Hewlett Packard Enterprise

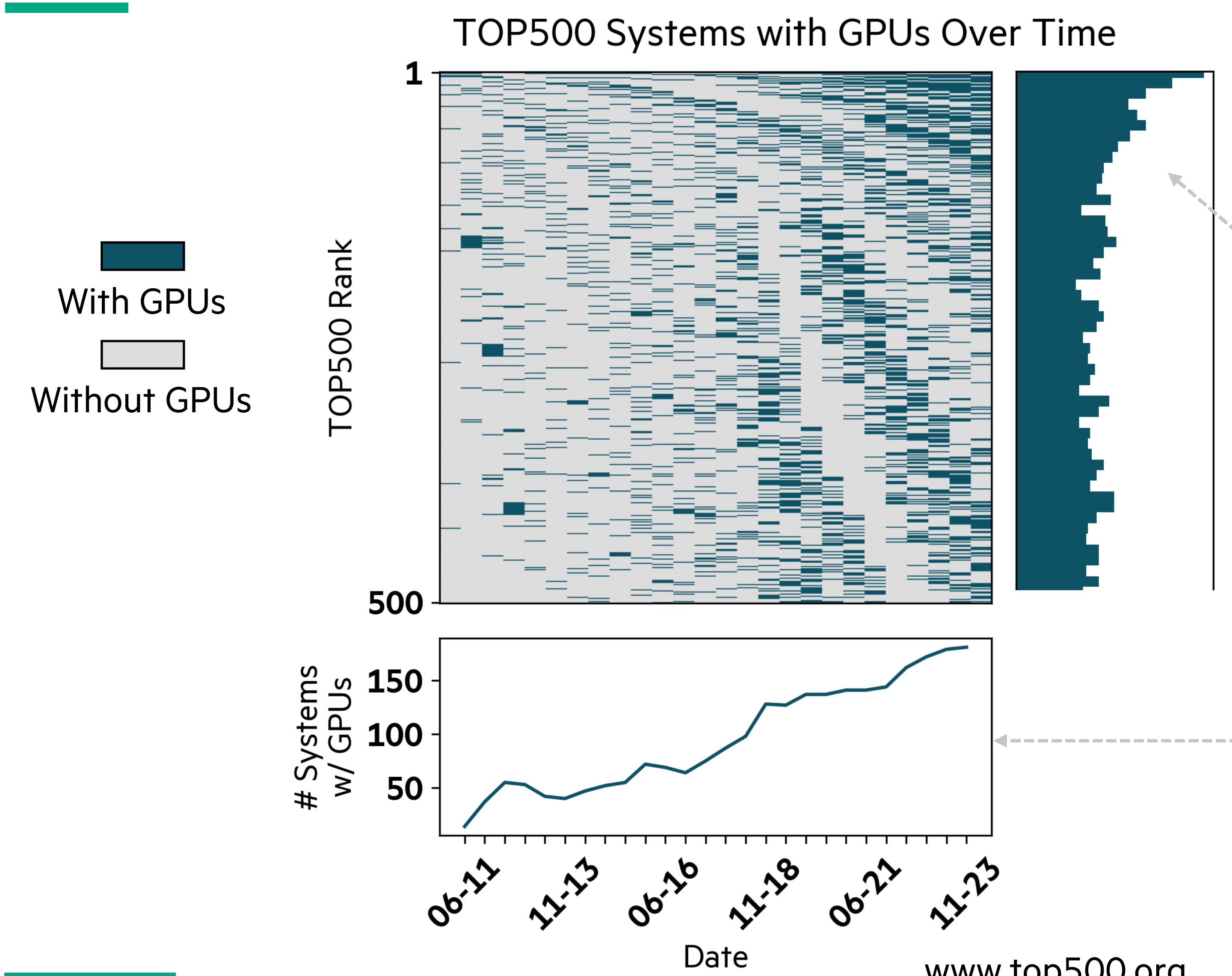
Vendor-Neutral GPU Programming In Chapel

Jade Abraham
jade.abraham@hpe.com
linkedin.com/in/jabraham17

Engin Kayraklioglu
engin@hpe.com
linkedin.com/in/engink

HPE Developer Meetup
July 31, 2024

It is Hard to Avoid GPUs



www.top500.org

41% of **top 10** have GPUs

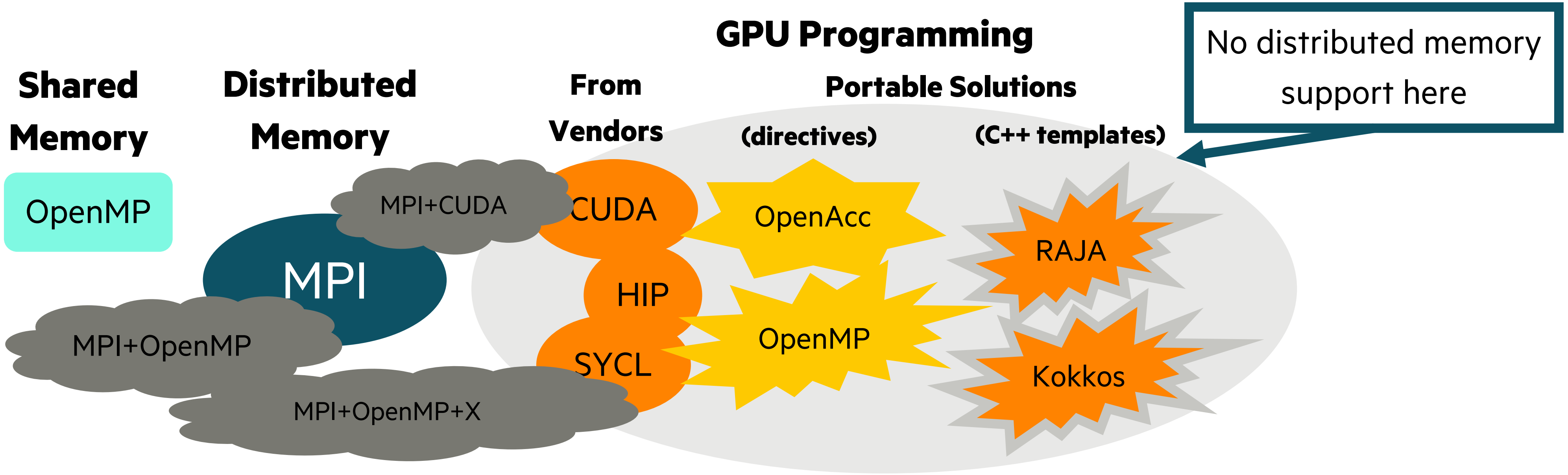
72% in the last **3 years**

27% of **top 100** have GPUs

52% in the last **3 years**

13x more systems w/GPUs

GPUs are Easy to Find… but difficult to program



All are effective, powerful, essential and tested technologies!

- … but programming for multiple nodes with GPUs appears to require at least 2 programming models
  - all of the models rely on C/C++/Fortran, which are different than the languages being taught these days
  - as a result, *using GPUs in HPC has a high barrier of entry*

Chapel is an alternative for productive distributed and shared memory GPU programming in a vendor-neutral way.

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

chapel-lang.org

What is Chapel?

