



**Hewlett Packard  
Enterprise**

# **CHAPEL TUTORIAL FOR PYTHON PROGRAMMERS: PRODUCTIVITY AND PERFORMANCE IN ONE LANGUAGE**

Michelle Strout and Chapel team members

RMACC Rocky Mountain Advanced Computing Consortium

May 18, 2023

# HOW TO PARTICIPATE IN THIS TUTORIAL

- **Poll Everywhere link:** [pollev.com/michellestrout402](https://pollev.com/michellestrout402)

- There will be fun questions throughout the tutorial

- **Attempt this Online website for running Chapel code**

- Go to main Chapel webpage at <https://chapel-lang.org/>
- Click on the little ATO icon on the lower left that is above the YouTube icon

- **Using a container on your laptop**

- First, install docker or podman for your machine and then start them up
- Then, the below commands work with docker (see github README.md for podman)

```
docker pull docker.io/chapel/chapel      # takes about 5 minutes
cd ChapelForPythonProgrammersMay2023    # assuming git clone has happened
docker run --rm -v "$PWD":/myapp -w /myapp chapel/chapel chpl hello.chpl
docker run --rm -v "$PWD":/myapp -w /myapp chapel/chapel ./hello
```

See

<https://github.com/mstrout/ChapelForPythonProgrammersMay2023> for more info and for example code.



# CHAPEL PROGRAMMING LANGUAGE

---

Chapel is a general-purpose programming language that provides **ease of parallel programming, high performance, and portability.**

And is being used in applications in various ways:

**refactoring** existing codes,

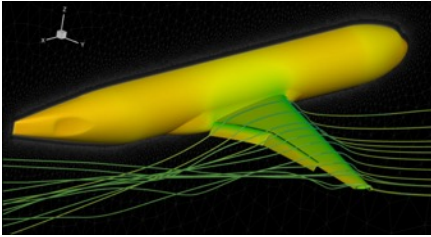
**developing** new codes,

serving high performance to Python codes (**Chapel server with Python client**), and

**providing distributed and shared memory parallelism** for existing codes.

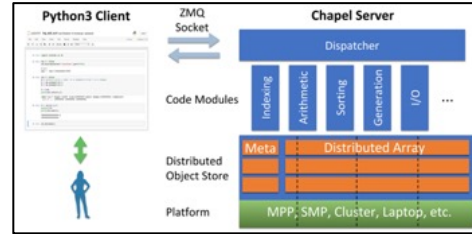


# APPLICATIONS OF CHAPEL



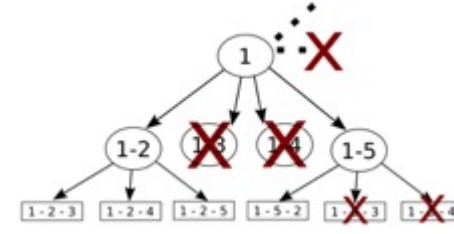
## CHAMPS: 3D Unstructured CFD

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



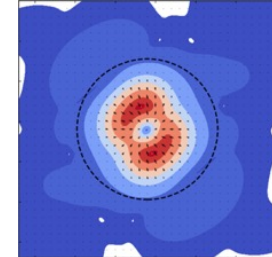
## Arkouda: Interactive Data Science at Massive Scale

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



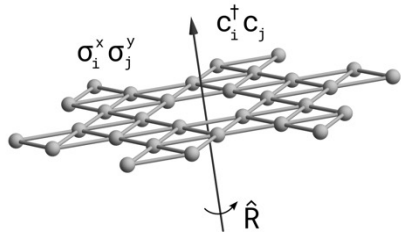
## ChOp: Chapel-based Optimization

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



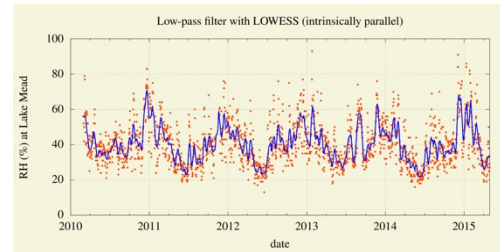
## ChpUltra: Simulating Ultralight Dark Matter

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



## Lattice-Symmetries: a Quantum Many-Body Toolbox

Tom Westerhout  
*Radboud University*



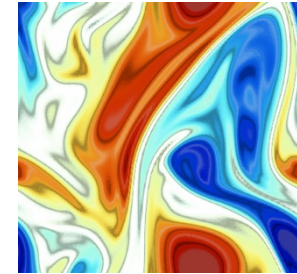
## Desk dot chpl: Utilities for Environmental Eng.

