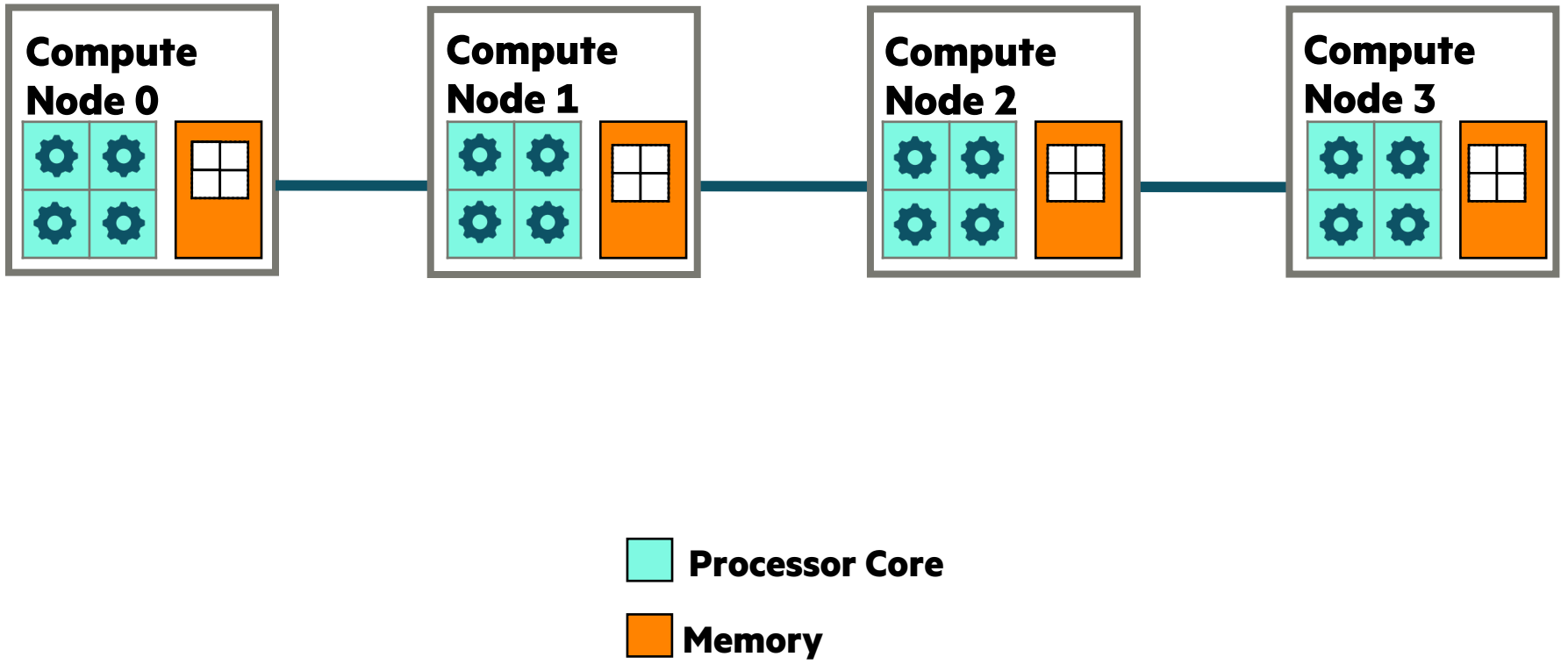
Chapel: Parallelism + Locality = Future-Proof Language Design?

Brad Chamberlain, HPE

NVIDIA
September 16, 2026

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?



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

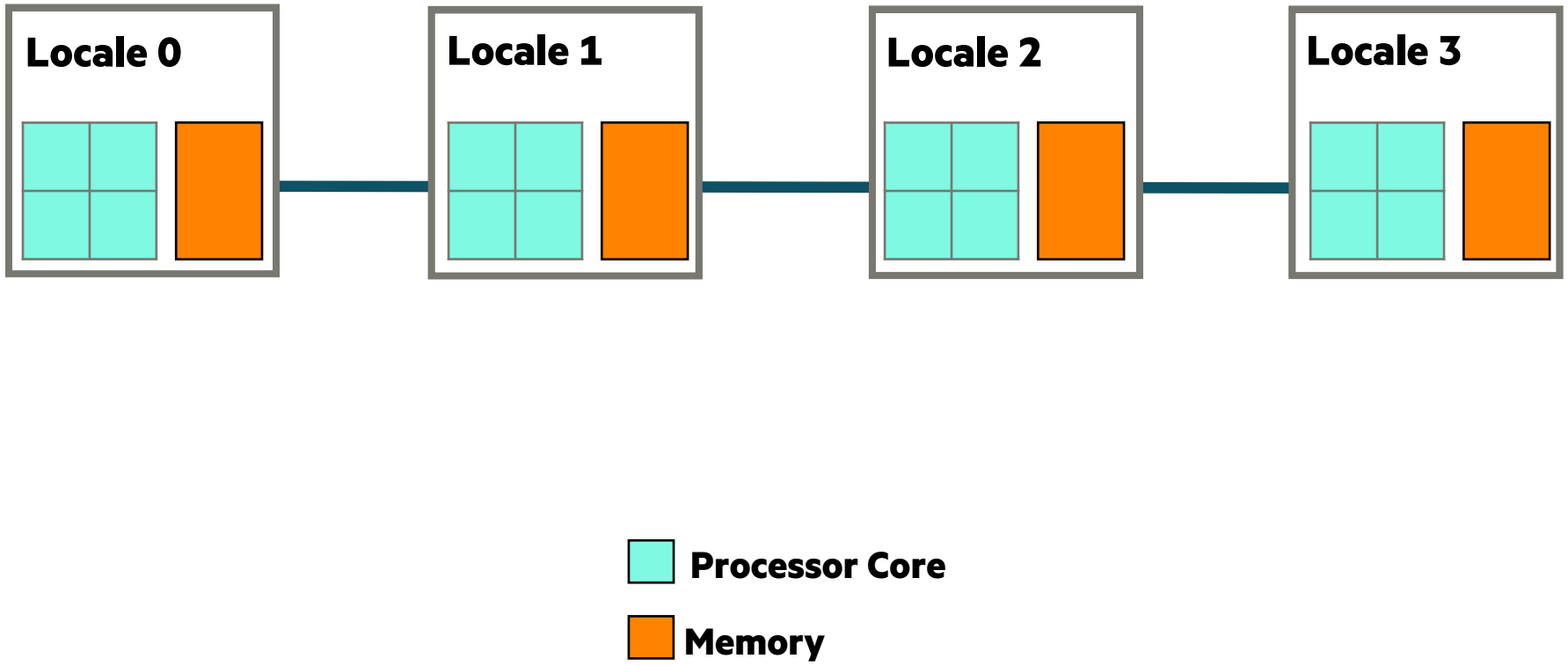
Approach: Express parallelism and locality using hardware-neutral, compiler optimizable abstractions

Locales in Chapel

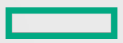
In Chapel, a *locale* refers to a compute resource with...

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

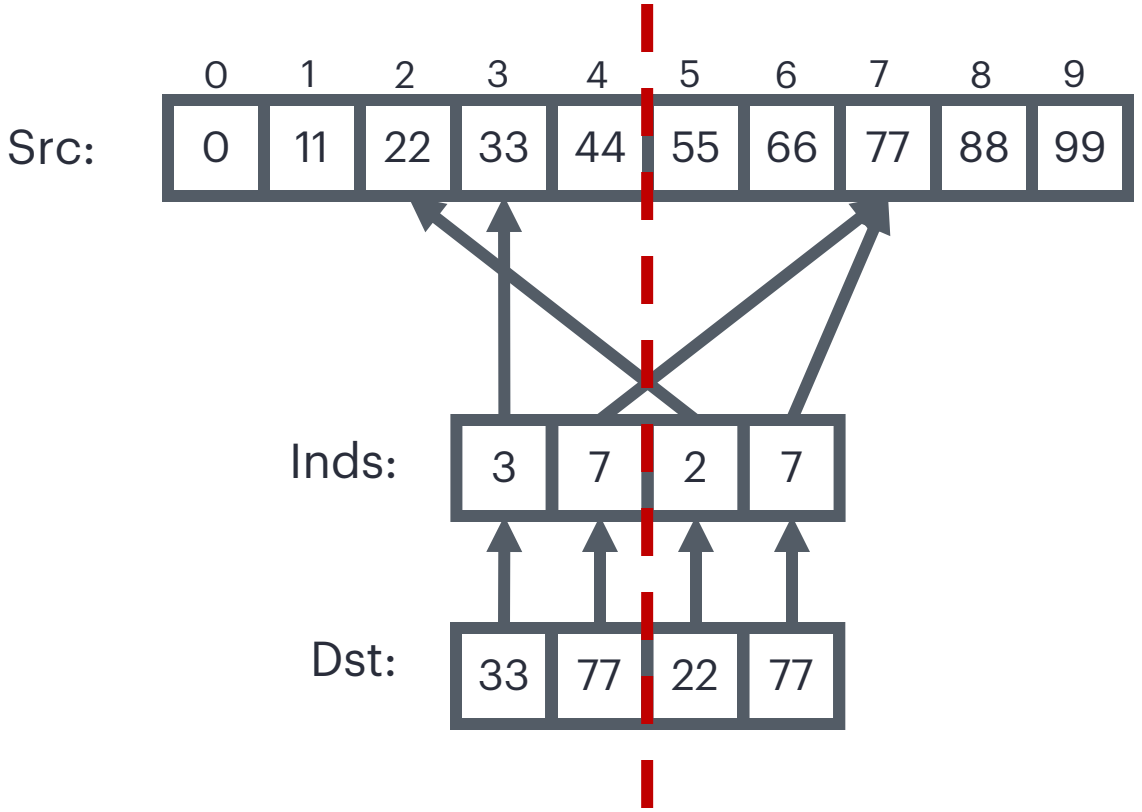
For now, think of each compute node as being a locale



High-Level Chapel By Example: Bale Index Gather



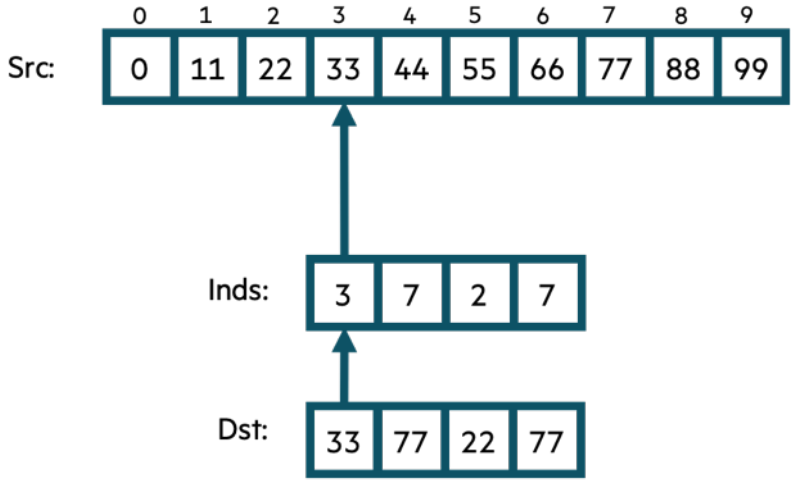
Bale Index Gather (IG): In Pictures



Bale IG in Chapel: Declarations and Main Loop (Sequential Version)