Chapel works everywhere

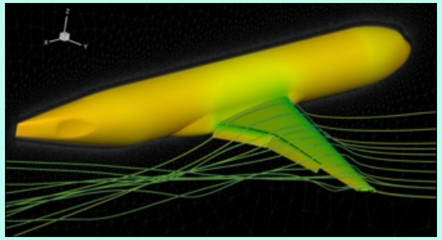
- you can develop on your laptop and have the code scale on a supercomputer
- GPUs can be targeted in a vendor-neutral way
- runs on Linux laptops/clusters, Cray systems, MacOS, WSL, AWS, Raspberry Pi
- shown to scale on Cray networks (Slingshot, Aries), InfiniBand, RDMA-Ethernet

Chapel makes distributed/shared memory parallel programming easy

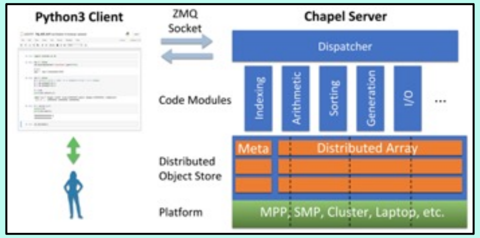
- data-parallel, locality-aware loops,
- ability to move execution and allocation to remote nodes,
- distributed arrays and bulk array operations
- different types of parallelism can be expressed with the same language features

Applications of Chapel

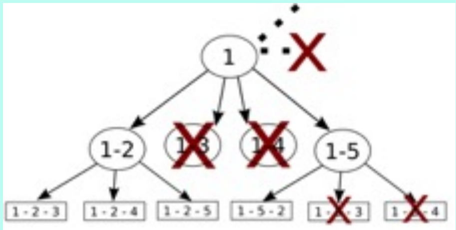
Active GPU efforts



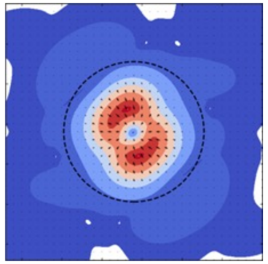
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



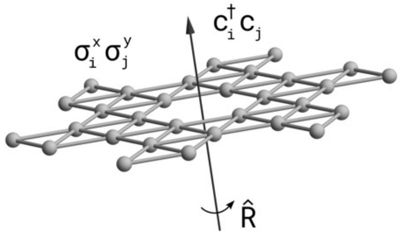
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



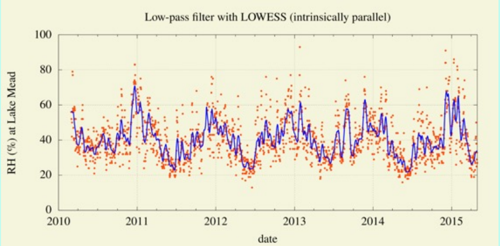
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

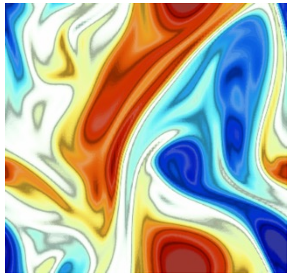


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

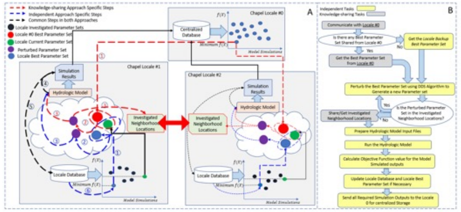


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

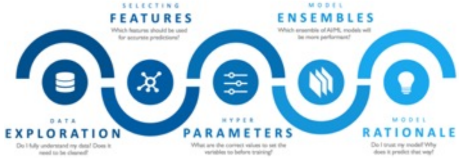
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



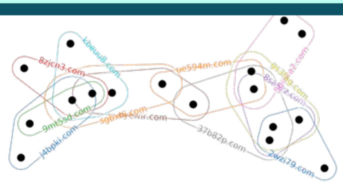
ChapQG: Layered Quasigeostrophic CFD
Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



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



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



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

?

Your Application Here?

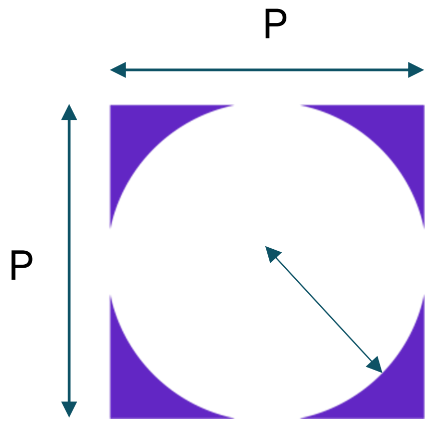
(images provided by their respective teams and used with permission)

Coral Reef Spectral Biodiversity

1. Read in a (M x N) raster image of habitat data

2. Create a (P x P) mask to find all points within a given radius.

3. Convolve this mask over the entire domain and perform a weighted reduce at each location.

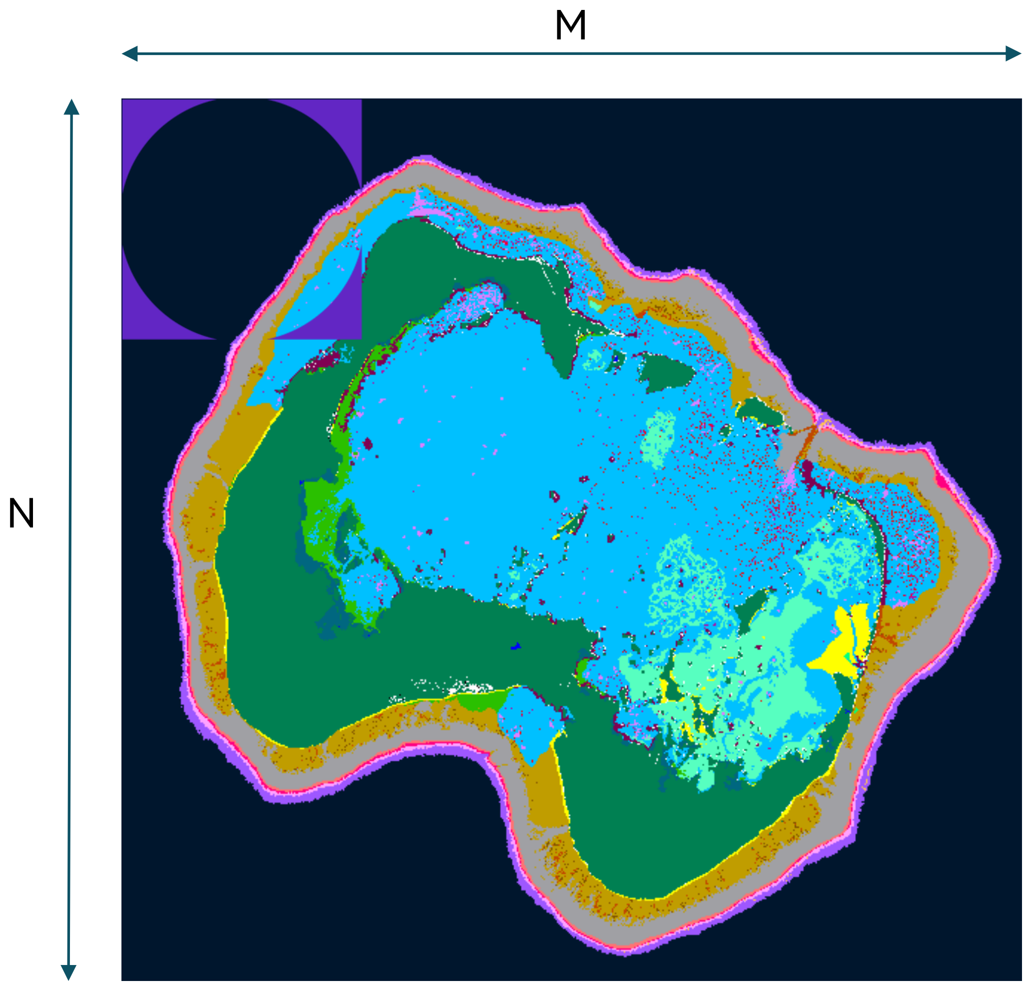


Algorithmic complexity: $O(MNP^3)$

Typically:

- M, N > 10,000

- P ~ 400



Coral Reef Spectral Biodiversity

```
proc convolve(InputArr, OutputArr) {    // 3D Input, 2D Output
 for ... {
  tonOfMath();
 }
}
proc main() {
 var InputArr: ...;
 var OutputArr: ...;

 convolve(InputArr, OutputArr);
}
```

Coral Reef Spectral Biodiversity

```
proc convolve(InputArr, OutputArr) {    // 3D Input, 2D Output
 foreach ... {
  tonOfMath();
 }
}
proc main() {
 var InputArr: ...;
 var OutputArr: ...;

 coforall loc in Locales do on loc {          // use all nodes in parallel...
  coforall gpu in here.gpus do on gpu {       // using GPUs on this node in parallel...
   coforall task in 0..#numWorkers {          // using numWorkers on this GPU in parallel.
    var MyInputArr = InputArr[...];
    var MyOutputArr: ...;
    convolve(MyInputArr, MyOutputArr);
    OutputArr[...] = MyOutputArr;
}}}}
```

Using a different loop flavor to enable GPU execution.

Multi-node, multi-GPU, multi-thread parallelism are expressed using the same language constructs.

High-level, intuitive array operations work across nodes and/or devices

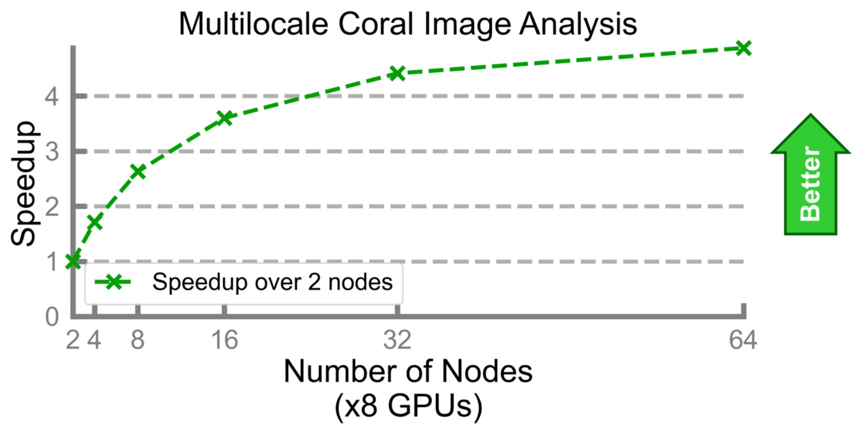
Coral Reef Spectral Biodiversity

```
proc convolve(InputArr, OutputArr) {   // 3D Inp
  foreach ... {
    tonOfMath();
  }
}
proc main() {
  var InputArr: ...;
  var OutputArr: ...;

  coforall loc in Locales do on loc {        // u
    coforall gpu in here.gpus do on gpu {   // u
      coforall task in 0..#numWorkers {  // using pa
        var MyInputArr = InputArr[...];
        var MyOutputArr: ...;
        convolve(MyInputArr, OutputArr);
        OutputArr[...] = MyOutputArr;
}}}}
```

Runs on Frontier!

- 5x improvement going from 2 to 64 nodes
  - (from 16 to 512 GPUs)
- Straightforward code changes:
  - from sequential Chapel code
  - to GPU-enabled one
  - to multi-node, multi-GPU, multi-thread



- Scalability improvements coming soon!

What We Will Discuss Today

- An overview of parallelism and locality concepts in Chapel
- A live demo showcasing GPU capabilities
- Stories from the Chapel community

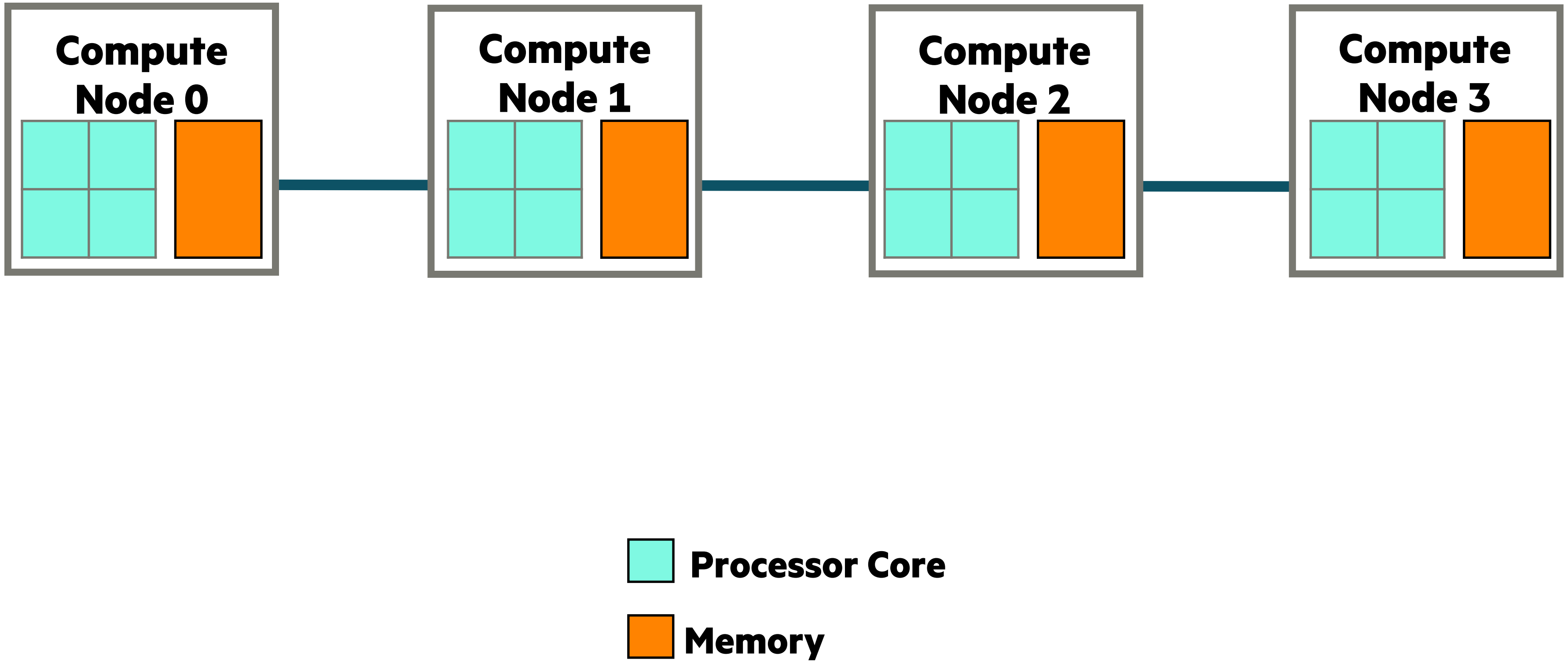
What we will not discuss today:

- Comprehensive list of Chapel features
  - (important ones will be covered)
- How GPU support is implemented
  - (happy to go over some backup slides, if there's interest)
- Everything you can do with GPUs using Chapel
  - (there's only so much time ☺)

GPU Programming in Chapel

Locales in Chapel

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

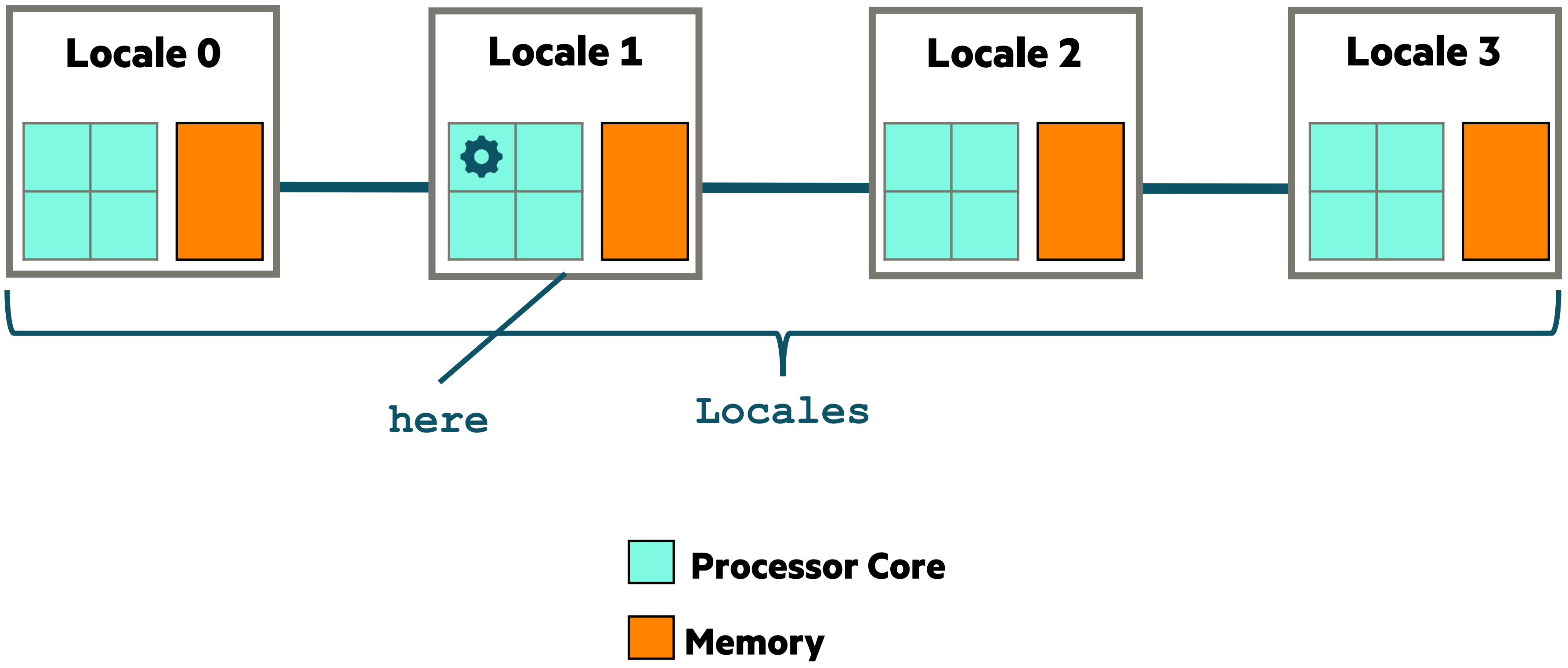


Key Built-In Types and Variables Related to Locales

`locale`: A type that represents system resources on which the program can run

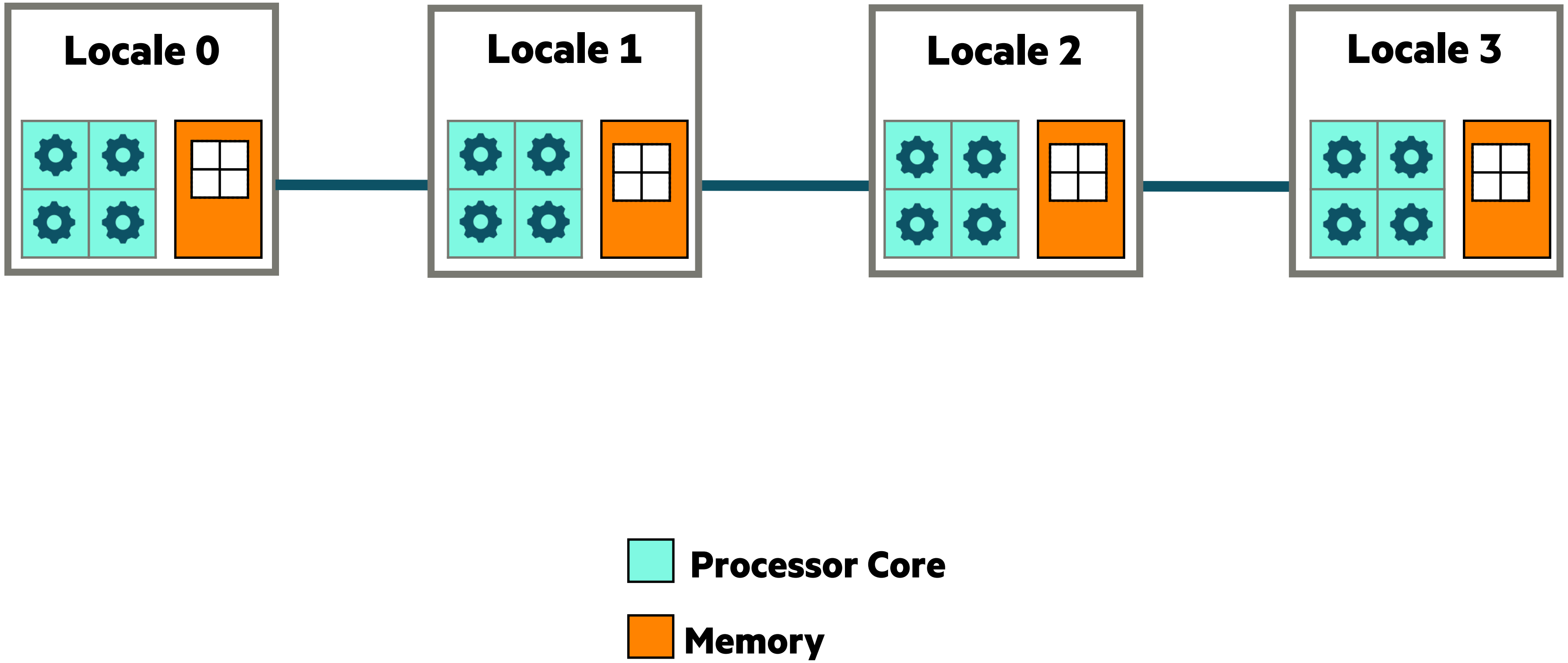