Nelson Luis Dias  
*The Federal University of Paraná, Brazil*



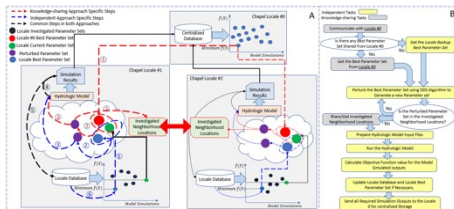
## RapidQ: Mapping Coral Biodiversity

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



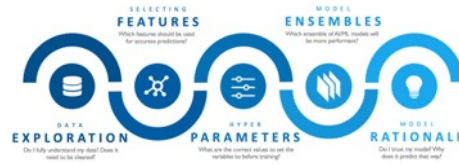
## ChapQG: Layered Quasigeostrophic CFD

Ian Grooms and Scott Bachman  
*University of Colorado, Boulder et al.*



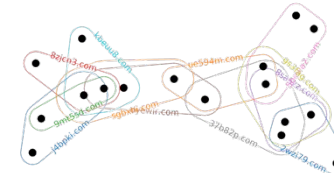
## Chapel-based Hydrological Model Calibration

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*



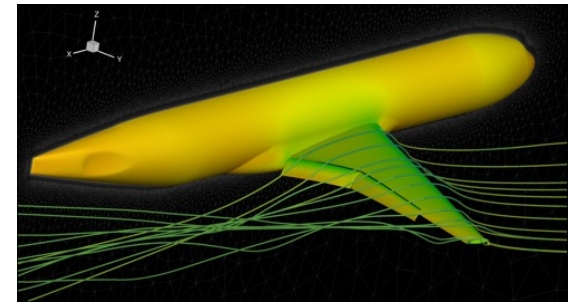
## Your Application Here?



# HIGHLIGHTS OF CHAPEL USAGE

## **CHAMPS:** Computational Fluid Dynamics framework for airplane simulation

- Professor Eric Laurendeau's team at Polytechnique Montreal
- Performance: achieves competitive results w.r.t. established, world-class frameworks from Stanford, MIT, etc.
- Programmability: *"We ask students at the master's degree to do stuff that would take 2 years and they do it in 3 months."*



## **Arkouda:** data analytics framework (<https://github.com/Bears-R-Us/arkouda>)

- Mike Merrill, Bill Reus, et al., US DOD
- Python front end client, Chapel server that processes dozens of terabytes in seconds
- April 2023: 1200 GiB/s for argsort on an HPE EX system




## **Recent Journal Paper on using Chapel for calibrating hydrologic models**

- Marjan Asgari et al, "Development of a knowledge-sharing parallel computing approach for calibrating distributed watershed hydrologic models", Environmental Modeling and Software.
- They report super-linear speedup



# INTRODUCTIONS

- Let's take some time to introduce ourselves

- Michelle Strout
  - Chapel team leader
  - Affiliate faculty in the Department of Computer Science at UArizona
- Current Chapel team 
  - Tech Lead: Brad Chamberlain
  - Visiting Scholar from NCAR: Scott Bachman

- **Participants, tell us some about yourself**

- Your institution
- Proudest HPC accomplishment
- Biggest HPC challenge



# LEARNING OBJECTIVES FOR TODAY'S TUTORIAL

---

- Compile and run Chapel programs in a web browser and/or on your laptop
- Familiarity with the Chapel execution model including how to run codes in parallel on a single node, across nodes, and both
- Experiment compiling and running provided Chapel code examples
  - k-mer counting (bioinformatics application)
  - Processing files in parallel using parallelism over multiple nodes and threads
  - Solving a diffusion PDE (partial differential equation)
  - Image processing (coral reef diversity example)
  - Same code can be compiled to run on a multi-core CPU AND a GPU
- Where to get help and how you can participate in the Chapel community



# HOW TO PARTICIPATE IN THIS TUTORIAL

- **Poll Everywhere link:** [pollev.com/michellestrout402](https://pollev.com/michellestrout402)

- There will be fun questions throughout the tutorial

- **Attempt this Online website for running Chapel code**

- Go to main Chapel webpage at <https://chapel-lang.org/>
- Click on the little ATO icon on the lower left that is above the YouTube icon