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



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

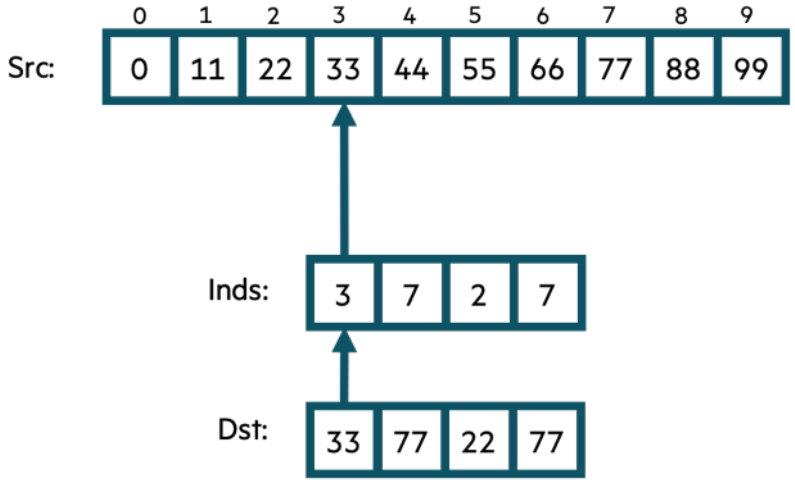
```
$
```



Bale IG in Chapel: Compiling

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



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

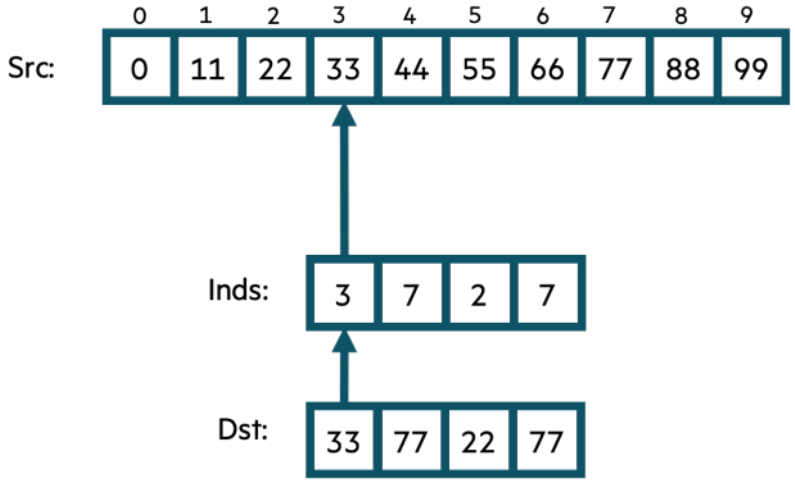
```
$ chpl bale-ig.chpl
$
```

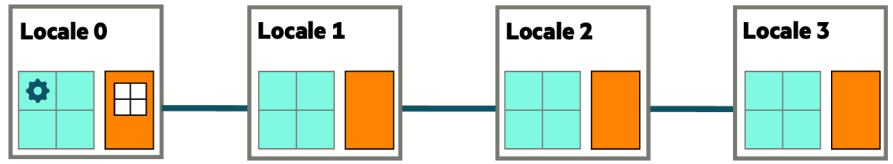


Bale IG in Chapel: Executing

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



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

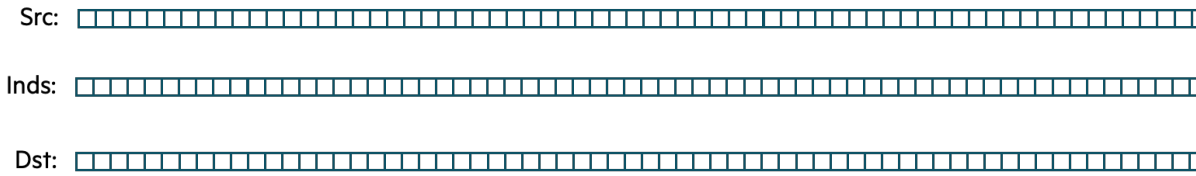
```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4
$
```



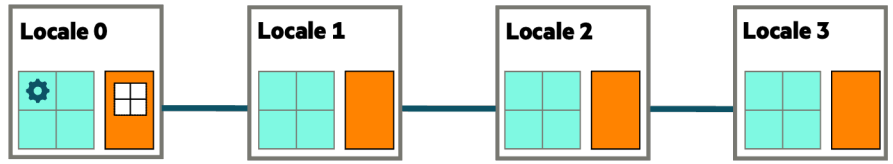


Bale IG in Chapel: Executing, Overriding Configs

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



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

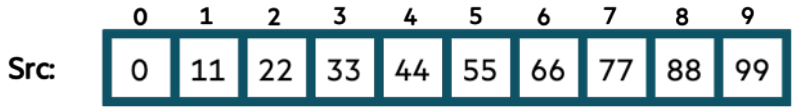



```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4 --n=1_000_000 --m=1_000_000
$
```

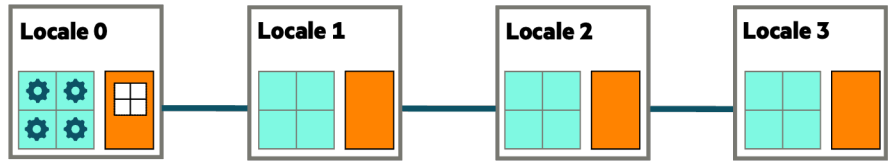


Bale IG in Chapel: Array Initialization

```
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);
```



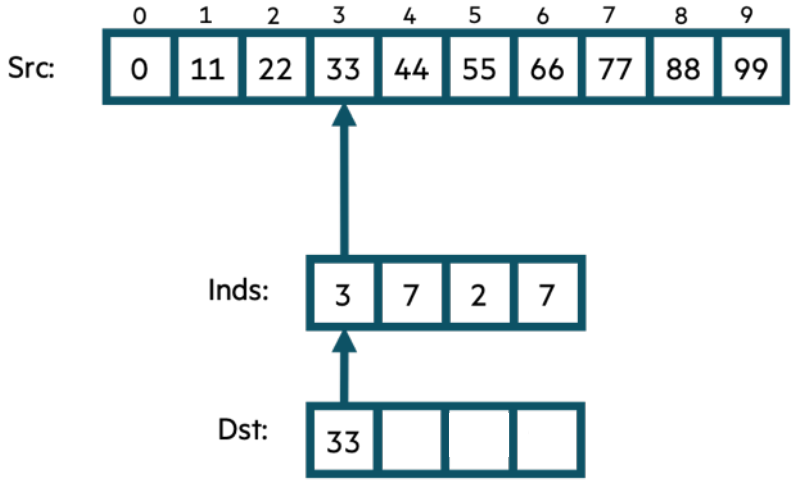

```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4
$
```

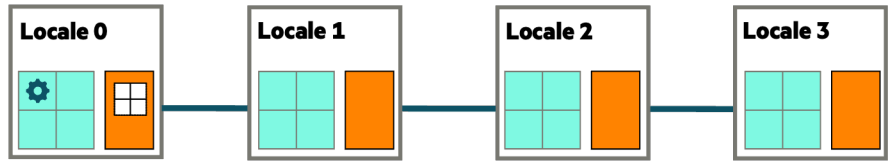


Bale IG in Chapel: Sequential Version

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




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

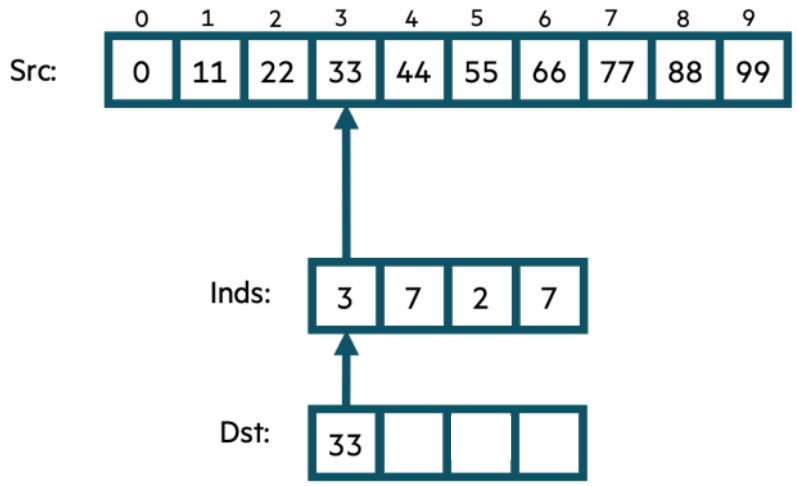
```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4
$
```



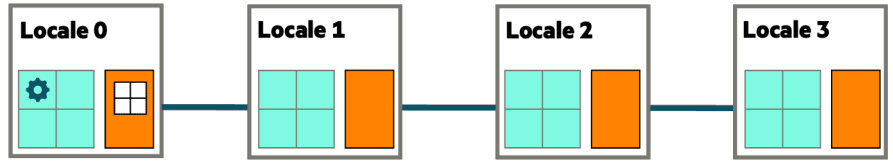


Bale IG in Chapel: Sequential, Zippered Version

```
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];
```

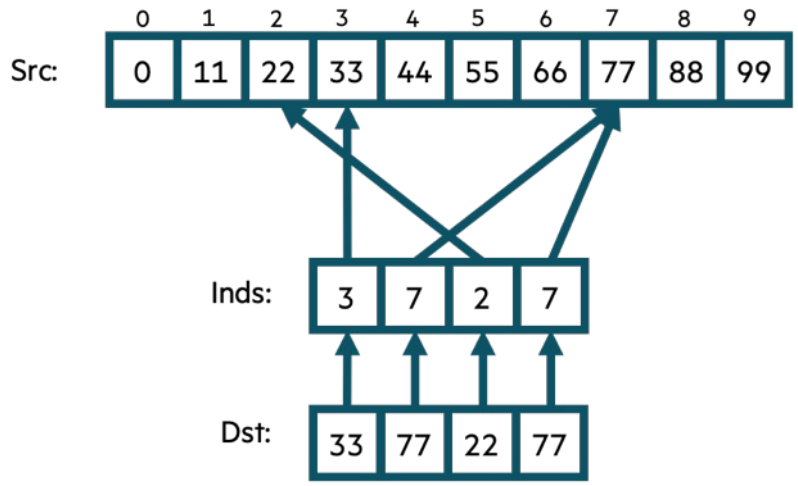



```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4
$
```

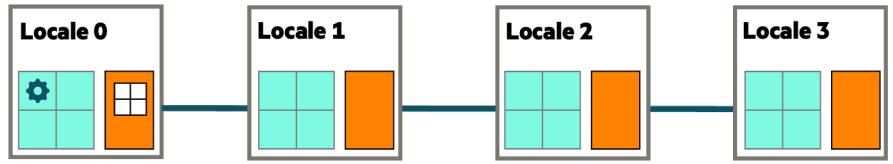


Bale IG in Chapel: Parallel, Zippered Version (Vectorized)

```
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];
```

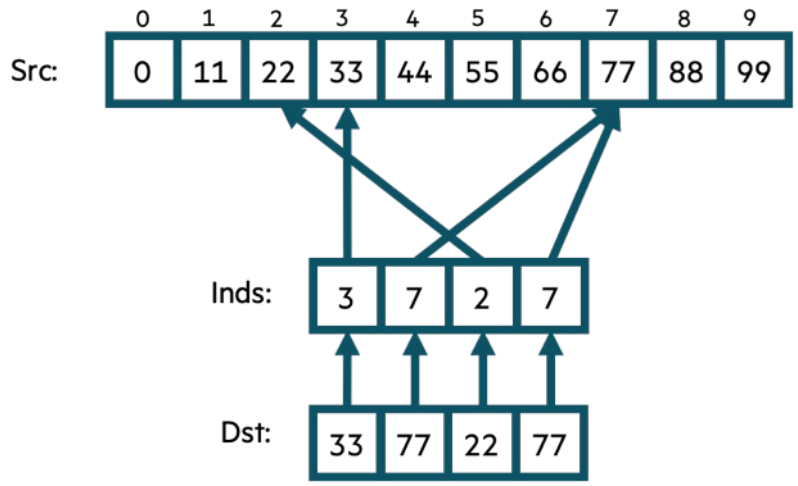



```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4
$
```

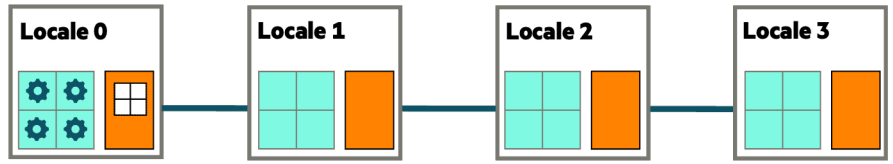


Bale IG in Chapel: Parallel, Zippered Version (Multicore)

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




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

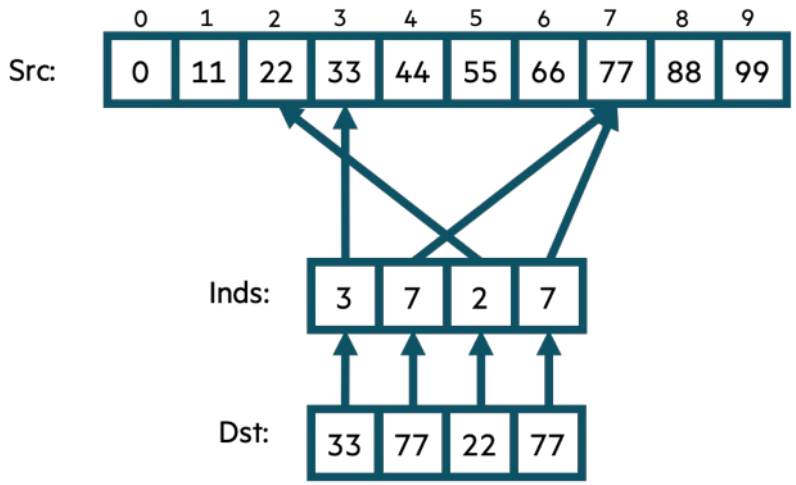



```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4
$
```

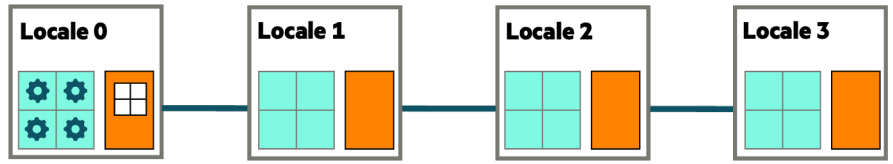


Bale IG in Chapel: Parallel Promoted Version (equivalent to previous version)

