PRACTICAL EXAMPLES OF PRODUCTIVITY AND PERFORMANCE IN CHAPEL

Brad Chamberlain
PASC’23
June 26, 2023
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
Imagine a programming language for parallel computing that was as...

...**programmable** as Python

...yet also as...

...**fast** as Fortran/C/C++

...**scalable** as MPI/SHMEM

...**GPU-ready** as CUDA/OpenMP/OpenCL/OpenACC/...

...**portable** as C
OUTLINE

• What is Chapel?
• Applications of Chapel
• Chapel Intro on CPUs and GPUs
• Wrap-up
### CHAPEL IS COMPACT, CLEAR, AND COMPETITIVE

#### STREAM TRIAD: C + MPI + OPENMP

```chapel
use BlockDist;
config const n = 1_000_000, alpha = 0.01;
const Dom = Block.createDomain([1..n]);
var A, B, C: [Dom] real;
B = 2.0;
C = 1.0;
A = B + alpha * C;
```

#### HPCC RA: MPI KERNEL

```chapel
forall (_, r) in zip(Updates, RAStream()) do
    T[r & indexMask].xor(r);
```
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
INRIA, IMEC, et al.

ChpiUltra: 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

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

ChapQQ: 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

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

**What is it?**
- 3D unstructured CFD framework for airplane simulation
- ~85k lines of Chapel written from scratch in ~3 years

**Who wrote it?**
- Professor Éric Laurendeau’s students + postdocs at Polytechnique Montreal

**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 COMMUNITY HIGHLIGHTS

- Team participated in the 7th AIAA High-lift Prediction Workshop and 1st AIAA Ice Prediction Workshop
  - Generating comparable results to high-profile sites: Boeing, Lockheed Martin, NASA, JAXA, Georgia Tech, ...
- Five papers published last summer at 2022 AIAA Aviation
- While on sabbatical, Éric presented CHAMPS and Chapel at ONERA, DLR, Université de Strasbourg, ...
- Student presentations at CASI/IASC Aero 21 Conference and to CFD Society of Canada (CFDSC)

(slide images taken from Éric Laurendeau’s SIAM PP22 talk, A Case Study on the Impact of Chapel within an Academic Computational Aerodynamic Laboratory, with permission)
“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.”

“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_KyB8al?t=1904 (hyperlink jumps to the section quoted here)
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
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

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

(images provided by their respective teams and used with permission)
Motivation: Imagine you’ve got...
  ...HPC-scale data science problems to solve
  ...a bunch of Python programmers
  ...access to HPC systems

How will you leverage your Python programmers to get your work done?
ARKOUDA: A PYTHON FRAMEWORK FOR INTERACTIVE HPC

Arkouda Client
(written in Python)

Arkouda Server
(written in Chapel)

User writes Python code in Jupyter, making familiar NumPy/Pandas calls
ARKOUDA SUMMARY

What is it?
- A Python client-server framework supporting interactive supercomputing
  - Computes massive-scale results (TB-scale arrays) within the human thought loop (seconds to a few minutes)
  - Initial focus has been on a key subset of NumPy and Pandas for Data Science
- ~30k lines of Chapel + ~25k lines of Python, written since 2019
- Open-source: https://github.com/Bears-R-Us/arkouda

Who wrote it?
- Mike Merrill, Bill Reus, et al., US DoD

Why Chapel?
- close to Pythonic
  - enabled writing Arkouda rapidly
  - doesn’t repel Python users who look under the hood
- achieved necessary performance and scalability
- ability to develop on laptop, deploy on supercomputer
ARKOUDA ARGSORT PERFORMANCE

HPE Apollo (May 2021)
- HDR-100 Infiniband network (100 Gb/s)
- 576 compute nodes
- 72 TiB of 8-byte values
- ~480 GiB/s (~150 seconds)

HPE Cray EX (April 2023)
- Sling114,688shot network (200 Gb/s)
- 896 nodes
- 28 TiB of 8-byte values
- ~1200 GiB/s (~24 seconds)

HPE Cray EX (May 2023)
- Slingshot network (200 Gb/s)
- 8192 nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)

A notable performance achievement in ~100 lines of Chapel
ARKOUDA ARGSORT PERFORMANCE

HPE Apollo (May 2021)
- HDR-100 Infiniband network (100 Gb/s)
- 576 compute nodes
- 72 TiB of 8-byte values
- ~480 GiB/s (~150 seconds)

HPE Cray EX (April 2023)
- Slingshot-11 network (200 Gb/s)
- 896 compute nodes
- 28 TiB of 8-byte values
- ~1200 GiB/s (~24 seconds)

A notable performance achievement in ~100 lines of Chapel
ARKOUDA ARGSORT PERFORMANCE

