



Hewlett Packard
Enterprise

ChapelCon 2024: State of the Chapel Project

Brad Chamberlain

June 7, 2024

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 2.0

Chapel 2.0 has been released!

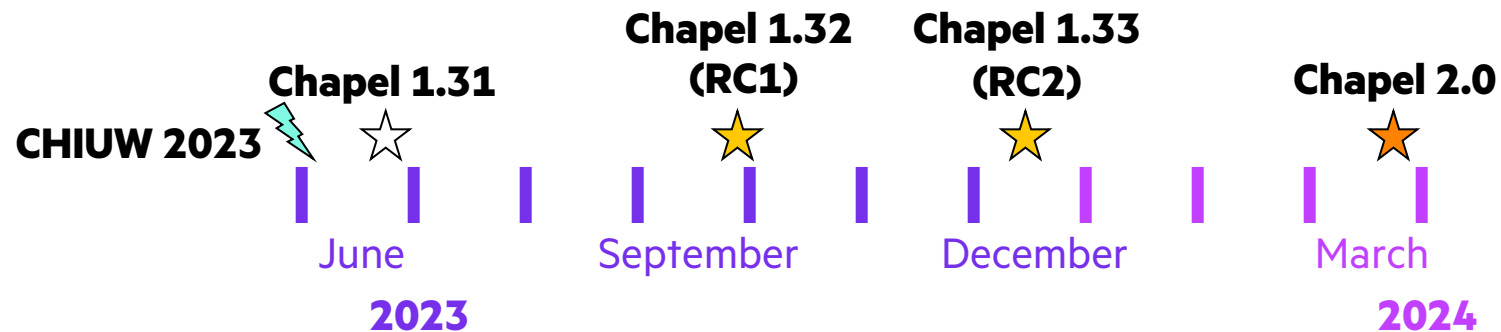
What is Chapel 2.0?

- A milestone release!
- Stabilizes core language and library features
 - these features should not have breaking changes in the future
- **Released:** March 21, 2024
- Chapel 1.32/1.33 served as release candidates

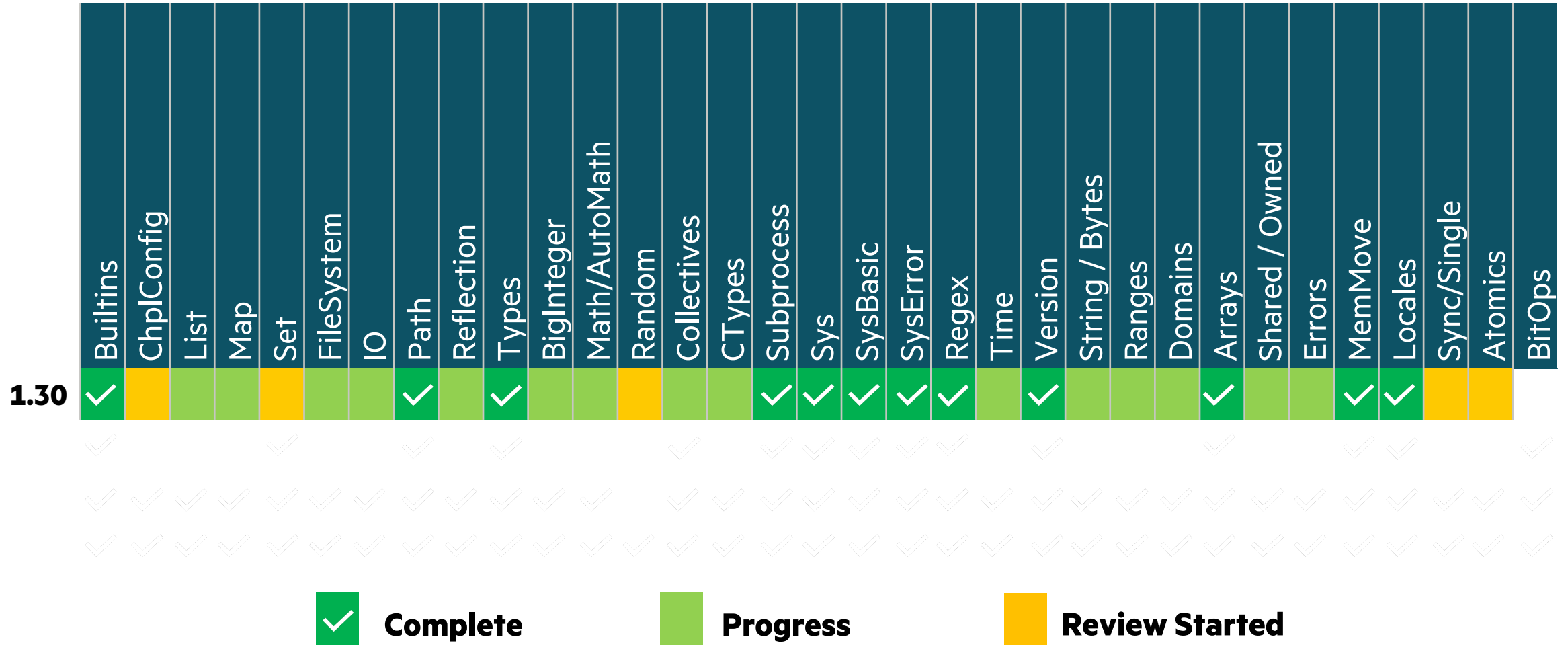


The screenshot shows a blog post from the Chapel Language Blog. The title is "Chapel 2.0: Scalable and Productive Computing for All", posted on March 21, 2024, by Daniel Fedorin. The post includes a "Table of Contents" with links to sections like "What Are People Doing With Chapel?", "Arkouda: Interactive Data Analysis at Scale", "CHAMPS: A Framework for Computational Fluid Dynamics", and "Coral Biodiversity Computation". The main text discusses the team's excitement about the release and highlights improvements in usability and performance, as well as the stability of the core language and library features.

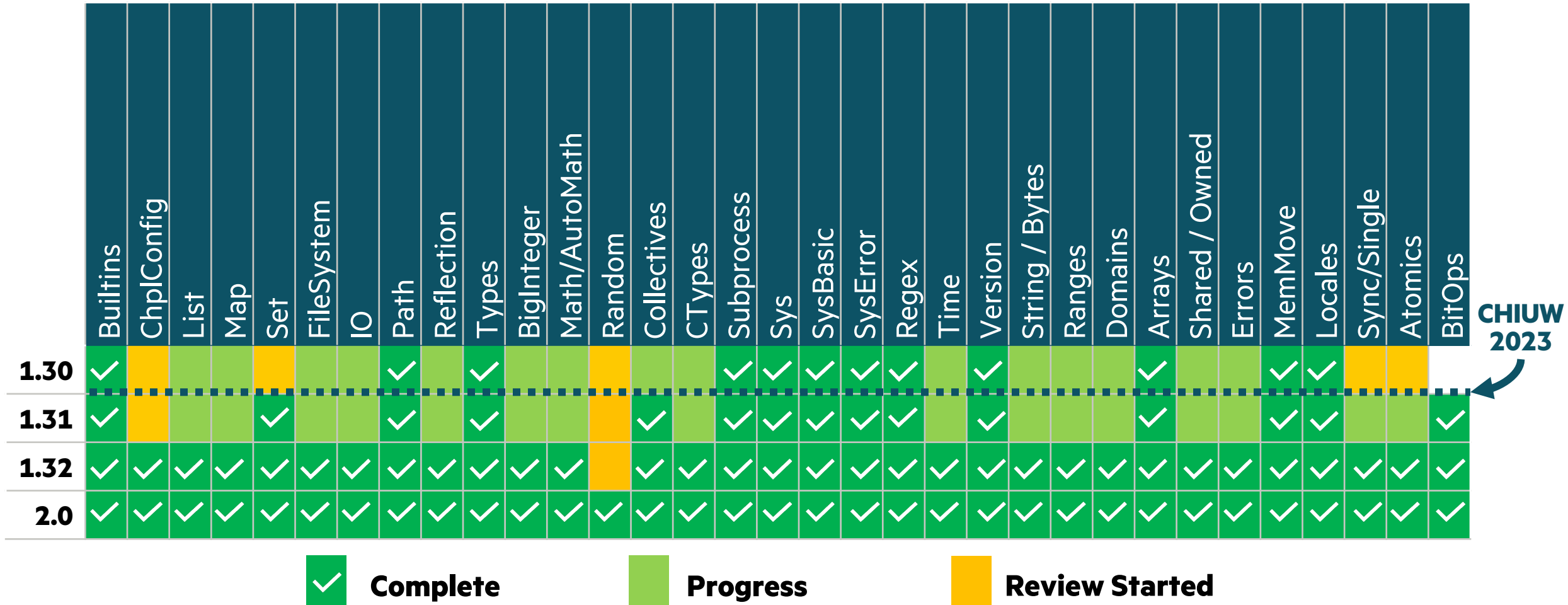
<https://chapel-lang.org/blog/posts/announcing-chapel-2.0/>



Chapel 2.0 Library Stabilization: Status as of CHI UW 2023



Chapel 2.0 Library Stabilization: Progress since CHI UW 2023



CHI UW 2023



Chapel 2.0 Language Stabilization: Highlights Since CHI UW 2023

Language:

- Visibility of generic types
- Handling of numeric types, including generics
- ‘range’/ ‘domain’/array/distribution improvements
- ‘string’/‘bytes’/‘c_string’ stabilization
- ‘sync’/‘single’ stabilization
- Intents: defaults for arrays, return/yield intents, ...
- Protection of special method names
- Lifetimes of temporaries
- Classes: casting, lifetime management, ...
- Implicit ‘param’ to ‘const ref’ conversions
- Made ‘serial’ statements unstable
- Marked ‘local’ statements unstable
- ...



Chapel 2.0: What's Next?

- **Chapel 2.1:** scheduled for June 27
- **Continue stabilizing features**
 - Prioritize those identified by users / developers
 - e.g., ‘Sort’ module, ‘dmapped’ keyword, ...
 - use ‘--warn-unstable’ to identify unstable features
- **Establish means of making future changes**
 - e.g., create a Chapel language advisory board?