`Locales`: An array of `locale` values

`here` : The `locale` on which the current task is executing



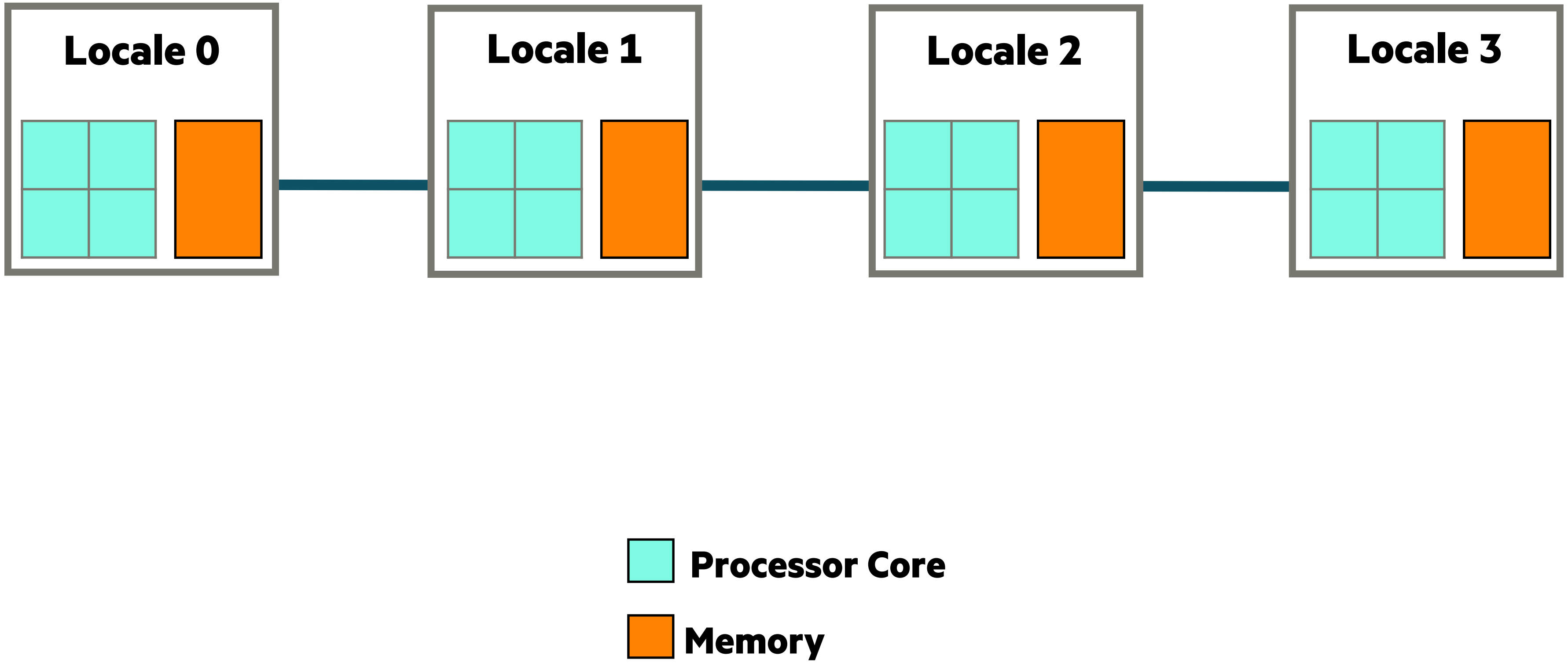
Key Concerns for Scalable Parallel Computing

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



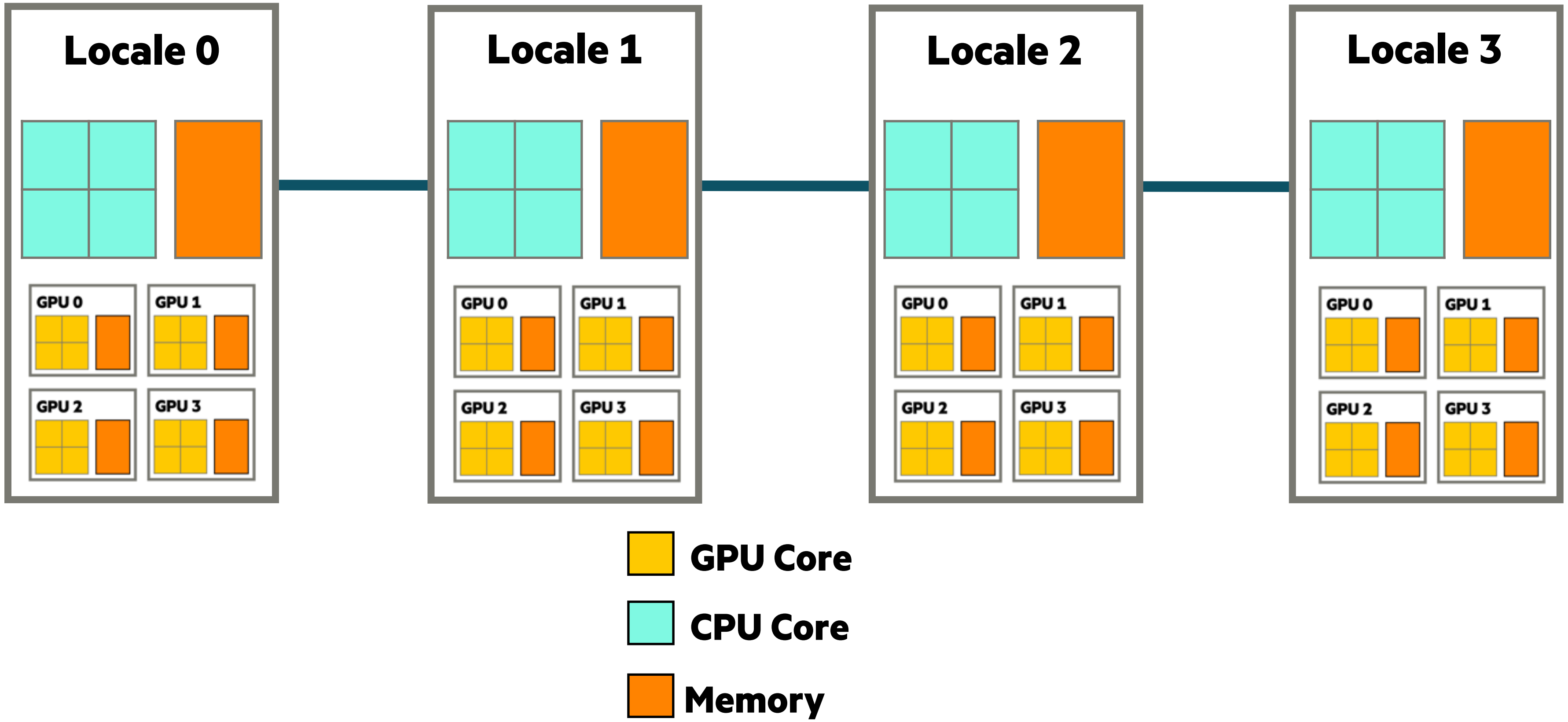
Key Concerns for Scalable Parallel Computing

1. **parallelism:** Which tasks should run simultaneously?
2. **locality:** Where should tasks run? Where should data be allocated?
   - complicating matters, compute nodes now often have GPUs with their own processors and memory



