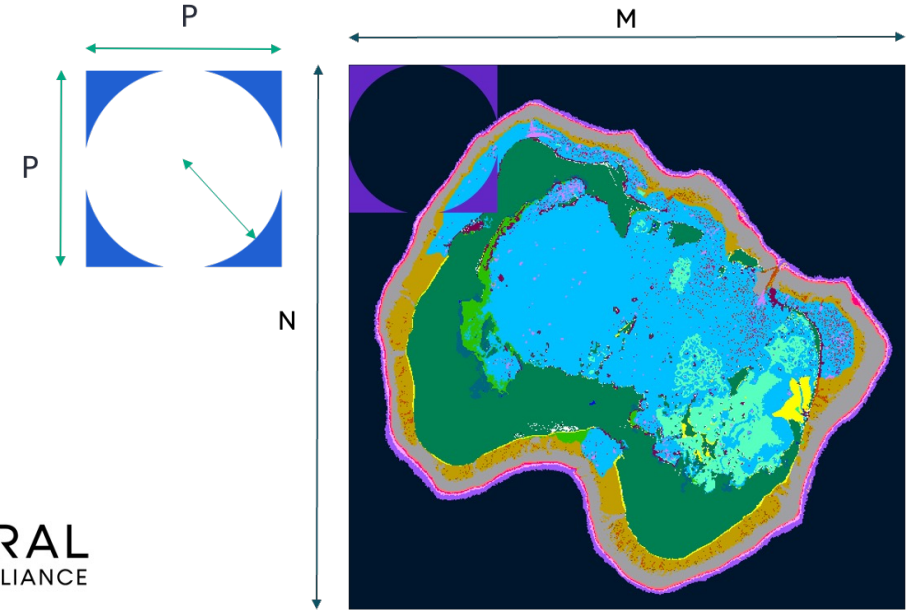
RapidQ Coral Biodiversity Application

What is it?

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

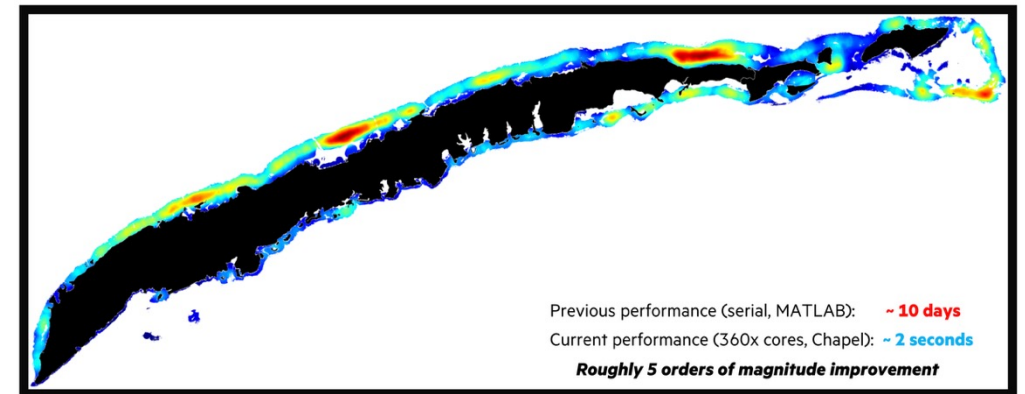
Who wrote it?

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



Why Chapel?

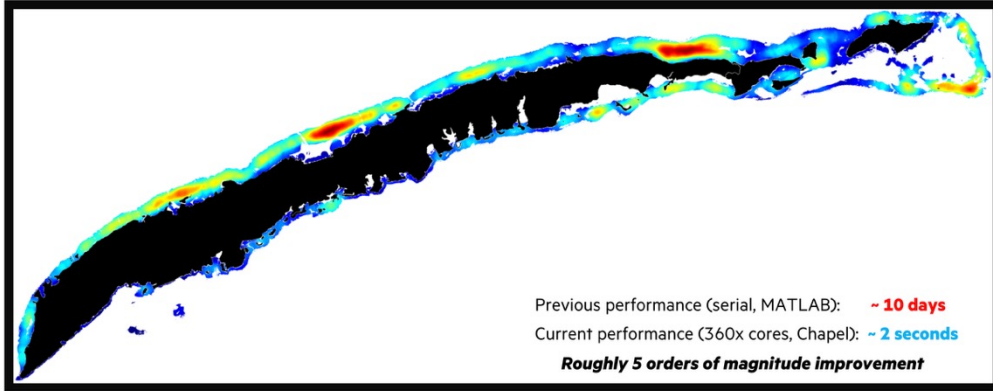
- easy transition from Python/Matlab, which were being used
- massive performance improvement:
 - ~10-day Matlab run finished in ~2 seconds using 360 cores
- enabled unexpected algorithmic improvements



