INTRODUCING CHAPEL: A PROGRAMMING LANGUAGE FOR PRODUCTIVE PARALLEL COMPUTING FROM LAPTOPS TO SUPERCOMPUTERS

Brad Chamberlain, Distinguished Technologist
LinuxCon, May 11, 2023
Parallel Computing: Using the processors and memories of multiple compute resources

- in order to run a program...
  - faster than we could otherwise
  - and/or using larger problem sizes
PARALLEL COMPUTING HAS BECOME UBIQUITOUS

Traditional parallel computing:
- supercomputers
- commodity clusters

Today:
- multicore processors
- GPUs
- cloud computing

![Diagram of Compute Nodes](image)
OAK RIDGE NATIONAL LABORATORY'S FRONTIER SUPERCOMPUTER

• 74 HPE Cray EX cabinets
• 9,408 AMD CPUs, 37,632 AMD GPUs
• 700 petabytes of storage capacity, peak write speeds of 5 terabytes per second using Cray ClusterStor storage system
• HPE Slingshot networking cables providing 100 GB/s network bandwidth.

TOP500

1

Built by HPE, ORNL’s Frontier supercomputer is #1 on the TOP500.

1.1 exaflops of performance.

GREEN500

2

Built by HPE, ORNL’s TDS and full system are ranked #2 & #6 on the Green500.

62.68 gigaflops/watt power efficiency for ORNL’s TDS system, 52.23 gigaflops/watt power efficiency for full system.

HPL-MxP

1

Built by HPE, ORNL’s Frontier supercomputer is #1 on the HPL-MxP list.

7.9 exaflops on the HPL-MxP benchmark (formerly HPL-AI).

HPC BENCHMARKS USING CONVENTIONAL PROGRAMMING APPROACHES

STREAM TRIAD: C + MPI + OPENMP

```c
#include <hpcc.h>
#include <omp.h>
#include <stdlib.h>

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

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commCount;
  int rv, errcount = 0;
  rv = HPCC_InitStream( params, 0 == MPI_COMM_WORLD);
  comm = MPI_COMM_WORLD;
  MPI_Comm_rank(comm, &myRank);
  if (myRank == 0) {    /* Set vector sizes. */
    VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0);
    a = HPCC_MALLOC( VectorSize );
    b = HPCC_MALLOC( VectorSize );
    c = HPCC_MALLOC( VectorSize );
  }
  rv = HPCC_StarStream(comm, a, b, c, /* Update Updates */);
  MPI_Reduce(&rv, &rv, 1, MPI_INT, MPI_MAX, 0, comm);
  return rv;
}
```

if (a | b | c) {
  if (myRank == 0) {    /* Set vector sizes. */
    VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0);
    a = HPCC_MALLOC( VectorSize );
    b = HPCC_MALLOC( VectorSize );
    c = HPCC_MALLOC( VectorSize );
  }
  rv = HPCC_StarStream( params, 0 == MPI_COMM_WORLD);
  MPI_Comm_rank(comm, &myRank);
  if (myRank == 0) {    /* Set vector sizes. */
    VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0);
    a = HPCC_MALLOC( VectorSize );
    b = HPCC_MALLOC( VectorSize );
    c = HPCC_MALLOC( VectorSize );
  }
  rv = HPCC_StarStream(comm, a, b, c, /* Update Updates */);
  MPI_Reduce(&rv, &rv, 1, MPI_INT, MPI_MAX, 0, comm);
  return rv;
}
```

if (params) {
  int sz = sizeof(HPCC_Params);
  /* We got a done message. */
  MPI_Waitall(1, inreq, &have_done);
  rv = 0;
  /* Finish everyone else up... */
  MPI_Isend(&LocalSendBuffer, localBufferSize, tparams.dtype64,
            sSrc, UPDATE_TAG, proc_count, MPI_COMM_WORLD, &outreq);
  MPI_Waitall(numProcs - 1, outreq, &have_done);
  /* Send our done messages */
  MPI_Isend(&LocalSendBuffer, localBufferSize, tparams.dtype64,
            sDest, UPDATE_TAG, proc_count, MPI_COMM_WORLD, &outreq);
  MPI_Waitall(numProcs - 1, outreq, &have_done);
  /* Send garbage... */
  MPI_Isend(ran, 0, tparams.dtype64, proc_count, FINISHED_TAG, MPI_COMM_WORLD, &outreq);
  MPI_Waitall(1, outreq, &have_done);
  if (NumberReceiving > 0) {
    /* update our update buffers */
    if (NumberReceiving > 0) {
      /* update our update buffers */
      if (NumberReceiving > 0) {
        if (NumberReceiving > 0) {
          if (NumberReceiving > 0) {
            if (NumberReceiving > 0) {
              if (NumberReceiving > 0) {
                if (NumberReceiving > 0) {
                  if (NumberReceiving > 0) {
                    if (NumberReceiving > 0) {
                      if (NumberReceiving > 0) {
                        if (NumberReceiving > 0) {
                          if (NumberReceiving > 0) {
                            if (NumberReceiving > 0) {
                              if (NumberReceiving > 0) {
                                if (NumberReceiving > 0) {
                                  if (NumberReceiving > 0) {
                                    if (NumberReceiving > 0) {
                                      if (NumberReceiving > 0) {
                                        if (NumberReceiving > 0) {
                                          if (NumberReceiving > 0) {
                                            if (NumberReceiving > 0) {
                                              if (NumberReceiving > 0) {
                                                if (NumberReceiving > 0) {
                                                  if (NumberReceiving > 0) {
                                                    if (NumberReceiving > 0) {
                                                      if (NumberReceiving > 0) {
                                                        if (NumberReceiving > 0) {
                                                          if (NumberReceiving > 0) {
                                                            if (NumberReceiving > 0) {
                                                              if (NumberReceiving > 0) {
                                                                if (NumberReceiving > 0) {
                                                                  if (NumberReceiving > 0) {
                                                                    if (NumberReceiving > 0) {
                                                                      if (NumberReceiving > 0) {
                                                                        if (NumberReceiving > 0) {
                                                                          if (NumberReceiving > 0) {
                                                                            if (NumberReceiving > 0) {
                                                                              if (NumberReceiving > 0) {
                                                                                if (NumberReceiving > 0) {
                                                                                  if (NumberReceiving > 0) {
 Adam */
```
Imagine having 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
...fun as [your favorite programming language]  