HPE Apollo (May 2021)
- HDR-100 Infiniband network (100 Gb/s)
- 576 compute nodes
- 72 TiB of 8-byte values
- ~480 GiB/s (~150 seconds)

HPE Cray EX (April 2023)
- Slingshot-11 network (200 Gb/s)
- 896 compute nodes
- 28 TiB of 8-byte values
- ~1200 GiB/s (~24 seconds)

HPE Cray EX (May 2023)
- Slingshot-11 network (200 Gb/s)
- 8192 compute nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)

A notable performance achievement in ~100 lines of Chapel
APPLICATIONS OF CHAPEL: LINKS TO USERS’ TALKS (SLIDES + VIDEO)

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

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

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

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

ChplUltra: Simulating Ultralight Dark Matter
Chop: Chapel-based Optimization
INRIA, IMEC, et al.

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

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

Nelson Luis Dias
The Federal University of Paraná, Brazil

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

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


RapidQ: Mapping Coral Biodiversity

ChapQG: Layered Quasigeostrophic CFD

Your Application Here?

(images provided by their respective teams and used with permission)
INTRODUCTION TO CHAPEL ON CPUS AND GPUS

(BY EXAMPLE USING STREAM TRIAD)
STREAM TRIAD: A TRIVIAL CASE OF PARALLELISM + LOCALITY

**Given:** \( n \)-element vectors \( A, B, C \)

**Compute:** \( \forall i \in 1..n, A_i = B_i + \alpha \cdot C_i \)

**In pictures:**

\[
\begin{array}{c}
\text{A} \\
\text{B} \\
\text{C}
\end{array}
\begin{array}{c}
= \\
+ \\
\cdot
\end{array}
\begin{array}{c}
\alpha
\end{array}
\]
STREAM TRIAD: A TRIVIAL CASE OF PARALLELISM + LOCALITY

**Given:** $n$-element vectors $A, B, C$

**Compute:** $\forall i \in 1..n, A_i = B_i + \alpha \cdot C_i$

**In pictures, in parallel** (shared memory / multicore):

[Diagram of parallel computation with shared memory]
STREAM TRIAD: A TRIVIAL CASE OF PARALLELISM + LOCALITY

**Given:** $n$-element vectors $A$, $B$, $C$

**Compute:** $\forall i \in 1..n, A_i = B_i + \alpha \cdot C_i$

**In pictures, in parallel** (distributed memory, global-view):

![Diagram showing the computation process]
STREAM TRIAD: A TRIVIAL CASE OF PARALLELISM + LOCALITY

**Given:** \( n \)-element vectors \( A, B, C \)

**Compute:** \( \forall i \in 1..n, A_i = B_i + \alpha \cdot C_i \)

**In pictures, in parallel** (distributed memory multicore, global-view):

![Diagram showing the stream triad computation in parallel](diagram.png)
**STREAM TRIAD: AN ALTERNATE APPROACH**

**Given:** $n$-element vectors $A, B, C$ on each locale

**Compute:** $\forall i \in 1..n, A_i = B_i + \alpha \cdot C_i$

**In pictures, in parallel** (distributed memory multicore, local-view):

![Diagram showing parallel computation of the stream triad](image-url)
STREAM TRIAD: SHARED MEMORY VERSION

```
stream-ep.chpl
config const n = 1_000_000,
    alpha = 0.01;

var A, B, C: [1..n] real;
A = B + alpha * C;
```

So far, this is simply a multi-core program
Nothing refers to remote locales, explicitly or implicitly

declare three arrays of size ‘n’
whole-array operations result in parallel computation
STREAM TRIAD: DISTRIBUTED MEMORY, EP VERSION

```chapel
config const n = 1_000_000,
    alpha = 0.01;

coforall loc in Locales {
    on loc {
        var A, B, C: [1..n] real;
        A = B + alpha * C;
    }
}
```

create a task per locale...
...running ‘on’ its locale
then run multi-core Stream on local arrays, as before
STREAM TRIAD: DISTRIBUTED MEMORY, GLOBAL VERSION

```
stream-glbl.chpl

config const n = 1_000_000,
    alpha = 0.01;

use BlockDist;

const Dom = Block.createDomain({1..n});
var A, B, C: [Dom] real;
A = B + alpha * C;
```

- 'use' the standard block-distribution module
- create a distributed domain (index set)...
- ...and distributed arrays
- these whole-array operations will use all cores on all locales
HPC BENCHMARKS: CONVENTIONAL APPROACHES VS. CHAPEL

STREAM TRIAD: C + MPI + OPENMP

HPCC RA: MPI KERNEL

use BlockDist;

config const n = 1_000_000, alpha = 0.01;

const Dom = Block.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);
HPC BENCHMARKS: CONVENTIONAL APPROACHES VS. CHAPEL