- **Using a container on your laptop**

- First, install docker or podman for your machine and then start them up
- Then, the below commands work with docker (see github README.md for podman)

```
docker pull docker.io/chapel/chapel      # takes about 5 minutes
cd ChapelForPythonProgrammersMay2023    # assuming git clone has happened
docker run --rm -v "$PWD":/myapp -w /myapp chapel/chapel chpl hello.chpl
docker run --rm -v "$PWD":/myapp -w /myapp chapel/chapel ./hello
```

Try one of these options for  
using Chapel



# Which option did you choose to try out Chapel during this tutorial?

Attempt This Online

Container on your  
laptop

Doing the polls and  
watching a neighbor

Learning from the  
examples in the slides

# PARALLELISM ACROSS NODES AND WITHIN NODES

- **Parallel hello world**

- ExamplesInSlides/hellopar.chpl

- **Key concepts**

- 'coforall'
- configuration constants, 'config const'
- range values, '0..#tasksPerLocale'
- 'writeln'
- inline comments start with '//'

```
// can be set on the command line with --tasksPerLocale=2
config const tasksPerLocale = 1;

// parallel loops over nodes and then over threads
coforall loc in Locales on loc {
    coforall tid in 0..#tasksPerLocale {

        writeln("Hello world! ",
                "(from task ", tid,
                " of ", tasksPerLocale,
                " on locale ", here.id,
                " of ", numLocales, ")" );

    }
}
```

# CHAPEL EXECUTION MODEL AND TERMINOLOGY: LOCALES

- Locales can run tasks and store variables
  - Think “compute node” on a parallel system
  - User specifies number of locales on executable’s command-line

```
prompt> ./myChapelProgram --numLocales=4 # or '-nl 4'
```

**Locales** array :



User's code starts running as a single task on locale 0

# TASK-PARALLEL “HELLO WORLD”

helloTaskPar.chpl

```
const numTasks = here.numPUs();  
coforall tid in 1..numTasks do  
    writef("Hello from task %n of %n on %s\n",  
          tid, numTasks, here.name);
```



# TASK-PARALLEL “HELLO WORLD”

helloTaskPar.chpl

```
const numTasks = here.numPUs();  
coforall tid in 1..numTasks do  
    writef("Hello from task %n of %n on %s\n",  
          tid, numTasks, here.name);
```

‘here’ refers to the locale on which we’re currently running

how many processing units (think “cores”) does my locale have?

what’s my locale’s name?



# TASK-PARALLEL “HELLO WORLD”

helloTaskPar.chpl

```
const numTasks = here.numPUs();  
coforall tid in 1..numTasks do  
    writef("Hello from task %n of %n on %s\n",  
          tid, numTasks, here.name);
```

a 'coforall' loop executes each iteration as an independent task

```
prompt> chpl helloTaskPar.chpl  
prompt> ./helloTaskPar  
Hello from task 1 of 4 on n1032  
Hello from task 4 of 4 on n1032  
Hello from task 3 of 4 on n1032  
Hello from task 2 of 4 on n1032
```



# TASK-PARALLEL “HELLO WORLD”

helloTaskPar.chpl

```
const numTasks = here.numPUs();  
coforall tid in 1..numTasks do  
    writef("Hello from task %n of %n on %s\n",  
          tid, numTasks, here.name);
```

```
prompt> chpl helloTaskPar.chpl  
prompt> ./helloTaskPar  
Hello from task 1 of 4 on n1032  
Hello from task 4 of 4 on n1032  
Hello from task 3 of 4 on n1032  
Hello from task 2 of 4 on n1032
```

**So far, this is a shared-memory program**

Nothing refers to remote locales,  
explicitly or implicitly

# TASK-PARALLEL “HELLO WORLD” (DISTRIBUTED VERSION)

helloTaskPar.chpl

```
coforall loc in Locales {  
  on loc {  
    const numTasks = here.numPUs();  
    coforall tid in 1..numTasks do  
      writef("Hello from task %n of %n on %s\n",  
            tid, numTasks, here.name);  
  }  
}
```



# TASK-PARALLEL “HELLO WORLD” (DISTRIBUTED VERSION)

helloTaskPar.chpl

```
coforall loc in Locales {  
  on loc {  
    const numTasks = here.numPUs();  
    coforall tid in 1..numTasks do  
      writef("Hello from task %n of %n on %s\n",  
            tid, numTasks, here.name);  
  }  
}
```

create a task per locale  
on which the program is running

have each task run 'on' its locale

then print a message per core,  
as before

```
prompt> chpl helloTaskPar.chpl  
prompt> ./helloTaskPar -nl=4  
Hello from task 1 of 4 on n1032  
Hello from task 4 of 4 on n1032  
Hello from task 1 of 4 on n1034  
Hello from task 2 of 4 on n1032  
Hello from task 1 of 4 on n1033  
Hello from task 3 of 4 on n1034  
Hello from task 1 of 4 on n1035  
...
```