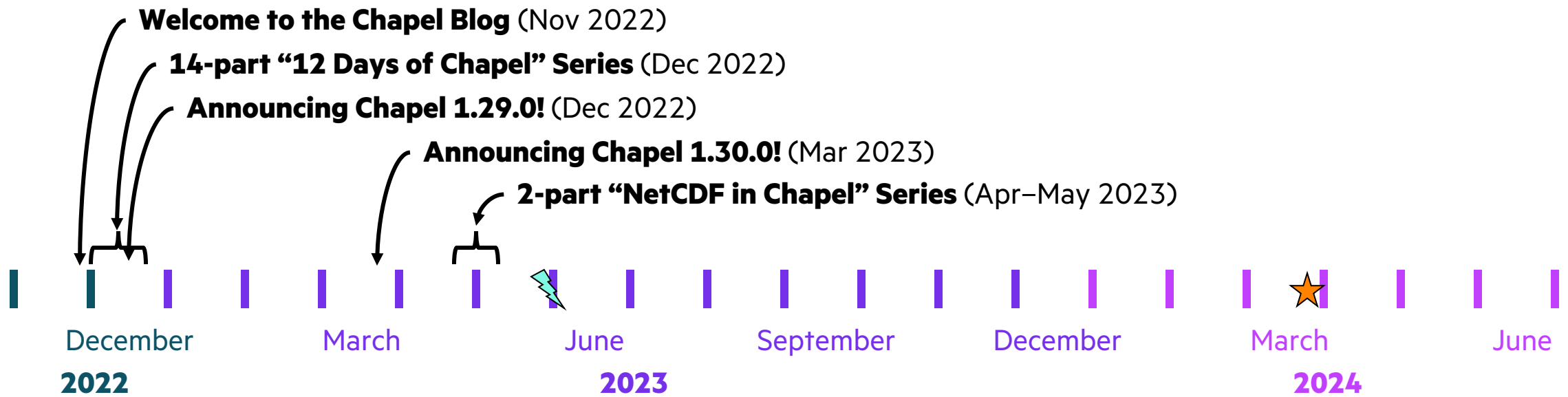
Community Focus

Focus on the Chapel Community

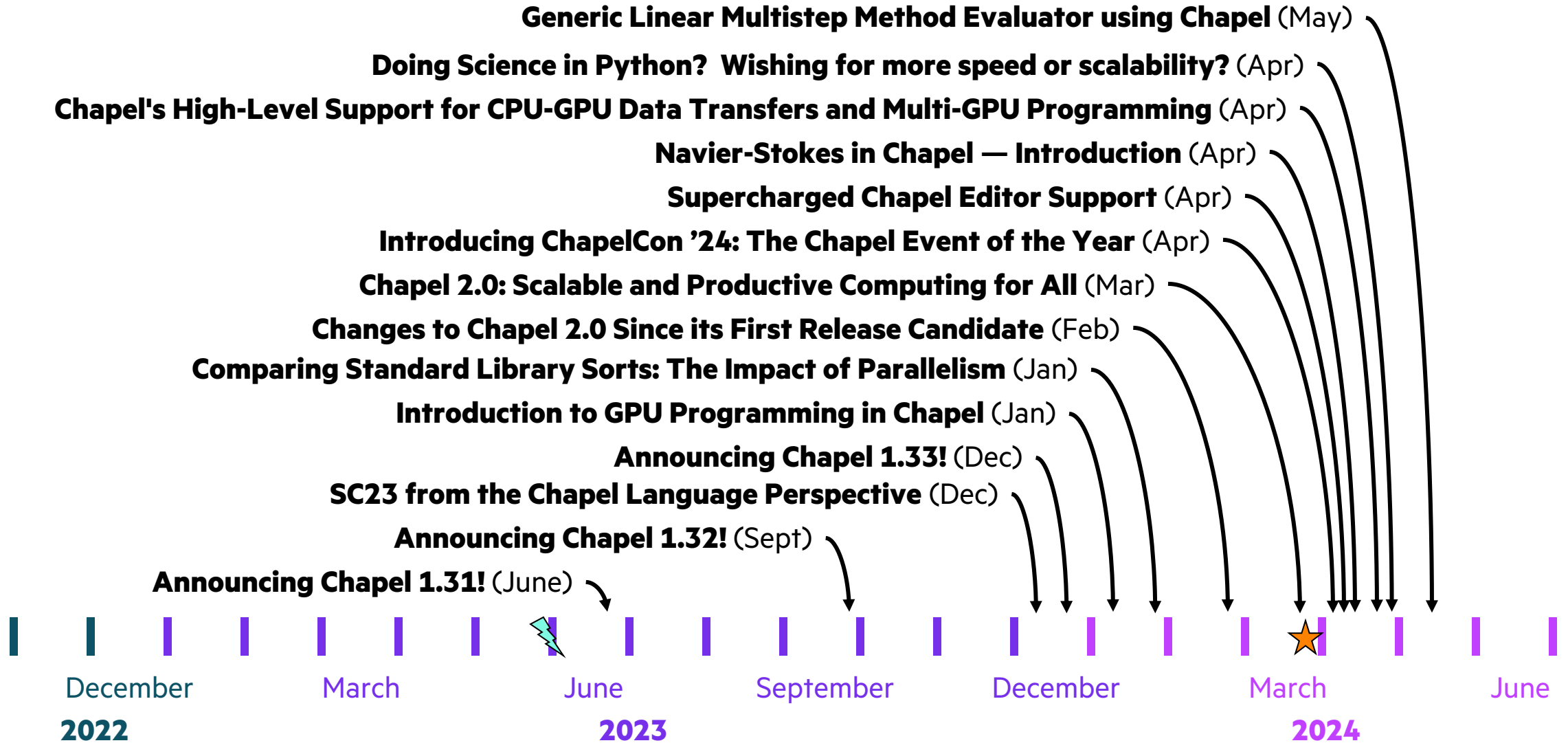
- With 2.0 wrapped up, we have shifted focus toward nurturing the Chapel community
 - Supporting existing users
 - Seeking out new use-cases and users
 - Amplifying our message about Chapel
- This has become our new “all-hands” activity
- Since CHI UW 2023, we have hired our first-ever community manager, Sarah Coghlan
- We’ve also kicked off several initiatives with this community focus in mind:
 - Rebranding CHI UW to **ChapelCon**, expanding its scope and format
 - Renewed focus on resolving **user GitHub issues**
 - 140 closed since CHI UW 2023
 - 88 closed the year prior to that
 - Plus...



Chapel Blog: Articles prior to CHI UW 2023

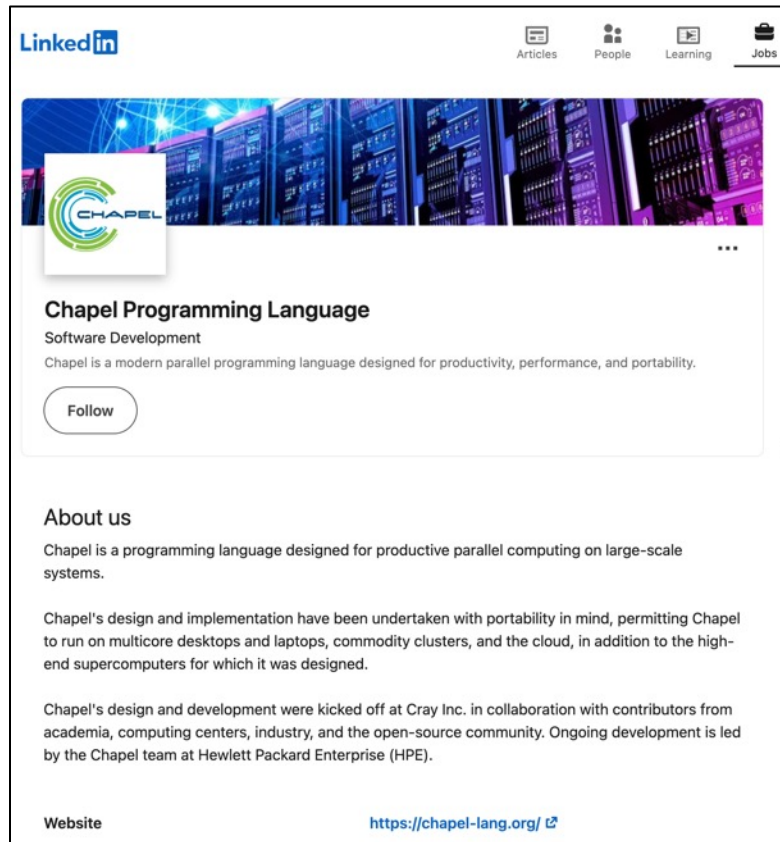


Chapel Blog: Articles since CHI UW 2023

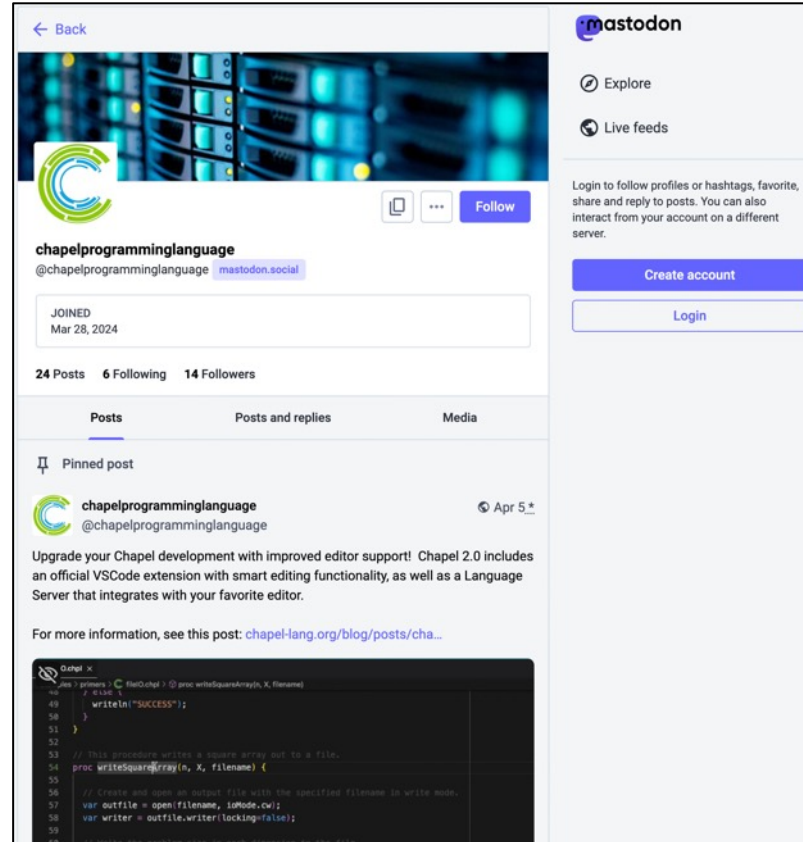


Chapel Social Media

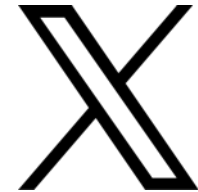
- As with the blog, we're striving to improve the frequency of our social media posts
 - Goal: a tweet every day or two during the work-week
- Recently launched LinkedIn and Mastadon project accounts—follow us there!




The screenshot shows the LinkedIn profile for 'Chapel Programming Language'. The profile picture is the Chapel logo, a green 'C' with a circular arrow. The banner image shows server racks. The profile name is 'Chapel Programming Language' with the industry 'Software Development'. The bio states: 'Chapel is a modern parallel programming language designed for productivity, performance, and portability.' There is a 'Follow' button. The 'About us' section describes Chapel as a programming language for productive parallel computing on large-scale systems, mentioning its design for portability and its development at Cray Inc. in collaboration with academia, industry, and the open-source community. A website link 'https://chapel-lang.org/' is provided at the bottom.



The screenshot shows the Mastadon profile for '@chapelprogramminglanguage'. The profile picture is the Chapel logo. The bio includes the handle '@chapelprogramminglanguage' and the server 'mastodon.social'. It shows 'JOINED Mar 28, 2024', '24 Posts', '6 Following', and '14 Followers'. A pinned post from April 5th is visible, with the text: 'Upgrade your Chapel development with improved editor support! Chapel 2.0 includes an official VSCode extension with smart editing functionality, as well as a Language Server that integrates with your favorite editor.' Below the text is a code snippet from a terminal window showing a successful write operation.



Website Redesign



The Chapel Parallel Programming Language

What is Chapel?

Chapel is a programming language designed for productive parallel computing at scale.

Why Chapel? Because it simplifies parallel programming through elegant support for:

- **data parallelism** to trivially use the cores of a laptop, cluster, or supercomputer
- **task parallelism** to create concurrency within a node or across the system
- **a global namespace** supporting direct access to local or remote variables
- **GPU programming** in a vendor-neutral manner using the same features as above
- **distributed arrays** that can leverage thousands of nodes' memories and cores

Chapel Characteristics

- **productive:** code tends to be similarly readable/writable as Python
- **scalable:** runs on laptops, clusters, the cloud, and HPC systems
- **fast:** performance *competes with or beats* conventional HPC programming models
- **portable:** compiles and runs in virtually any *nix environment
- **open-source:** hosted on [GitHub](#), permissively [licensed](#)
- **production-ready:** used in *real-world applications* spanning diverse fields

New to Chapel?

As an introduction to Chapel, you may want to...