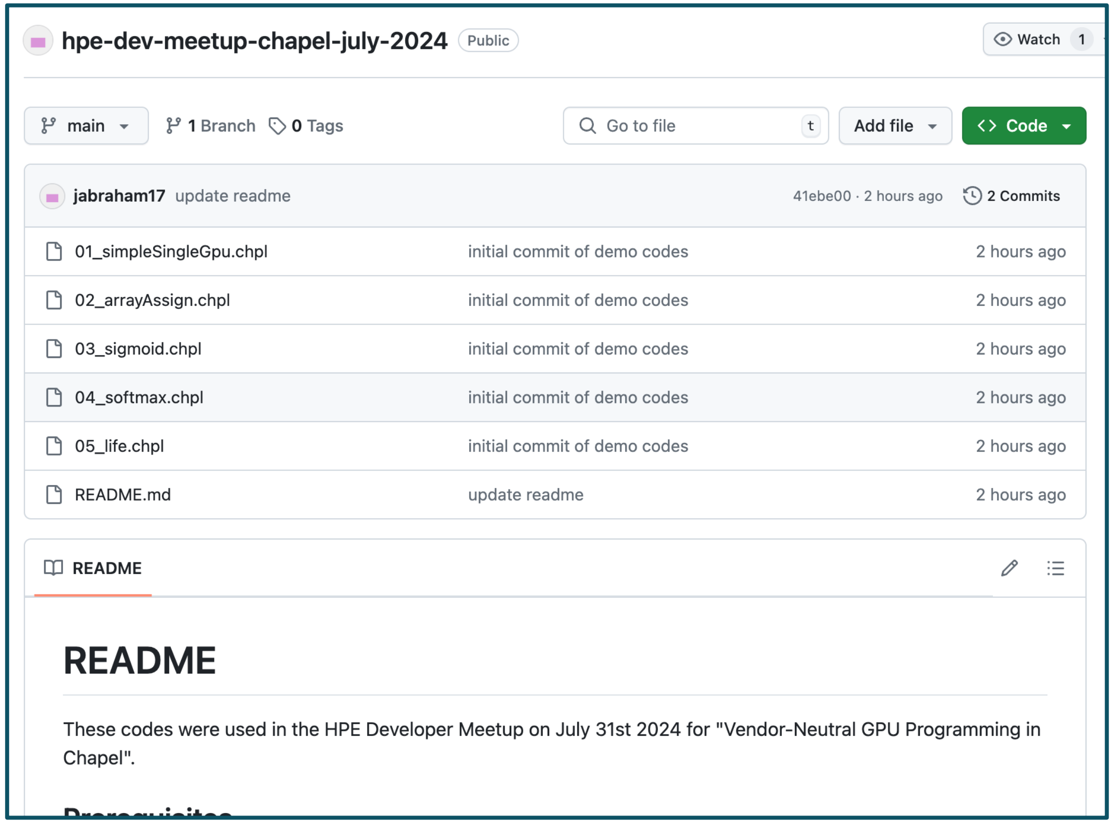
Key Concerns for Scalable Parallel Computing

1. **parallelism:** Which tasks should run simultaneously?
2. **locality:** Where should tasks run? Where should data be allocated?
   - complicating matters, compute nodes now often have GPUs with their own processors and memory
   - we represent these as *sub-locales* in Chapel



Live Demo

Example Codes Are Available



https://github.com/jabraham17/hpe-dev-meetup-chapel-july-2024

Stories From The Chapel Community

Chapel Performance on Different GPU and CPUs

- Comparing Chapel's performance
  …against OpenMP, Kokkos, CUDA and HIP
  …on different GPU and CPUs
  …using BabelStream, miniBUDE and TeaLeaf

- Recently presented at
  - Heterogeneity in Computing Workshop (HCW)
  - In conjunction with IPDPS

Performance Portability of the Chapel Language on Heterogeneous Architectures

Josh Milthorpe
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
Australian National University
Canberra, Australia
ORCID: 0000-0002-3588-9896

Xianghao Wang
Australian National University
Canberra, Australia

Ahmad Azizi
Australian National University
Canberra, Australia

***Abstract*—A performance-portable application can run on a variety of different hardware platforms, achieving an acceptable level of performance without requiring significant rewriting for each platform. Several performance-portable programming models are now suitable for high-performance scientific application development, including OpenMP and Kokkos. Chapel is**

other heterogeneous programming models that allow single-source programming for diverse hardware platforms.

We seek to answer the question: how well does Chapel support the development of *performance-portable* application codes compared to more widely-used programming models

Paper is available at milthorpe.org/wp-content/uploads/2024/03/Milthorpe_HCW2024.pdf

miniBUDE

- Proxy for BUDE (a protein docking simulation)
  - The computation is very arithmetically intensive and makes significant use of trigonometric functions

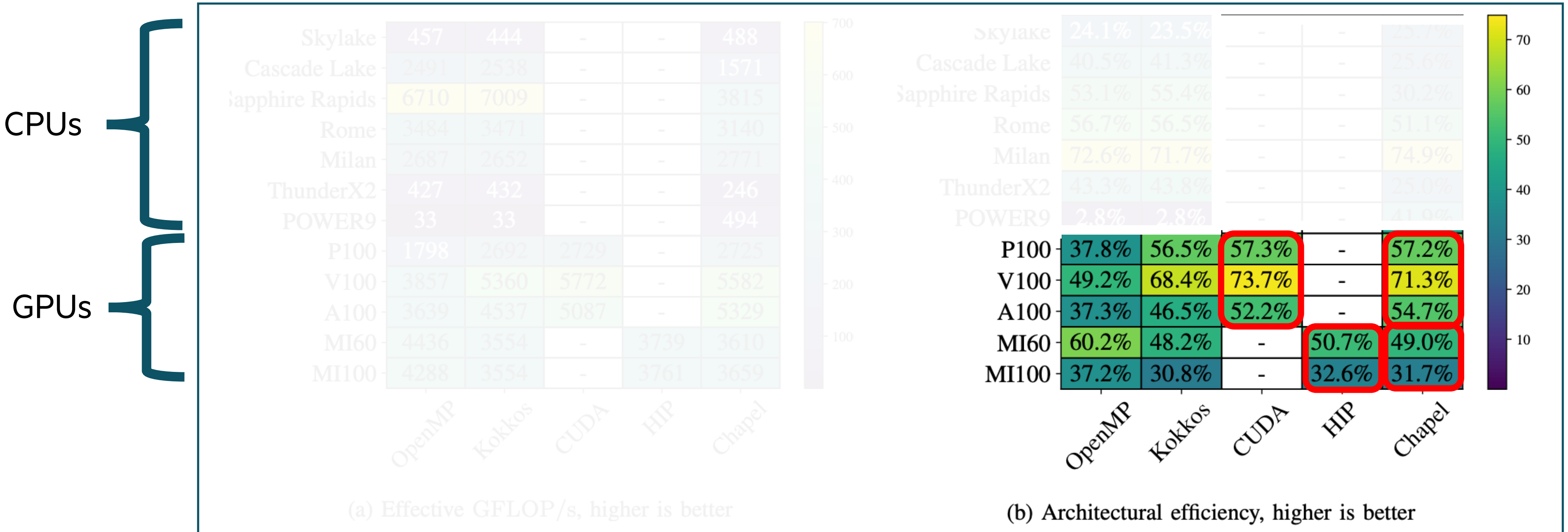


Fig. 2: miniBUDE results for small deck bm1

Figure from: "Performance Portability of the Chapel Language on Heterogeneous Architectures". Josh Milthorpe (Oak Ridge National Laboratory, Australian National University), Xianghao Wang (Australian National University), Ahmad Azizi (Australian National University) Heterogeneity in Computing Workshop (**HCW**)