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



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

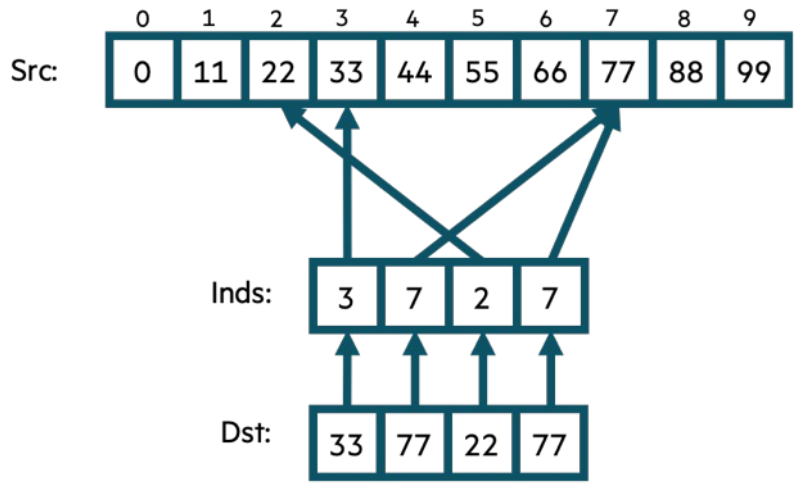



```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4
$
```

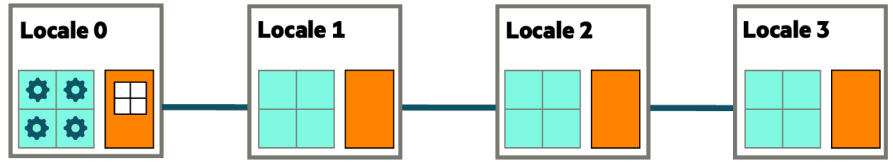


Bale IG in Chapel: Parallel, Zippered Version (Multicore, again)

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




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

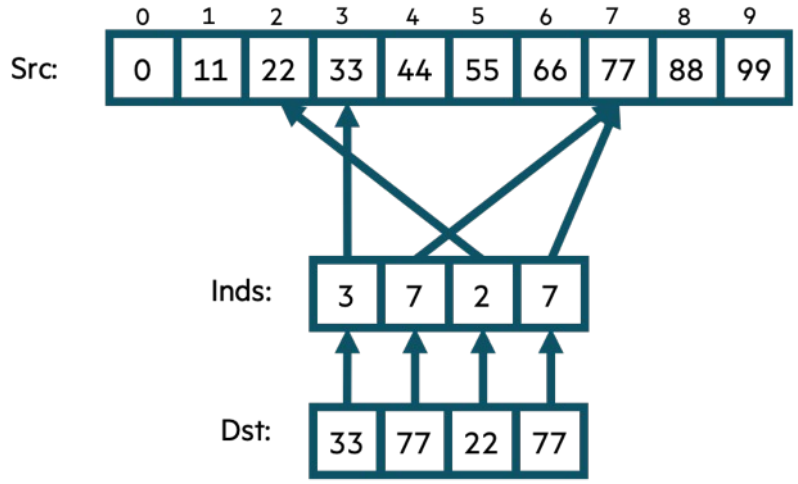



```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4
$
```

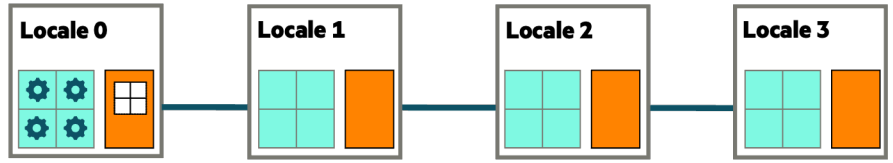


Bale IG in Chapel: Parallel, Zippered Version with Named Domains (Multicore)

```
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];
```

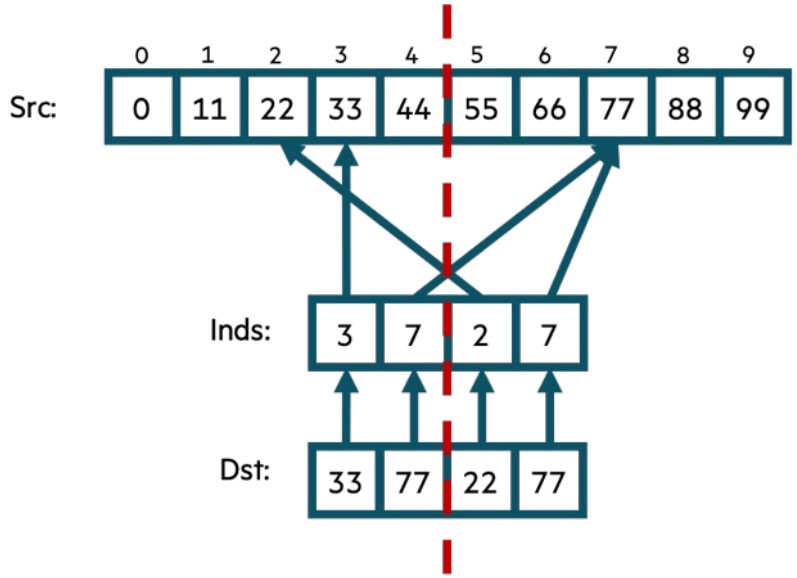



```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4
$
```

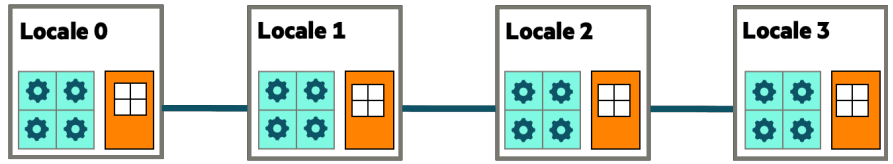


Bale IG in Chapel: Distributed Parallel Version

```
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];
```

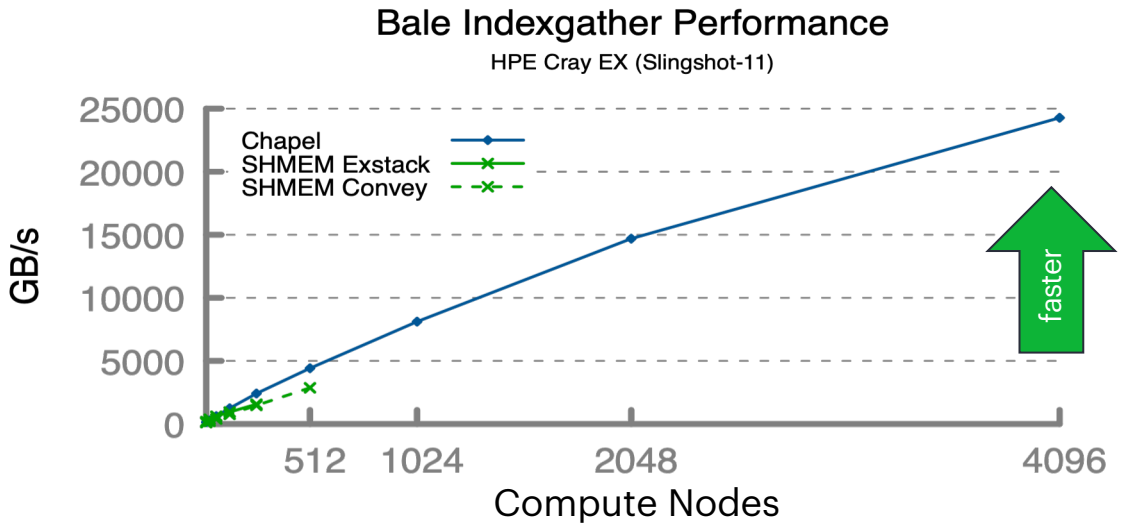



```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4
$
```



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 IG in Chapel vs. SHMEM on HPE Cray EX

Chapel

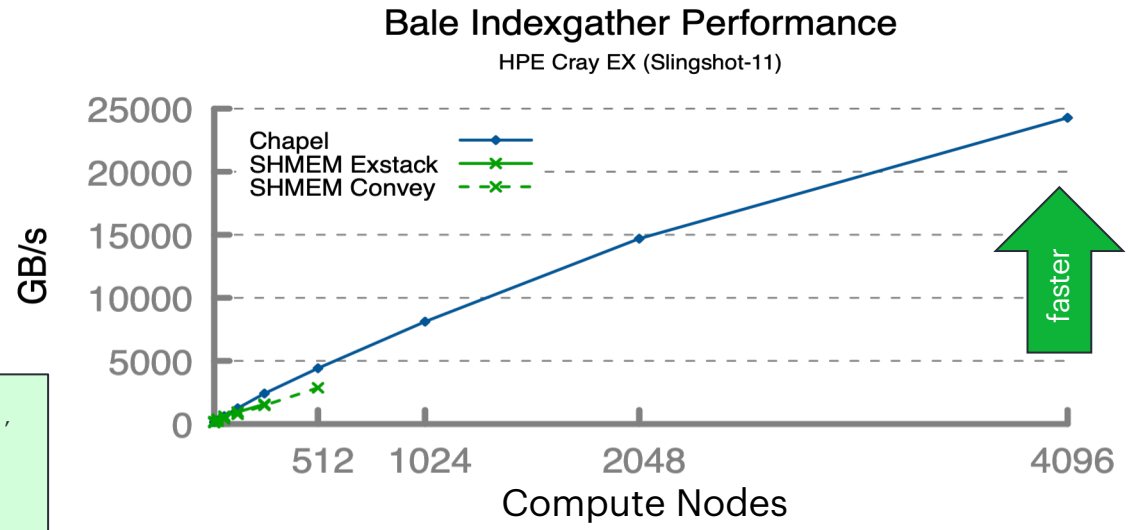
```
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

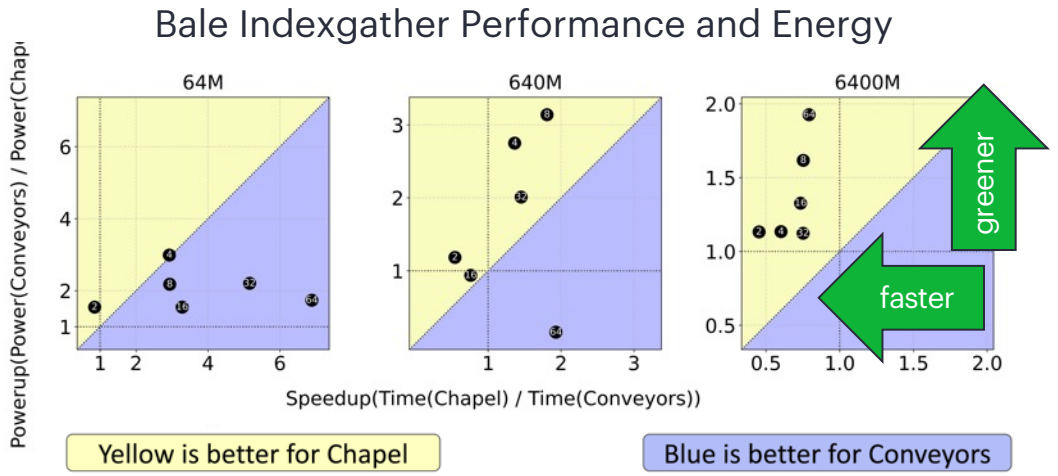
SHMEM (Exstack version)

```
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)

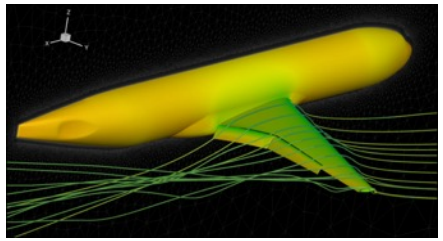
```
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
       more | convey_advance(replies, !more)) {

  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;
}
```



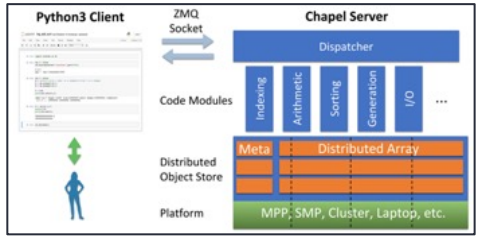


[performance and energy results from *Heading Towards Energy Savings or Just Performance: Case-Study on Message Aggregation-Based Runtimes,* Shubhendra Pal Singhal et al., CUG 2026, April 28, 2026]

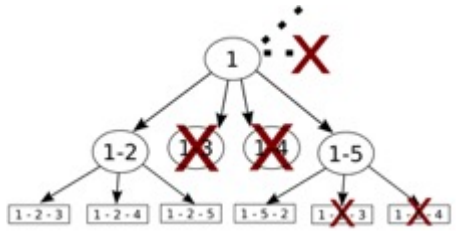
Applications of Chapel



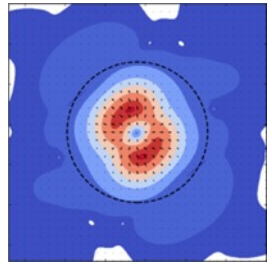
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



