Hewlett Packard Enterprise

Productive Parallel Programming from the Desktop to the Supercomputer with Chapel

Brad Chamberlain

FNWI Colloquium, Radboud University

October 16, 2025

Q: What makes Chapel unique?

A: It’s one of the few programming languages designed for scalable parallel computing from the outset.

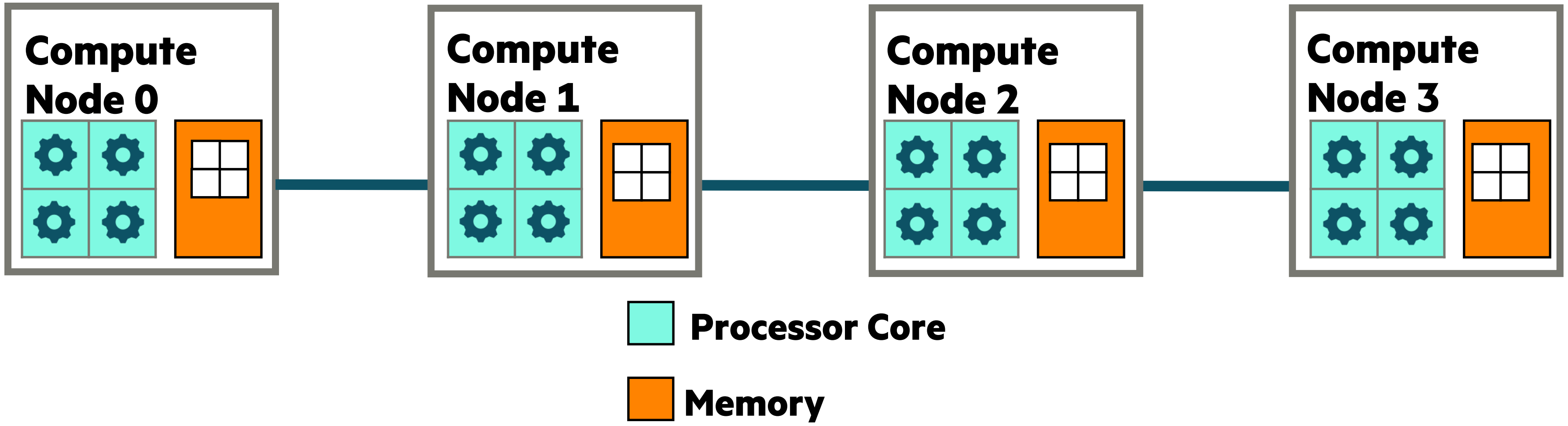
What is [Scalable] Parallel Computing?

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

- Why? To run a program…

  …faster than we could otherwise

  …and/or using larger problem sizes



Scalable Parallel Computing: As more processors and memory are added, benefits increase

HPC = High Performance Computing

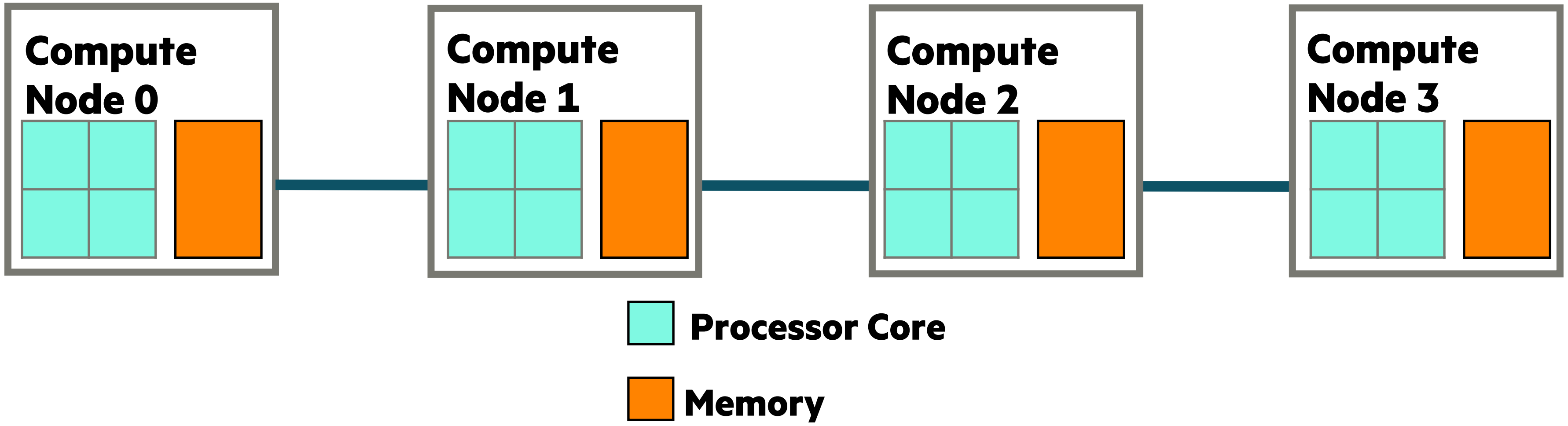
Parallel Computing has become Ubiquitous

Parallel computing, historically:

- supercomputers
- commodity clusters

Additional, ubiquitous parallelism today:

- multicore processors
- cloud computing
- GPUs



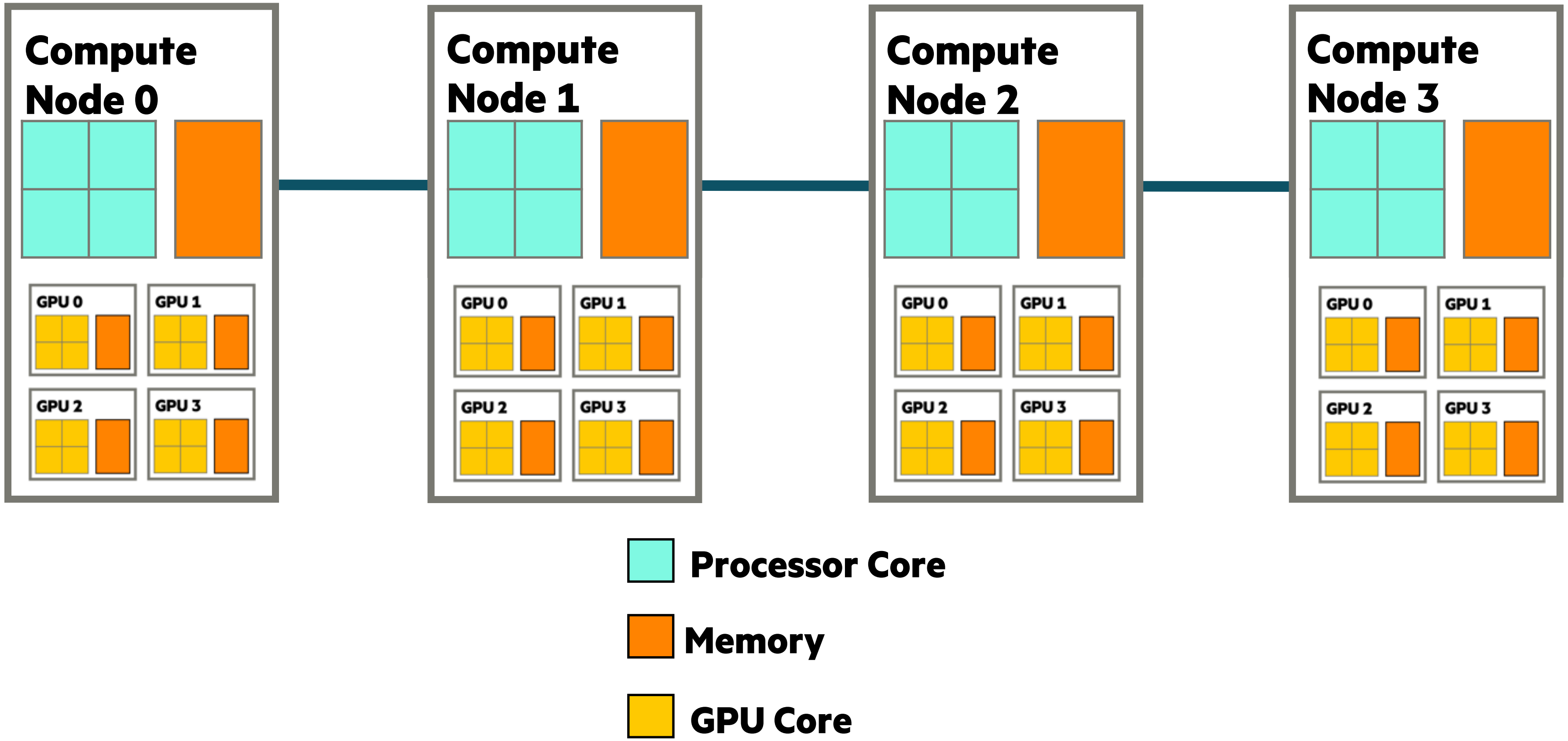
Parallel Computing has become Ubiquitous

Parallel computing, historically:

- supercomputers
- commodity clusters

Additional, ubiquitous parallelism today:

- multicore processors
- cloud computing
- GPUs



Highlights from HIPS 2025 Keynote:
“Reflections on 30 Years of HPC Programming”

30 Years Ago vs. Today: Top HPC Systems

Top 5 systems in the Top500, June 1995:

- **Cores:** 80–3680 cores
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D
- **Networks:** crossbar, mesh, 3D torus

Top 5 systems in the Top 500, June 2025:

- **Cores:** 2,073,600–11,039,616 **(~563x–138,000x)**
- **Rmax:** ~477.9–1742.0 PFlop/s **(~2,810,000x–17,600,000x)**
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure
- **Networks:** Slingshot-11, InfiniBand NDR

TOP500 LIST - JUNE 1995

R_{max} and R_{peak} values are in GFlop/s. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

← 1-100 101-200 201-300 301-400 401-500 →

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	

TOP500 LIST - JUNE 2025

R_{max} and R_{peak} values are in PFlop/s. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

← 1-100 101-200 201-300 301-400 401-500 →

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	11,039,616	1,742.00	2,746.38	29,581
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	793.40	930.00	13,088
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. Today: Top HPC Systems

Top 5 systems in the Top500, June 1995:

- **Cores:** 80–3680 cores
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D
- **Networks:** crossbar, mesh, 3D torus

HPC HW has become far more capable...

Top 5 systems in the Top 500, June 2025:

- **Cores:** 2,073,600–11,039,616 **(~563x–138,000x)**
- **Rmax:** ~477.9–1742.0 PFlop/s **(~2,810,000x–17,600,000x)**
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure
- **Networks:** Slingshot-11, InfiniBand NDR

TOP500 LIST - JUNE 1995

R_{max} and R_{peak} values are in GFlop/s. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

← | 1-100 | 101-200 | 201-300 | 301-400 | 401-500 | →

Rank	System			
1	Numerical Wind Tunnel, Fujitsu National Aerospace Laboratory of Japan Japan			
2	XP/S140, Intel Sandia National Laboratories United States			
3	XP/S-MP 150, Intel DOE/SC/Oak Ridge National Laboratory United States			
4	T3D MC1024-8, Cray/HPE Government United States			
5	VPP500/80, Fujitsu National Lab. for High Energy Physics Japan	80	98.90	128.00

And complex!