# Which Chapel code does the same thing as this python code?

```
x = 42
str = "answer"
print(str, " = ", x)
```

A

B

C

A

```
var x = 42;
var str = "answer";
writeln(str, " = ", x);
```

B

```
config const tasksPerLocale = 2;
coforall tid in 0..#tasksPerLocale {
  var message = "answer = ";
  message += 42:string;
  writeln(message);
}
```

C

```
var x = 42;
var str = "answer";
coforall loc in Locales {
  on loc {
    writeln(x, " = ", str);
  }
}
```

# K-MER COUNTING FROM BIOINFORMATICS

kmer.chpl

```
use Map, IO;

config const infilename = ("kmer_large_input.txt");
config const k = 4;

var sequence, line : string;
var f = open(infilename, ioMode.r);
var infile = f.reader();
while infile.readLine(line) {
    sequence += line.strip();
}
infile.close();

var nkmerCounts : map(string, int);

for ind in 0..<(sequence.size-k) {
    nkmerCounts[sequence[ind..#k]] += 1;
}
```

'Map' and 'IO' are two of the standard libraries provided in Chapel. A 'map' is like a dictionary in python.

'config const' indicates a configuration constant, which result in built-in command-line parsing

Reading all of the lines from the input file into the string 'sequence'.

The variable 'nkmerCounts' is being declared as a dictionary mapping strings to ints

Counting up each kmer in the sequence

# EXPERIMENTING WITH THE K-MER EXAMPLE

- **Some things to try out with 'ExamplesInSlides/kmer.chpl'**

```
chpl kmer.chpl
```

```
./kmer
```

```
./kmer --k=10
```

```
./kmer --infilename="kmer.chpl"
```

```
./kmer --k=10 --infilename="kmer.chpl"
```

See

<https://github.com/mstrout/ChapelForPythonProgrammersMay2023> for  
*more info and for example code.*

```
# can change k
```

```
# can change the infilename
```

```
# can change both
```





# What Chapel code does the same thing as this python code?

```
# read in a file into a list of strings
# where each string has a line with the newline at the end removed
with open("filename.txt") as file:
    lines = [line.strip() for line in file]

print(lines)
```

A

B

C

A

```
// declare a dictionary/map to store the count per kmer
var nkmerCounts : map(string, int);

// count up the number of times each kmer occurs
for ind in 0..<(sequence.size-k) {
    nkmerCounts[sequence[ind..#k]] += 1;
}
```

B

```
var sequence, line : string;
var f = open(infile, ioMode.r);
var infile = f.reader();
while infile.readLine(line) {
    sequence += line.strip();
}
```

C

```
use List, IO;

var line : string;
var lines : list(string);
var infile = open("filename.txt", ioMode.r).reader();
while infile.readLine(line) {
    lines.append(line.strip());
}

writeln(lines);
```

## 2D DIFFUSION PARTIAL DIFFERENTIAL EQUATION EXAMPLE

- See 'ExamplesInSlides/diffusion.chpl' in the repository
- Some things to try out with 'diffusion.chpl'

```
chpl diffusion.chpl  
./diffusion
```

```
--xLen=4 --yLen=4 --nx=61 --ny=61 # doubles the size of the domain  
                                     # along each dimension, keeping the  
                                     # density of points the same  
  
--nu=0.025                           # reduces the fluid viscosity  
  
--nt=100                             # twice as many timesteps
```

See

<https://github.com/mstrout/ChapelForPythonProgrammersMay2023> for  
more info and for example code.



**Based on this code, we can conclude that Chapel can do summation, min, and max reductions over lists and arrays.**

```
var oneDimArray : [1..4] int = [20, 30, 40, 50];
writeln("oneDimArray = ", oneDimArray);
writeln("+ reduce oneDimArray = ", + reduce oneDimArray);

use List;
var aList : list(real) = new list([50, 20, 30, 40]);

writeln("aList = ", aList);
writeln("min reduce aList = ", min reduce aList);
```

---

True

False

# WRITING OUT EVERYTHING EXAMPLE

---

- See 'ExamplesInSlides/writelnExamples.chpl' in the repository
- **Key points**
  - The Chapel compiler provides default 'writeThis' routines for every standard library and user-defined datatype
  - This helps enable "printf" debugging through the use of 'writeln' calls

See

<https://github.com/mstrout/ChapelForPythonProgrammersMay2023> for more info and for example code.