- watch an [overview talk](#) or browse its [slides](#)
- read a [chapter-length introduction](#) to Chapel
- learn about [projects powered by Chapel](#)
- check out [performance highlights](#) like these:

PRK Stencil Performance (Gflop/s)

Locales (x 36 cores / locale)	MPI+OpenMP	Chapel
16	~1000	~1000
32	~2000	~2000
64	~3000	~3000
128	~4000	~4000
256	~5000	~10000

NPB-FT Performance (Gop/s)

Locales (x 36 cores / locale)	UPC	Chapel	MPI
16	~1000	~1000	~1000
32	~2000	~2000	~2000
64	~3000	~3000	~3000
128	~4000	~4000	~4000
256	~4500	~4500	~3000

Home

What is Chapel?
What's New?

Blog

Upcoming Events
Job Opportunities

How Can I Learn Chapel?

Contributing to Chapel
Community

Download Chapel
Try Chapel Online

Documentation
Release Notes


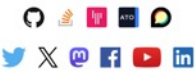
Performance
Powered by Chapel

Presentations
Papers / Publications
Tutorials

ChapelCon
CHUG

Contributors / Credits

chapel+info@discoursemail.com



Arkouda

[github](#) [documentation](#) [gitter](#) [discourse](#) [community](#) [about](#)

Massive scale data science, from the comfort of your laptop

Arkouda Ready for supercomputers **NumPy** Industry-standard

```
import arkouda as ak

ak.startup('localhost', 5555)

# Generate two large arrays
a = ak.randint(0, 2**32, 2**10)
b = ak.randint(0, 2**32, 2**10)

# add them
c = a + b

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

[Get Started](#) [Documentation](#) [Chat on Gitter](#)

Other Community-Oriented Things That Are Cooking

- Uptick in **talks, tutorials**, outreach
- Diversifying **modes of obtaining Chapel**
 - GitHub **Codespaces** (already available)
 - **Ubuntu** and **Enterprise 9** packages (already available)
 - **deb/rpm** packages
 - **Spack** package
 - **AWS AMIs** (Amazon Machine Images)?
- Applying to **HPSF** (High Performance Software Foundation)
- “How-to” **YouTube videos**
- Chapel **educators’ forum**
- Quarterly **newsletter**?
- Community **office hours**?
- [...**Your ideas here**...]
- And your help in growing awareness of Chapel is always appreciated!

Luca speaking today @ 12:00 PT

Chapel Community Survey

- **We launched our first-ever Chapel Community Survey**
 - We would appreciate if everyone attending ChapelCon were to fill it out
 - (Note that there is also a survey about ChapelCon itself—both would be ideal)

Chapel Community Survey

Whether you are a seasoned Chapel veteran or new the language, we would like to hear from you!

Please share this link with others on your team or community that are interested in Chapel.

Note that all questions are optional and that we appreciate any information you share with us.

Thank you!

[Sign in to Google](#) to save your progress. [Learn more](#)

Name

Your answer _____

Email

Your answer _____

Affiliation / Home Institution

Your answer _____

Your answer _____

What do you view as Chapel's biggest strengths?

Your answer _____

What sorts of improvements would you like to see take place in Chapel or its community? (e.g., new features, libraries, performance or portability improvements, events, ...)

Your answer _____

In what situations would you be most likely to recommend a colleague try using Chapel?

Your answer _____

Is there anything else you want share with us?

Your answer _____

[Submit](#) [Clear form](#)



GPUs

GPU Highlights Since CHI UW 2023

- **Improved AMD support**

- multi-locale + multi-GPU runs now supported
- features are in parity with NVIDIA

- **CPU-as-device mode**

- develop CPU+GPU computations without access to a GPU

- **Improved integration with Chapel language**

- ‘reduce’ expressions and intents
- ‘foreach’ intents
- greater use of ‘@attribute’ syntax

- **Plus, performance, scaling, and significant increases in community usage...**



GPU Performance Improvements Since CHI UW 2023

Performance in previous release was lacking

Significantly improved CPU array initialization

```

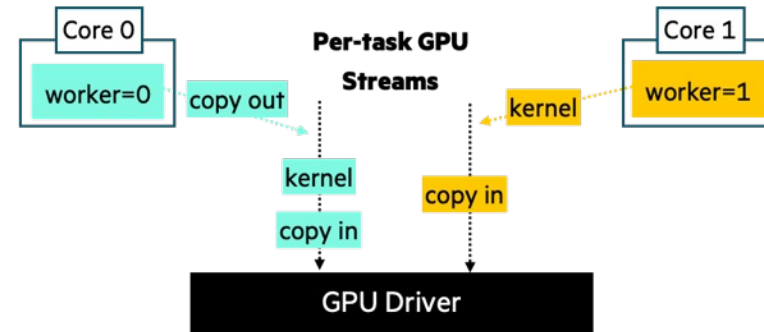
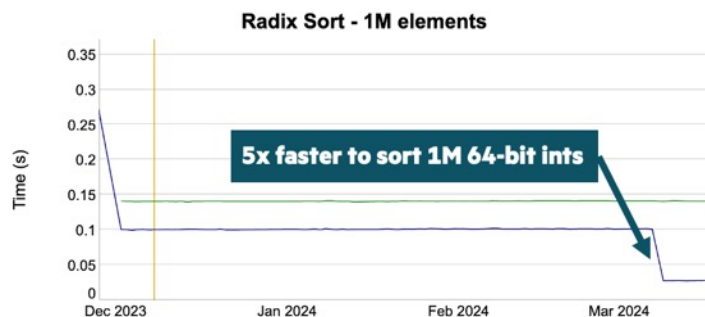
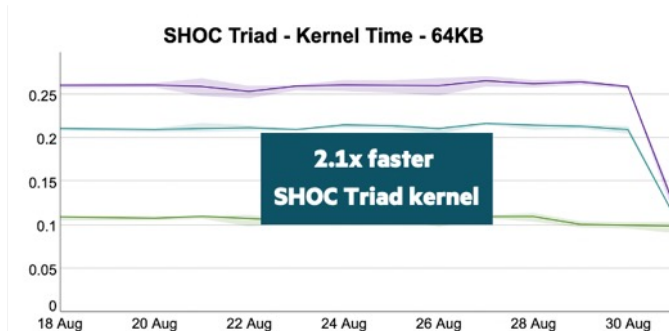
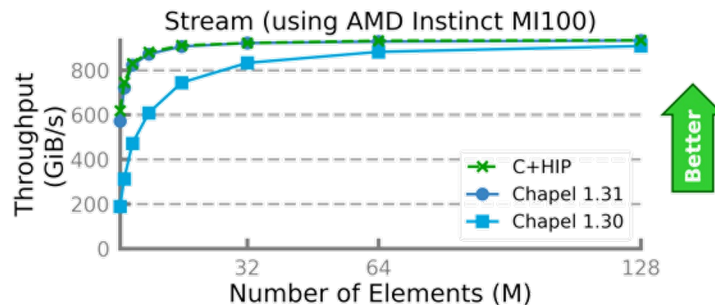
var CpuArr: [1..n] int;

on here.gpus[0] {
  var GpuArr: [1..n] int;