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

(Often in ways that hurt programmability)

TOP500 LIST - JUNE 2025

R_{max} and R_{peak} values are in PFlop/s. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

← | 1-100 | 101-200 | 201-300 | 301-400 | 401-500 | →

System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
tan - HPE Cray EX255a, AMD 4th Gen EPYC 24C D Instinct MI300A, Slingshot-11, TOSS, HPE NL	11,039,616	1,742.00	2,746.38	29,581
235a, AMD Optimized 3rd z, AMD Instinct MI250X, PE oratory	9,066,176	1,353.00	2,055.72	24,607
xascale Compute Blade, GHz, Intel Data Center GPU ional Laboratory	9,264,128	1,012.00	1,980.01	38,698
oster - BullSequana XH3000, GH Superchip z, NVIDIA GH200 Superchip, Quad-Rail NVIDIA Band NDR200, RedHat Enterprise Linux, EVIDEN roHPC/FZJ Germany	4,801,344	793.40	930.00	13,088
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. Today: Top HPC Systems and Programming Notations

Top 5 systems in the Top500, June 1995:

- **Cores:** 80–3680 cores
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D
- **Networks:** crossbar, mesh, 3D torus

HPC HW has become far more capable…

Top 5 systems in the Top 500, June 2025:

- **Cores:** 2,073,600–11,039,616 **(~563x–138,000x)**
- **Rmax:** ~477.9–1742.0 PFlop/s **(~2,810,000x–17,600,000x)**
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure
- **Networks:** Slingshot-11, InfiniBand NDR

Broadly-adopted HPC programming notations:

- **Languages:** C, C++, Fortran
- **Inter-node:** MPI, SHMEM
- **Intra-node:** vendor-specific pragmas & intrinsics
  - OpenMP on the horizon: 1997
- **Scripting:** Perl, [[t]c]sh

Broadly-adopted HPC programming notations:

- **Languages:** C, C++, Fortran
- **Inter-node:** MPI, SHMEM
- **Intra-node:** OpenMP, vendor-specific pragmas & intrinsics
- **GPUs:** CUDA, HIP, SYCL, Kokkos, OpenMP, OpenACC, …
- **Scripting:** Python, bash

30 Years Ago vs. Today: Top HPC Systems and Programming Notations

Top 5 systems in the Top500, June 1995:

- **Cores:** 80–3680 cores
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D
- **Networks:** crossbar, mesh, 3D torus

HPC HW has become far more capable…

Top 5 systems in the Top 500, June 2025:

- **Cores:** 2,073,600–11,039,616 **(~563x–138,000x)**
- **Rmax:** ~477.9–1742.0 PFlop/s **(~2,810,000x–17,600,000x)**
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure
- **Networks:** Slingshot-11, InfiniBand NDR

Broadly-adopted HPC programming notations:

- **Languages:** C, C++, Fortran
- **Inter-node:** MPI, SHMEM
- **Intra-node:** vendor-specific pragmas & intrinsics
  - OpenMP on the horizon: 1997
- **Scripting:** Perl, [[t]c]sh

…while HPC notations have largely stayed the same, modulo GPU computing

Broadly-adopted HPC programming notations:

- **Languages:** C, C++, Fortran
- **Inter-node:** MPI, SHMEM
- **Intra-node:** OpenMP, vendor-specific pragmas & intrinsics
- **GPUs:** CUDA, HIP, SYCL, Kokkos, OpenMP, OpenACC, …
- **Scripting:** Python, bash

Meanwhile, in Mainstream Computing…

- Consider all the currently relevant languages that emerged or rose to prominence during those 30 years:
  - **Java** (~1995)
  - **Javascript** (~1995)
  - **Python** (~1989; v2.0 ~2000)
  - **C#** (~2000)
  - **Go** (~2009)
  - **Rust** (~2012)
  - **Julia** (~2012)
  - **Swift** (~2014)

Recurring themes: productivity, safety, portability, performance

Such languages have become favorite day-to-day languages for many users across multiple disciplines

Why can't HPC have nice things too? (Or maybe we can…?)

Outline

- Background & Motivation
- Introduction to Chapel
- Chapel Applications
- Wrap-up

What is Chapel?

Chapel: A modern parallel programming language

- Portable & scalable
- Open-source & collaborative

Goals:

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

Productive Parallel Programming: One Definition

Imagine a programming language for parallel computing that is as…

…**readable and writeable** as Python

…yet also as…

…**fast** as Fortran / C / C++

…**scalable** as MPI / SHMEM

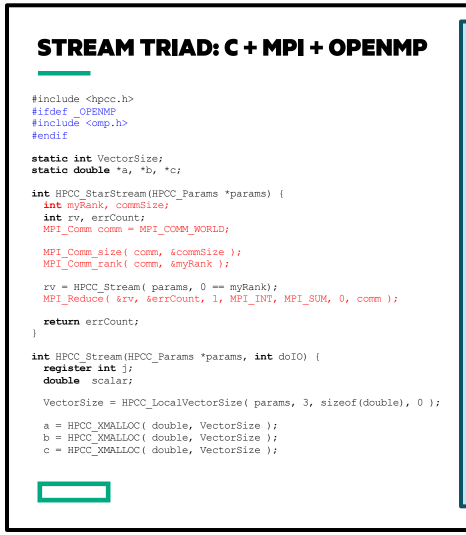
…**GPU-ready** as CUDA / HIP / OpenMP / Kokkos / OpenCL / OpenACC / …

…**portable** as C

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

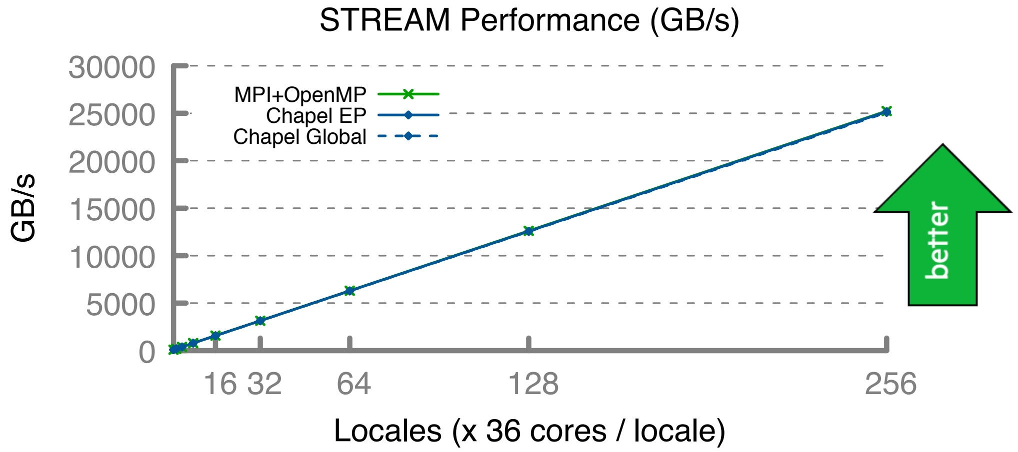
This is our motivation for Chapel

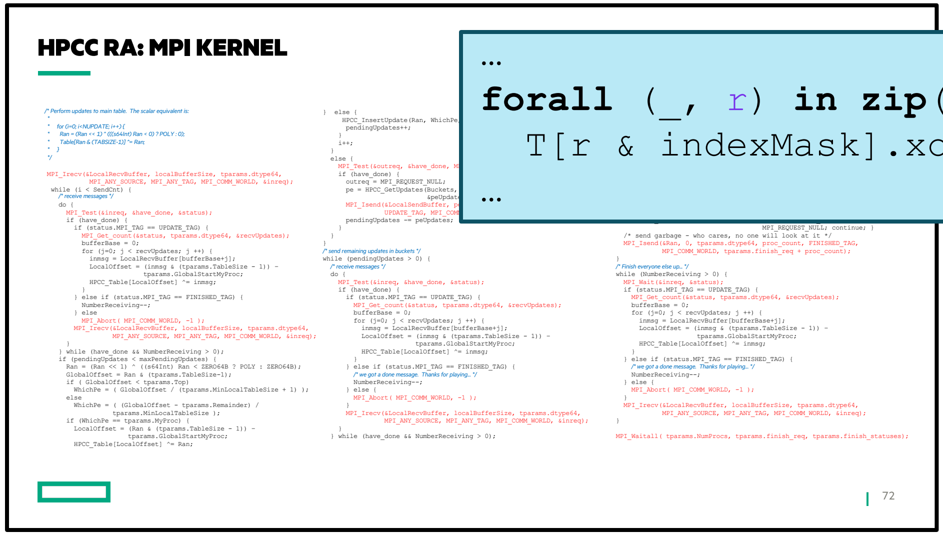
HPCC Stream Triad and RA in C + MPI + OpenMP vs. Chapel



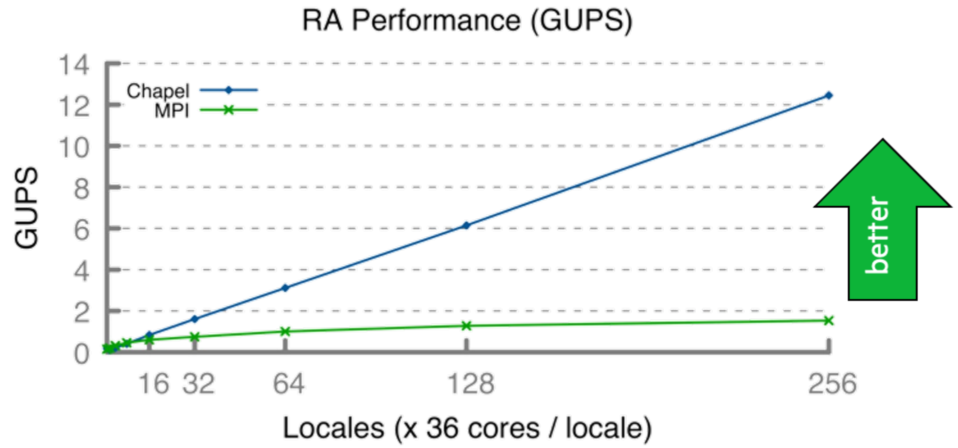

```
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

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

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

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

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

  return errCount;
}

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

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

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

```
use BlockDist;

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

B = 2.0;
C = 1.0;