From Scott Bachman's CHI UW 2023 talk:
<https://youtu.be/IJhh9KLL2X0>

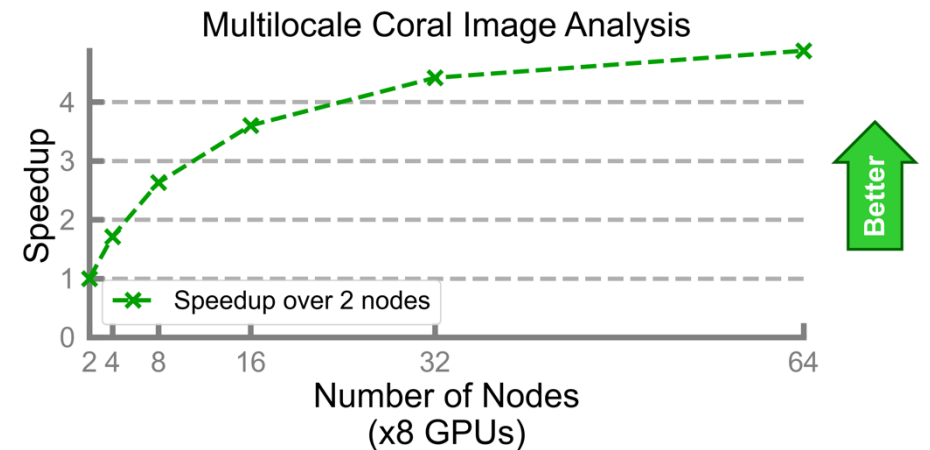
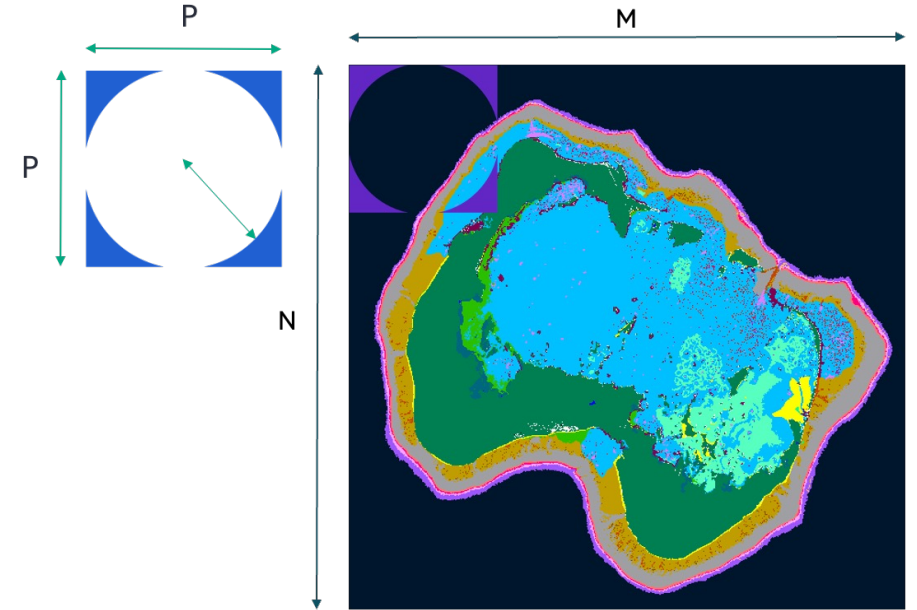
RapidQ Improvements on GPUs

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

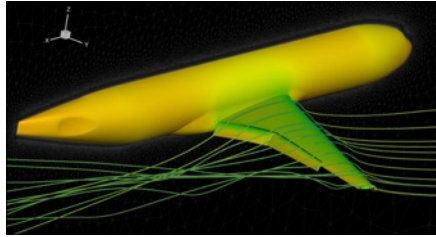


Improved algorithm: Spectral Diversity, $O(M \cdot N \cdot P^3)$

- Chapel run was estimated to require ~4 weeks on an 8-core desktop
- code was updated to leverage GPUs during Chapel's early GPU days
 - required adding ~90 lines of code for a total of ~320
 - ran on Frontier, using NVIDIA K20X Kepler GPUs
- ran in ~87 minutes on 2 nodes (16 GPUs)
- and in ~17 minutes on 64 nodes (512 GPUs), a ~5x improvement