  GpuArr = CpuArr;
  CpuArr = GpuArr;
}
    
```

Time (s) (RTX A2000)		
	Unified Memory	Array on Device
	0.12	0.18
	0.038	0.018

	Unified Memory	Array on Device
	0.25	0.033
	0.14	0.034



Throughput (GiB/s)

enableGpuP2P=true

	0	1	2	3
0		86.2	86.3	86.3
1	86.1		86.4	86.3
2	86.6	86.5		86.1
3	86.6	86.5	86.5	

	Speedup	
	NVIDIA A100	AMD MI250X
coral	1.80x	1.25x
miniBUDE*	1.82x	1.92x

GPU Highlights Since CHIUX 2023: Community Papers at IPDPS Workshops

Performance Portability of the Chapel Language on Heterogeneous Architectures

Josh Milthorpe, Xianghao Wang, Ahmad Azizi (ORNL / ANU), **HCW 2024**

Performance Portability of the Chapel Language on Heterogeneous Architectures

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

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 a parallel programming language that supports the productive development of high-performance scientific applications and has recently added support for GPU architectures through native code generation.

Using three mini-apps — BabelStream, miniBUDE, and TeLeaf — we evaluate the Chapel language’s performance portability across various CPU and GPU platforms. In our evaluation, we replicate and build on previous studies of performance portability using mini-apps, comparing Chapel against OpenMP, Kokkos, and the vendor programming models CUDA and HIP. We find that Chapel achieves comparable performance portability to OpenMP and Kokkos and identify several implementation issues that limit Chapel’s performance portability on certain platforms.

Index Terms—performance portability, Chapel language, mini-app, parallel programming, general-purpose GPU programming

Architecture	OpenMP	Kokkos	CUDA	HIP	Chapel
Skylake	457	444	-	-	488
Cascade Lake	2491	2538	-	-	1571
Sapphire Rapids	6710	7009	-	-	3815
Rome	3484	3471	-	-	3140
Milan	2687	2652	-	-	2771
ThunderX2	427	432	-	-	246
POWER9	33	33	-	-	494
P100	1798	2692	2729	-	2725
V100	3857	5360	5772	-	5582
A100	3639	4537	5087	-	5329
MI60	4436	3554	-	3739	3610
MI100	4288	3554	-	3761	3659

(a) Effective GFLOP/s, higher is better

Architecture	OpenMP	Kokkos	CUDA	HIP	Chapel
Skylake	24.1%	23.5%	-	-	25.7%
Cascade Lake	40.5%	41.3%	-	-	25.6%
Sapphire Rapids	53.1%	55.4%	-	-	30.2%
Rome	56.7%	56.5%	-	-	51.1%
Milan	72.6%	71.7%	-	-	74.9%
ThunderX2	43.3%	43.8%	-	-	25.0%
POWER9	2.8%	2.8%	-	-	41.9%
P100	37.8%	56.5%	57.3%	-	57.2%
V100	49.2%	68.4%	73.7%	-	71.3%
A100	37.3%	46.5%	52.2%	-	54.7%
MI60	60.2%	48.2%	-	50.7%	49.0%
MI100	37.2%	30.8%	-	32.6%	31.7%

(b) Architectural efficiency, higher is better

Fig. 2: miniBUDE results for small deck bm1

CUDA, and HIP programming models across of diverse set of hardware architectures.

Josh speaking today @ 9:40 PT

GPU-Accelerated Tree-Search in Chapel: Comparing Against CUDA and HIP on Nvidia and AMD GPUs

Guillaume Helbecque, Ezhilmathi Krishnasamy, Nouredine Melab, Pascal Bouvry (U. Luxembourg / U. Lille), **PDCO 2024**

GPU-Accelerated Tree-Search in Chapel versus CUDA and HIP

Guillaume Helbecque^{*†}, Ezhilmathi Krishnasamy^{*}, Nouredine Melab[†] and Pascal Bouvry^{*}
^{*}University of Luxembourg, DCS-FSTM/SnT, Luxembourg
[†]Université de Lille, CNRS/CRISIAL UMR 9189, Centre Inria de l’Université de Lille, France

Abstract—In the context of exascale programming, the PGAS-based Chapel is among the rare languages targeting the holistic handling of high-performance computing issues including the productivity-aware harnessing of Nvidia and AMD GPUs. In this paper, we propose a pioneering proof-of-concept dealing with this latter issue in the context of tree-based exact optimization. Actually, we revisit the design and implementation of a generic multi-pool GPU-accelerated tree-search algorithm using Chapel. This algorithm is instantiated on the backtracking method and experimented on the N-Queens problem. For performance evaluation, the Chapel-based approach is compared to Nvidia CUDA and AMD HIP low-level counterparts. The reported results show that in a single-GPU setting, the high GPU abstraction of Chapel results in a loss of only 8% (resp. 16%) compared to CUDA (resp. HIP). In a multi-GPU setting, up to 80% (resp. 71%) of the baseline speed-up is achieved for coarse-grained problem instances on Nvidia (resp. AMD) GPUs.

Index Terms—Chapel, Tree-Search, GPU computing, CUDA, HIP, N-Queens, AMD, Nvidia

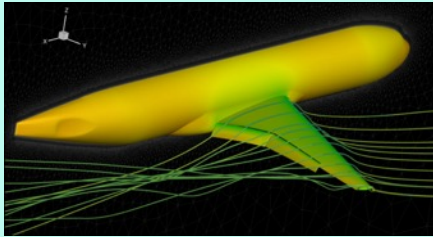
using a combination of a low-level GPU programming model, such as CUDA or HIP, to target the hardware architecture of the GPU. The memory access pattern of the GPU architecture is modeled with in the reasonware initial mode. In based UPC, support [12].

In this work, we focus on the Chapel programming language [13]. It is a versatile PGAS-based parallel programming

Fig. 2: Illustration of the parallel evaluation of nodes model.

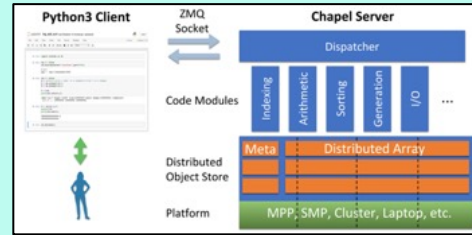
Guillaume speaking today @ 12:45 PT

GPU Highlights Since CHIUW 2023: GPUs and User Codes



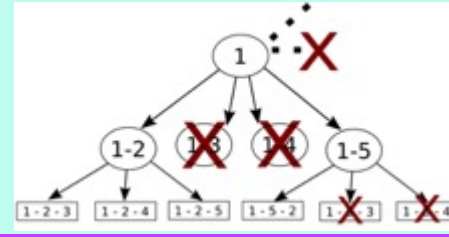
CHAMPS: 3D Unstructured CFD

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



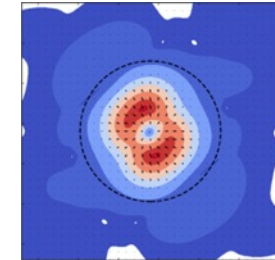
Arkouda: Interactive Data Science at Massive Scale

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



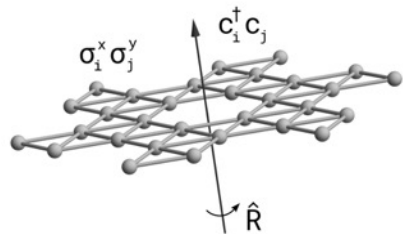
ChOp: Chapel-based Optimization

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



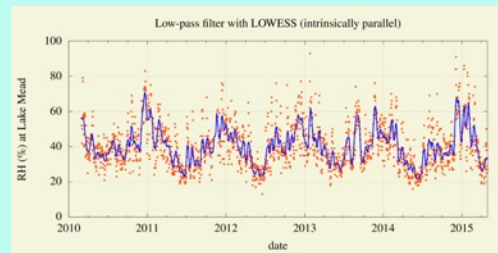
ChpUltra: Simulating Ultralight Dark Matter

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



Lattice-Symmetries: a Quantum Many-Body Toolbox

Tom Westerhout
Radboud University



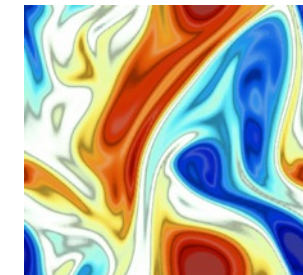
Desk dot chpl: Utilities for Environmental Eng.

Nelson Luis Dias
The Federal University of Paraná, Brazil



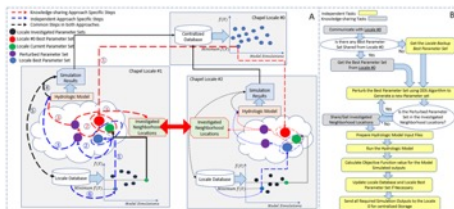
RapidQ: Mapping Coral Biodiversity

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



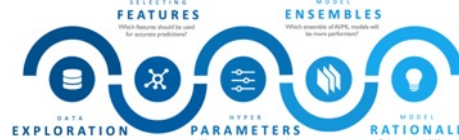
ChapQG: Layered Quasigeostrophic CFD

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)

Active GPU efforts

GPU Highlights Since CHI UW 2023: Scaling on DOE Systems

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

Tiago Carneiro, Engin Kayraklioglu, Guillaume Helbecque, Nouredine Melab (IMEC / HPE / U. Luxembourg, U. Lille),

[Euro-Par 2024](#)

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

Tiago Carneiro¹[0000-0002-6145-8352], Engin Kayraklioglu²[0000-0002-4966-3812],
Guillaume Helbecque^{3,4}[0000-0002-8697-3721], and Nouredine Melab⁴

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

Abstract. The Top500 list features supercomputers powered by accelerators from different vendors. This variety brings, along with the heterogeneity challenge, both the code and performance portability challenges. In this context, Chapel's native GPU support comes as a solution for code portability between different vendors. In this paper, we investigate the viability of using the Chapel high-productivity language as a tool to achieve both code and performance portability in large-scale tree-based search. As a case study, we implemented a distributed backtracking for solving permutation combinatorial problems. Extensive experiments conducted on big N-Queens problem instances, using up to 512 NVIDIA GPUs and 1024 AMD GPUs on Top500 supercomputers, reveal that it is possible to scale on the two different systems using the same tree-based search written in Chapel. This trade-off results in a performance decrease of less than 10% for the biggest problem instances.