A = B + alpha * C;
```



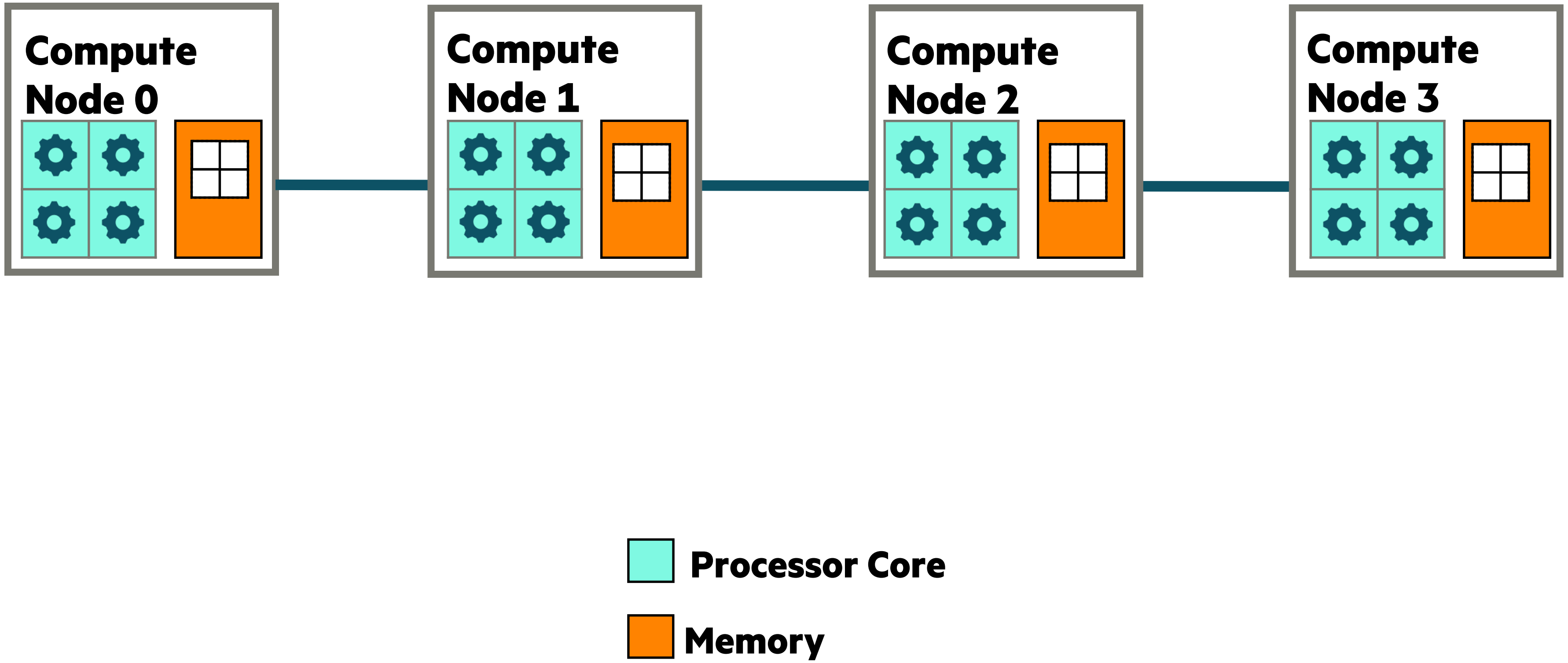



```
…
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask].xor(r);
…
```



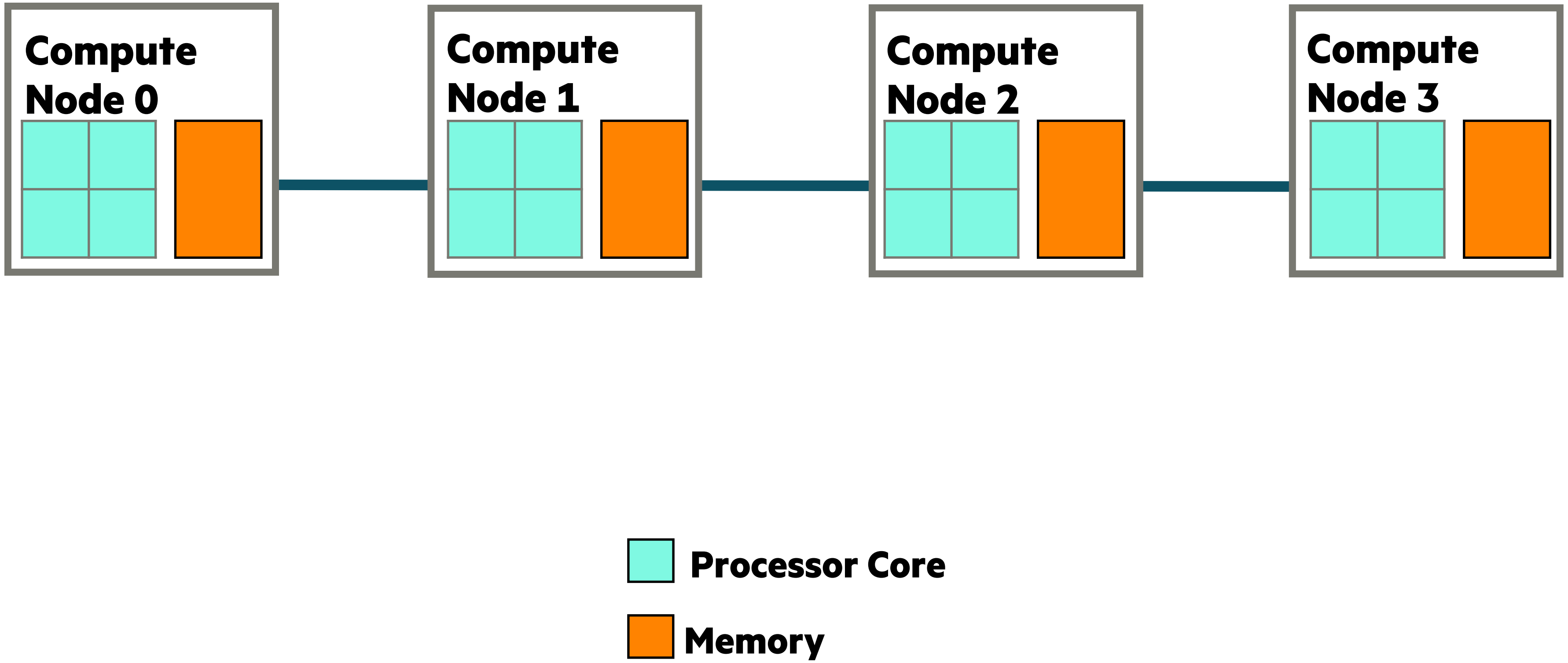
Key Concerns for Scalable Parallel Computing

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



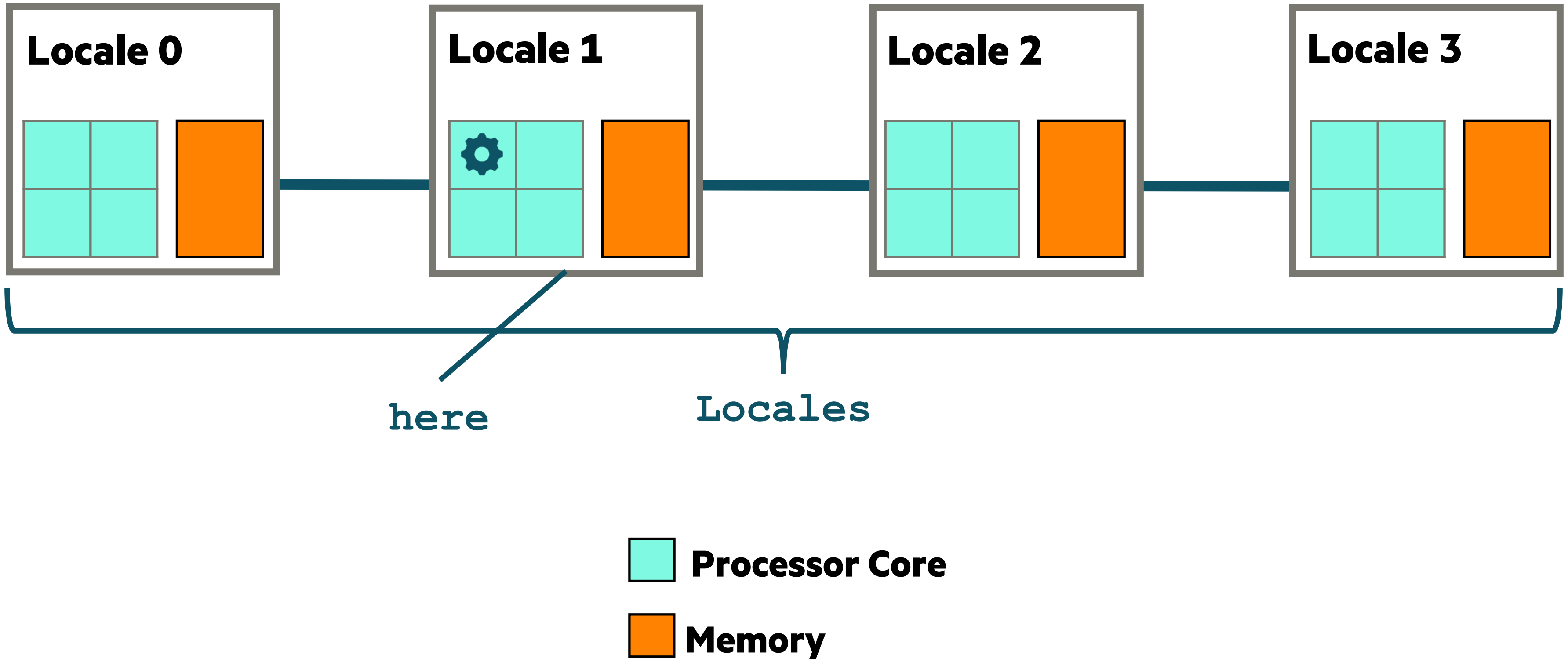
Locales in Chapel

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



Built-In Locale Variables in Chapel

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



“Low-level” parallelism and locality in Chapel

Basic Features for Locality

on.chpl

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

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

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

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

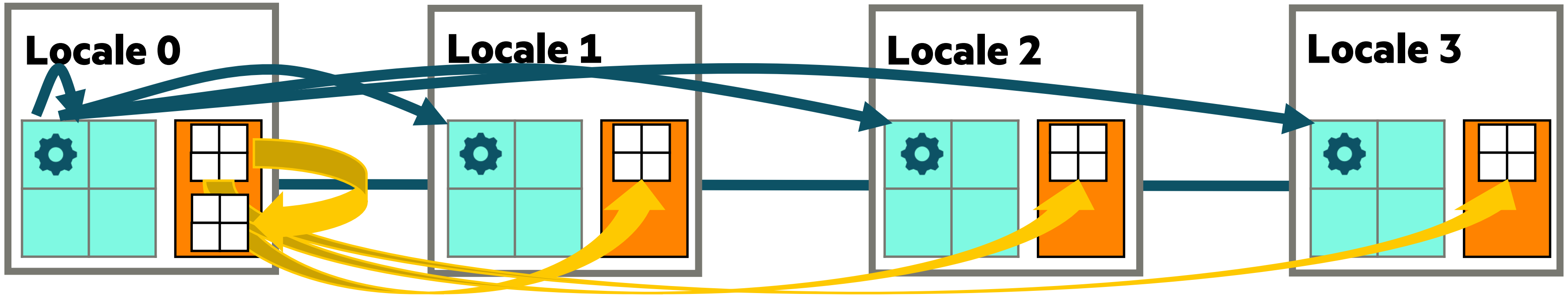
Variables are stored using the memory local to the current task

This loop will serially iterate over the program's locales

on-clauses move tasks to target locales

remote variables can be accessed directly

This is a distributed, yet serial, computation



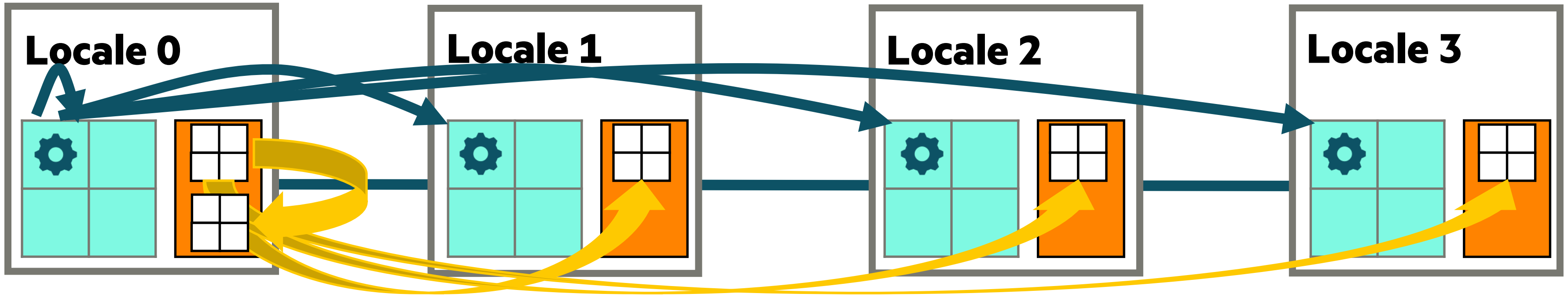
Mixing Locality with Task Parallelism

coforall.chpl

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

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

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

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

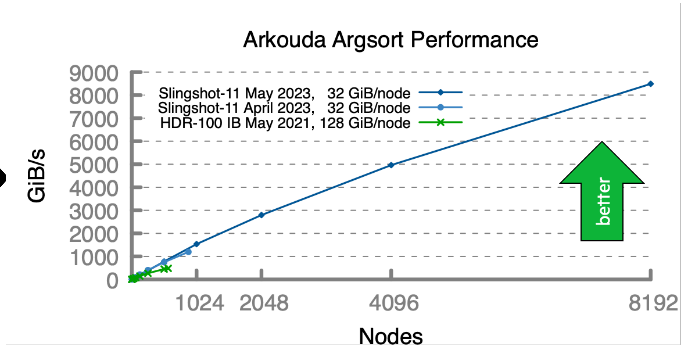
This is a distributed parallel computation



Chapel's Adaptability

Chapel pre-dates all of the architectural changes mentioned previously, apart from commodity vectors

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



Yet it supports all of these HW features

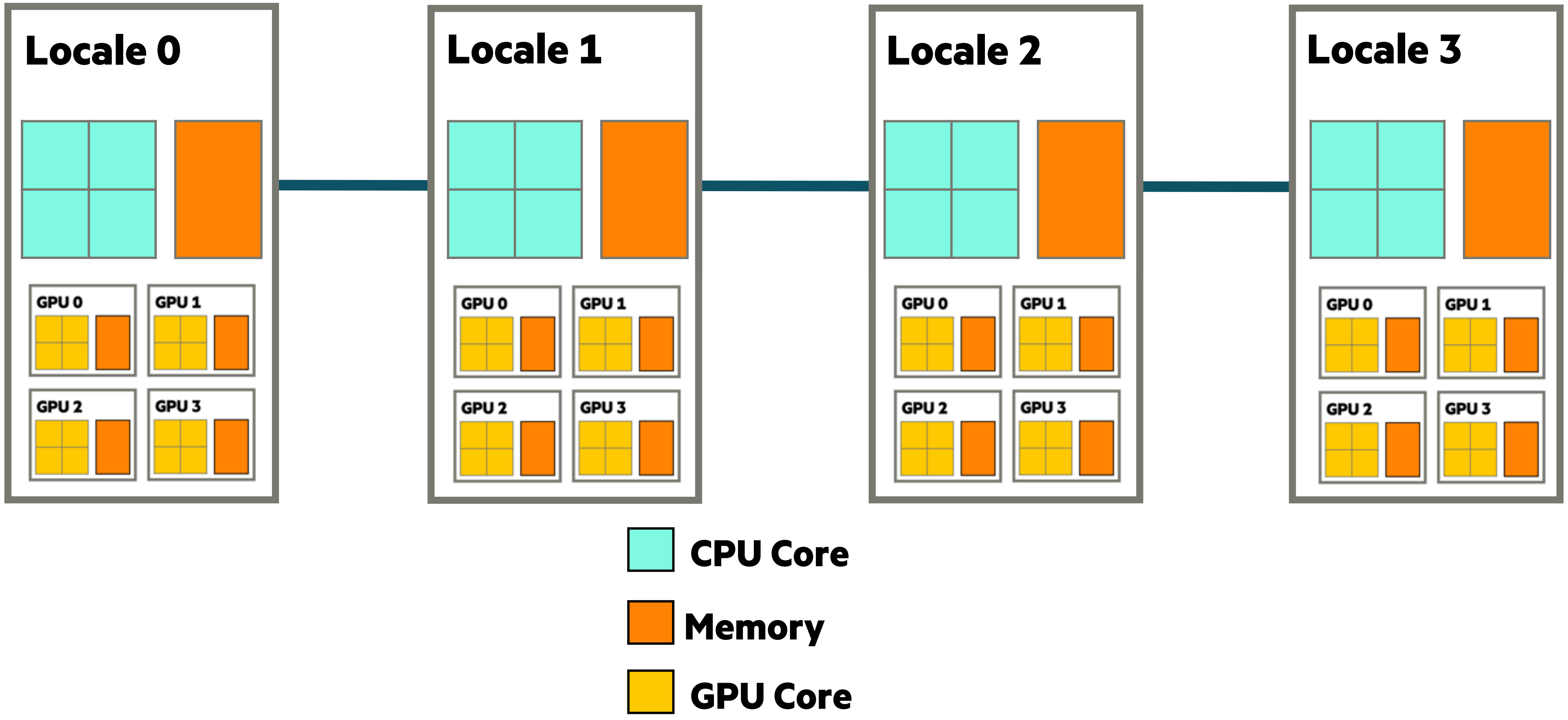
– Using essentially the same language features as ~20 years ago

– How? By expressing parallelism and locality independently from HW mechanisms

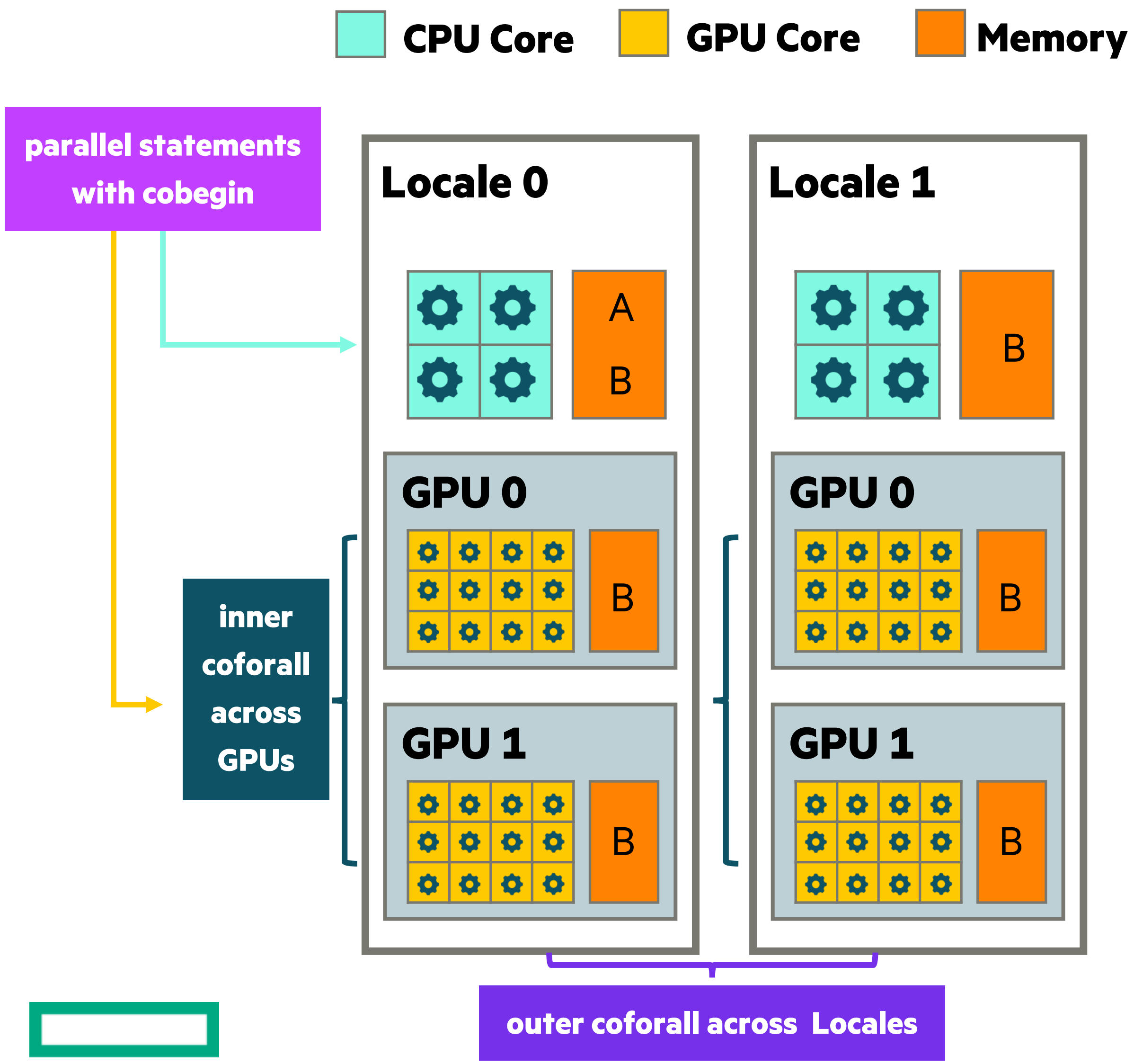
Representing GPUs in Chapel

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