Native GPU Programming in Chapel at Scale

- Comparing Chapel's native GPU programming
  …against interoperability with HIP and CUDA
  …on Frontier and Perlmutter
  …using N-Queens as proxy for combinatorial optimization

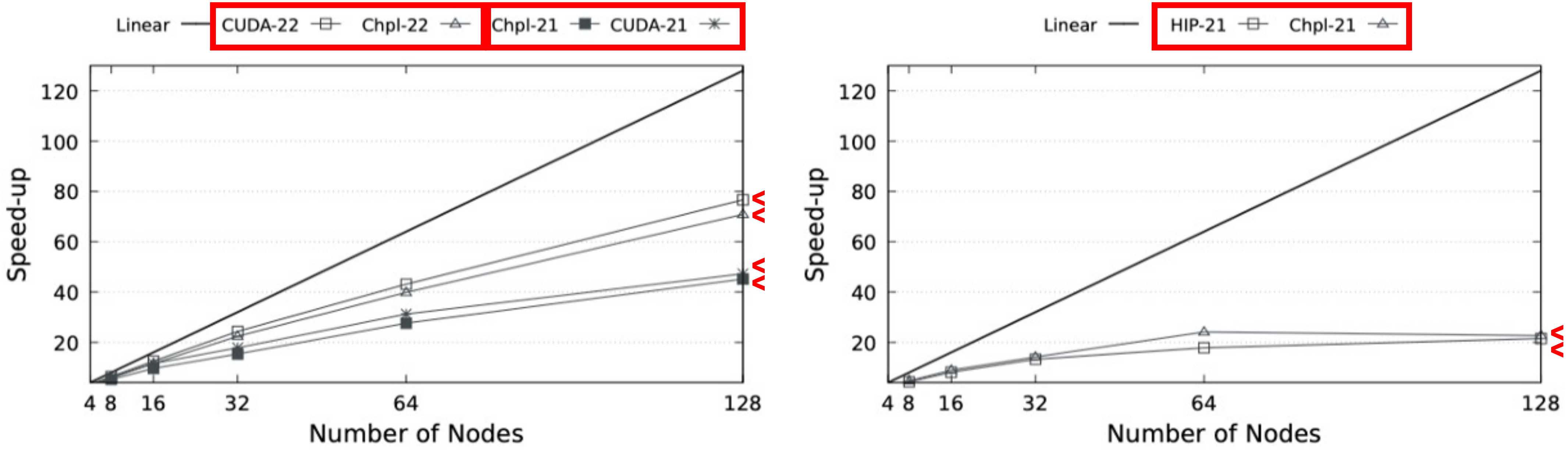
- To be presented at Euro-Par 2024
  - 26-30 August
  - Madrid, Spain

Investigating Portability in Chapel for Tree-based Optimization on GPU-powered Clusters

Tiago Carneiro[1][0000−0002−6145−8352], Engin Kayraklioglu[2][0000−0002−4966−3812], Guillaume Helbecque[3,4][0000−0002−8697−3721], and Nouredine Melab[4]

[1] Interuniversity Microelectronics Centre (IMEC), Belgium
tiago.carneiropessoa@imec.be
[2] Hewlett Packard Enterprise, USA
engin@hpe.com
[3] University of Luxembourg, Luxembourg
guillaume.helbecque@uni.lu
[4] Université de Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, France
nouredine.melab@univ-lille.fr

Native GPU Programming in Chapel at Scale



(a) NVIDIA-based System

(b) AMD-based system

Figure from: "Investigating Portability in Chapel for Tree-Based Optimizations on GPU-powered Clusters". Tiago Carneiro, Engin Kayraklioglu, Guillaume Helbecque, Nouredine Melab Europar 2024

Keynote at ChapelCon '24

A Case for Parallel-First Languages in Post-Serial, Accelerated World

Paul Sathre, Virginia Tech

ChapelCon '24 Keynote: A Case for Parallel-First Languages in a Post-Serial, Accelerated World

Chapel Parallel Programming Language
356 subscribers
Subscribe
12 Share Download Clip

Slides and recording are available on
chapel-lang.org/ChapelCon24.html#keynote

Summary

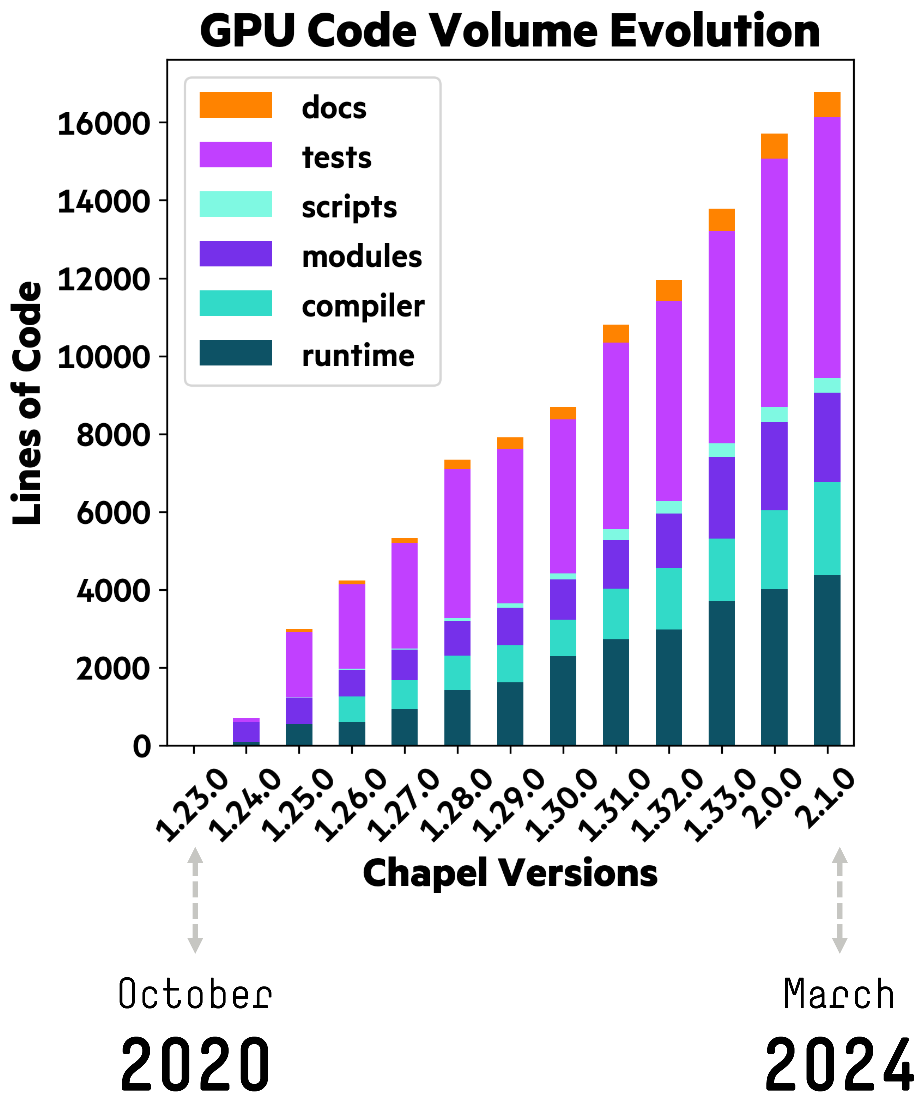
Where We Are Today

Over ~3 years we have been steadily improving

- NVIDIA, AMD GPUs are supported
- Multiple nodes with multiple GPUs can be used
- Parallel tasks can use GPUs concurrently
- GPU features can be emulated on CPUs