Frontier (ORNL)

128 nodes x 8 GPUs = 1024 GPUs

GPUs: AMD Instinct MI250X

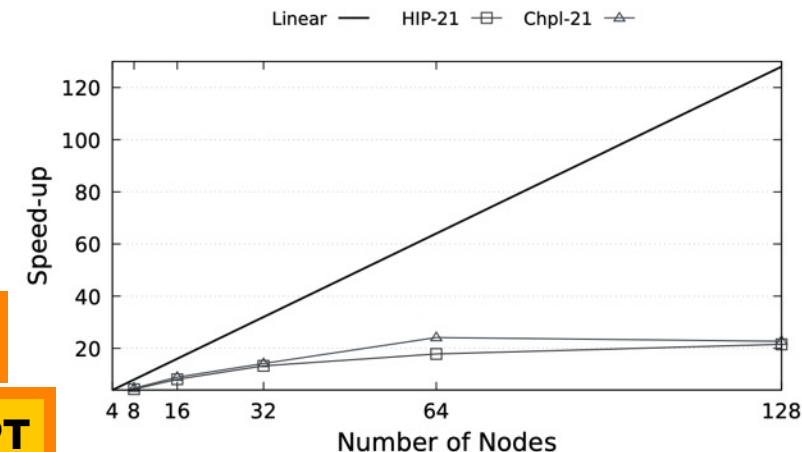
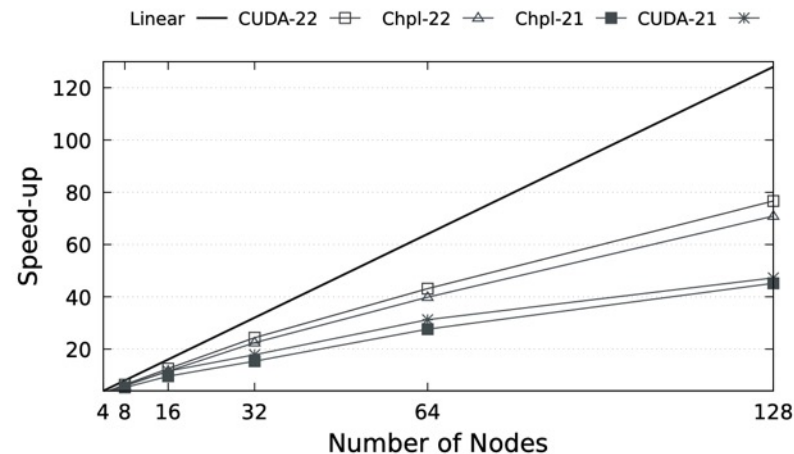
Perlmutter (NERSC)

128 nodes x 4 GPUs = 512 GPUs

GPUs: NVIDIA A100


Tiago speaking today @ 10:00 PT

also see Brett's talk @ 10:10 PT



Additional GPU Resources

[GPU Programming in Chapel](#) blog series



Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts

GPU Programming in Chapel

This series showcases Chapel's support for vendor-neutral GPU programming.

★ **Introduction to GPU Programming in Chapel**

Posted on January 10, 2024

This post gives a beginner's introduction to Chapel's GPU programming features


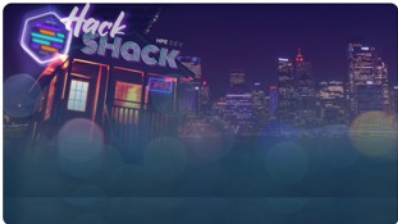
Chapel's High-Level Support for CPU-GPU Data Transfers and Multi-GPU Programming

Posted on April 25, 2024

This post covers how Chapel's arrays, parallelism, and locality features enable moving data between CPUs and GPUs.

[HPE Developer Meetup:](#)


- [Vendor-Neutral GPU Programming in Chapel](#)
- July 31, Free and online

Support English

Vendor-Neutral GPU Programming in Chapel

Date & Time	Jul 31, 2024 08:00 AM in Pacific Time (US and Canada)
Description	Speakers: Jade Abraham, HPE Engin Kayraklioglu, HPE

Abstract:
Writing programs on modern computers requires parallelism to achieve maximum performance. This is



GPU Highlights: ChapelCon 2024 Keynote

[A Case for Parallel-First Languages in a Post-Serial, Accelerated World](#)

Paul speaking today @ 10:45 PT

Paul Sathre (Virginia Tech)

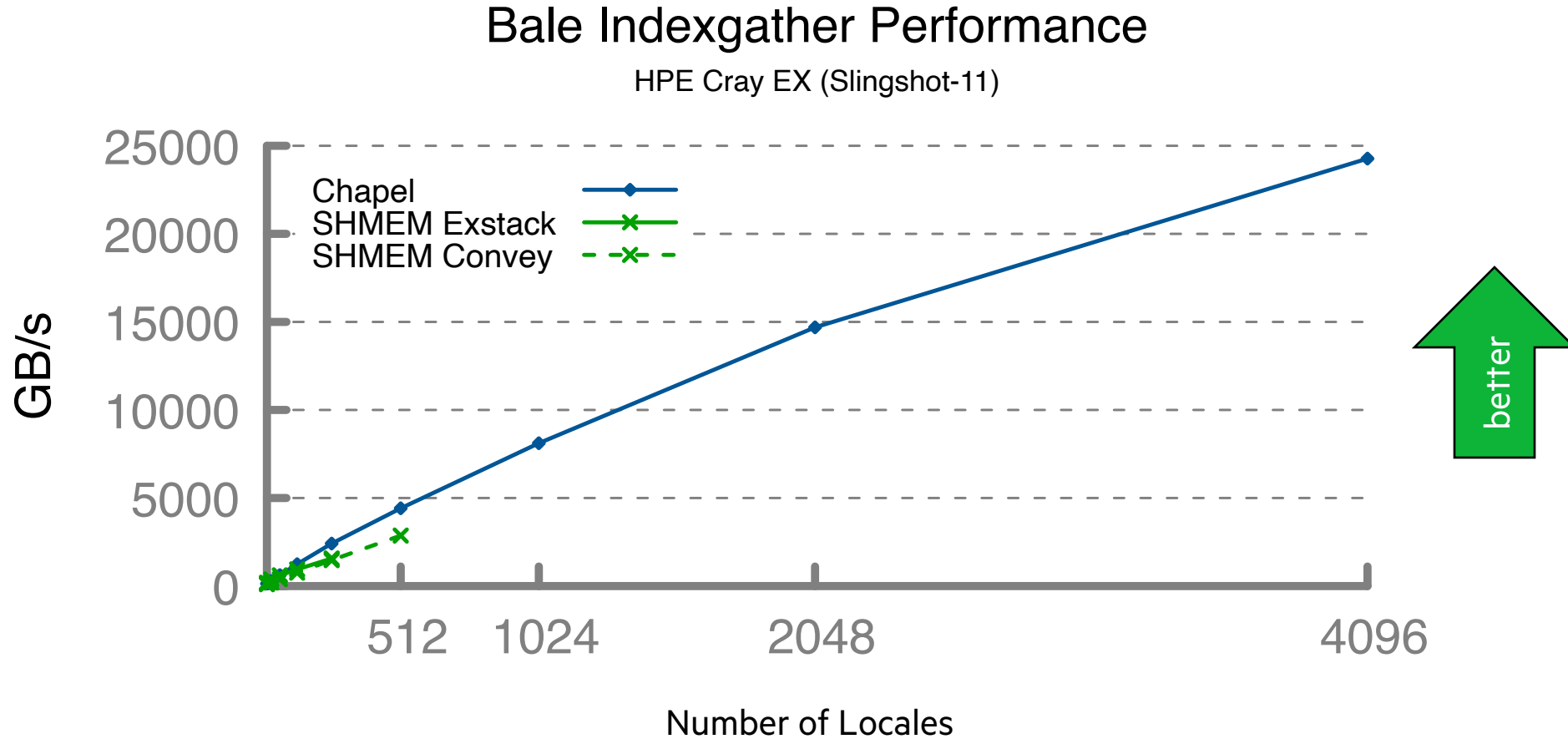


Abstract: Parallel processors have finally dominated all scales of computing hardware, from the personal and portable to the ivory tower datacenters of yore. However, dominant programming models and pedagogy haven't kept pace, and languish in a post-serial mix of libraries and language extensions. Further, heterogeneity in the form of GPUs has dominated the performance landscape of the last decade, penetrating casual user markets thanks to data science, crypto and AI booms. Unfortunately GPUs' performance remains largely constrained to expert users by the lack of more productive and portable programming abstractions. **This talk will probe questions about how to rethink and democratize parallel programming for the masses.** By reflecting on lessons learned from a decade and a half of accelerated computing, **I'll show where Chapel 2.0 fits into the lineage of GPU computing, can capitalize on GPU momentum, and lead a path forward.**

Bio: Paul Sathre is a Research Software Engineer in the Synergy Lab and NSF Center for Space, High-performance, and Resilient Computing (SHREC) at Virginia Tech. His research interests encompass systems software and tools and programming systems, particularly with respect to democratizing access to high-performance computing. More recently, his work has focused on the intersection of computational co-design with portable and productive languages, tools, and libraries for heterogeneous computing.

Scalability and Performance

Bale IndexGather in Chapel vs. SHMEM on HPE Cray EX

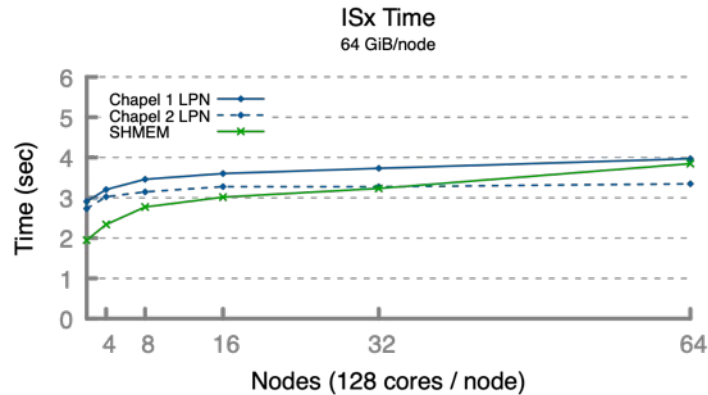


Network Performance Evaluation

Since CHI UW 2023, we have evaluated performance of various communication patterns x networks

- For example, ISx on Slingshot-11, InfiniBand, and AWS:

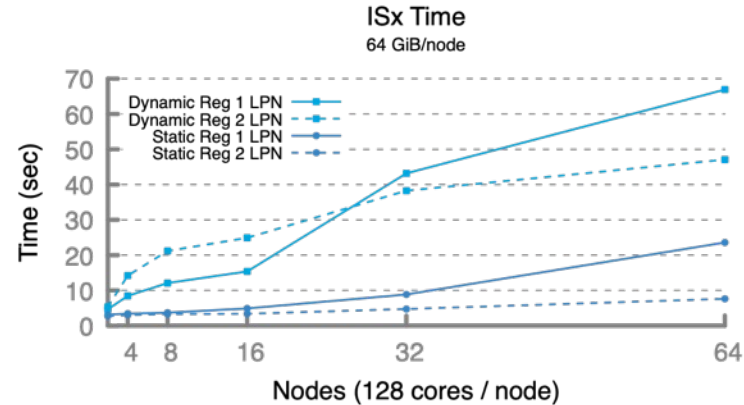
HPE Cray EX



Network: Slingshot-11 (single NIC)

Processor: Dual-socket AMD Milan

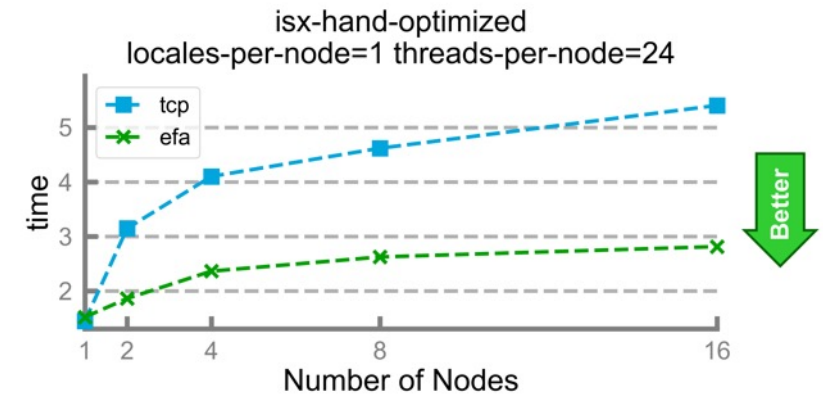
HPE Apollo



Network: InfiniBand HDR-200 (single NIC)

Processor: Dual-socket AMD Milan

AWS



Network: AWS (Ethernet and EFA)

Processor: AWS Graviton3

Notes:

- These AWS results should be considered preliminary and have almost certainly improved since this study
- We also looked briefly at RDMA-enabled Ethernet (not shown here)



Chapel Co-Locales

- A **locale** in Chapel is a part of the target system that can run tasks and store variables
 - Traditionally, each compute node has been a locale
- We've recently added support for multiple **co-locals** per compute node:
 - Command-line interface: `$./myChplProg -nl 8x2 # use 8 nodes w/ 2 locales each...`
 - Typical cases:
 - locale per NIC
 - locale per socket
 - locale per NUMA domain
 - locale per L3 cache
- Co-locals can improve performance via:
 - better network utilization
 - better memory locality and affinity

Stream TRIAD on dual-socket node, 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
"first touch"	470	31.7%	N/A

We've also improved serial/vector performance this year

- In 2023, our fastest CLBG n-body lagged the baseline by 1.4x
 - Rust, Julia, Fortran, and C++ versions all outperformed it
- This year, we became the baseline after the 2.0 release
 - With no source changes!

The Computer Language 23.03 Benchmarks Game

n-body description

First a few simple programs.

Then optimisations, multicore parallelism, [\[pdf\] vector parallelism](#).

Last [hand-written](#) vector instructions and "unsafe" programs.

source	secs	mem	gz
Chapel #3	5.60	10,940	960
C clang	5.98	11,392	1173
Java	7.76	40,724	1430

× source	secs	mem	gz	cpu secs	cpu load
1.0 Rust #6	3.92	11,068	1790	3.92	0% 0% 0% 100%
1.0 Julia #8	4.11	226,264	1111	4.38	2% 2% 2% 100%
1.1 Classic Fortran #6	4.20	10,960	1524	4.20	0% 0% 100% 0%
1.2 C++ g++ #9	4.88	10,960	1530	4.88	0% 0% 0% 100%
1.4 Classic Fortran #2	5.37	10,960	1500	5.36	100% 0% 0% 0%
1.4 Classic Fortran	5.48	10,960	1393	5.48	0% 0% 0% 99%
1.4 Chapel #3	5.60	10,940	960	5.61	99% 0% 0% 0%

The Computer Language 24.04 Benchmarks Game

n-body description

First a few simple programs.

Then optimisations, multicore parallelism, [\[pdf\] vector parallelism](#).

Last [hand-written](#) vector instructions and "unsafe" programs.

source	secs	mem	gz
Chapel #3	3.89	19,648	967
C clang	5.65	19,684	1179
Java	7.85	41,512	1437

× source	secs	mem	gz	cpu secs	cpu load
1.0 Chapel #3	3.89	19,648	967	3.90	100% 0% 0% 0%
1.0 Rust #2	3.92	19,660	1809	3.92	0% 100% 0% 1%
1.0 Rust #6	3.95	19,788	1796		
1.1 Julia #8	4.16	272,512	1129		
1.1 Classic Fortran #6	4.21	19,856	1530		
1.2 Rust #8	4.51	19,788	1774	4.50	100% 0% 0% 1%
1.3 C gcc #6	4.96	19,520	1186	4.96	0% 0% 100% 0%

**I'll speak briefly
on the CLBG today @ 9:30 PT**

Tool Improvements

Chapel Tools: Background

- Tools have been a classic chicken-and-egg problem for Chapel
 - Users don't want to use a new language without tools
 - Tools teams don't want to support a new language until it has users
 - Meanwhile, our team has traditionally had trouble prioritizing tools work
- What tools we have typically come from the open-source community

Happily, this has started to change through our Dyno compiler re-work effort

see Henry, Drake, and Cole's talk @ 8:35 PT

Three New Tools Since CHIUW 2023

chpl-language-server (CLS):

- Enables features within editors that support the Language Server Protocol (LSP) — VSCode, vim, emacs, ...
- Provides real-time features to navigate, query, and refactor Chapel code

chplcheck:

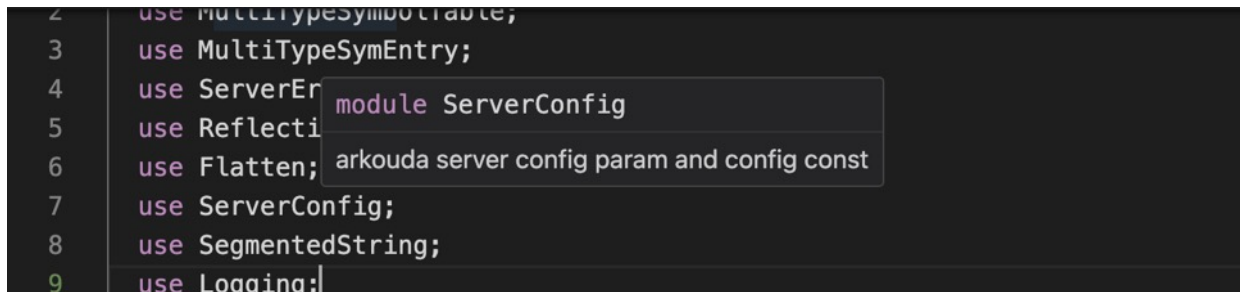
- A Chapel linter that provides style checks and helps prevent common errors
- Can be run from the command-line or an editor (via LSP)

see Daniel and Jade's talk @ 8:55 PT

VSCode extension for Chapel:

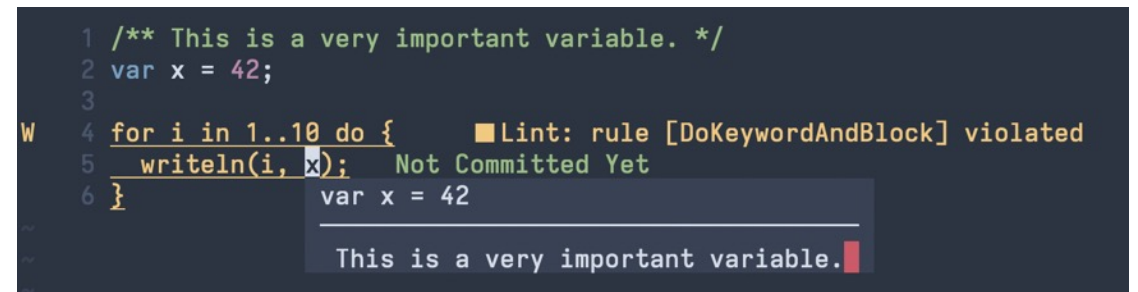
- Supports the two tools above, along with syntax highlighting, autofill, GUI breakpoints, ...