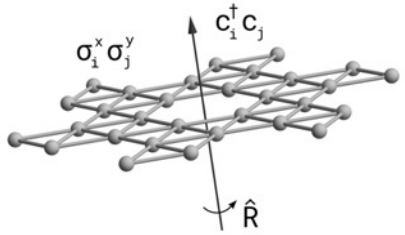
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



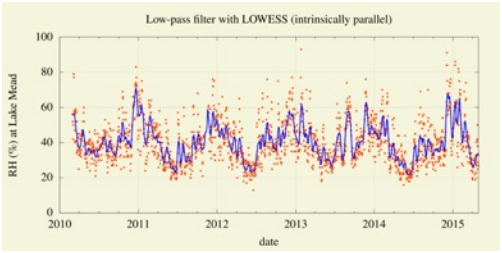
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

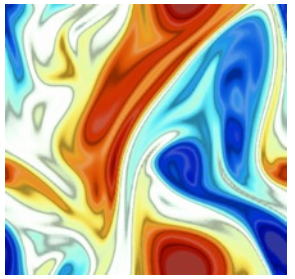


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

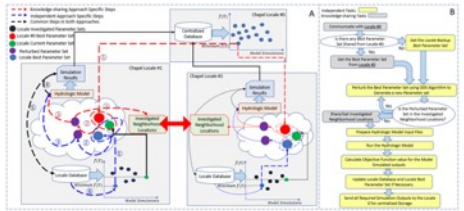


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

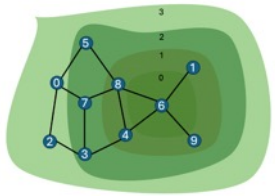
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



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

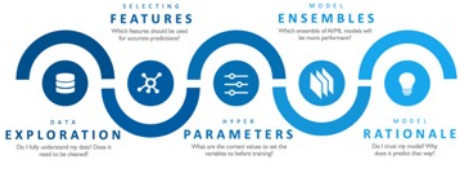


Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



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

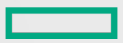
Modeling Ocean Carbon Dioxide Removal
Scott Bachman Brandon Neth, et al.
[C]Worthy



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

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

Future-proof languages?
A look back at 25 years of HPC



25 Years Ago vs. Today: Top HPC Systems in the Top500

Top 5 systems in the Top500, June 2001:

- **Cores:** 1,152–9,632
- **Rmax:** ~1,709–7,226 GFlop/s

Rank	System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)	Power (kW)
1	ASCI White, SP Power3 375 MHz, IBM Lawrence Livermore National Laboratory United States	8,192	7,226.00	12,288.00	
2	SP Power3 375 MHz 16 way, IBM DOE/SC/LBNL/NERSC United States	2,528	2,526.00	3,792.00	
3	ASCI Red, Intel Sandia National Laboratories United States	9,632	2,379.00	3,207.00	
4	ASCI Blue-Pacific SST, IBM SP 604e, IBM Lawrence Livermore National Laboratory United States	5,808	2,144.00	3,856.50	
5	SR8000/MPP, Hitachi University of Tokyo Japan	1,152	1,709.10	2,074.00	

Top 5 systems in the Top 500, June 2026:

- **Cores:** 4,801,344–13,789,440
- **Rmax:** ~1,000–2,198 PFlop/s

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**LineShine** - LingKun, LX2 304C 1.55GHz, LingQi, Kylin OS, Shenzhen Cloud Computing Center Co., Ltd. National Supercomputing Centre in Shenzhen (NSCS) China	13,789,440	2,198.40	2,735.82	42,220
2	**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
3	**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
4	**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
5	**JUPITER Booster** - BullSequana XH3000, GH Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Enterprise Linux, Bull EuroHPC/FZJ Germany	4,801,344	1,000.00	1,226.28	15,794

25 Years Ago vs. Today: Top HPC Systems in the Top500

Top 5 systems in the Top500, June 2001:

- **Cores:** 1,152–9,632
- **Rmax:** ~1,709–7,226 GFlop/s

Cores: ~500x–12,000x
Rmax: ~138,400x–1,286,000x

Top 5 systems in the Top 500, June 2026:

- **Cores:** 4,801,344–13,789,440
- **Rmax:** ~1,000–2,198 PFlop/s

Rank	System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)
1	ASCI White, SP Power3 375 MHz, IBM Lawrence Livermore National Laboratory United States	8,192	7,226.00	12,288.00
2	SP Power3 375 MHz 16 way, IBM DOE/SC/LBNL/NERSC United States			
3	ASCI Red, Intel Sandia National Laboratories United States			
4	ASCI Blue-Pacific SST, IBM SP 604e, IBM Lawrence Livermore National Laboratory United States			
5	SR8000/MPP, Hitachi University of Tokyo Japan			

HPC HW has become far more capable…

And complex!

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

(Often in ways that hurt programmability)

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	Lin Shine - LingKun, LX2 304C 1.55GHz, LingQi, Kylin OS …nzhen Cloud Computing Center Co., Ltd. Na…upercomputing Centre in Shenzhen (NSCS) Ch…	13,789,440	2,198.40	2,735.82	42,220
2	El… …y EX255a, AMD 4th Gen EPYC 24C …I300A, Slingshot-11, TOSS, HPE	11,340,000	1,809.00	2,821.10	29,685
	…timized 3rd …ct MI250X,	9,066,176	1,353.00	2,055.72	24,607
	…pute Blade, …ata Center GPU …tory	9,264,128	1,012.00	1,980.01	38,698
	…Sequana XH3000, GH Superchip …200 Superchip, Quad-Rail NVIDIA …, RedHat Enterprise Linux, Bull Inf… Eu… Ge…	4,801,344	1,000.00	1,226.28	15,794

25 Years Ago vs. Today: Well-established HPC Programming Notations

Top 5 systems in the Top500, June 2001:

- **Cores:** 1,152–9,632
- **Rmax:** ~1,709–7,226 GFlop/s

HPC HW has become far more capable…

Top 5 systems in the Top 500, June 2026:

- **Cores:** 4,801,344–13,789,440
- **Rmax:** ~1,000–2,198 PFlop/s

Well-established HPC notations, June 2001:

- **Languages:** Fortran, C, C++, UPC
- **Inter-node:** MPI, PVM, SHMEM
- **Intra-node:** Pthreads, OpenMP
- **Scripting:** Perl, sh/csh/tcsh, Tcl/TK

Well-established HPC notations, June 2026:

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

25 Years Ago vs. Today: Well-established HPC Programming Notations

Top 5 systems in the Top500, June 2001:

- **Cores:** 1,152–9,632
- **Rmax:** ~1,709–7,226 GFlop/s

HPC HW has become far more capable…

Top 5 systems in the Top 500, June 2026:

- **Cores:** 4,801,344–13,789,440
- **Rmax:** ~1,000–2,198 PFlop/s

Well-established HPC notations, June 2001:

- **Languages:** Fortran, C, C++, UPC
- **Inter-node:** MPI, PVM, SHMEM
- **Intra-node:** Pthreads, OpenMP
- **Scripting:** Perl, sh/csh/tcsh, Tcl/TK

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

Well-established HPC notations, June 2026:

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

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

seemingly, each new flavor of hardware parallelism brings a brand new software notation

25 Years Ago vs. Today: The Ubiquity of Parallelism

Top 5 systems in the Top500, June 2001:

- **Cores:** 1,152–9,632
- **Rmax:** ~1,709–7,226 GFlop/s

HPC HW has become far more capable…

Top 5 systems in the Top 500, June 2026:

- **Cores:** 4,801,344–13,789,440
- **Rmax:** ~1,000–2,198 PFlop/s

Well-established HPC notations, June 2001:

- **Languages:** Fortran, C, C++, UPC
- **Inter-node:** MPI, PVM, SHMEM
- **Intra-node:** Pthreads, OpenMP
- **Scripting:** Perl, sh/csh/tcsh, Tcl/TK

HPC SW notations have largely stayed the same

Well-established HPC notations, June 2026:

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

Availability of parallelism, June 2001

- HPC systems at national labs, universities, industry

Parallelism has expanded beyond HPC, dramatically!

Availability of parallelism, June 2026:

- HPC systems at national labs, universities, industry
- laptops / desktops with multicore CPUs and GPUS
- single-board computers (SBCs)
- cloud computing, AI data centers, …

25 Years Ago vs. Today: How does Chapel fit in?

Top 5 systems in the Top500, June 2001:

- **Cores:** 1,152–9,632

Chapel pre-dates all of the HW technology changes mentioned earlier.

Rank	System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)	Power (kW)
1	ASCI White, SP Power3 375 MHz, IBM Lawrence Livermore National Laboratory United States	8,192	7,226.00	12,288.00	
2	SP Power3 375 MHz 16 way, IBM DOE/SC/LBNL/NERSC United States				
3	ASCI Red, Intel Sandia National Laboratories United States				
4	ASCI Blue-Pacific SST, IBM SP 604e, IBM Lawrence Livermore National Laboratory United States				
5	SR8000/MPP, Hitachi University of Tokyo Japan				

Top 5 systems in the Top 500, June 2026:

- **Cores:** 4,801,344–13,789,440
- **Rmax:** ~1,000–2,198 PFlop/s

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	Li Shine - LingKun, LX2 304C 1.55GHz, LingQi, Kylin OS nzhen Cloud Computing Center Co., Ltd. Na percomputing Centre in Shenzhen (NSCS) Ch	13,789,440	2,198.40	2,735.82	42,220
2	El y EX255a, AMD 4th Gen EPYC 24C I300A, Slingshot-11, TOSS, HPE	11,340,000	1,809.00	2,821.10	29,685
	timized 3rd ct MI250X,	9,066,176	1,353.00	2,055.72	24,607
	pute Blade	9,264,128	1,012.00	1,980.01	38,698
	Inf Eu Ge				

And complex!

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

(Often in ways that hurt programmability)

Yet it supports them using the same features for parallelism and locality that we designed 20+ years ago

25 Years Ago vs. Today: How does Chapel fit in?

Top 5 systems in the Top500, June 2001:

- **Cores:** 1,152–9,632
- **Rmax:** ~1,709–7,226 GFlop/s

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

Availability of parallelism, June 2001

- HPC systems at national labs, universities, industry

Top 5 systems in the Top 500, June 2026:

- **Cores:** 4,801,344–13,789,440
- **Rmax:** ~1,000–2,198 PFlop/s

Unified

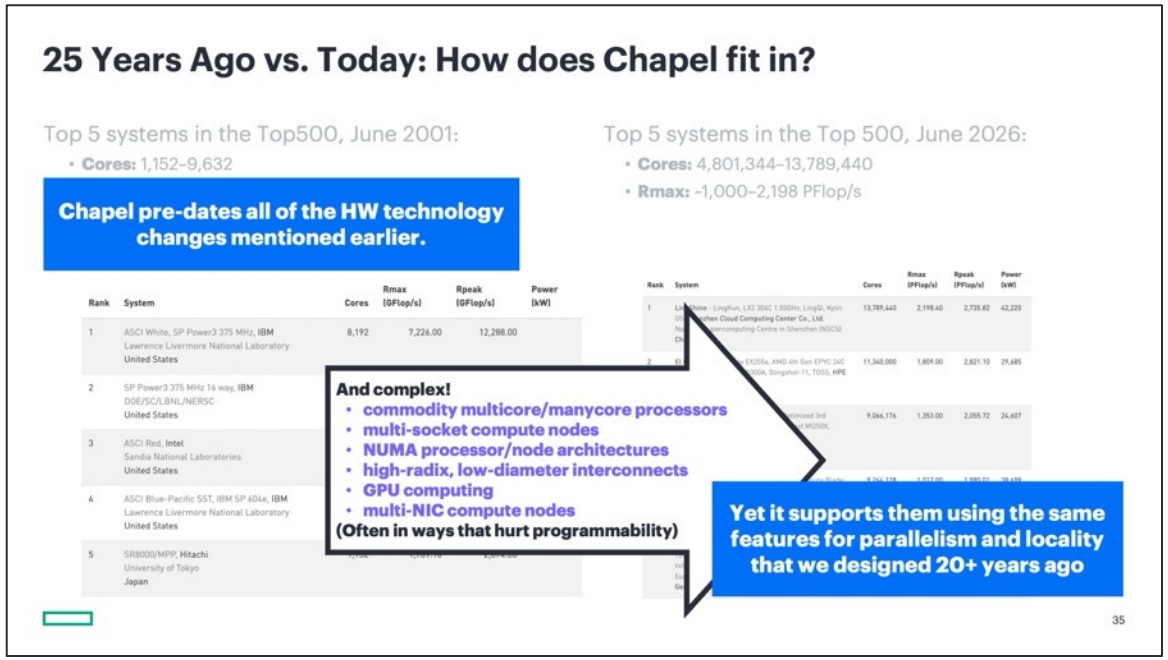
~~Well-established~~ HPC notation~~s~~, June 2026:

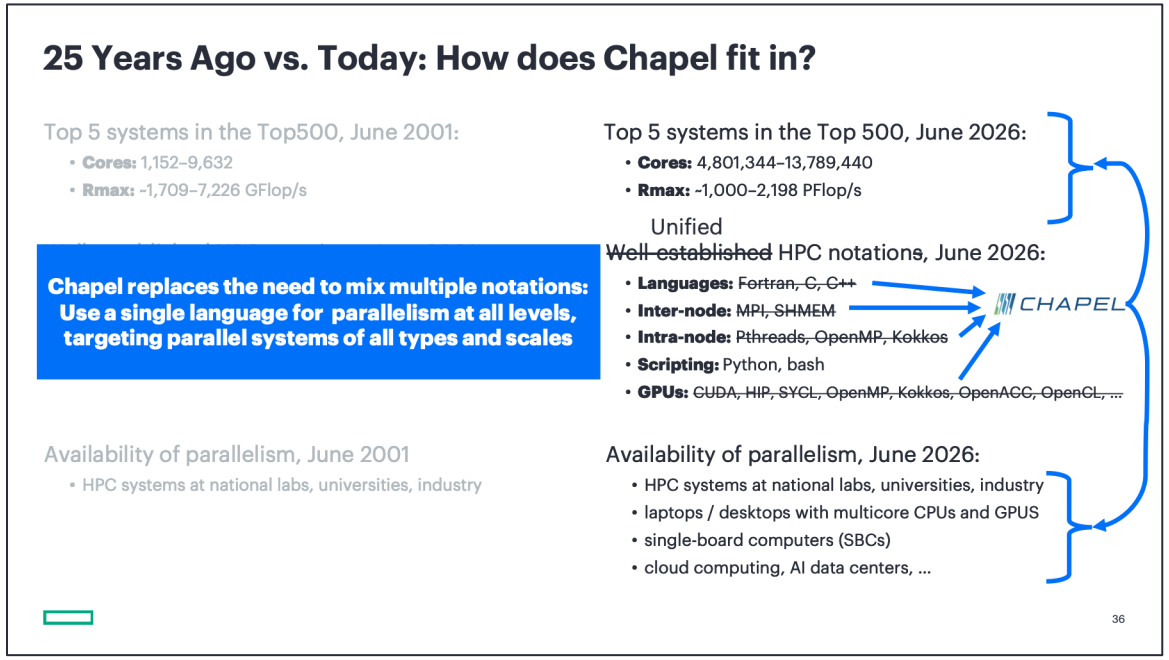
- **Languages:** ~~Fortran, C, C++~~
- **Inter-node:** ~~MPI, SHMEM~~
- **Intra-node:** ~~Pthreads, OpenMP, Kokkos~~
- **Scripting:** Python, bash
- **GPUs:** ~~CUDA, HIP, SYCL, OpenMP, Kokkos, OpenACC, OpenCL, …~~

Availability of parallelism, June 2026:

- HPC systems at national labs, universities, industry
- laptops / desktops with multicore CPUs and GPUS
- single-board computers (SBCs)
- cloud computing, AI data centers, …

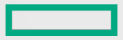
25 Years Ago vs. Today: How does Chapel fit in?





key to Chapel's longevity and flexibility: express parallelism and locality independently of hardware mechanisms

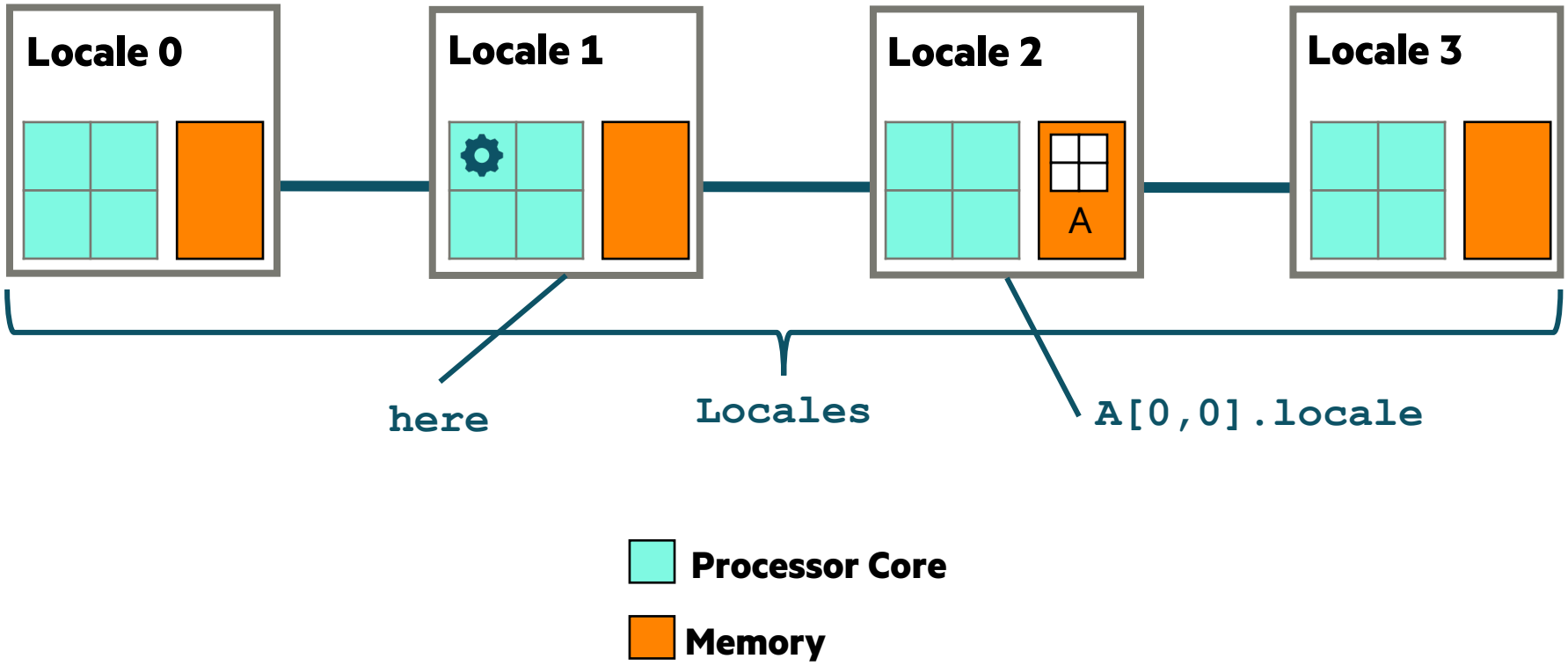
“Low-level” Chapel Features and GPU support



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

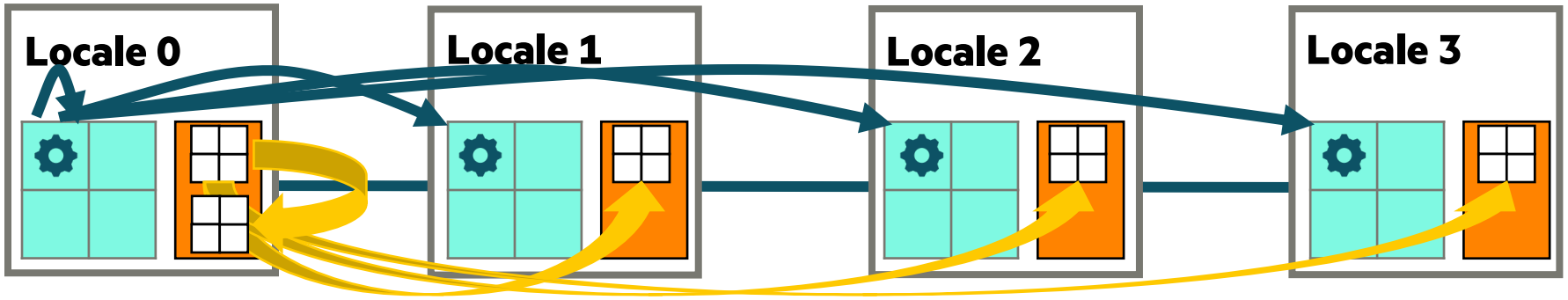
Variables are stored using the memory local to the current task

This loop will iterate sequentially over the program's locales

on-clauses move tasks to target locales

remote variables can be accessed directly

This is a distributed, yet sequential, computation

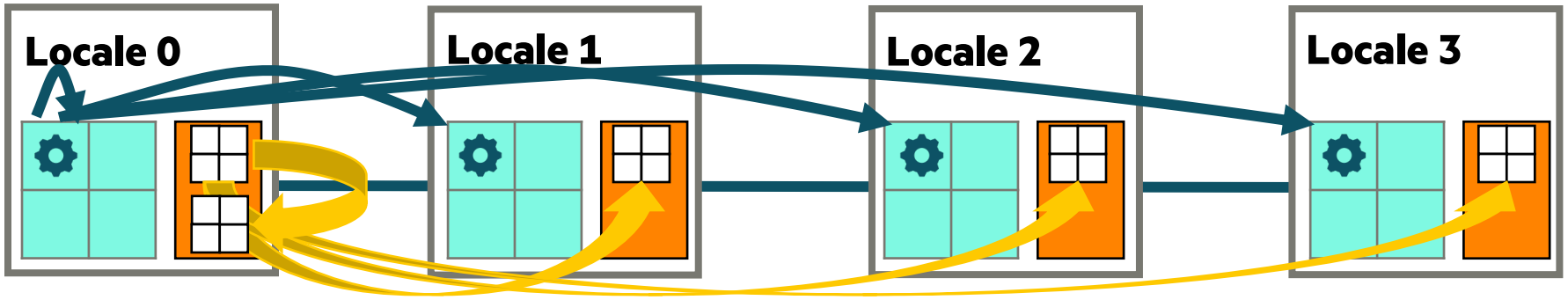


Mixing Locality with Task Parallelism

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

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

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

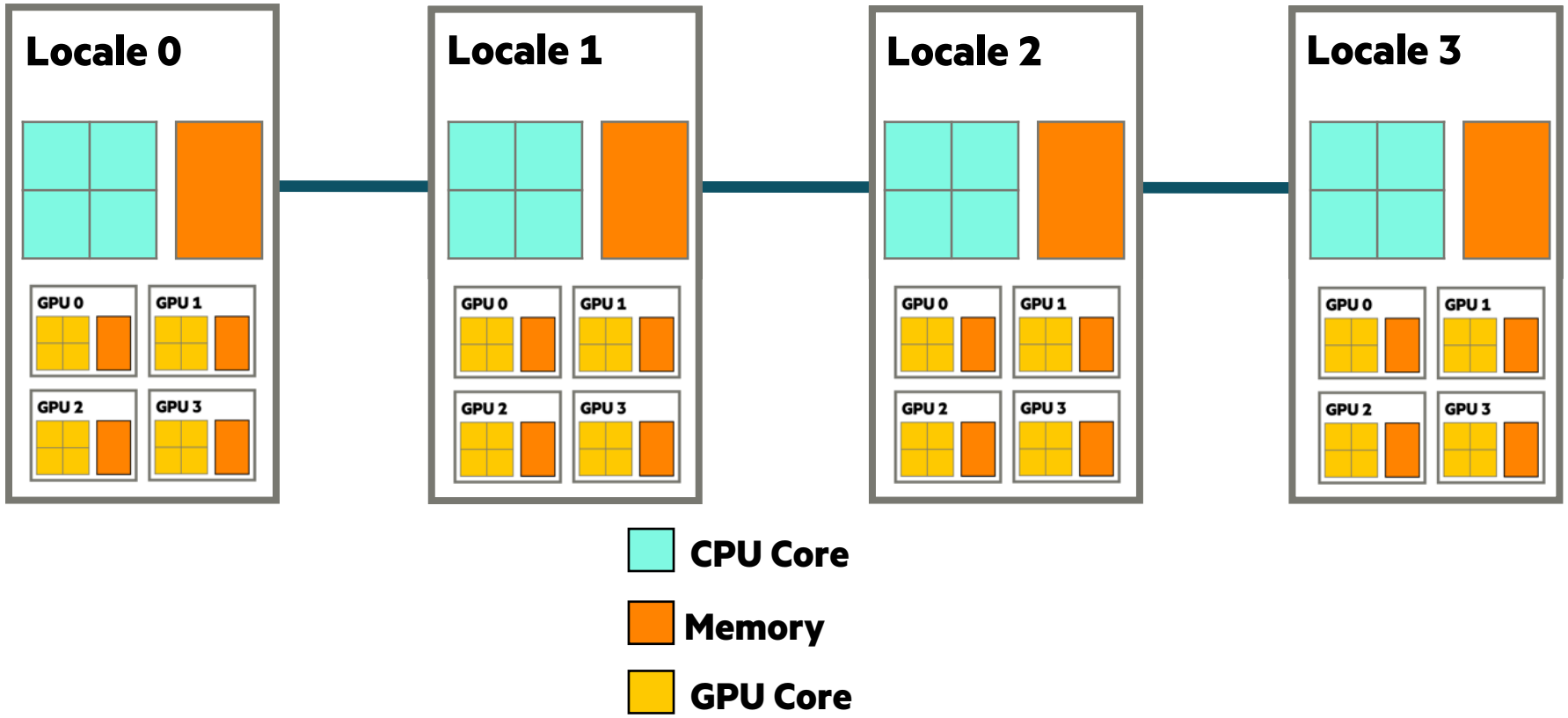
The coforall loop creates
a parallel task per iteration
(in this case, a task per locale)