```
on here.gpus[0] { … }                    coforall gpu in here.gpus do on gpu { … }
```



Targeting CPUs and GPUs using Parallelism and Locality




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

High-level parallelism and locality in Chapel

Data Parallelism

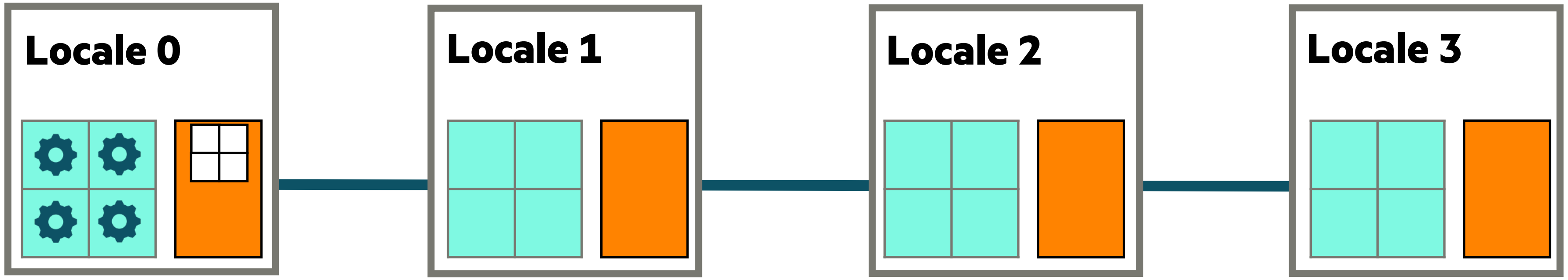
forall.chpl

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

forall a in A {
  a += 1;
}
```

The forall loop's iterand specifies how parallelism is implemented

A 'forall' over a local array, like 'A' here, creates a task per core, dividing the work evenly

This results in a local parallel computation



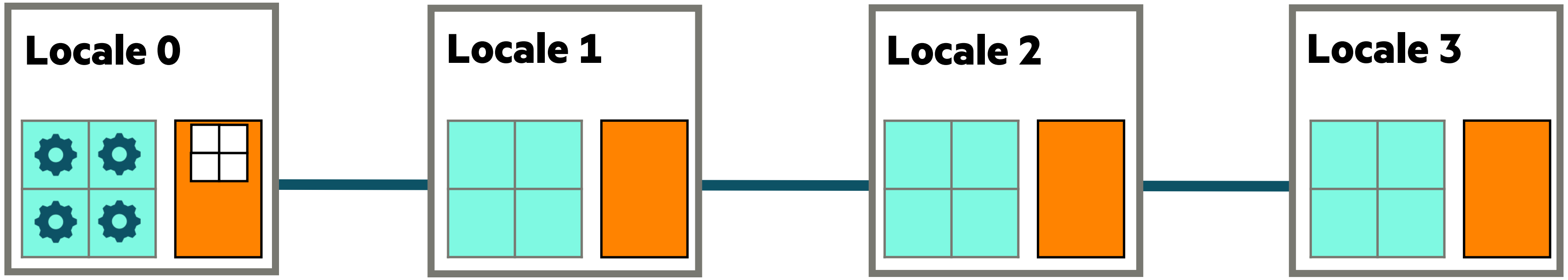
Data Parallelism using Domains

forall-dom.chpl

```
const D = {1..2, 1..2};
var A: [D] real;

forall a in A {
  a += 1;
}
```