STREAM TRIAD: C + MPI + OPENMP

```
use BlockDist;

config const n = 1_000_000,
    alpha = 0.01;
const Dom = Block.createDomain([1..n]);
var A, B, C: [Dom] real;
B = 2.0;
C = 1.0;
A = B + alpha * C;
```

These programs are all CPU-only

Nothing refers to GPUs, explicitly or implicitly
STREAM TRIAD: DISTRIBUTED MEMORY, GPUS ONLY

```chpl
config const n = 1_000_000,
        alpha = 0.01;

coforall loc in Locales do on loc {
    coforall gpu in here.gpus do on gpu {
        var A, B, C: [1..n] real;
        A = B + alpha * C;
    }
}
```

This is a GPU-only program

Nothing other than coordination code runs on the CPUs.

Use a similar ‘coforall’ + ‘on’ idiom to run a Triad concurrently on each of this locale’s GPUs.
The program uses all CPUs and GPUs across all of our compute nodes.

```chpl
stream-ep.chpl

config const n = 1_000_000,
alpha = 0.01;

coforall loc in Locales do on loc {
  cobegin {
    coforall gpu in here.gpus do on gpu {
      var A, B, C: [1..n] real;
      A = B + alpha * C;
    }
    { 
      var A, B, C: [1..n] real;
      A = B + alpha * C;
    }
  }
}
```

- 'cobegin { ... }' creates a task per child statement.
- One task runs our multi-GPU triad.
- The other runs the multi-CPU triad.
STREAM TRIAD: DISTRIBUTED MEMORY, GPUS AND CPUs (REFACTOR)

```chpl
config const n = 1_000_000,
    alpha = 0.01;

coforall loc in Locales do on loc {
    cobegin {
        coforall gpu in here.gpus do on gpu {
            runTriad();
        }
        runTriad();
    }
}

proc runTriad() {
    var A, B, C: [1..n] real;
    A = B + alpha * C;
}
```

we can also refactor the repeated code into a procedure for re-use

the compiler creates CPU and GPU versions of this procedure
# CHAPEL ON GPUS: STATUS

**Status:** Compiling Chapel to GPUs is still reasonably new:

<table>
<thead>
<tr>
<th></th>
<th>NVIDIA</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>via interop?</td>
<td>via interop?</td>
</tr>
<tr>
<td></td>
<td>heroic?</td>
<td>heroic?</td>
</tr>
<tr>
<td>Chapel loops?</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>multi-GPU?</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>multi-node?</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

pre-2021: 

- ✓ (Chapel loops?)
- ✓ (multi-GPU?)
- ✓ (multi-node?)
### CHAPEL ON GPUS: STATUS

**Status:** Compiling Chapel to GPUs is still reasonably new:

<table>
<thead>
<tr>
<th></th>
<th><strong>NVIDIA</strong></th>
<th></th>
<th><strong>AMD</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-2021</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mar 2021</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
**CHAPEL ON GPUS: STATUS**

**Status:** Compiling Chapel to GPUs is still reasonably new:

<table>
<thead>
<tr>
<th></th>
<th><strong>NVIDIA</strong></th>
<th><strong>AMD</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-2021</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Mar 2021</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sep 2021</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
## CHAPEL ON GPUs: STATUS

### Status:
Compiling Chapel to GPUs is still reasonably new:

<table>
<thead>
<tr>
<th></th>
<th>NVIDIA</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>via interop?</td>
<td>heroic?</td>
</tr>
<tr>
<td></td>
<td>Chapel loops?</td>
<td>multi-GPU?</td>
</tr>
<tr>
<td></td>
<td>multi-node?</td>
<td>via interop?</td>
</tr>
<tr>
<td></td>
<td>heroic?</td>
<td>Chapel loops?</td>
</tr>
<tr>
<td></td>
<td>multi-GPU?</td>
<td>multi-node?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-2021</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mar 2021</td>
<td>v1.24</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sep 2021</td>
<td>v1.25</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mar 2022</td>
<td>v1.26</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## CHAPEL ON GPUS: STATUS

**Status:** Compiling Chapel to GPUs is still reasonably new:

<table>
<thead>
<tr>
<th></th>
<th>NVIDIA</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-2021</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Mar 2021 v1.24</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sep 2021 v1.25</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mar 2022 v1.26</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Jun 2022 v1.27</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
**CHAPEL ON GPUS: STATUS**

**Status:** Compiling Chapel to GPUs is still reasonably new:

<table>
<thead>
<tr>
<th></th>
<th>NVIDIA</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-2021</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Mar 2021 v1.24</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sep 2021 v1.25</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mar 2022 v1.26</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Jun 2022 v1.27</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Dec 2023 v1.29</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
## CHAPEL ON GPUS: STATUS