This is a distributed parallel computation



GPUs add complexity

Of course, modern compute nodes also have GPUs, with their own processor and memory types



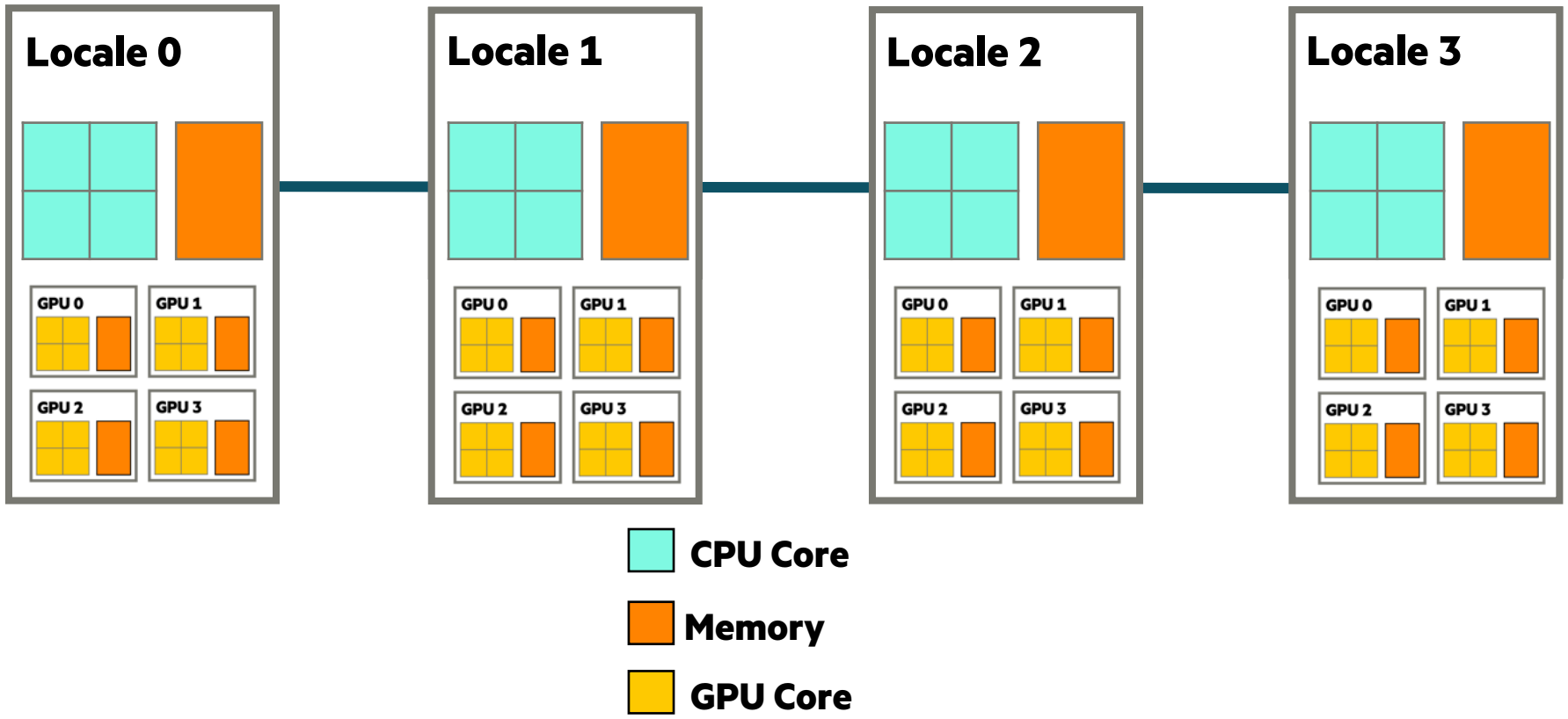
Adapting Chapel for GPUs

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

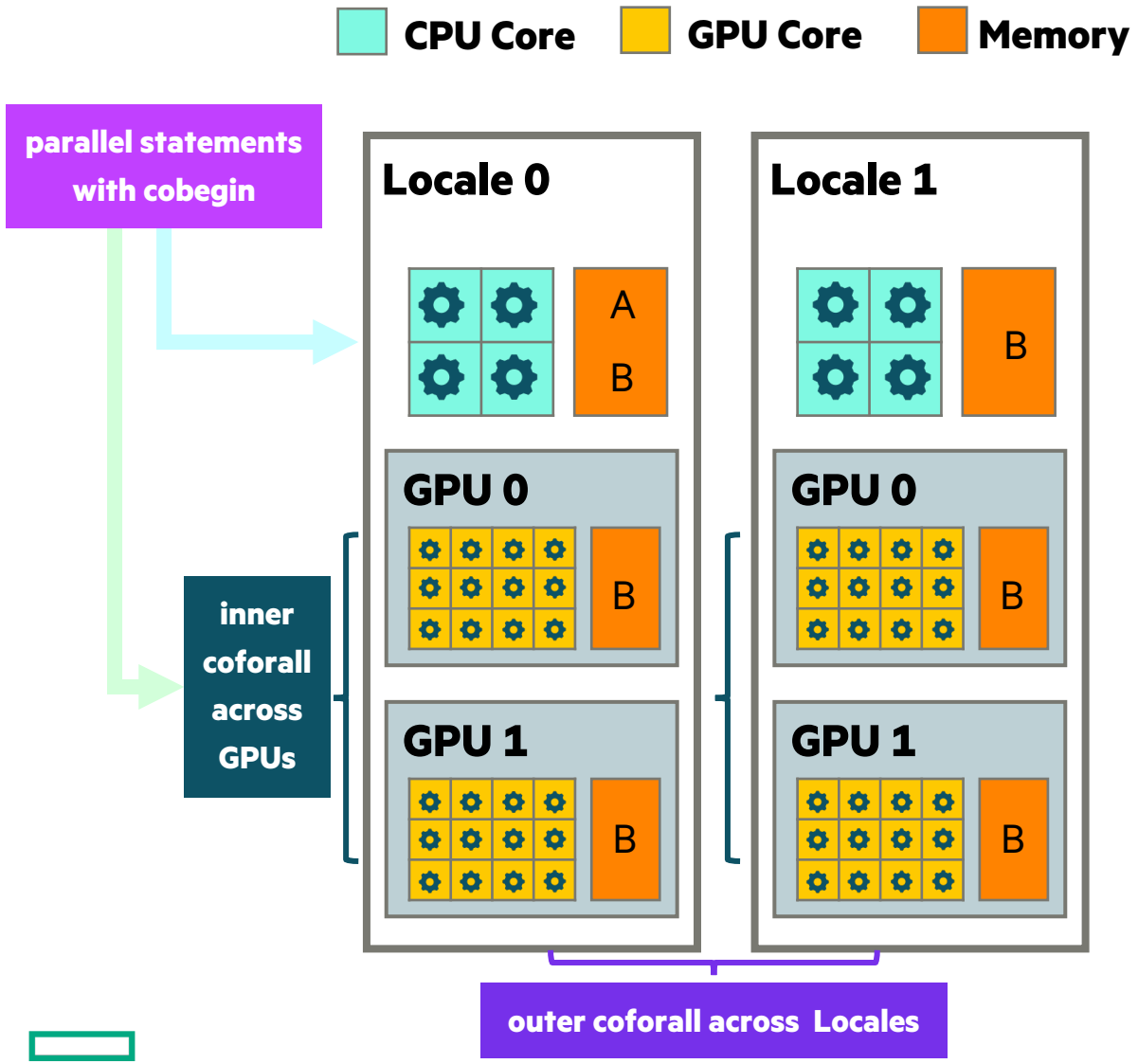
- Each top-level locale may have an array of locales called ‘gpus’
- We can then target them using Chapel’s traditional features for parallelism + locality

```
on here.gpus[0] { … }
```

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



Targeting CPUs and GPUs using Parallelism and Locality




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

For More on Chapel and GPUs

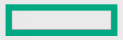
Selected Publications:

- **Productive, Vendor-Neutral GPU Programming Using Chapel**
  Engin Kayraklioglu, Andy Stone, *WACCPD at SC24*, November 18, 2024
- **Performance Portability of the Chapel Language on Heterogeneous Architectures**
  Josh Milthorpe, Xianghao Wang, Ahmad Azizi, *Heterogeneity in Computing Workshop (HCW 2024) at IPDPS 2024*, May 27, 2024
- **Investigating Portability in Chapel for Tree-based Optimization on GPU-powered Clusters**
  Tiago Carneiro, Engin Kayraklioglu, Guillaume Helbecque, Nouredine Melab, *Euro-PAR 2024*, August 28, 2024

Selected Presentations / Demos / Posters:

- **Vendor-Neutral GPU Programming in Chapel**
  Jade Abraham and Engin Kayraklioglu, *HPE Developer Meetup, online*, July 31, 2024
- **The Secret Sauce of Vendor-Neutral GPU Programming (in Chapel)**
  Jade Abraham, *PNW PLSE 2025*, May 7, 2025
- **GPU-Based Monte Carlo Simulation of Light Transport in Tissue: A Chapel Implementation**
  Engin Kayraklioglu, Hui Wang, *NVIDIA GTC 2025*, March 17, 2025

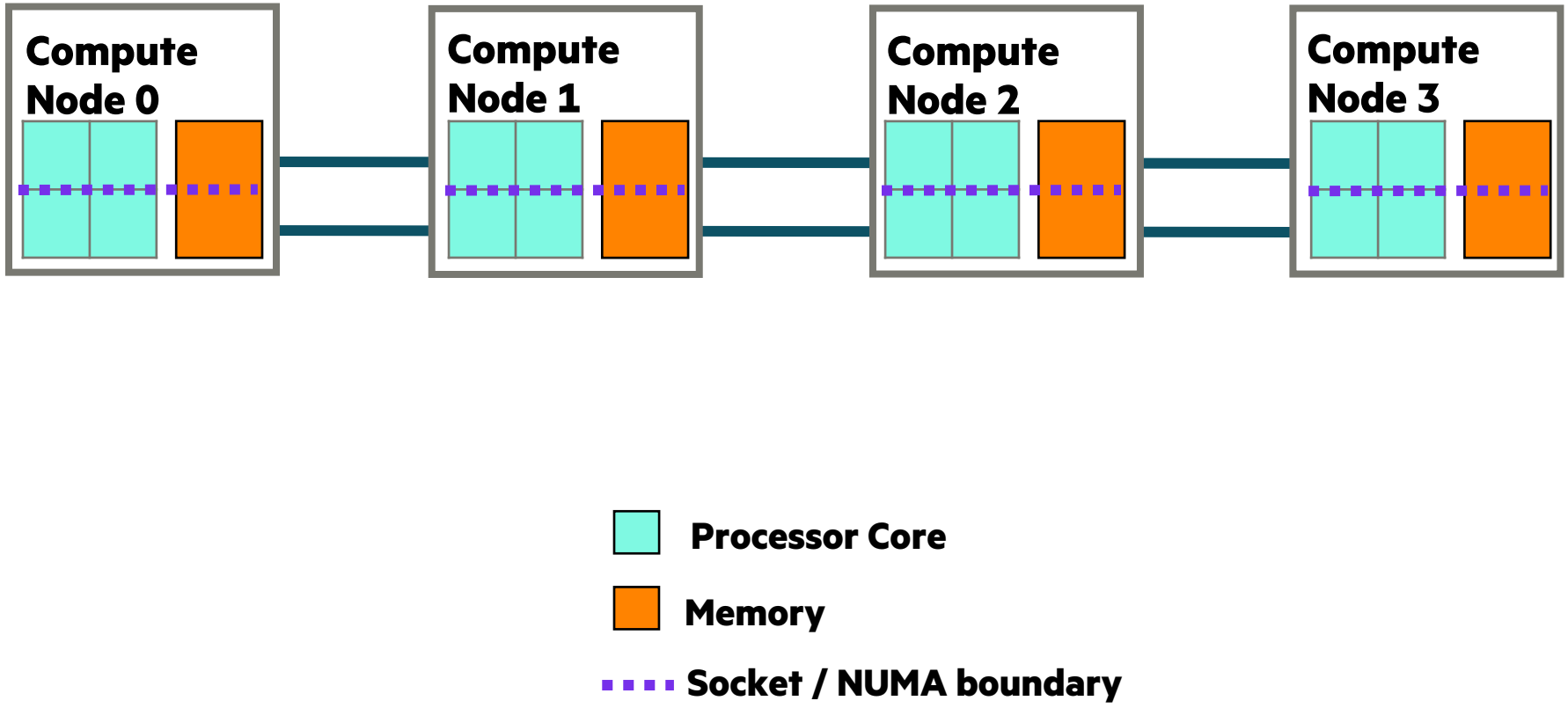
Intra-node locality & co-locales



Locality Concerns within Compute Nodes

Compute nodes have become increasingly hierarchical and locality-sensitive over time