```
2 use MultiTypeSymbolTable;
3 use MultiTypeSymEntry;
4 use ServerEr
5 use Reflecti
6 use Flatten;
7 use ServerConfig;
8 use SegmentedString;
9 use Logging;
```



CLS in VSCode

```
1 /** This is a very important variable. */
2 var x = 42;
3
4 for i in 1..10 do {
5     writeln(i, x);
6 }
```



CLS in Neovim



Time's running short...
Yet there's so much I could cover!

There Are Many Other Exciting Talks Today (that I couldn't weave into this talk)

12:20–12:30 **Exploring Machine Learning Capabilities in Chapel: An Internship Journey**

Iain Moncrief (*Oregon State University*)

Abstract: This talk recounts a

Iain speaking today @ 12:20 PT

from a performance and language experience with learning Chapel

12:55–1:15 **Arrays as Arguments in First-Class Functions: the Levenberg-Marquardt Algorithm in Chapel**

Nelson Dias, Débora Roberti and Vanessa Arruda Dias (*Federal University of Paraná, Federal University of Santa Maria*)

Nelson speaking today @ 12:55 PT

1:15–1:35 **On the Design of Graph Analytical Software in Chapel**

Oliver Alvarado Rodriguez, David A. Bader and Zhihui Du (*New Jersey Institute of Technology*)

Oliver speaking today @ 1:15 PT

1:35–1:45 **Implementing Imaginary Elementary Mathematical Functions**

Damian McGuckin, Peter Harding (*Pacific ESI*)

Damian speaking today @ 1:35 PT

2:00–2:10 **Chapel in a Petabyte-Scale GPU Database Engine with Voltron Data's Theseus**

Trent Nelson and Fernanda Foertter (*Voltron Data*)

Trent speaking today @ 2:00 PT

2:10–2:20 **Chplx: an HPX Foundation for Chapel**

Shreyas Atre, Chris Taylor, Patrick Diehl and Hartmut Kaiser (*Louisiana State University, Tactical Computing Labs, LLC*)

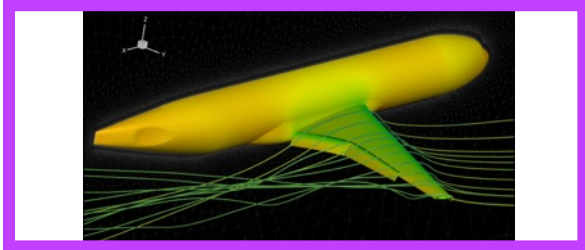
Chris speaking today @ 2:10 PT

2:20–2:40 **Follow-Up on Chapel-Powered HPC Workflows for Python**

John Byrne, Harumi Kuno, Chinmay Ghosh, Porno Shome, Amitha C, Sharad Singhal, Clarete Riana Crasta, David Emberson and Abhishek Dwaraki (*Hewlett Packard Enterprise*)

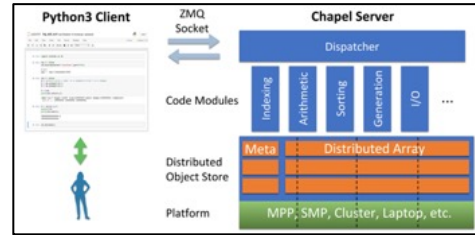
Harumi speaking today @ 2:20 PT

Updates from other CHI UW Alumni



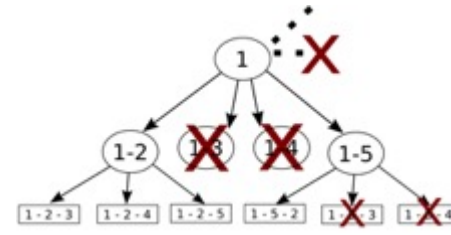
CHAMPS: 3D Unstructured CFD

[CHI UW 2021](#) [CHI UW 2022](#)



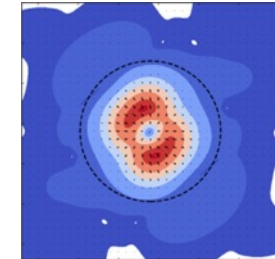
Arkouda: Interactive Data Science at Massive Scale

[CHI UW 2020](#) [CHI UW 2023](#)



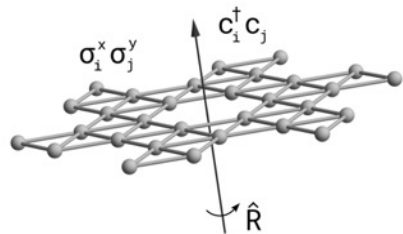
ChOp: Chapel-based Optimization

[CHI UW 2021](#) [CHI UW 2023](#)



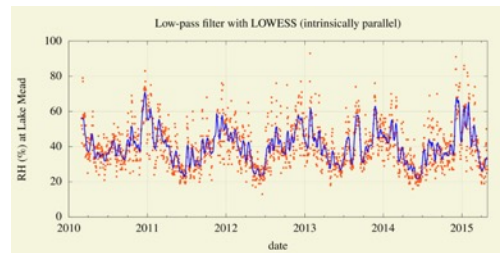
ChpUltra: Simulating Ultralight Dark Matter

[CHI UW 2020](#) [CHI UW 2022](#)



Lattice-Symmetries: a Quantum Many-Body Toolbox

[CHI UW 2022](#)



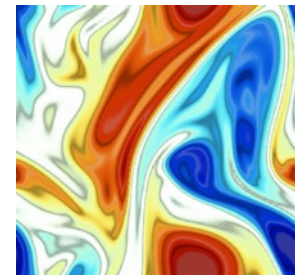
Desk dot chpl: Utilities for Environmental Eng.

[CHI UW 2022](#)

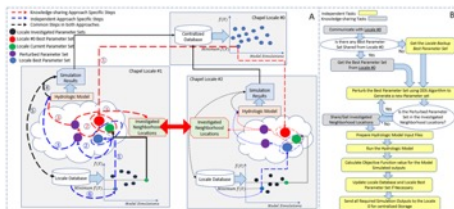


RapidQ: Mapping Coral Biodiversity

[CHI UW 2023](#)

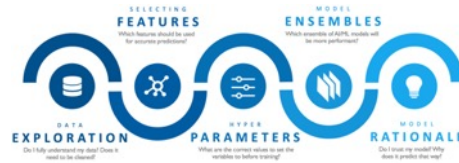


ChapQG: Layered Quasigeostrophic CFD



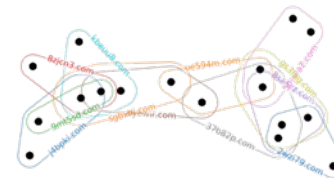
Chapel-based Hydrological Model Calibration

[CHI UW 2023](#)



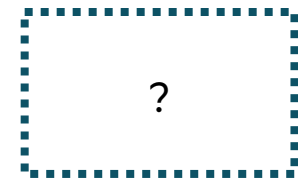
CrayAI HyperParameter Optimization (HPO)

[CHI UW 2021](#)



CHGL: Chapel Hypergraph Library

[CHI UW 2020](#)



Your Application Here?

Update from the CHAMPS Team

CHAMPS status update

- CHAMPS has seen very rapid development progress since its initiation, in many ways due to the efficiency of Chapel. Alongside the technical developments of our Computational Fluid Dynamics (CFD) capabilities, scientific progress has been very fruitful.
- **The research group is now into more holistic research**, slowing down production quantity to examine more fundamental, high-impact research of similar high quality but **with computational breath that is one-order of magnitude more complex**.
For instance:
 - **problem size has now reached 2 Billion unknowns**
 - Reynolds-Averaged Navier-Stokes has moved from steady to unsteady flow analysis, **requiring 1000x more calculations/solutions**.
- This is due to the industrial pull, which requires very advanced computational workflows to examine **novel aircraft configurations to attain environmental targets** such as Bombardier Eco-Jet or NASA/Boeing Truss-Braced Wing concepts.
- **Important advances will be presented at the American Institute of Aeronautics and Astronautics AVIATION conference in late July 2024**. Stay tuned!

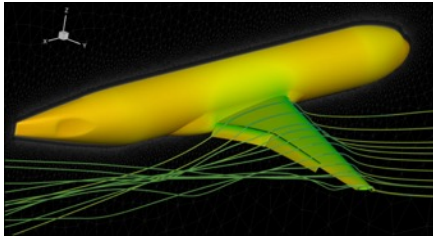


Bombardier Eco-jet
Blended Wing Body



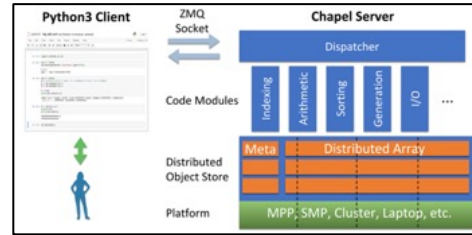
NASA/Boeing
Truss-Braced Wing

Updates from other CHIUW Alumni



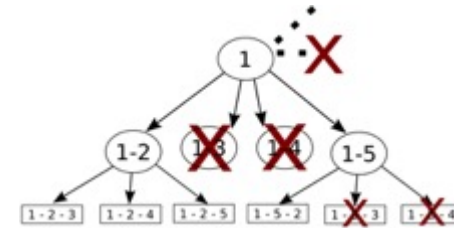
CHAMPS: 3D Unstructured CFD

[CHIUW 2021](#) [CHIUW 2022](#)



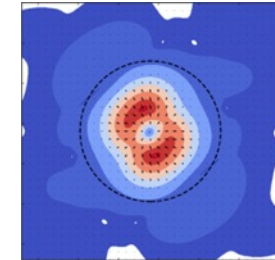
Arkouda: Interactive Data Science at Massive Scale

[CHIUW 2020](#) [CHIUW 2023](#)



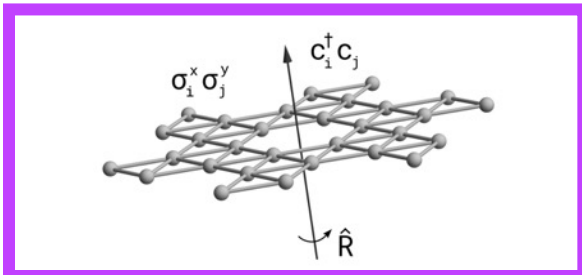
ChOp: Chapel-based Optimization

[CHIUW 2021](#) [CHIUW 2023](#)



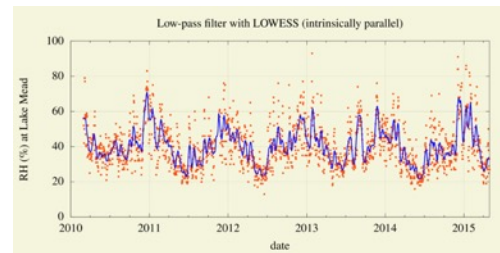
ChpUltra: Simulating Ultralight Dark Matter

[CHIUW 2020](#) [CHIUW 2022](#)



Lattice-Symmetries: a Quantum Many-Body Toolbox

[CHIUW 2022](#)



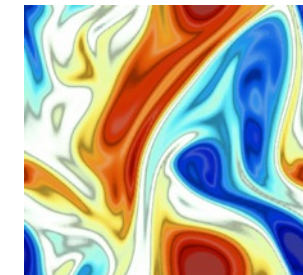
Desk dot chpl: Utilities for Environmental Eng.

[CHIUW 2022](#)

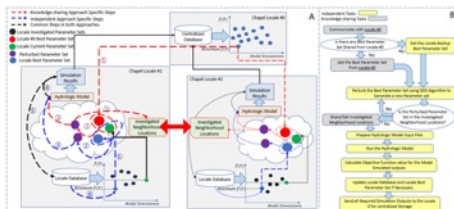


RapidQ: Mapping Coral Biodiversity

[CHIUW 2023](#)

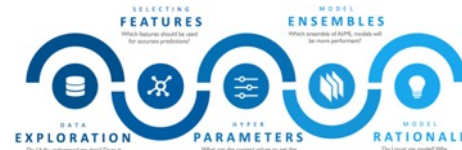


ChapQG: Layered Quasigeostrophic CFD



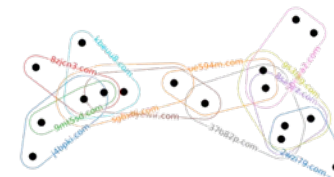
Chapel-based Hydrological Model Calibration

[CHIUW 2023](#)



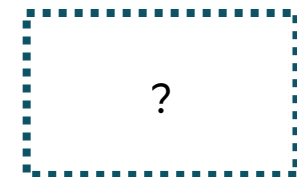
CrayAI HyperParameter Optimization (HPO)

[CHIUW 2021](#)