A domain is a named index set that can be used to declare arrays…

This is equivalent to the previous slide



Data Parallelism using Domains

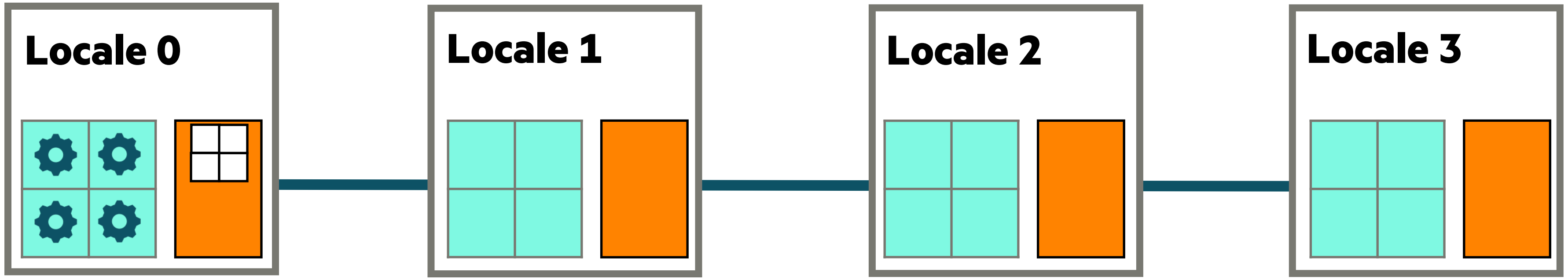
forall-dom-loop.chpl

```
const D = {1..2, 1..2};
var A: [D] real;

forall i in D {
  A[i] += 1;
}
```

A domain is a named index set that can be used to declare arrays…

…and to drive loops
(using the same parallelization
as local arrays)

This is also equivalent to the previous slides



Data Parallelism using Distributed Domains

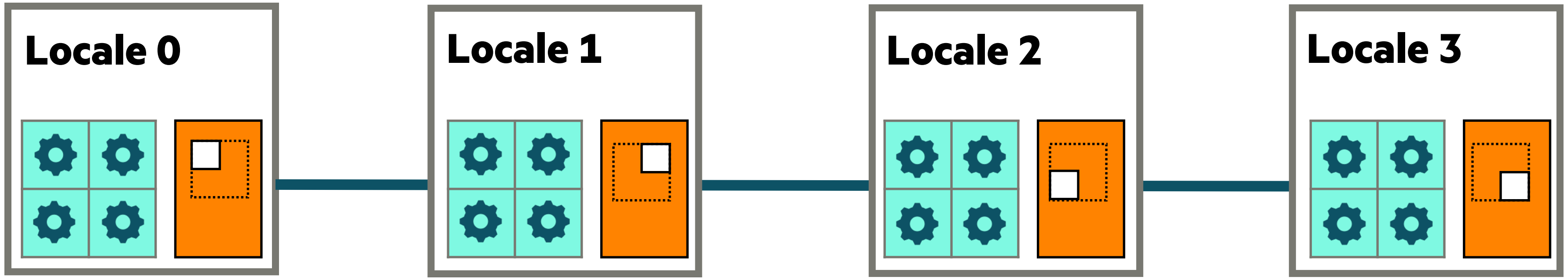
forall-dist-dom.chpl

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

forall i in D {
  A[i] += 1;
}
```

Distributed domains distribute their indices—and their arrays' elements—across the target locales

Forall loops over distributed domains use all the cores on all locales owning a subdomain

This results in a distributed parallel computation



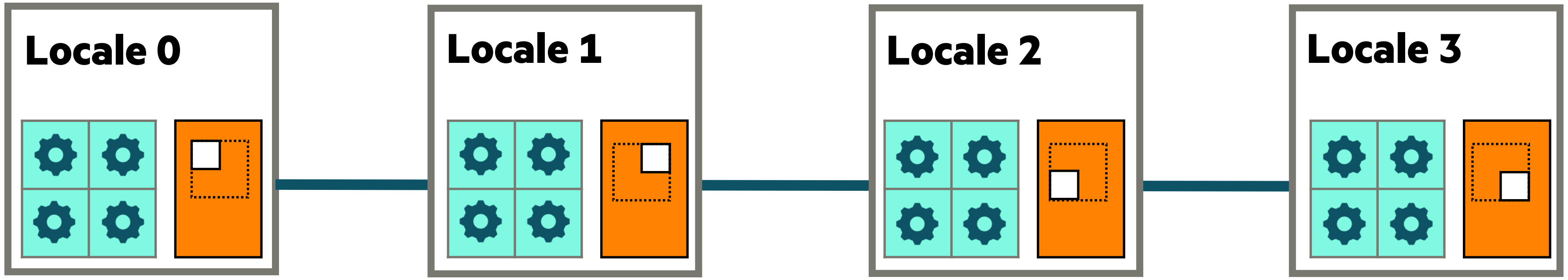
Data Parallelism using Distributed Arrays

forall-dist-arr.chpl

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

forall a in A {
  a += 1;
}
```

Forall loops over distributed arrays act similarly

This is equivalent to the previous slide



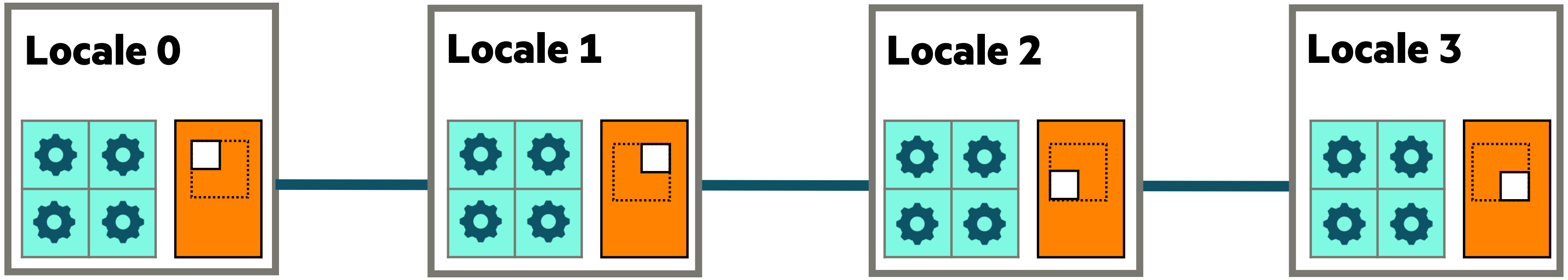
Data Parallelism using Promotion over a Distributed Array