This is our motivation for Chapel
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
FIVE KEY CHARACTERISTICS OF CHAPEL

1. **compiled**: to generate the best performance possible
2. **statically typed**: to avoid simple errors after hours of execution
3. **interoperable**: with C, Fortran, Python, …
4. **portable**: runs on laptops, clusters, the cloud, supercomputers
5. **open-source**: to reduce barriers to adoption and leverage community contributions
OUTLINE

• What is Chapel, and Why?
• Chapel Benchmarks and Apps
• Intro to Chapel, by Example
• Applications of Chapel
• Wrap-up
CHAPEL BENCHMARKS AND APPS
FOR DESKTOP BENCHMARKS, CHAPEL IS COMPACT AND FAST

[plot generated by summarizing data from https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html as of Feb 8, 2023]
FOR DESKTOP BENCHMARKS, CHAPEL IS COMPACT AND FAST (ZOOMED)

(plot generated by summarizing data from [https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html](https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html) as of Feb 8, 2023)
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;
```

HPCC RA: MPI KERNEL

```
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.

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)
INTRODUCTION TO CHAPEL,
BY EXAMPLE
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 CONCERNS FOR SCALABLE PARALLEL COMPUTING

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

![Diagram of locales with processor cores and memory]
BASIC FEATURES FOR LOCALITY

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

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

on Locales[1] {
    var B: [1..2, 1..2] real;
    B = 2 * A;
}
```

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

Variables are stored using the memory local to the current task

on-clauses move tasks to other locales

remote variables can be accessed directly

This is a serial, but distributed computation
basic-for.chpl

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

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

for loc in Locales {
    on loc {
        var B = A;
    }
}
MIXING LOCALITY WITH TASK PARALLELISM

```chpl
import basics-coforall.chpl;

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

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

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

The coforall loop creates a parallel task per iteration.

This results in a parallel distributed computation.
ARRAY-BASED PARALLELISM AND LOCALITY

basics-distarr.chpl

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

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

use BlockDist;

var D = Block.createDomain({1..2, 1..2});
var B: [D] real;

B = A;

Chapel also supports distributed domains (index sets) and arrays

They also result in parallel distributed computation

Locale 0

Locale 1

Locale 2

Locale 3
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:**

```
A = B + C * \alpha
```
**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 showing parallel computation of A, B, C with vector addition and scalar multiplication]
**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):

---

![Diagram](image-url)
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):

![Diagram showing parallel operations on vectors A, B, C, and scalar α]
STREAM TRIAD: SHARED MEMORY

```
stream-epl.chpl

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

'config' declarations support command-line overrides

```
$ chpl stream-epl.chpl
$ ./stream-epl
$ ./stream-epl --n=10 --alpha=3.0
```

compile the program
run with the default values
override those values
STREAM TRIAD: SHARED MEMORY

```chpl
stream-ep.chpl

config const n = 1_000_000,
    alpha = 0.01;

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

- **Define the configuration constants**
- **Declare variables**
- **Assignment statement**

---

**So far, this is simply a multi-core program**

Nothing refers to remote locales, explicitly or implicitly.
STREAM TRIAD: DISTRIBUTED MEMORY (EP VERSION)

```chpl
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)

```chpl
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

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

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

STREAM Performance (GB/s)

RA Performance (GUPS)

better
STREAM TRIAD: MPI + OPENMP VS. CHAPEL

STREAM Performance (GB/s)