CHGL: Chapel Hypergraph Library

[CHIUW 2020](#)

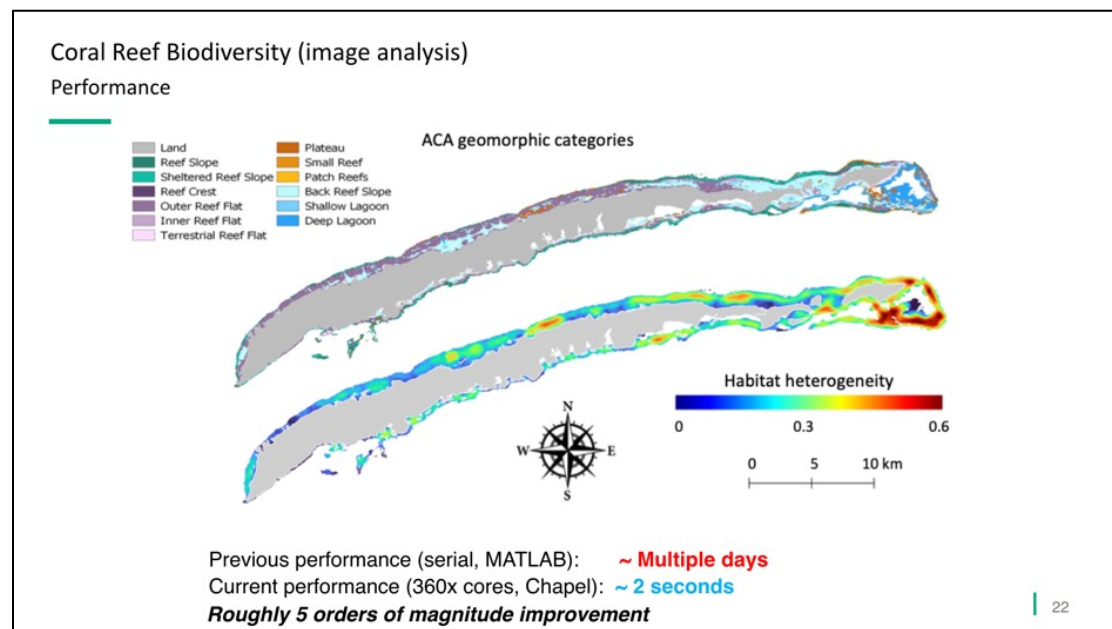


Your Application Here?

Publications at PAW-ATM 2023 (an SC23 workshop)

High-Performance Programming and Execution of a Coral Biodiversity Mapping Algorithm Using Chapel,

Scott Bachman (NCAR) et al.



Implementing Scalable Matrix-Vector Products for the Exact Diagonalization Methods in Quantum Many-Body Physics,

Tom Westerhout (Radboud University) et al.

Implementing Scalable Matrix-Vector Products for the Exact Diagonalization Methods in Quantum Many-Body Physics

Tom Westerhout
tom.westerhout@ru.nl
Institute for Molecules and Materials, Radboud University
Nijmegen, The Netherlands

Bradford L. Chamberlain
bradford.chamberlain@hpe.com
Hewlett Packard Enterprise

ABSTRACT

Exact diagonalization is a well-established method for simulating small quantum systems. Its applicability is limited by the exponential growth of the Hamiltonian matrix that needs to be diagonalized. Physical symmetries are usually utilized to reduce the matrix dimension, and distributed-memory parallelism is employed to explore larger systems. This paper focuses on an implementation of the conventional distributed algorithms, with a special emphasis on the matrix-vector product. Instead of the conventional MPI+X paradigm, Chapel is chosen as the language in this work.

We provide a comprehensive description of the algorithms and present performance and scalability tests. Our implementation outperforms the state-of-the-art MPI-based solution by a factor of 7-8 on 32 compute nodes or 4096 cores and scales well through 256 nodes or 32 768 cores. The implementation has 3 times fewer software lines of code than the current state of the art, but is still able to handle generic Hamiltonians.

CCS CONCEPTS

- Computing methodologies → Parallel computing methodologies;
- Applied computing → Physics;
- Theory of computation → Distributed algorithms.

Package	Spins	Generic Hamiltonians	Matrix-free representation	Lattice symmetries	Distributed-memory parallelism	Software lines of code (excluding tests)
lattice-symmetries [30]	✓	✓	✓	✓	✓	8500
SPINPACK [23]	✓	✓	✓	✓	✓	26000
QuSpin [28, 29]	✓	✓	✓	✓	✓	26000
quantum_basis [27]	✓	✓	✓	✓	✓	12500
Hydra [24]	✓	✓	✓	either one, but not both	✓	18000
libcommute [15]	✓	✓	✓	✓	✓	4500
H4 [13]	✓	✓	✓	✓	✓	29000
Pomerol [3]	✓	✓	✓	✓	✓	5000
EDLib [11]	✓	✓	✓	✓	✓	4000
EDJpack [2]	✓	✓	✓	✓	✓	11000

the Hamiltonian, researchers have employed two key techniques:

First, various symmetries of the Hamiltonian are used to transform the problem into multiple smaller problems that can be solved in parallel. Second, distributed-memory parallelism is used to solve the system size that can be studied. These techniques, systems of about 10¹⁴ spins (called *spins*) can be studied. The dimension of $2^{48} \approx 3 \times 10^{14}$ can be reduced to about 10^{11} . Research groups have been able to exploit both symmetries and its complexity has limited the size of the system. In an endeavor to demonstrate

Speedup over the fastest 1-node run
 (60 spins LS: 124 s, 42 spins LS: 309 s)

Number of nodes (128 cores per node)



**Consider submitting work on
applications in Chapel to:**



The 7th Annual Parallel Applications Workshop, Alternatives To MPI+X

November 17 | 18, 2024

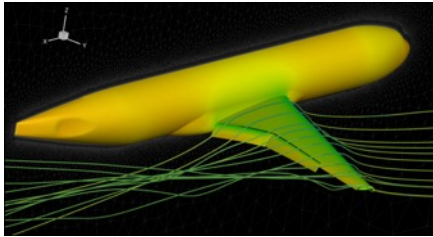
Held in conjunction with SC24



Deadline: July 24, 2024

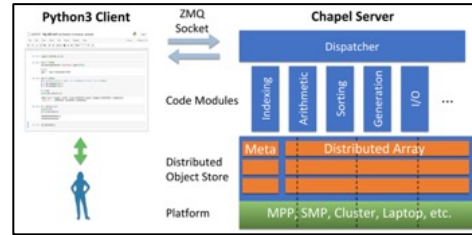
Submission Styles: Papers / Talks

Updates from other CHI UW Alumni



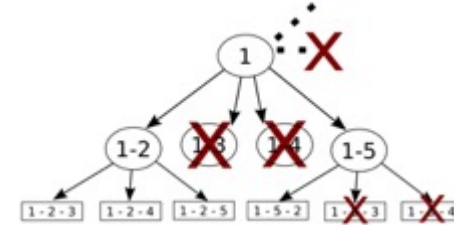
CHAMPS: 3D Unstructured CFD

[CHI UW 2021](#) [CHI UW 2022](#)



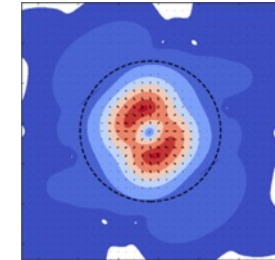
Arkouda: Interactive Data Science at Massive Scale

[CHI UW 2020](#) [CHI UW 2023](#)



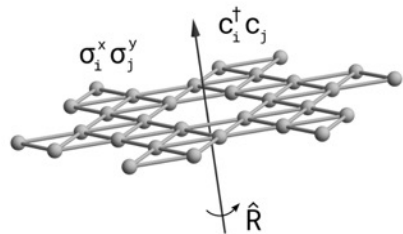
ChOp: Chapel-based Optimization

[CHI UW 2021](#) [CHI UW 2023](#)



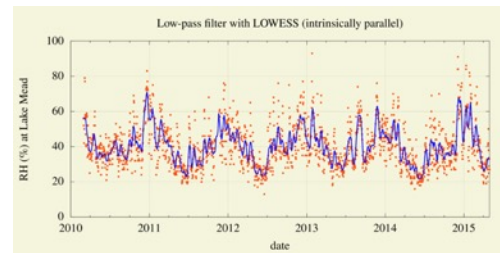
ChpUltra: Simulating Ultralight Dark Matter

[CHI UW 2020](#) [CHI UW 2022](#)



Lattice-Symmetries: a Quantum Many-Body Toolbox

[CHI UW 2022](#)



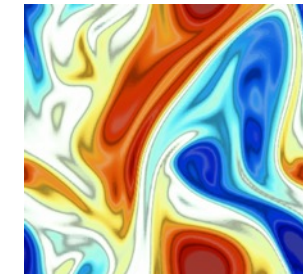
Desk dot chpl: Utilities for Environmental Eng.

[CHI UW 2022](#)

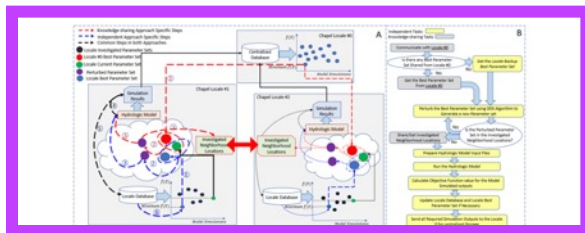


RapidQ: Mapping Coral Biodiversity

[CHI UW 2023](#)

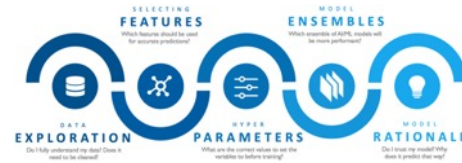


ChapQG: Layered Quasigeostrophic CFD



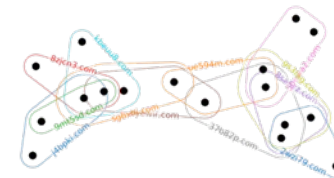
Chapel-based Hydrological Model Calibration

[CHI UW 2023](#)



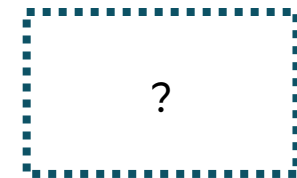
CrayAI HyperParameter Optimization (HPO)

[CHI UW 2021](#)



CHGL: Chapel Hypergraph Library

[CHI UW 2020](#)



Your Application Here?

Newly Minted PhDs

- Since her CHI UW 2023 talk, **Marjan Asgari...**
 - Defended her Ph.D. on parallel computing for calibrating hydrological models
 - Her software is still in use by her former research group at University of Guelph
 - Now works for **Natural Resources Canada** focused on parallel computing
- Since his CHI UW 2023 talk, **Thomas Rolinger...**
 - Defended his Ph.D. on compiler optimizations for irregular memory access patterns in PGAS languages
 - Published a paper on his work at IPDPS 2024
 - Now works on **NVIDIA**'s back-end compiler team

Congratulations, Marjan and Thomas!!!



Wrapping Up

The Chapel Team at HPE



State of the Chapel Project: Summary

As a team and community, we have accomplished a ton since CHIUIW 2023

- Released Chapel 2.0 and made ongoing improvements
- Accomplished and published great scientific results
- Improved GPU, tool, and performance features
- Got a great start on renewed focus on community