promotion-dist.chpl

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

A += 1;
```

Scalar functions and operators
(like += here)
can be called with array arguments

This is also equivalent to the last few slides



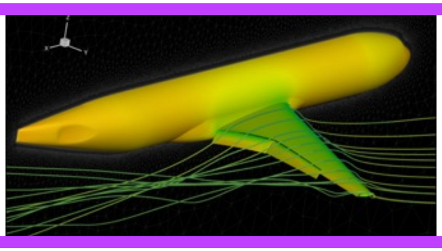
And much, much more…

This has just been a small taste of Chapel… there's much more

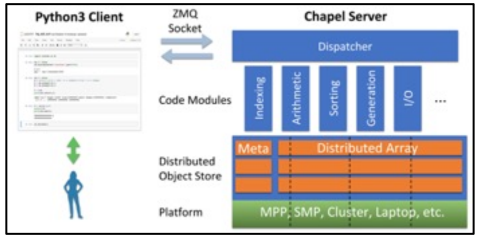
- atomic and sync types for synchronizing between tasks
- additional ways to create tasks and parallel loops
- object-oriented features
- iterators
- generics, polymorphism, overloading
- default arguments, keyword-based argument passing
- namespacing
- interoperability
- etc.

Chapel Applications

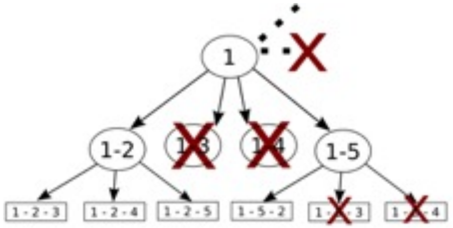
Applications of Chapel



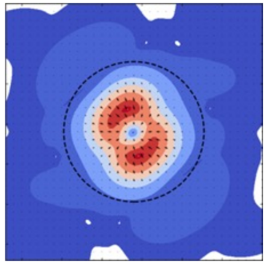
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



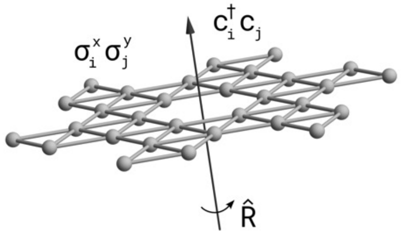
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



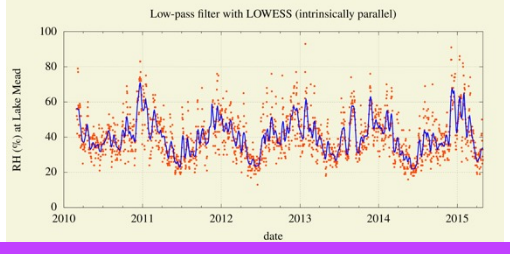
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

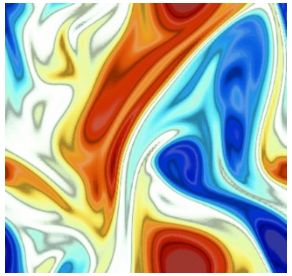


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

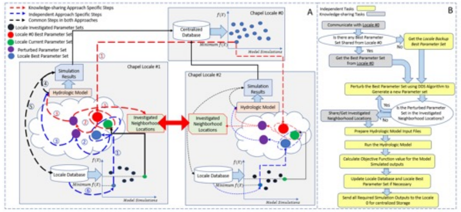


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

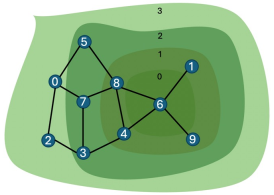
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



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

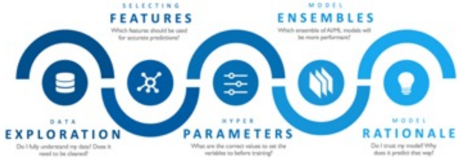


Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



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

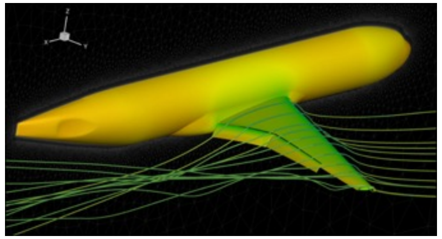
Modeling Ocean Carbon Dioxide Removal
Scott Bachman Brandon Neth, et al.
[C]Worthy



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

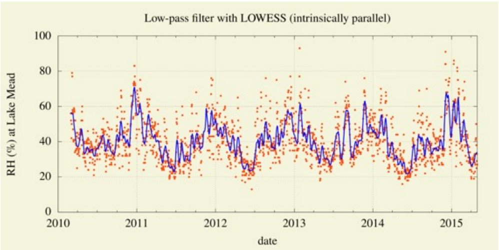
[images provided by their respective teams and used with permission]

Diversity in Application Scales (both in terms of code and systems)



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

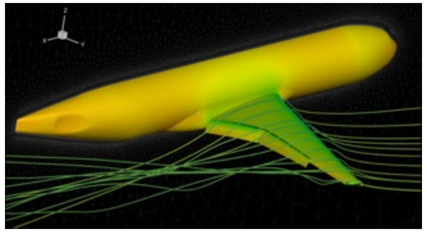


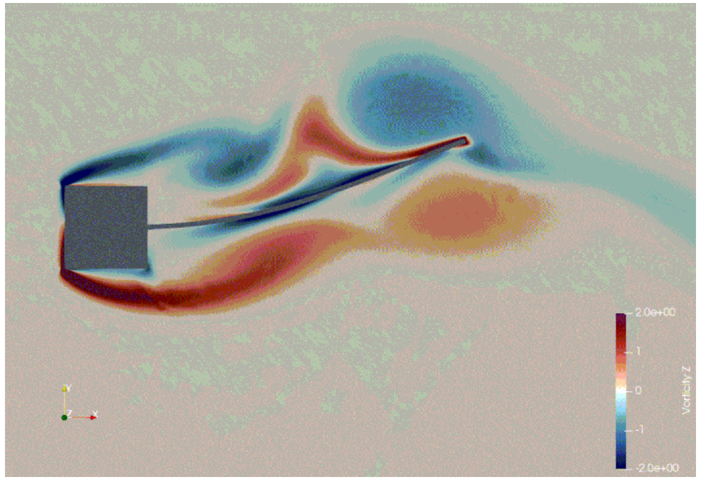
Computation: Atmospheric data analysis
Code size: 5000+ lines
Systems: Desktops, sometimes w/ GPUs

CHAMPS Summary

What is it?

- 3D unstructured CFD framework for airplane simulation
- ~100+k lines of Chapel written since 2019

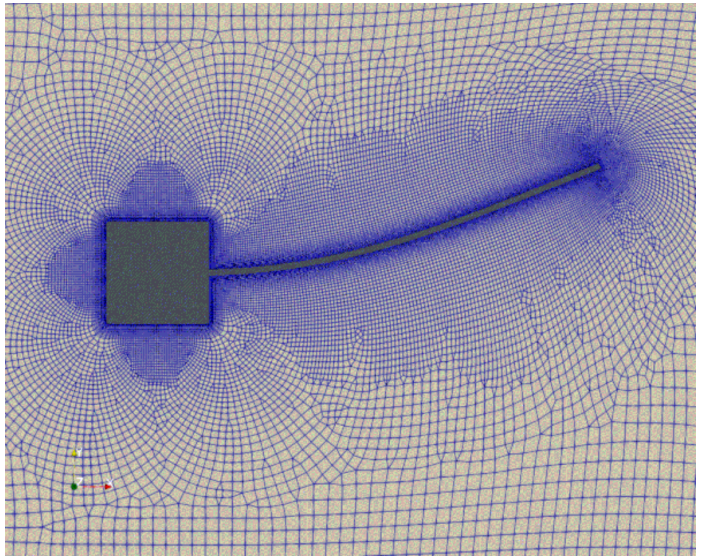




Who wrote it?

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

POLYTECHNIQUE MONTRÉAL



Why Chapel?

- performance and scalability competitive with MPI + C++
- students found it far more productive to use
- enabled them to compete with more established CFD centers

(images provided by the CHAMPS team and used with permission)

CHAMPS: Excerpt from Éric's CHIUW 2021 Keynote (transcript)

HPC Lessons From 30 Years of Practice in CFD Towards Aircraft Design and Analysis (June 4, 2021)

*"To show you what Chapel did in our lab… [our previous framework] ended up 120k lines. And my students said, 'We can't handle it anymore. It's too complex, we lost track of everything.' And today, they went **from 120k lines to 48k lines, so 3x less**.*

*But the code is not 2D, it's 3D. And it's not structured, it's unstructured, which is way more complex. And it's multi-physics… **So, I've got industrial-type code in 48k lines.**"*

*"[Chapel] promotes the programming efficiency … **We ask students at the master's degree to do stuff that would take 2 years and they do it in 3 months.** So, if you want to take a summer internship and you say, 'program a new turbulence model,' well they manage. And before, it was impossible to do."*

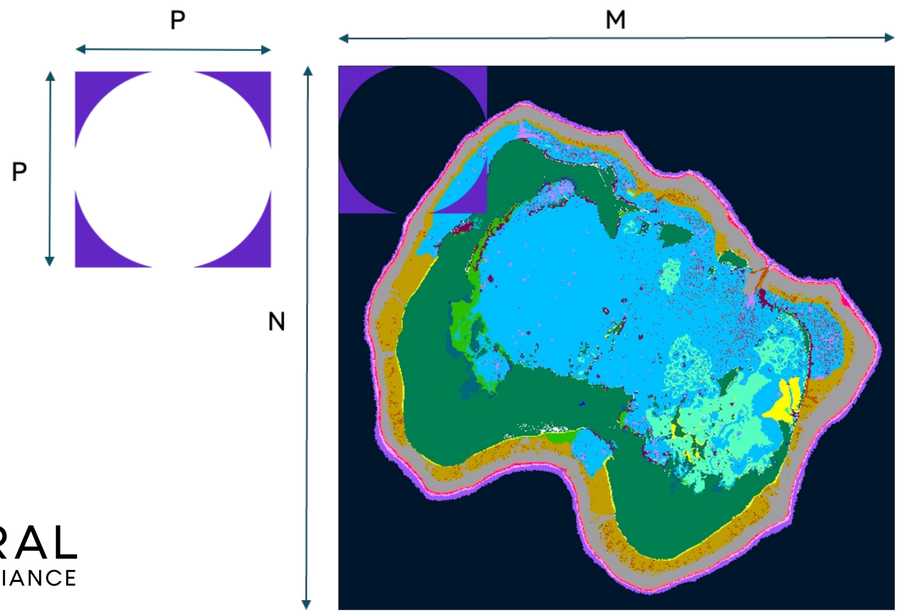
POLYTECHNIQUE MONTRÉAL

*"So, for me, this is like the proof of the benefit of Chapel, **plus the smiles I have on my students everyday in the lab because they love Chapel as well.** So that's the key, that's the takeaway."*

Talk available online: https://youtu.be/wD-a_KyB8aI?t=1904 (hyperlink jumps to the section quoted here)

(images provided by the CHAMPS team and used with permission)

RapidQ Coral Biodiversity Summary



What is it?

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

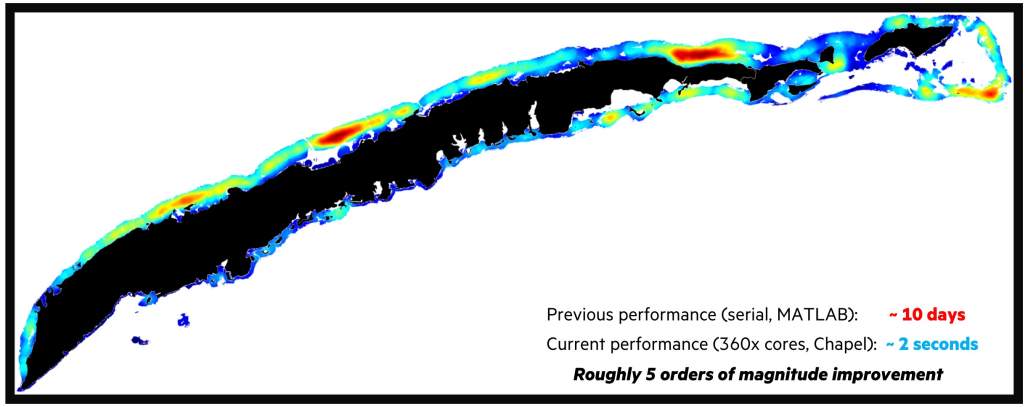
Who wrote it?

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

NCAR Operated by UCAR
[C]Worthy
CORAL REEF ALLIANCE

Why Chapel?

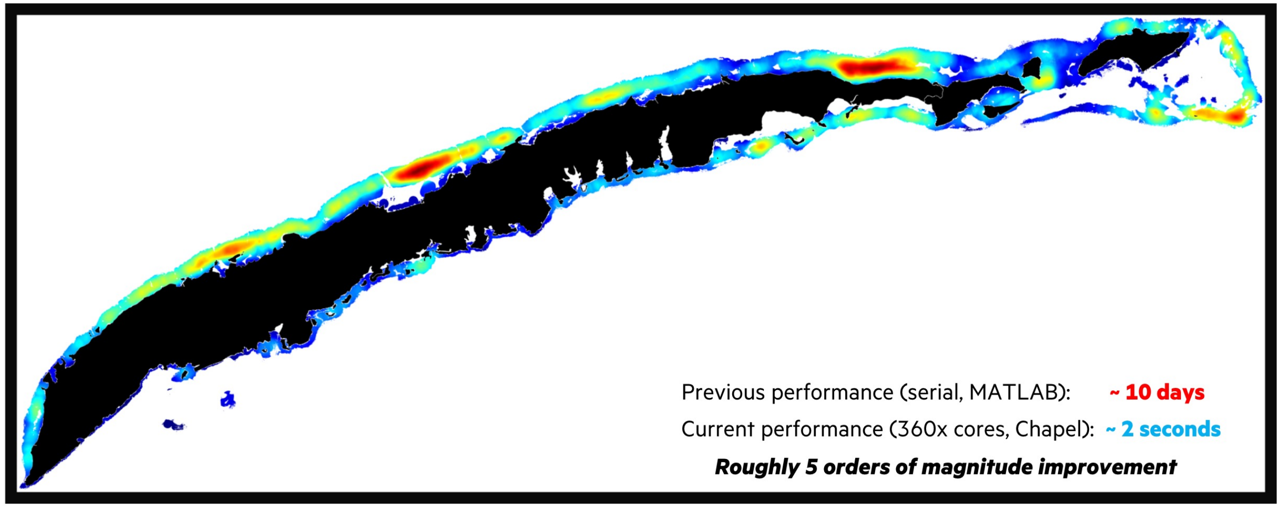
- easy transition from Matlab/Python, which were being used
- massive performance improvement:
  previous ~10-day run finished in ~2 seconds using 360 cores
- enabled unexpected algorithmic improvements



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

Coral Reef Spectral Biodiversity: Productivity and Performance

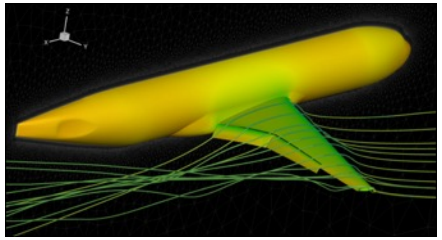
Original algorithm: Habitat Diversity, $O(M \cdot N \cdot P)$



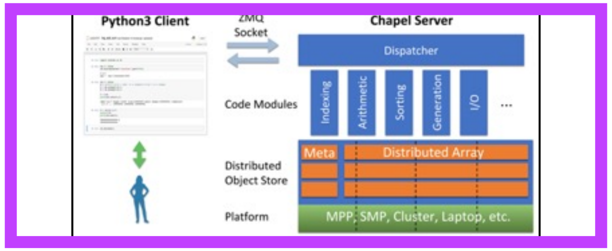
Improved algorithm: Spectral Diversity, $O(M \cdot N \cdot P^3)$

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

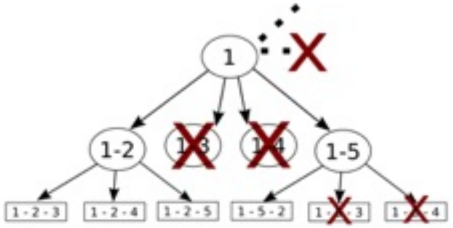
Applications of Chapel



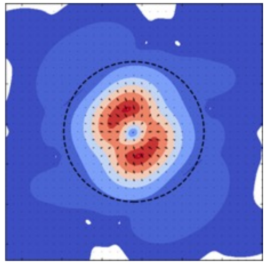
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



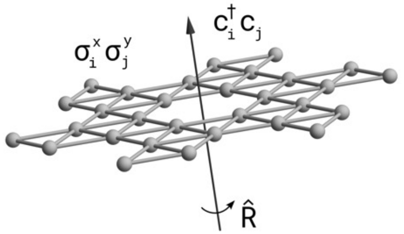
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



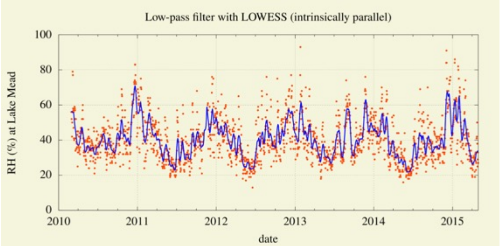
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

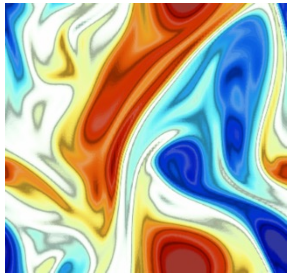


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

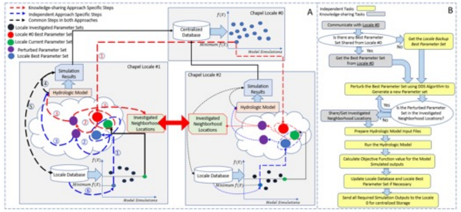


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

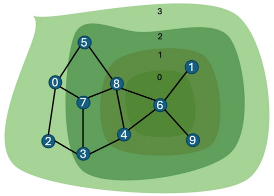
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



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

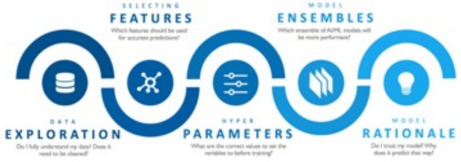


Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



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

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



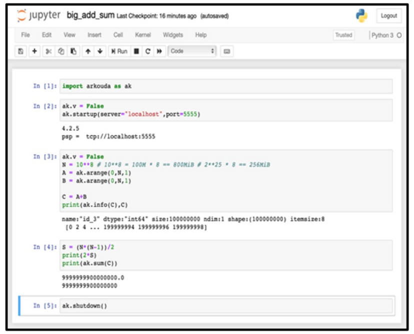
CrayAI HyperParameter Optimization (HPO)
Ben Albrecht et al.
Cray Inc. / HPE

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

What is Arkouda?

Q: “What is Arkouda?”

Arkouda Client
(written in Python)

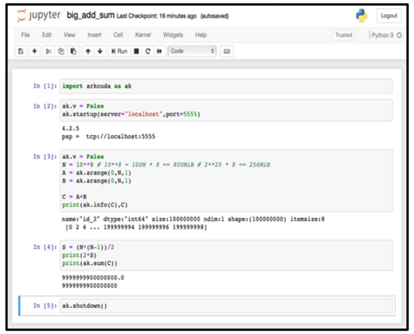


User writes Python code
making familiar NumPy/Pandas calls

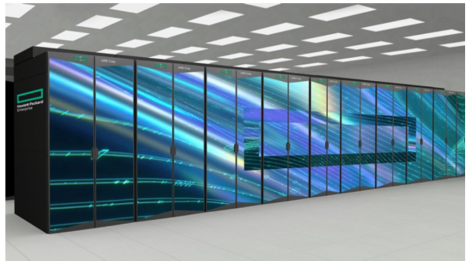
What is Arkouda?

Q: “What is Arkouda?”

Arkouda Client
(written in Python)



Arkouda Server
(written in Chapel)



User writes Python code
making familiar NumPy/Pandas calls

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

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

Performance and Productivity: Arkouda Argsort

HPE Cray EX

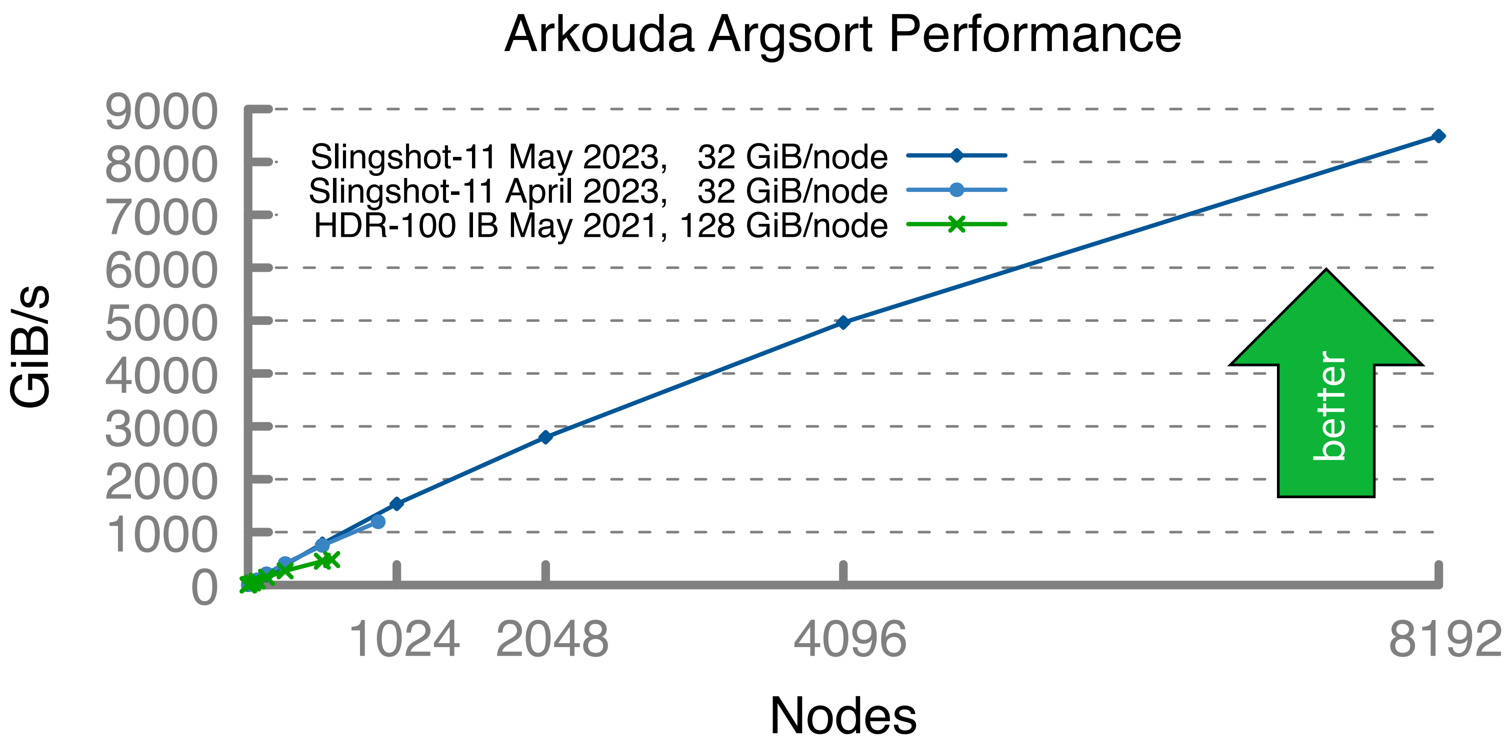
- Slingshot-11 network (200 Gb/s)
- 8192 compute nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)

HPE Cray EX

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

HPE Apollo

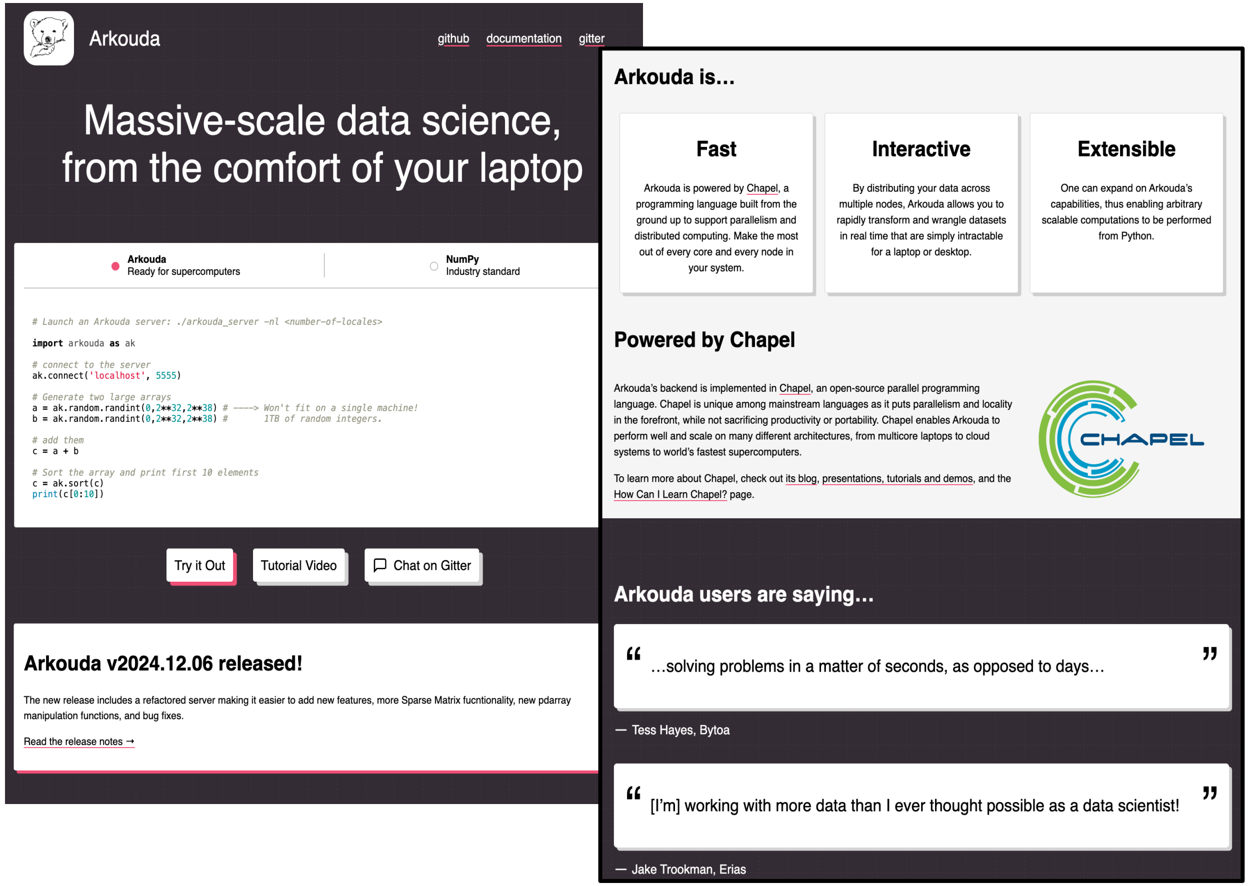
- HDR-100 InfiniBand network (100 Gb/s)
- 576 compute nodes
- 72 TiB of 8-byte values
- ~480 GiB/s (~150 seconds)



Implemented using ~100 lines of Chapel

For More Information on Arkouda

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



“7 Questions for Chapel Users” Interview series

A good way to learn more about Chapel users’ apps and experiences

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

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

Posted on November 6, 2024.

Tags: User Experiences | Interviews | Graph Analytics | Arkouda

By: Engin Kayraklioglu, Brad Chamberlain

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

Posted on July 30, 2025.

Tags: User Experiences | Interviews

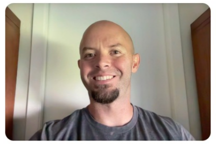
By: Engin Kayraklioglu, Brad Chamberlain

7 Questions for Marjan Asgari: Optimizing Hydrological Models with Chapel

Posted on September 15, 2025.

Tags: User Experiences | Interviews | Earth Sciences

By: Engin Kayraklioglu, Brad Chamberlain



7 Questions for Scott Bachman: Analyzing Coral Reefs with Chapel

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

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

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

Chapel Language Blog

About | Chapel Website | Featured | Series | Tags | Authors | All Posts

7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

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

7 Questions for Nelson Luís Dias: Atmospheric Turbulence in Chapel

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

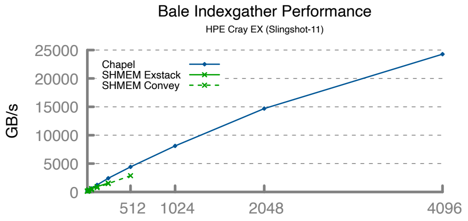
Wrap-up

Summary

Chapel is unique among programming languages

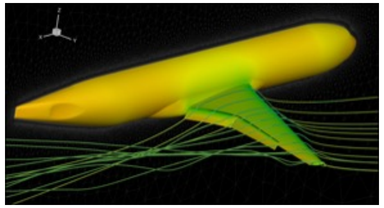
- supports first-class concepts for parallelism and locality
- ports and scales from laptops to supercomputers
- supports clean, concise code relative to conventional approaches
- supports GPUs in a vendor-neutral manner

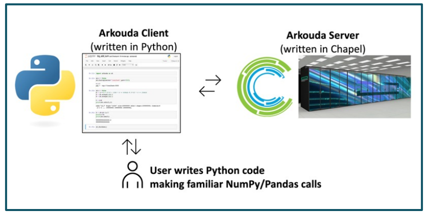
```
use BlockDist;