Locales (x 36 cores / locale)

GB/s

0 5000 10000 15000 20000 25000 30000

16 32 64 128 256

MPI+OpenMP
Chapel EP
Chapel Global

better
KEY CONCERNS FOR SCALABLE PARALLEL COMPUTING

1. **parallelism**: What 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:** What 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

![Diagram](image)
OAK RIDGE NATIONAL LABORATORY’S FRONTIER SUPERCOMPUTER

- 74 HPE Cray EX cabinets
- 9,408 AMD CPUs, 37,632 AMD GPUs
- 700 petabytes of storage capacity, peak write speeds of 5 terabytes per second using Cray ClusterStor storage system
- HPE Slingshot networking cables providing 100 GB/s network bandwidth.

1.1 exaflops of performance.

Built by HPE, ORNL’s Frontier supercomputer is #1 on the TOP500.

62.68 gigaflops/watt power efficiency for ORNL’s TDS system, 52.23 gigaflops/watt power efficiency for full system.

Built by HPE, ORNL’s TDS and full system are ranked #2 & #6 on the Green500.

7.9 exaflops on the HPL-MxP benchmark (formerly HPL-AI).

 Built by HPE, ORNL’s Frontier supercomputer is #1 on the HPL-MxP list.

STREAM TRIAD: DISTRIBUTED MEMORY, CPUS ONLY

These programs are both CPU-only

Nothing refers to GPUs, explicitly or implicitly

---

stream-glbl.chpl

```chapel
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;
```

---

stream-ep.chpl

```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;
    }
}
```
STREAM TRIAD: DISTRIBUTED MEMORY, GPUS ONLY

```
stream-ep.chpl

config const n = 1_000_000,
    alpha = 0.01;

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

Use a similar ‘coforall’ + ‘on’ idiom to run a Triad concurrently on each of this locale’s GPUs

This is a GPU-only program

Nothing other than coordination code runs on the CPUs
This program uses all CPUs and GPUs across all of our compute nodes.
| STREAM TRIAD: DISTRIBUTED MEMORY, GPUS AND CPUS (REFACTOR) |

```chpl
stream-ep.chpl

config const n = 1_000_000,
    alpha = 0.01;

coforall loc in Locales {
    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
Performance vs. reference versions has become increasingly competitive over the past 4 months
APPLICATIONS OF CHAPEL
APPLICATIONS OF CHAPEL

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

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

ChOp: Chapel-based Optimization
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)
**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)
HPC Lessons From 30 Years of Practice in CFD Towards Aircraft Design and Analysis (June 4, 2021)

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

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

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

“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)
APPLICATIONS OF CHAPEL

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

CHOp: Chapel-based Optimization
INRIA, IMEC, et al.

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

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

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

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

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 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
SCALABILITY OF ARKOUDA’S ARGSORT ROUTINE

HPE Cray EX (spring 2023)
• 114,688 cores of AMD Rome
• Slingshot-11 network (200 Gb/s)
• 28 TiB of 8-byte values
• 1200 GiB/s
  – 24 seconds elapsed time

HPE Apollo (summer 2021)
• 73,728 cores of AMD Rome
• HDR Infiniband network (100 Gb/s)
• 72 TiB of 8-byte values
• 480 GiB/s
  – 2.5 minutes elapsed time

Arkouda Argsort Performance

A notable performance achievement in ~100 lines of Chapel
WRAP-UP
THE CHAPEL TEAM AT HPE
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
• targets GPUs in a vendor-neutral manner

Chapel is being used for productive parallel computing at scale
• users are reaping its benefits in practical, cutting-edge applications
• applicable to domains as diverse as physical simulations and data science

If you or your users are interested in taking Chapel for a spin, let us know!
• we’re happy to work with users and user groups to help ease the learning curve
COMING UP: CHIUW 2023

The Chapel Parallel Programming Language

CHIUW 2023
The 10th Annual Chapel Implementers and Users Workshop
June 1–2, 2023
free and online in a virtual format

- **What?** The Chapel community’s annual workshop
- **When?** June 1–2
  - one day of interactive programming
  - one day of presentations
- **Where?** Online
- **Cost?** Free

Details at: [https://chapel-lang.org/CHIUW2023.html](https://chapel-lang.org/CHIUW2023.html)
CHAPEL RESOURCES

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

Social Media:
• Twitter: @ChapelLanguage
• Facebook: @ChapelLanguage
• YouTube: http://www.youtube.com/c/ChapelParallelProgrammingLanguage

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
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
• targets GPUs in a vendor-neutral manner

Chapel is being used for productive parallel computing at scale
• users are reaping its benefits in practical, cutting-edge applications
• applicable to domains as diverse as physical simulations and data science

If you or your users are interested in taking Chapel for a spin, let us know!
• we’re happy to work with users and user groups to help ease the learning curve
THANK YOU

https://chapel-lang.org
@ChapelLanguage