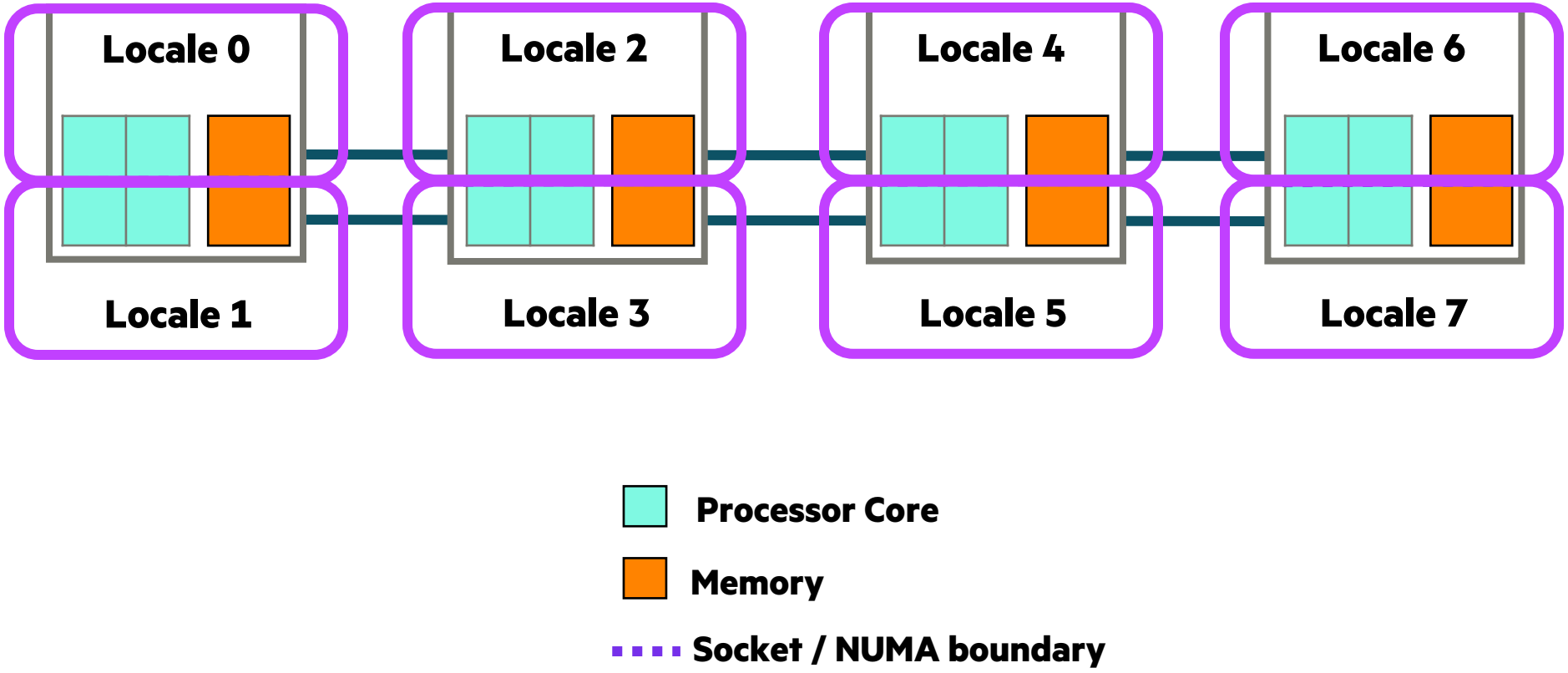
- multiple sockets
- chiplet-based architectures
- non-uniform memory access (NUMA) behaviors
- multiple NICs



Locality Concerns within Compute Nodes

Compute nodes have become increasingly hierarchical and locality-sensitive over time

- multiple sockets
- chiplet-based architectures
- non-uniform memory access (NUMA) behaviors
- multiple NICs



Co-Locales

Introduced in Chapel 1.32–2.0, ***co-locales*** support running multiple locales per compute node

— Command-line interface:

```
$ ./myChplProg -nl 8x2    # use 8 nodes w/ 2 locales each
```

— Typical cases:

- locale per socket
- locale per NUMA domain
- locale per last-level cache
- locale per NIC

Co-locales can improve performance via:

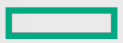
— better memory locality and affinity

— better network utilization

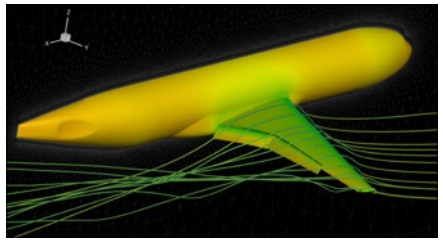
Stream TRIAD on dual-socket nodes, Milan CPUs, 64 cores/CPU

Configuration	GB/s	Improvement	Feature
-nl 2	357	N/A	N/A
-nl 2x2	460	28.9%	Socket
-nl 2x8	466	30.5%	NUMA
-nl 2x16	470	31.7%	L3 cache

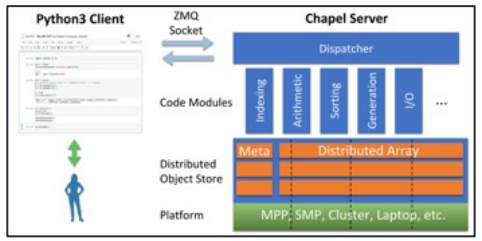
Applications of Chapel



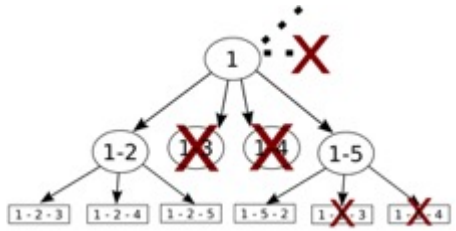
Applications of Chapel



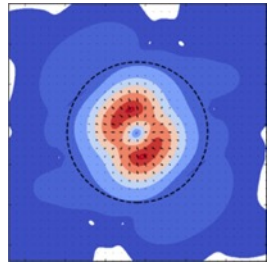
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



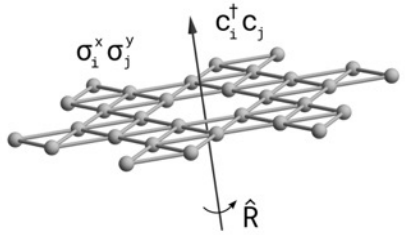
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



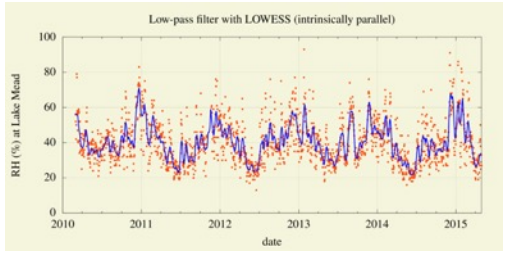
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

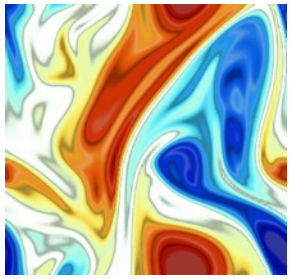


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

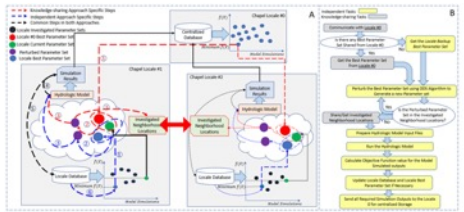


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

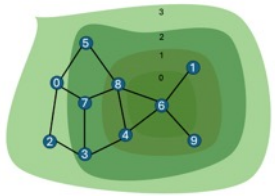
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



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

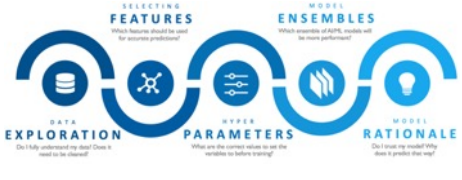


Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



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



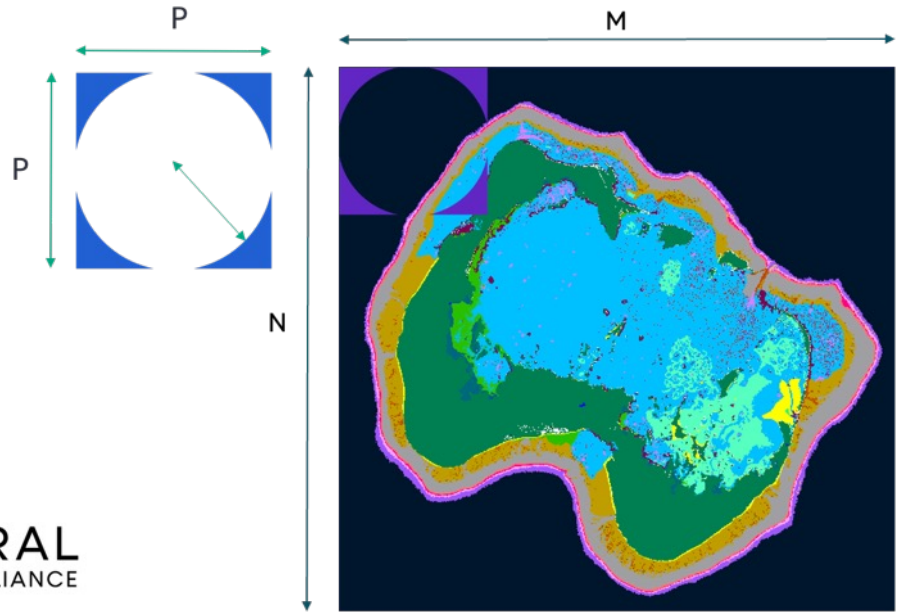
CrayAI HyperParameter Optimization (HPO)
Ben Albrecht et al.
Cray Inc. / HPE

[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

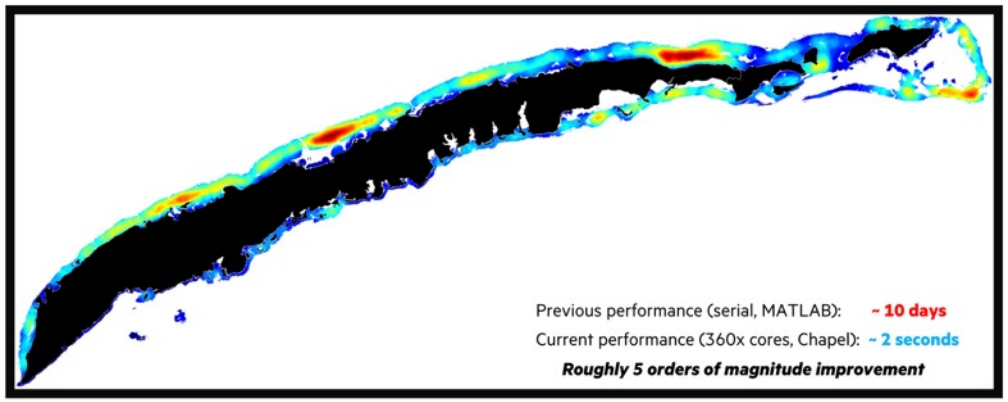


Who wrote it?

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

Why Chapel?