config const n = 10,
             m = 4;

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

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

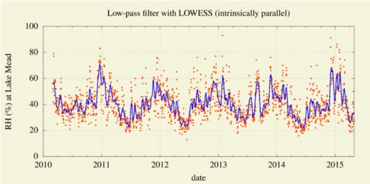


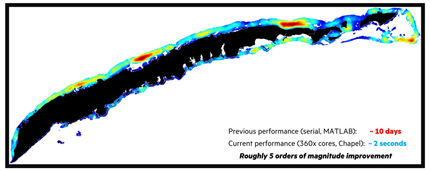
Chapel is being used for productive parallel computing at all scales

- users are reaping its benefits in practical, cutting-edge applications
- applicable to domains as diverse as physical simulations and data science
- Arkouda is a notable case, supporting interactive HPC







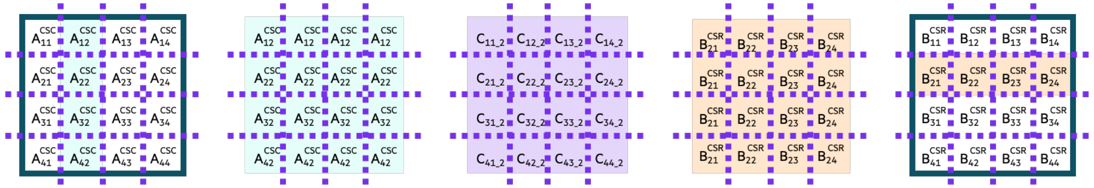


What's Cooking? What's Next?