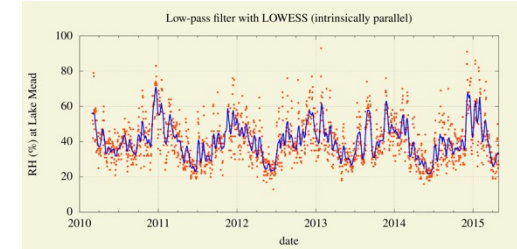
Productivity Across Diverse Application Scales (code and system size)



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




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



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


 **7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel**
Posted on September 17, 2024.
Tags: Computational Fluid Dynamics, User Experiences, Interviews
By: [Engin 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.”

 **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.”

 **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 Luis Dias from Brazil who is using Chapel to analyze data generated by the [Amazon Tall Tower Observatory \(ATTO\)](#), a project dedicated to long-term, 24/7 monitoring of greenhouse gas fluctuations. Read on

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

“7 Questions for Chapel Users” Interview Series

Chapel’s proven to be generally applicable & attractive for:  Chapel Language Blog

- Computational Fluid Dynamics
- Earth Sciences
- Exploratory Data Analytics
- Astrophysics
- Graph Analysis
- Artificial Intelligence
- ...

About Chapel Website Featured Series Tags Authors All Posts

7 Questions for Scott Bachman: Analyzing Coral Reefs with Chapel



Posted on October 1, 2024.

Tags: Earth Sciences Image Analysis GPUs

User Experiences Interviews

7 Questions for Akihiro Hayashi: Early Chapel GPU Support through Multiresolution Abstractions



Posted on March 18, 2026.

Tags: GPUs User Experiences Interviews ChapelCon

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

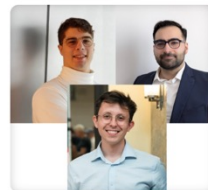


Posted on July 30, 2025.

Tags: Searching / Sorting GPUs User Experiences

Interviews

7 Questions for CHAMPS Developers: Empowering Academic R&D to Create Cutting-Edge CFD Apps in Chapel



Posted on March 26, 2026.

Tags: Computational Fluid Dynamics User Experiences

Interviews

By: Engin Kayraklioglu, Brad Chamberlain
Part of a series: [7 Questions for Chapel Users](#)

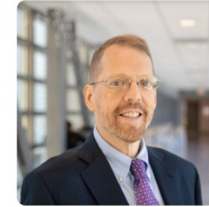
7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel



Posted on September 17, 2024.

Tags: Computational Fluid Dynamics User Experiences Interviews

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



Posted on November 6, 2024.

Tags: Graph Analytics Arkouda User Experiences Interviews

7 Questions for Marjan Asgari: Optimizing Hydrological Models with Chapel



Posted on September 15, 2025.

Tags: Earth Sciences User Experiences Interviews

7 Questions for Nelson Luís Dias: Atmospheric Turbulence in Chapel



Posted on October 15, 2024.

Tags: Earth Sciences Computational Fluid Dynamics

User Experiences Interviews Data Analysis

By: Engin Kayraklioglu, Brad Chamberlain
Part of a series: [7 Questions for Chapel Users](#)



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

Posted on February 12, 2025.

Tags: Data Analysis Arkouda User Experiences Interviews

7 Questions for Oliver Alvarado Rodriguez: Exploiting Chapel’s Distributed Arrays for Graph Analysis through Arachne



Posted on January 21, 2026.

Tags: Graph Analytics Arkouda Sparse Arrays

User Experiences Interviews

By: Engin Kayraklioglu, Brad Chamberlain
Part of a series: [7 Questions for Chapel Users](#)

Chapel Resources and News



Where Does Chapel Run?

In the Browser:

- Attempt This Online (ATO)
- GitHub Codespaces

Laptops/Desktops:

- Linux/UNIX
- Mac OS X
- Windows (leveraging WSL)

HPC Systems:

- Commodity clusters
- HPE/Cray supercomputers, such as:
 - Frontier
 - Perlmutter
 - Piz Daint
 - Polaris
 - ...
- Other vendors' supercomputers

Cloud:

- AWS
- Microsoft Azure (?)
- Google Cloud (?)