Mature enough to get started, big efforts are still underway

- Distributed arrays
- Intel support
- Improving language features to support GPU programming
- Performance improvements
- Bug fixes



Ongoing AI/ML Efforts

Chapel Tensor Library (github.com/Iainmon/gputil)

- PyTorch like interface for tensor operations:
  - Used for inference using NVIDIA and AMD GPUs
  - Builds up composable network layers like PyTorch
  - Supports loading in pretrained models from PyTorch
  - Tracks computational graph, supports backpropagation
- Ongoing effort is to support PyTorch interoperability and multi-locale inference

llm.chpl *Stay tuned!*

- A port of llm.c in Chapel (https://github.com/karpathy/llm.c)
- Even shorter, even more parallel implementation of GPT-2
  - Chapel's multidimensional arrays make implementation much simpler

MLPerf *Stay tuned!*

- We are actively looking into porting some MLPerf benchmarks in Chapel
- If interested, please reach out!

If You Want to Learn More About GPU Programming in Chapel

GPU Programming Blog Series: chapel-lang.org/blog/series/gpu-programming-in-chapel/

Introduction to GPU Programming in Chapel

Posted on January 10, 2024.

Tags: GPU Programming | How-To

By: Daniel Fedorin

Chapel's High-Level Support for CPU-GPU Data Transfers and Multi-GPU Programming

Posted on April 25, 2024.

Tags: GPU Programming | How-To

By: Engin Kayraklioglu

Technote: https://chapel-lang.org/docs/main/technotes/gpu.html

- Anything and everything about our GPU support
  – configuration, advanced features, links to some tests, caveats/limitations
- More of a reference manual than a tutorial

Previous talks

- **LinuxCon / Open Source Summit North America 2024 Talk:** GPU Programming in Chapel and a Live Demo
  – https://youtu.be/5-jLdKduaJE?si=ezaz5mDORvmTjgRL
- **CHIUW '23 Talk:** updates from May '22-May '23 period
  – https://chapel-lang.org/CHIUW/2023/KayrakliogluSlides.pdf
- **LCPC '22 Talk:** a lot of details on how the Chapel compiler works to create GPU kernels
  – https://chapel-lang.org/presentations/Engin-SIAM-PP22-GPU-static.pdf

Chapel is Open Source, Get Involved!

Check out "GPU Support" issues to contribute/report bugs

Join the discussion on Discourse

Join the discussion on Gitter

Try Chapel on GitHub Codespaces

github.com/chapel-lang/chapel-hello-world

See many other ways of trying Chapel

chapel-lang.org/download.html

GPU support coming soon!

Other Chapel Resources

Chapel homepage: https://chapel-lang.org

- (points to all other resources)

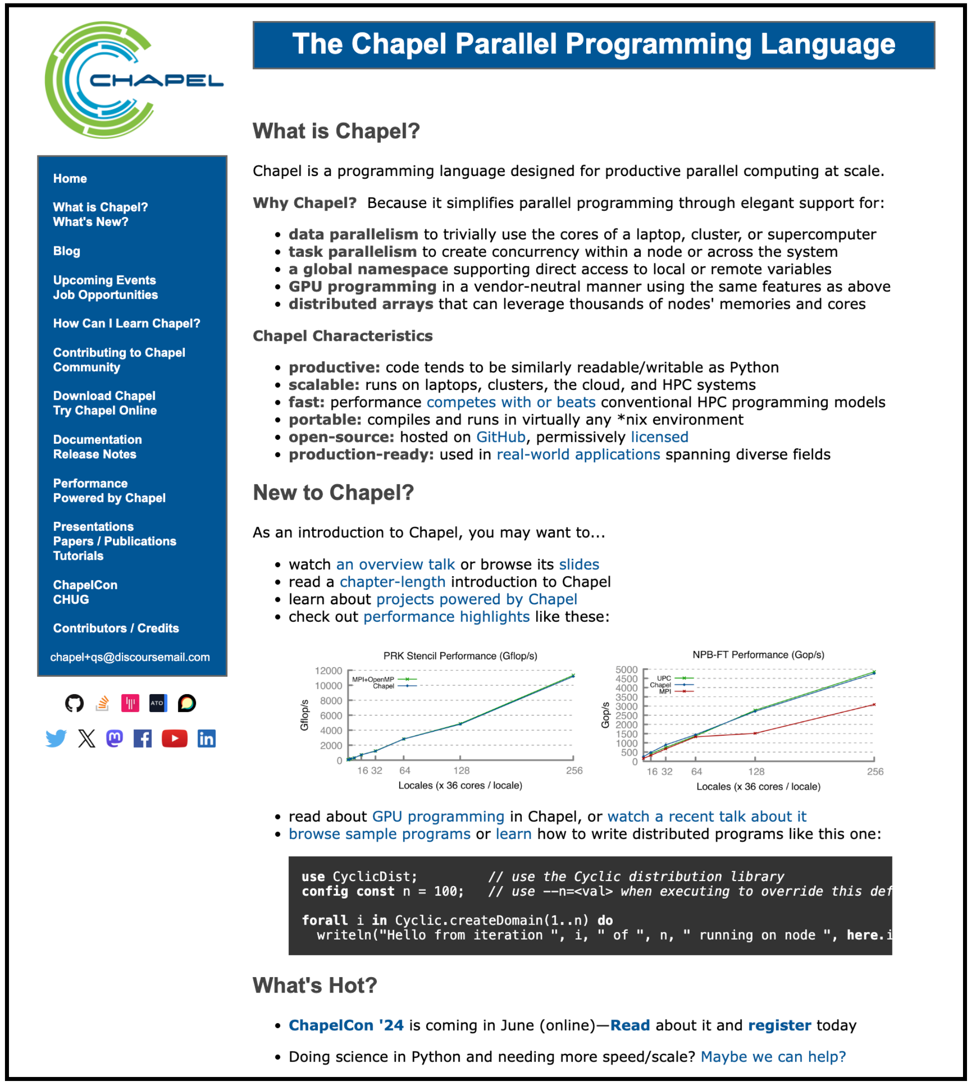
Blog: https://chapel-lang.org/blog/

Social Media:

- Facebook: @ChapelLanguage
- LinkedIn: ChapelLanguage
- Mastodon: @ChapelProgrammingLanguage
- X / Twitter: @ChapelLanguage
- YouTube: @ChapelLanguage

Community Discussion / Support:

- Discourse: https://chapel.discourse.group/
- Gitter: https://gitter.im/chapel-lang/chapel
- Stack Overflow: https://stackoverflow.com/questions/tagged/chapel
- GitHub Issues: https://github.com/chapel-lang/chapel/issues



Closing Thoughts

GPUs are becoming more common but programming them isn't getting easier.

- C/C++/Fortran are not good starting points for many potential users.
- GPU capability (and parallelism in general) is typically achieved by additional frameworks.

Parallel programming, GPUs and HPC should be more accessible.

- There are many potential use cases in different fields, including social sciences.
- Making GPUs more accessible (and parallelism in general) accelerates progress.

chapel-lang.org

Chapel makes parallel programming, GPUs and HPC more accessible.

- Existing applications prove that Chapel delivers on the promise.
- Its GPU support makes Chapel an all-inclusive solution for parallel programming.