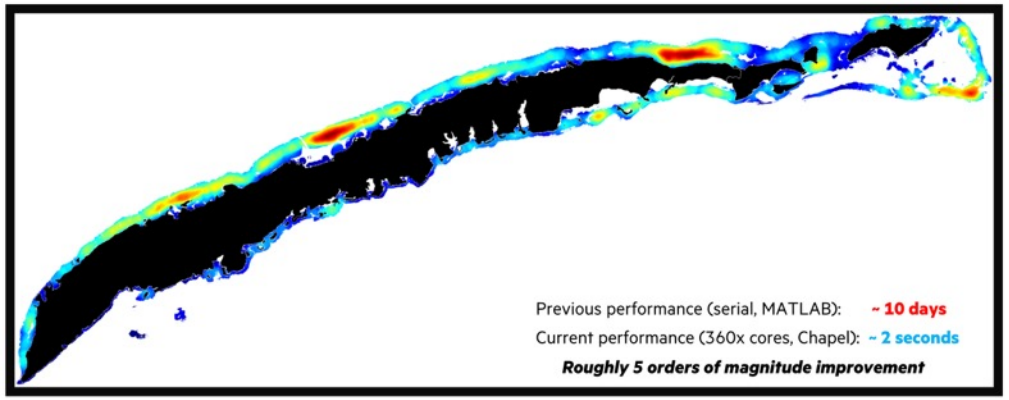
— easy transition from Python/Matlab, which were being used
— massive performance improvement:
  ~10-day Matlab run finished in ~2 seconds using 360 cores
— enabled unexpected algorithmic improvements

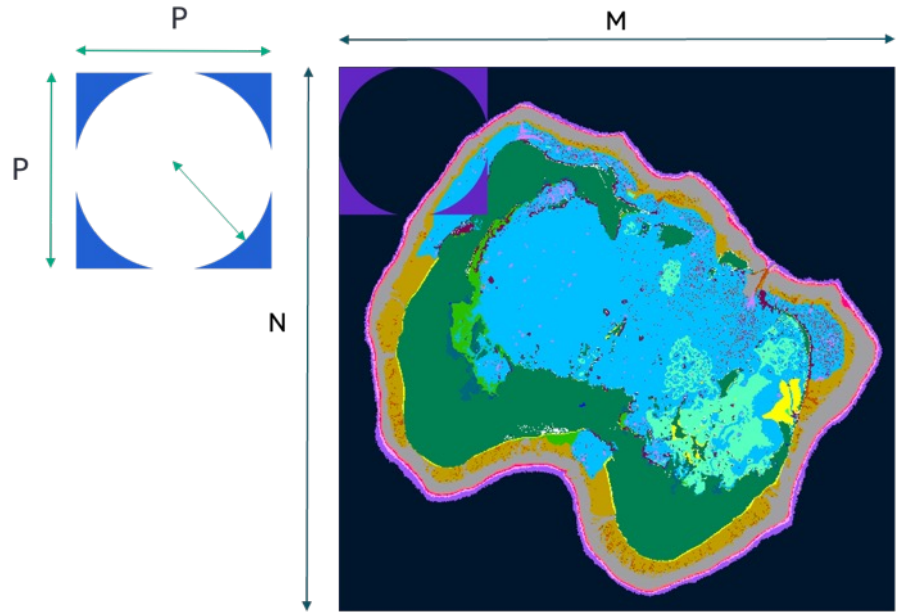


From Scott Bachman's CHIUW 2023 talk:
https://youtu.be/IJhh9KLL2X0

RapidQ Improvements on GPUs

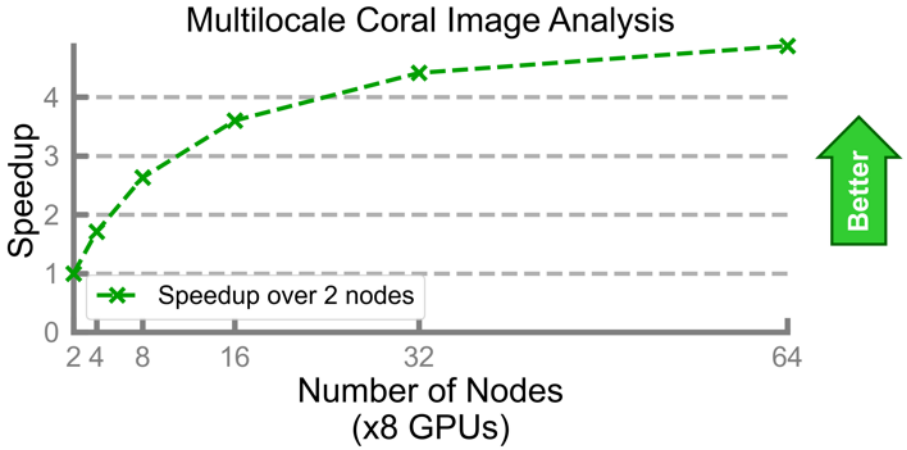
Original algorithm: Habitat Diversity, $O(M \cdot N \cdot P)$



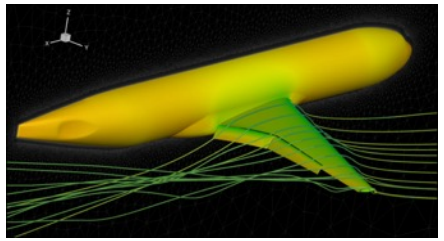


Improved algorithm: Spectral Diversity, $O(M \cdot N \cdot P^3)$

- 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

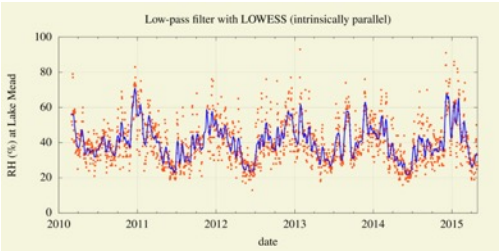


Productivity Across Diverse Application Scales (code and system size)

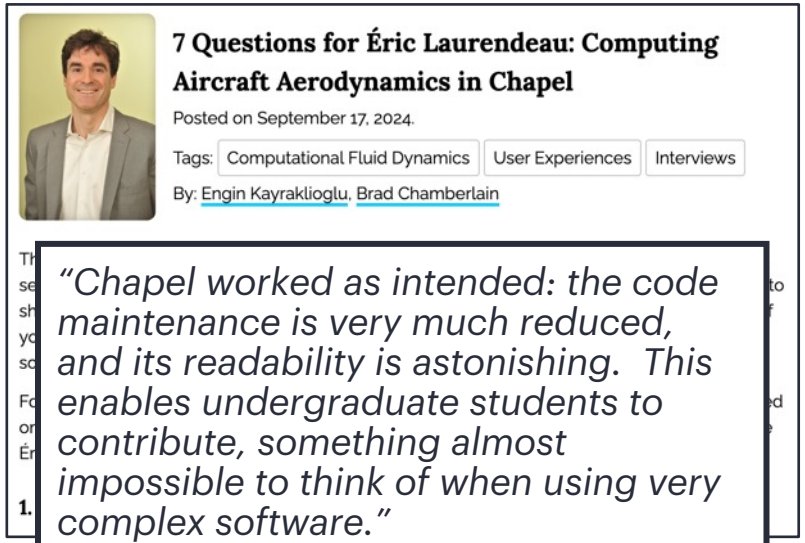


Computation: Aircraft simulation / CFD
Code size: 100,000+ lines
Systems: Desktops, HPC systems

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



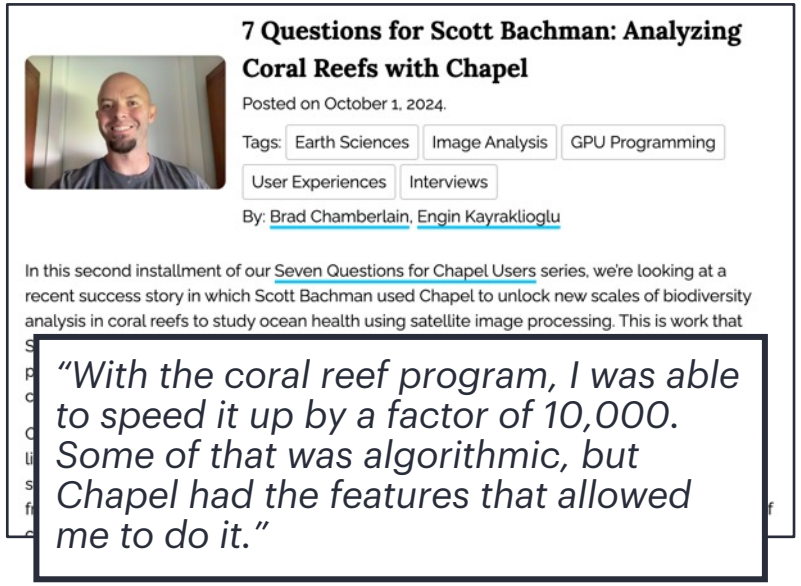
7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

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

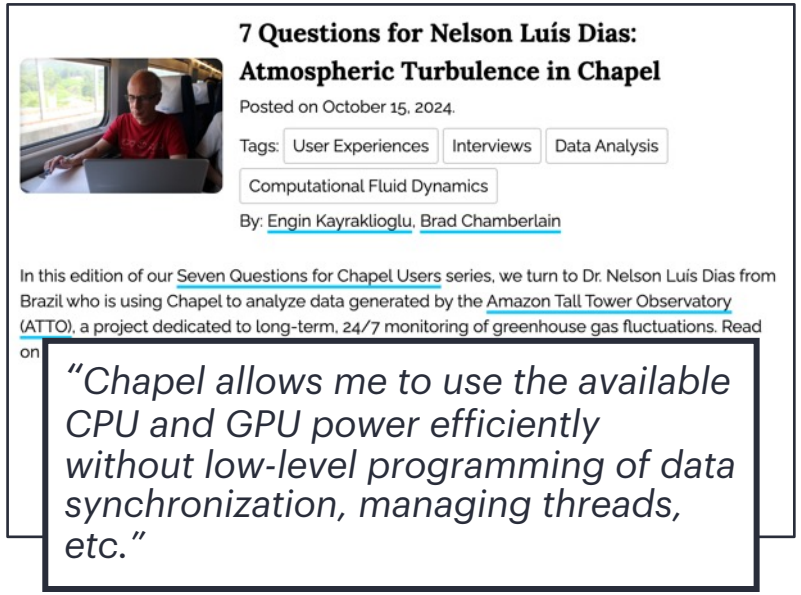
Posted on October 1, 2024.

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 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."

For more, see “7 Questions for Chapel Users” Blog Interviews

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

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

Chapel Language Blog

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 Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

Posted on September 17, 2024.

Tags: Computational Fluid Dynamics | 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 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 Bill Reus: Interactive Supercomputing with Chapel for Cybersecurity

Posted on February 12, 2025.

Tags: Data Analysis | Arkouda | User Experiences | Interviews

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 Marjan Asgari: Optimizing Hydrological Models with Chapel

Posted on September 15, 2025.

Tags: Earth Sciences | 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

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 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

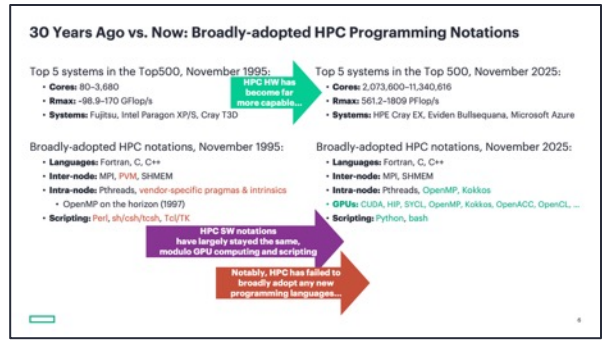
https://chapel-lang.org/blog/series/7-questions-for-chapel-users/

Wrap-up & Closing Thoughts

Summary

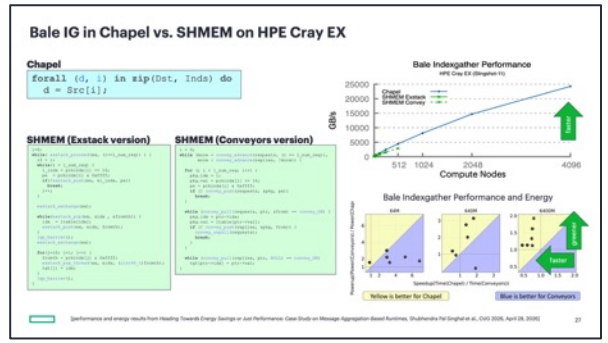
Scalable parallelism traditionally requires a mix of notations to leverage HW

- 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



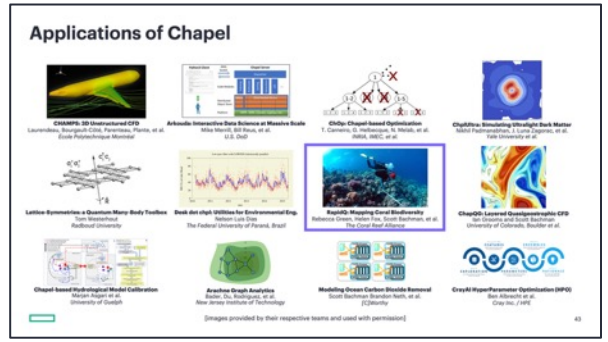
Chapel is a unique programming language

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



User apps are benefitting from Chapel, from laptops to supercomputers

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



A future-proof language?

Chapel has adapted well to architectural changes over its lifetime

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

At this point, the real challenges for being future-proof are social:

- growing the user community, particularly in the age of GenAI programming
- nurturing a multi-institution developer community
- paying off technical debt from the project's research roots
- sustainably funding the project

Such challenges were our motivation to move the project under HPSF/The Linux Foundation

AI and Parallel Languages

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

(* = your mileage may vary)

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

- Targeting a single, unified parallel language should be at least as successful as a mash-up of lower-level notations
- Leveraging higher-level abstractions and compiler optimizations conserves tokens by not re-solving problems
- 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;
}
```

Ways to engage with the Chapel Community

Synchronous Community Events

- Project Meetings, weekly
- Deep Dive / Demo Sessions, weekly timeslot
- ChapelCon (formerly CHIUW)

Asynchrounous Communications

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

Social Media

FOLLOW US

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

Discussion Forums

GET IN TOUCH

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

Ways to Use Chapel

GET STARTED

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

(from the footer of chapel-lang.org)

Thank You

@ChapelLanguage