CPUs:

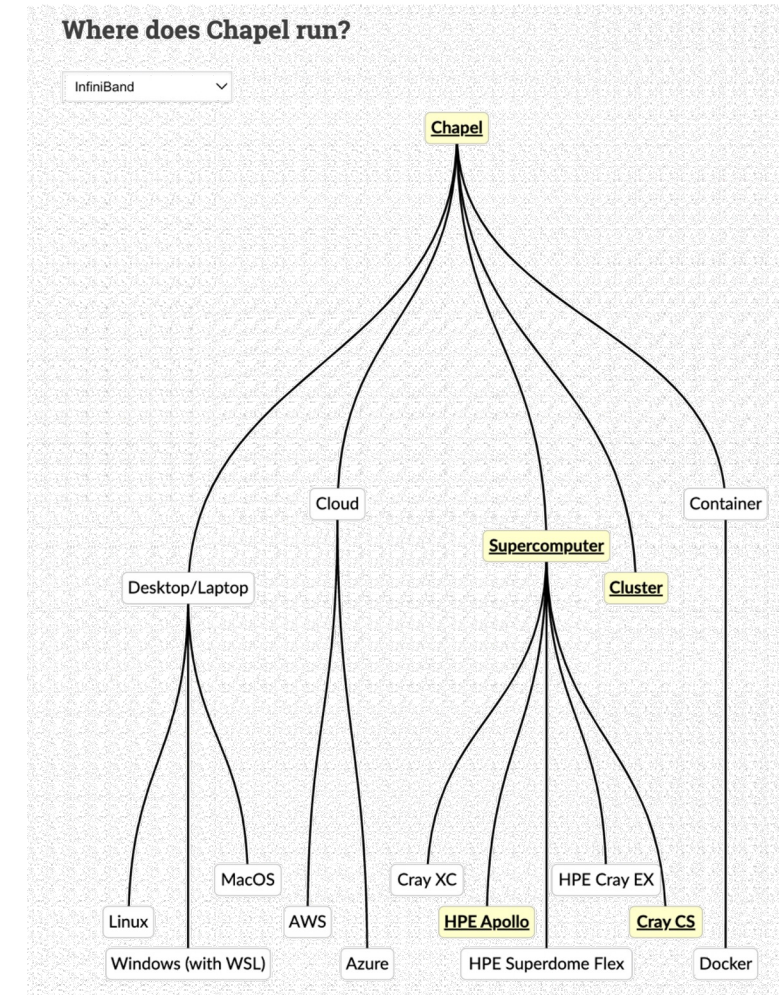
- Intel
- AMD
- Arm (M1/M2, Graviton, A64FX, Raspberry Pi, ...)

GPUs:

- NVIDIA
- AMD

Networks:

- Slingshot
- Aries/Gemini
- InfiniBand
- AWS EFA
- Ethernet

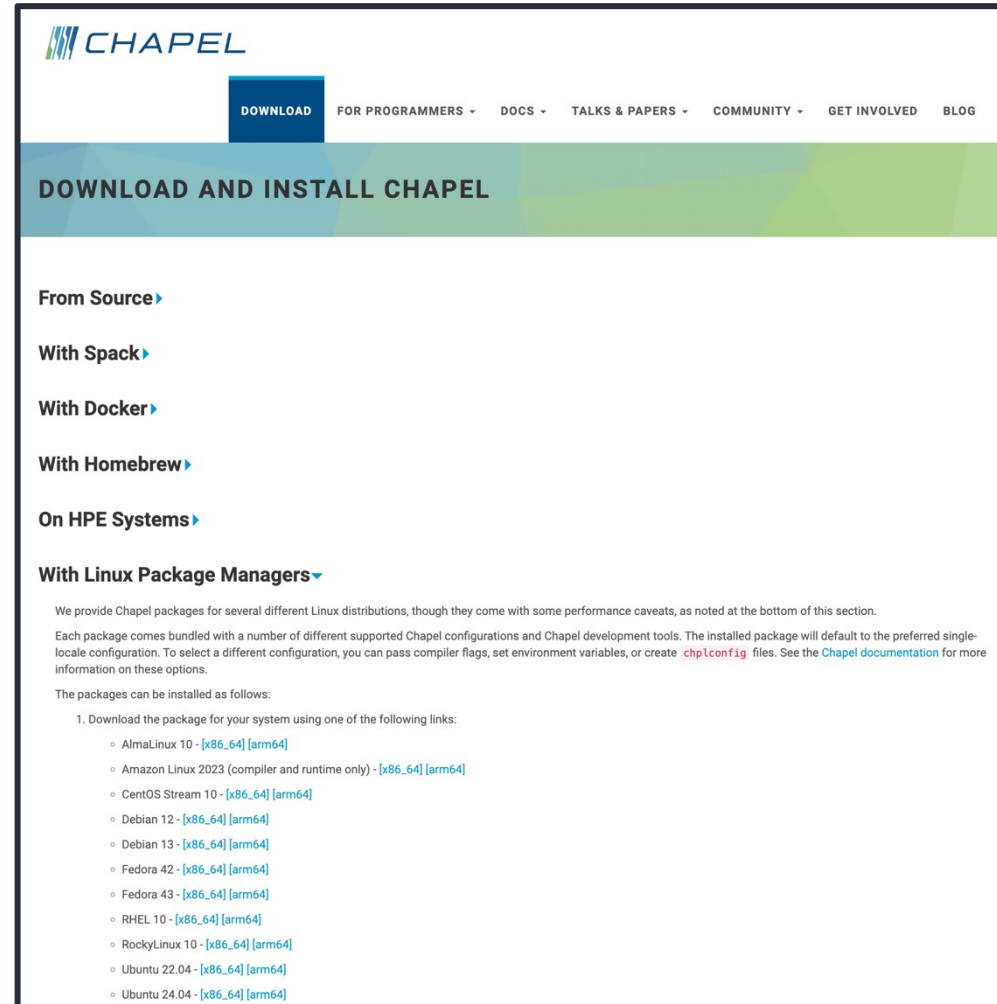


<https://chapel-lang.org/docs/usingchapel/portability.html>

Where can I get Chapel?

Release Formats:

- Source releases via GitHub
- Spack / E4S
- Docker
- Homebrew
- Modules on HPE Cray systems
- Linux packages via apt/rpm
- Attempt This Online / GitHub Codespaces



The screenshot shows the Chapel website's navigation and download page. At the top, the Chapel logo is on the left, and a navigation menu includes 'DOWNLOAD' (highlighted in a blue box), 'FOR PROGRAMMERS', 'DOCS', 'TALKS & PAPERS', 'COMMUNITY', 'GET INVOLVED', and 'BLOG'. Below the navigation is a green header with the text 'DOWNLOAD AND INSTALL CHAPEL'. The main content area lists several installation methods with right-pointing chevrons: 'From Source', 'With Spack', 'With Docker', 'With Homebrew', 'On HPE Systems', and 'With Linux Package Managers'. Under 'With Linux Package Managers', there is a paragraph of text explaining that packages are provided for various Linux distributions and that each package includes configuration options. Below this text, a list of links is provided for downloading packages for different Linux distributions and architectures, such as AlmaLinux 10, Amazon Linux 2023, CentOS Stream 10, Debian 12 and 13, Fedora 42 and 43, RHEL 10, RockyLinux 10, Ubuntu 22.04, and Ubuntu 24.04.

<https://chapel-lang.org/download/>



Ways to engage with the Chapel Community

Synchronous Community Events

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

Asynchronous Communications








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

Social Media








Discussion Forums

Ways to Use Chapel







FOLLOW US

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

GET IN TOUCH

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

GET STARTED

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

(from the footer of chapel-lang.org)



Recent News: Chapel 2.0 and HPSF

What is Chapel 2.0?

- **Released:** March 21, 2024
- Stabilizes core language and library features
 - In semantic versioning, essentially our “1.0” release
 - Non-core features marked as “unstable”
- Using an “editions” discipline to modify/add features now



The screenshot shows a blog post from the Chapel Language Blog. The title is "Chapel 2.0: Scalable and Productive Computing for All". It was posted on March 21, 2024, by Daniel Fedorin. The post is tagged with "Chapel 2.0" and "Release Announcements". A table of contents is visible on the left, listing various topics like "What Are People Doing With Chapel?", "Arkouda: Interactive Data Analysis at Scale", "CHAMPS: A Framework for Computational Fluid Dynamics", "Coral Biodiversity Computation", "A Language Built with Scalable Parallel Computing in Mind", "Rich Tooling Support", and "Conclusion and Looking Forward". The main text of the post discusses the release of Chapel 2.0, highlighting its stability and performance improvements.