GPU Highlights Since CHIUIW 2023: Uses by Users / User Codes

Active GPU efforts

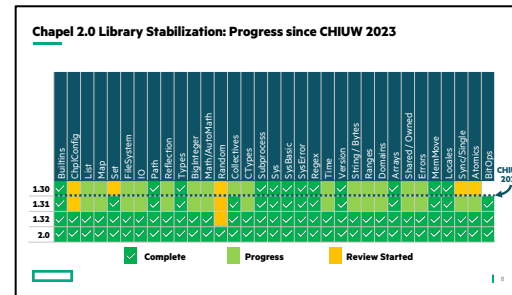
GPU Highlights Since CHIUIW 2023: Community Papers at IPDPS Workshops

Performance Portability of the Chapel Language on Heterogeneous Architectures
 Josh Millhorce, Xianghao Wang, Ahmad Azizi (ORNL / ANL), HCV 2024

GPU-Accelerated Tree-Search in Chapel: Comparing Against CUDA and HIP on Nvidia and AMD GPUs
 Guillaume Helbecque, Ezhilmathi Krishnasamy, Nouredine Melab, Pascal Bouvry (U. Luxembourg / U. Lille), FSDP 2024

Josh speaking today @ 9:40 PT

Guillaume speaking today @ 12:45 PT



Tools: Three New Tools Since CHIUIW 2023

chpl-language-server (CLS):

- Enables features within editors that support the Language Server Protocol (LSP) — VSCode, vim, emacs, ...
- Provides real-time features to navigate, query, and refactor Chapel code

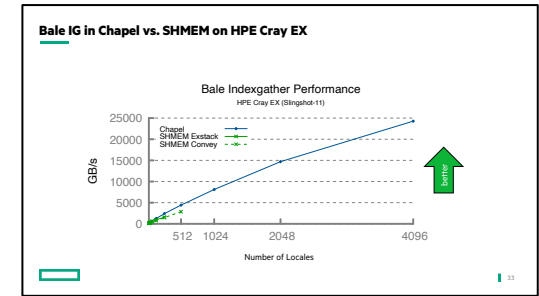
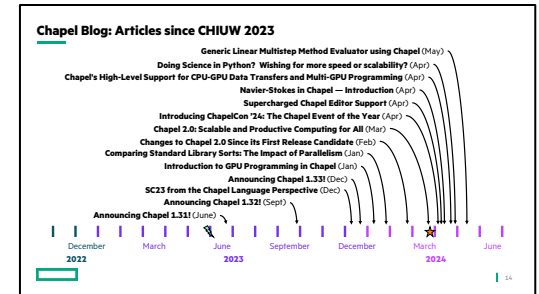
chplcheck:

- A Chapel linter that provides style checks and helps prevent common errors
- Can be run from the command-line or an editor (via LSP)

VSCode extension for Chapel:

- Supports the two tools above, along with syntax highlighting, autofill, GUI breakpoints, ...

see Daniel and Jade's talk @ 8:55 PT



Update from the CHAMPS Team

CHAMPS status update

- CHAMPS has seen very rapid development progress since its installation. In many ways due to the efficiency of Chapel. Alongside the technical developments of our Computational Fluid Dynamics (CFD) capabilities, scientific progress has been very fruitful.
- The research group is now into more holistic research, slowing down production quantity to examine more fundamental, high-impact research of similar high quality but with computational breadth that is one-order of magnitude more complex. For instance:
 - problem size has now reached 2 Billion unknowns
 - Reynolds-Averaged Navier-Stokes has moved from steady to unsteady flow analysis, requiring 1000s more calculations/iterations.
- This is due to the industrial pull, which requires very advanced computational workflows to examine novel aircraft configurations to attain environmental targets such as Bombardier Eco Jet or NASA/Boeing Truss-Braced Wing concepts.
- Important advances will be presented at the American Institute of Aeronautics and Astronautics AVIATION conference in late July 2024. Stay tuned!

Bombardier Eco Jet
 Boeing Truss-Braced Wing
 NASA/Boeing Truss-Braced Wing

Stay tuned to today's talks to hear more detail about many of these efforts!

We look forward to working with you all in the year to come!

Chapel Resources

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

- (points to all other resources)

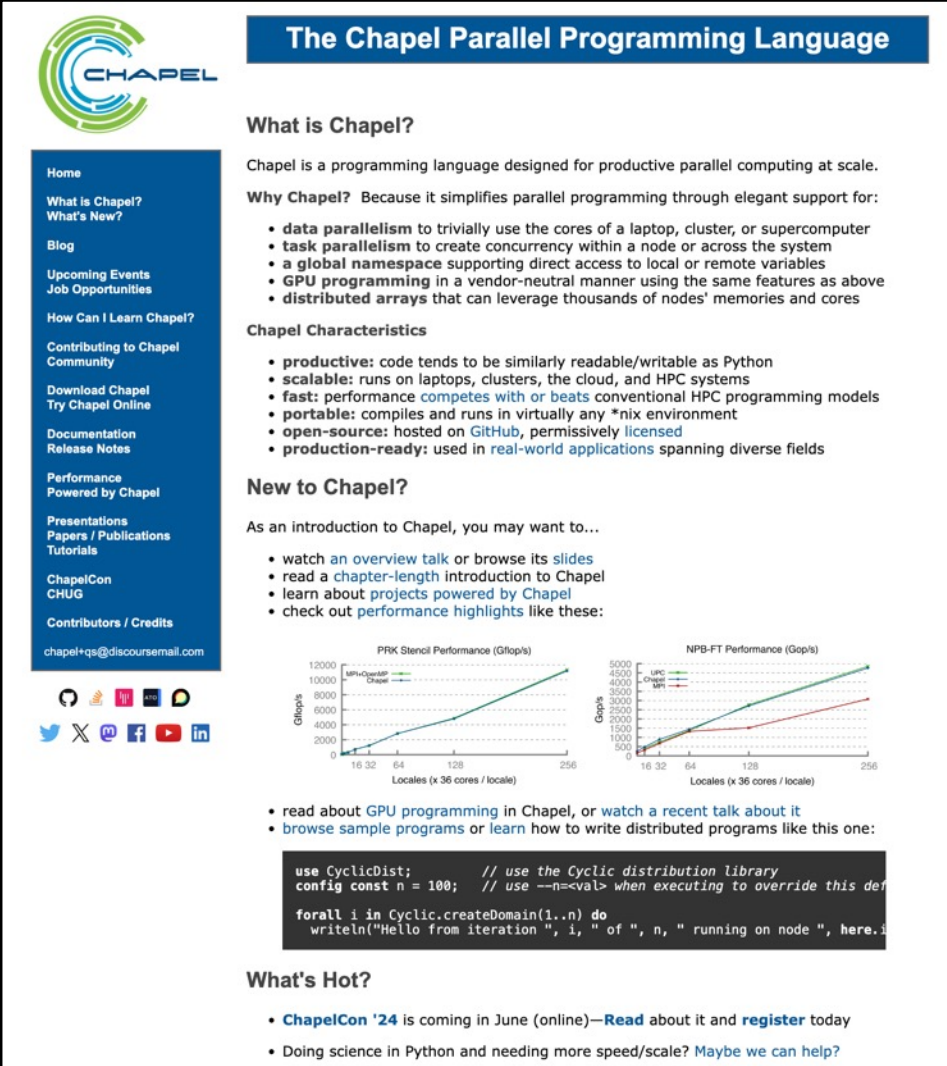
Blog: <https://chapel-lang.org/blog/>

Social Media:

- Facebook: [@ChapelLanguage](#)
- LinkedIn: [@chapel-programming-language](#)
- Mastadon: [@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>



The Chapel Parallel Programming Language

What is Chapel?

Chapel is a programming language designed for productive parallel computing at scale.

Why Chapel?

Because it simplifies parallel programming through elegant support for:

- **data parallelism** to trivially use the cores of a laptop, cluster, or supercomputer
- **task parallelism** to create concurrency within a node or across the system
- **a global namespace** supporting direct access to local or remote variables
- **GPU programming** in a vendor-neutral manner using the same features as above
- **distributed arrays** that can leverage thousands of nodes' memories and cores

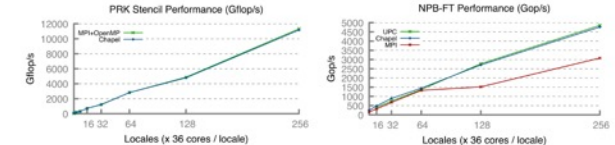
Chapel Characteristics

- **productive:** code tends to be similarly readable/writable as Python
- **scalable:** runs on laptops, clusters, the cloud, and HPC systems
- **fast:** performance *competes with or beats* conventional HPC programming models
- **portable:** compiles and runs in virtually any *nix environment
- **open-source:** hosted on GitHub, permissively licensed
- **production-ready:** used in *real-world applications* spanning diverse fields

New to Chapel?

As an introduction to Chapel, you may want to...

- watch an [overview talk](#) or browse its [slides](#)
- read a [chapter-length](#) introduction to Chapel
- learn about [projects powered by Chapel](#)
- check out performance highlights like these:



Locales (x 36 cores / locale)	Chapel	Other
16	~2000	~2000
32	~4000	~4000
64	~8000	~8000
128	~16000	~16000
256	~32000	~32000

Locales (x 36 cores / locale)	Chapel	Other
16	~1000	~1000
32	~2000	~2000
64	~4000	~4000
128	~8000	~8000
256	~16000	~16000

- read about [GPU programming in Chapel](#), or watch a [recent talk about it](#)
- browse [sample programs](#) or learn how to write distributed programs like this one:

```
use CyclicDist; // use the Cyclic distribution library
config const n = 100; // use --n=<val> when executing to override this def

forall i in Cyclic.createDomain(1..n) do
  writeln("Hello from iteration ", i, " of ", n, " running on node ", here.i)
```

What's Hot?

- **ChapelCon '24** is coming in June (online)—[Read](#) about it and [register](#) today
- Doing science in Python and needing more speed/scale? [Maybe we can help?](#)

**Consider submitting work on
applications in Chapel to:**



The 7th Annual Parallel Applications Workshop, Alternatives To MPI+X

November 17 | 18, 2024

Held in conjunction with SC24



Deadline: July 24, 2024

Submission Styles: Papers / Talks

Thank you

<https://chapel-lang.org>
@ChapelLanguage