**Status:** Compiling Chapel to GPUs is still reasonably new:

<table>
<thead>
<tr>
<th></th>
<th>NVIDIA</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-2021</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mar 2021</td>
<td>v1.24</td>
<td>✓</td>
</tr>
<tr>
<td>Sep 2021</td>
<td>v1.25</td>
<td>✓</td>
</tr>
<tr>
<td>Mar 2022</td>
<td>v1.26</td>
<td>✓</td>
</tr>
<tr>
<td>Jun 2022</td>
<td>v1.27</td>
<td>✓</td>
</tr>
<tr>
<td>Dec 2023</td>
<td>v1.29</td>
<td>✓</td>
</tr>
<tr>
<td>Mar 2023</td>
<td>v1.30</td>
<td>✓</td>
</tr>
</tbody>
</table>
**Status:** Compiling Chapel to GPUs is still reasonably new:

<table>
<thead>
<tr>
<th></th>
<th>NVIDIA</th>
<th></th>
<th></th>
<th></th>
<th>AMD</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>via</td>
<td>heroic</td>
<td>Chapel</td>
<td>multi-</td>
<td></td>
<td></td>
<td></td>
<td>multi-</td>
</tr>
<tr>
<td></td>
<td>interop</td>
<td></td>
<td>loops?</td>
<td>GPU?</td>
<td>multi-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pre-2021</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mar 2021</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sep 2021</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mar 2022</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Jun 2022</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Dec 2023</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mar 2023</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Jun 2023</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Performance vs. CUDA has become increasingly competitive over the past 6 months
WRAP-UP
THE CHAPEL TEAM AT HPE, JUNE 2023
Chapel is unique among programming languages
• built-in features for scalable parallel computing make it HPC-ready
• supports clean, concise code relative to conventional approaches
• ports and scales from laptops to supercomputers

Chapel is being used for productive parallel computing at scale
• users are reaping its benefits in practical, cutting-edge applications
• in diverse application domains: from physical simulation to data science
• scaling to thousands of nodes / millions of processor cores

Vendor-neutral GPU support is maturing rapidly
• fleshes out an overdue aspect of “any parallel hardware”

We’re interested in helping new users and fostering new collaborations
CHAPEL RESOURCES

Chapel homepage: [https://chapel-lang.org](https://chapel-lang.org)
- (points to all other resources)

Social Media:
- Blog: [https://chapel-lang.org/blog/](https://chapel-lang.org/blog/)
- Twitter: [@ChapelLanguage](https://twitter.com/ChapelLanguage)
- Facebook: [@ChapelLanguage](https://facebook.com/ChapelLanguage)
- YouTube: [@ChapelLanguage](https://youtube.com/ChapelLanguage)

Community Discussion / Support:
- Discourse: [https://chapel.discourse.group/](https://chapel.discourse.group/)
- Gitter: [https://gitter.im/chapel-lang/chapel](https://gitter.im/chapel-lang/chapel)
- Stack Overflow: [https://stackoverflow.com/questions/tagged/chapel](https://stackoverflow.com/questions/tagged/chapel)
- GitHub Issues: [https://github.com/chapel-lang/chapel/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:
- distributed arrays that can leverage thousands of nodes’ memories and cores
- a global namespace supporting direct access to local or remote variables
- 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

Chapel Characteristics
- productive: code tends to be similarly readable/writeable as Python
- scalable: runs on laptops, clusters, the cloud, and HPC systems
- fast: performance compares with or beats C/C++ & MPI & OpenMP
- portables: compiles and runs in virtually any *nix environment
- open-source: hosted on Github, permissively licensed

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:

- browse sample programs or learn how to write distributed programs like this one:

```c
// use the cyclic distribution library
cyclic count n = 16; // use --cyclic when compiling this example
parallel for (i = 0; i < n; i++)
    do
        printf("Hello from iteration ", i, " of ", n, " running on node ", node.id);
```
Chapel is unique among programming languages

• built-in features for scalable parallel computing make it HPC-ready
• supports clean, concise code relative to conventional approaches
• ports and scales from laptops to supercomputers

Chapel is being used for productive parallel computing at scale

• users are reaping its benefits in practical, cutting-edge applications
• in diverse application domains: from physical simulation to data science
• scaling to thousands of nodes / millions of processor cores

Vendor-neutral GPU support is maturing rapidly

• fleshes out an overdue aspect of “any parallel hardware”

SUMMARY

We’re interested in helping new users and fostering new collaborations
THANK YOU

https://chapel-lang.org
@ChapelLanguage