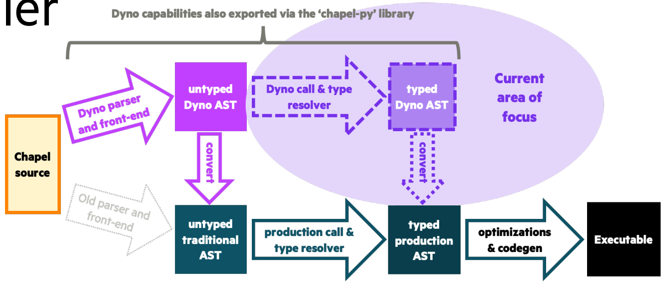
- Chapel has been accepted into the Linux Foundation's High Performance Software Foundation (HPSF)
  - moves governance of the project from HPE to the community
  - a logical next step in our open-source journey

HPSF
HIGH PERFORMANCE
SOFTWARE FOUNDATION

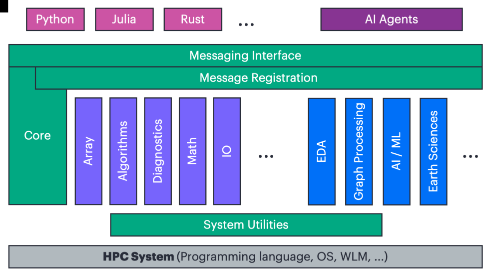
- A renewed focus on sparse matrix/array computations
  - e.g., sparse matrix-matrix multiplication



- Dyno compiler rework: Modernizing the Chapel compiler
  - faster
  - better error messages
  - support for programmer coding tools



- Honeycomb: A next-generation evolution of Arkouda
  - multi-lingual, including AI-based natural language interactions
  - user-extensible to support arbitrary computations



Ways to interact with or follow the Chapel Community

"Live" (Virtual) Community Events

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

Electronic Broadcasts

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

Social Media

FOLLOW US

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

Discussion Forums

GET IN TOUCH

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

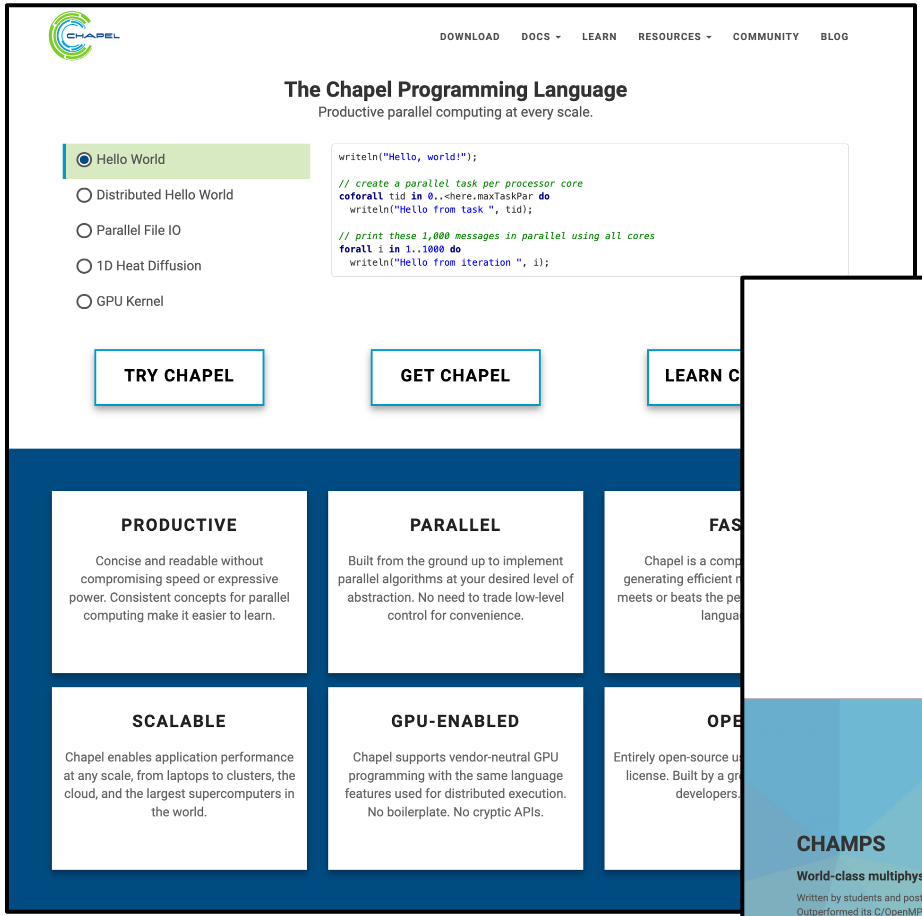
Ways to Use Chapel

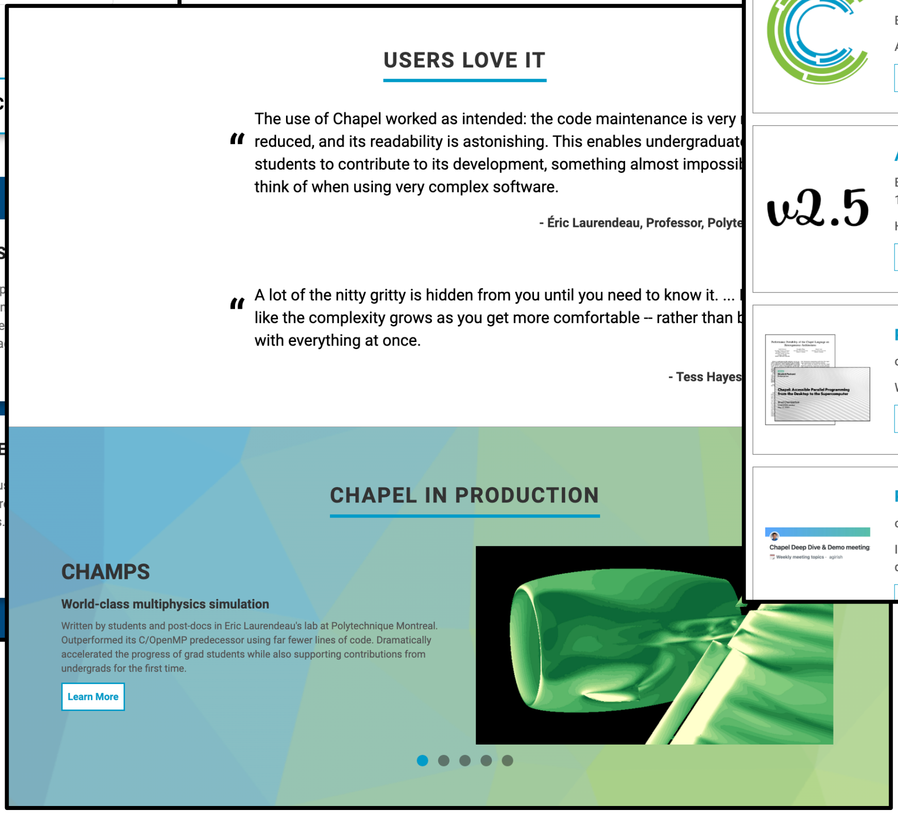
GET STARTED

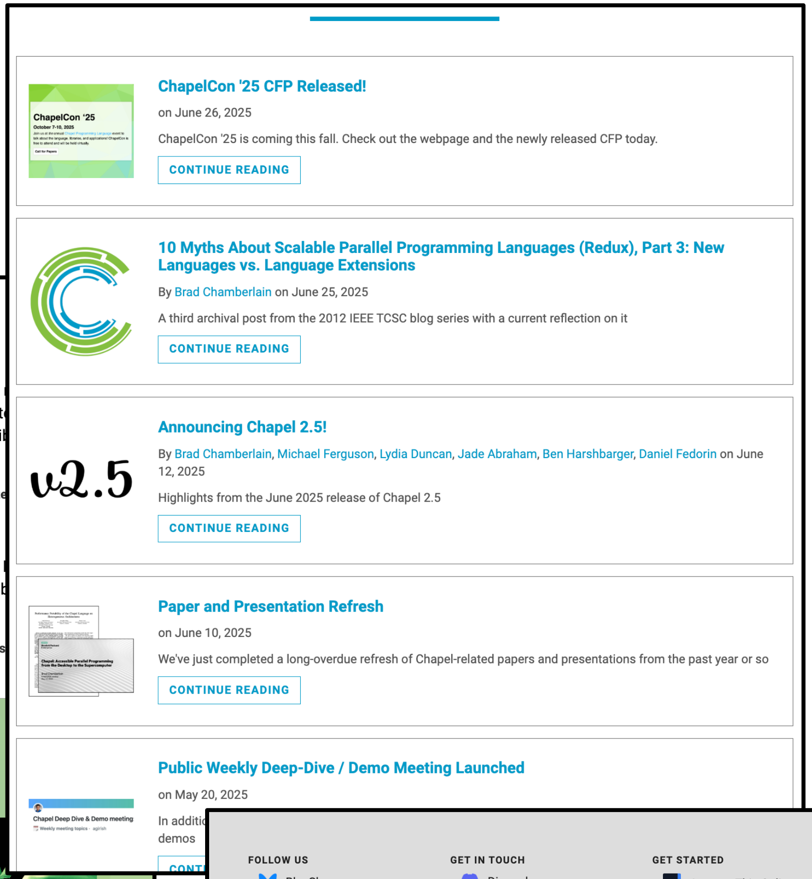
- Attempt This Online
- Docker
- E4S
- GitHub Releases
- Homebrew
- Spack

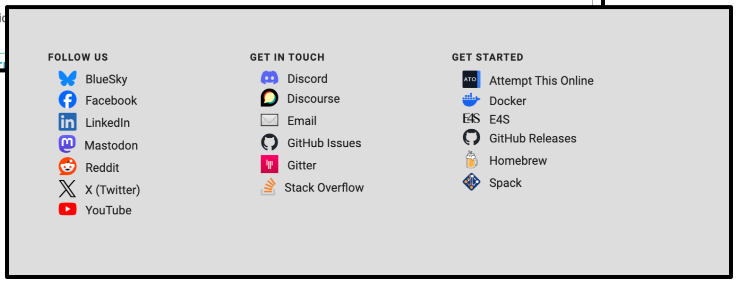
(from the footer of chapel-lang.org)

Chapel Website









chapel-lang.org

Thank you

https://chapel-lang.org
@ChapelLanguage