<https://chapel-lang.org/blog/posts/announcing-chapel-2.0/>

What is HPSF?

- A foundation within the Linux Foundation focused on open-source HPC software
 - Sponsored by AWS, Microsoft, DOE Labs, HPE, AMD, Arm, NVIDIA, CEA, and many others
 - <https://hpsf.io/>
- Chapel proudly became a member project in November 2025
 - As a result, we’ve opened up things like weekly meetings, governance, blog and website repos, etc.



Closing Thoughts



Summary

Parallel programming traditionally requires a mix of notations to leverage HW

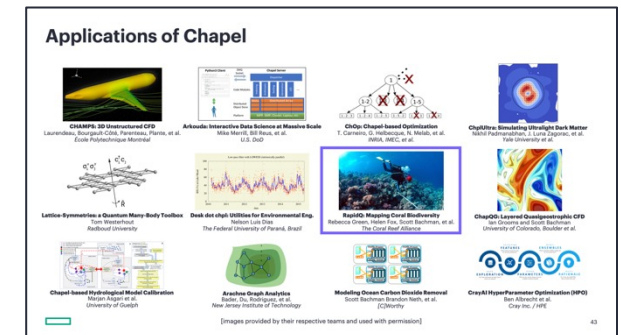
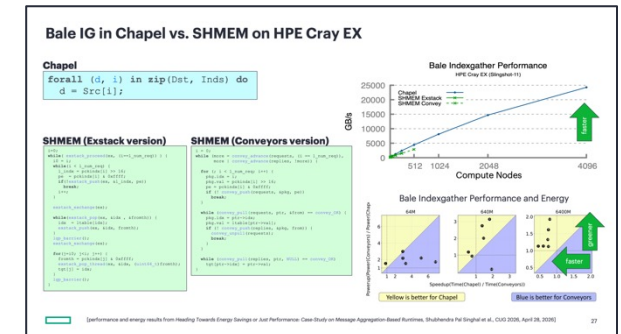
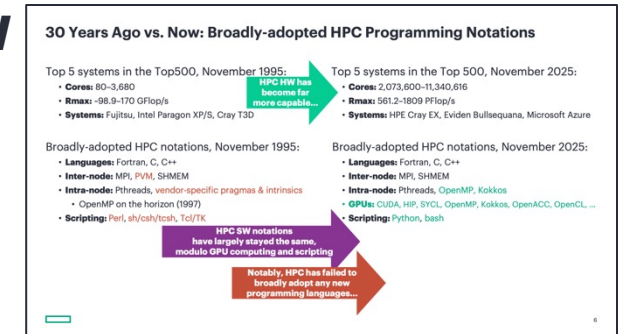
- e.g., C++ with MPI, OpenMP, and/or CUDA
- such programming models are sufficient, but leave much to be desired
- hardware has generally become more difficult to program over time
 - and at HPC scales, hardware speeds & feeds tend to dominate budgets and attention

Chapel is a unique parallel language

- high-level features for parallelism and locality
- portable across CPUs, GPUs, networks, vendors, scales, ...
- scalable, performant, energy-efficient (?)
- simplifies compiler optimizations on distributed systems (not discussed tonight)

Users are benefitting from Chapel, on laptops as well as supercomputers

- Coral Reef image processing
- Computational Fluid Dynamics
- Data Analysis
- ...



AI and Parallel Languages

Q: Now that AIs can program*, do languages like Chapel still have value?

A: I'd say "definitely", at least for the foreseeable future:

- An AI targeting a single, unified parallel language should be at least as successful as one targeting a mash-up of lower-level notations
- As long as humans need to validate and maintain AI-developed code, the clearer it is the better

Chapel

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

SHMEM (Exstack version)

```
i=0;  
while( exstack_proceed(ex, (i==l_num_req)) ) {  
  i0 = i;  
  while(i < l_num_req) {  
    l_indx = pckindx[i] >> 16;  
    pe = pckindx[i] & 0xffff;  
    if(!exstack_push(ex, &l_indx, pe))  
      break;  
    i++;  
  }  
  exstack_exchange(ex);  
  while(exstack_pop(ex, &idx, &fromth)) {  
    idx = ltable[idx];  
    exstack_push(ex, &idx, fromth);  
  }  
  lgp_barrier();  
  exstack_exchange(ex);  
  for(j=i0; j<i; j++) {  
    fromth = pckindx[j] & 0xffff;  
    exstack_pop_thread(ex, &idx, (uint64_t)fromth);  
    tgt[j] = idx;  
  }  
  lgp_barrier();  
}
```

SHMEM (Conveyors version)

```
i = 0;  
while (more = convey_advance(requests, (i == l_num_req)),  
       more | convey_advance(replies, !more)) {  
  for (; i < l_num_req; i++) {  
    pkg.idx = i;  
    pkg.val = pckindx[i] >> 16;  
    pe = pckindx[i] & 0xffff;  
    if (!convey_push(requests, &pkg, pe))  
      break;  
  }  
  while (convey_pull(requests, ptr, &from) == convey_OK) {  
    pkg.idx = ptr->idx;  
    pkg.val = ltable[ptr->val];  
    if (!convey_push(replies, &pkg, from)) {  
      convey_unpull(requests);  
      break;  
    }  
  }  
  while (convey_pull(replies, ptr, NULL) == convey_OK)  
    tgt[ptr->idx] = ptr->val;  
}
```

(* = your mileage may vary)

“This all sounds great... So, what’s the catch?”

(Or: “What keeps Brad up at night?”)

— **Adoption of Chapel:**

- a challenge for any new language
- most people aren’t actively shopping for new languages, particularly in an age with Python, Rust, Swift, Julia, ...
- and even fewer are actively grappling with scalable parallel computing
- **how I get back to sleep:** Users like Nelson for whom Chapel replaces conventional desktop programming languages

— **Technical debt / incomplete efforts**

- Chapel is now 20+ years old, and in many respects, the code reflects this
- Many language features and libraries feel rock-solid; others are squishier than we’d like
- Meanwhile, preserving portability and tracking third-party software is taking more and more time (see Jade’s [recent HPSFCon talk](#))
- **how I get back to sleep:** “This is the nature of SW and open-source projects”; seeing how quickly things can improve with effort

— **Funding**

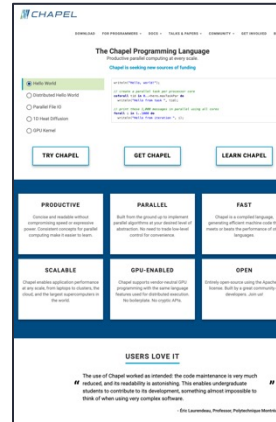
- Chapel has enjoyed long-term stability and growth in funding, but that has recently stalled and reversed
- Without new sources of funding or staffing/collaboration, we expect to move to a maintenance-only mode this fall
- See our recent [Chapel Project Seeks New Funding](#) blog post for more information
- **how I get back to sleep:** Currently, I often don’t



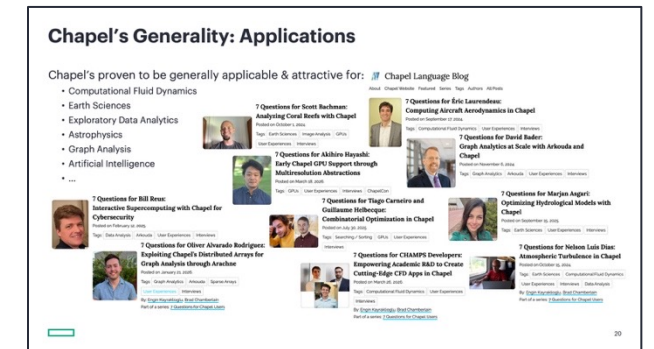
For more about...

...Chapel:

- Browse the [Chapel website](#) and [blog](#):
- Reach out to [our team and community](#)



...applications of Chapel: Browse the [7 Questions for Chapel Users](#) interview series:



...me grappling with socio-technical aspects of parallel language adoption:

- [Reflections on 30 Years of HPC Programming: So many hardware advances, so little adoption of new languages](#)
- [10 Myths About Scalable Parallel Programming Languages \(Redux\)](#)
 - Consider starting with [its summary](#)



Closing Statements

I consider current and aspiring [scalable] parallel programmers to be at least as worthy of modern languages as the Python, Rust, Swift, and Julia communities are

Over the next 30 years, I hope we see the number of broadly adopted languages for scalable parallelism grow to be ≥ 1 , rather than the current 0



Thank You

@ChapelLanguage