# ANALYZING MULTIPLE FILES USING PARALLELISM

parfilekmer.chpl

```
use FileSystem;
config const dir = "DataDir";
var fList = findFiles(dir);
var filenames =
    Block.createArray(0..#fList.size, string);
filenames = fList;

// per file word count
forall f in filenames {
    ...
    // code from kmer.chpl
    ...
}
```

```
prompt> chpl --fast parfilekmer.chpl
prompt> ./parfilekmer
prompt> ./parfilekmer -nl 4
```

Shared and Distributed-Memory  
Parallelism using forall, a distributed  
array, and command line options to  
indicate number of locales



# PROCESSING FILES IN PARALLEL

- See 'ExamplesInSlides/parfilekmer.chpl' in the repository
- Some things to try out with 'parfilekmer.chpl'

```
# put more and bigger files into DataDir/  
# or set the config const dir to something else  
chpl parfilekmer.html  
./parfilekmer --dir="SomethingElse/"
```

```
./parfilekmer --k=10
```

```
# can also change k
```

See

<https://github.com/mstrout/ChapelForPythonProgrammersMay2023> for more info and for example code.



# What does the following Chapel code do?

```
var array = [1,2,3,4];
var result = "";
for num in array {
    result += num:string + ":";
}
result = result[0..#result.size-1];
var sum : int;
for substr in result.split(":") {
    sum += substr : int;
}
writeln("sum = ", sum);
```

Converts an array of strings to integers and then prints their sum.

Converts an array of integers to strings, concatenates them with a colon in-between, then splits that string and sums up resulting integers.

Sums an array of integers and then concatenates them into a string.

# IMAGE PROCESSING EXAMPLE

- **See 'image\_analysis\_example/' subdirectory in the repository**

- Coral reef diversity analysis written by Scott Bachman
- Calls out to libpng to read and write PNG files
- Uses distributed and shared memory parallelism

- **'image\_analysis\_example/README.md' explains how to compile and run it**

- **Some things to try out when running 'main'**

```
./main -nl 4 --inname=Roatan_benthic_r3_gray.png --outname=out1.png --radius=10
```

```
./main -nl 4 --inname=Roatan_benthic_r3_gray.png --outname=out2.png --radius=100
```

```
# Can also change the number of locales, but only up to the -N number given to salloc
```

See

<https://github.com/mstrout/ChapelForPythonProgrammersMay2023> for more info and for example code.



# GPU SUPPORT IN CHAPEL

- **Generate code for GPUs**

- Support for NVIDIA and AMD GPUs
- Exploring Intel support

- **Chapel code calling CUDA examples**

- <https://github.com/chapel-lang/chapel/blob/main/test/gpu/interop/stream/streamChpl.chpl>
- <https://github.com/chapel-lang/chapel/blob/main/test/gpu/interop/cuBLAS/cuBLAS.chpl>

- **Key concepts**

- Using the 'locale' concept to indicate execution and data allocation on GPUs
- 'forall' and 'foreach' loops will be converted to kernels
- Arrays declared in 'on here.gpus[i]' blocks are allocated on the GPU

- **For more info...**

- <https://chapel-lang.org/docs/technotes/gpu.html>

```
use GpuDiagnostics;
startGpuDiagnostics();

var operateOn =
    if here.gpus.size>0 then here.gpus
    else [here,];

// Same code can run on GPU or CPU
coforall loc in operateOn do on loc {
    var A : [1..10] int;
    foreach a in A do a+=1;
    writeln(A);
}

stopGpuDiagnostics();
writeln(getGpuDiagnostics());
```

# STREAM TRIAD: SHARED MEMORY

stream-ep.chpl

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

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

Declare three arrays of size 'n'

Whole-array operations compute  
Stream Triad in parallel

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

Nothing refers to remote locales (nodes),  
explicitly or implicitly

# STREAM TRIAD: DISTRIBUTED MEMORY

stream-ep.chpl

```
config var n = 1_000_000,  
          alpha = 0.01;  
  
coforall loc in Locales {  
  on loc {  
    var A, B, C: [1..n] real;  
    A = B + alpha * C;  
  }  
}
```

'coforall' loops execute each iteration as an independent task

the array of locales (nodes) on which this program is running

have each task run 'on' its locale

then run multi-core Stream, as before

**This is a CPU-only program**

Nothing refers to GPUs, explicitly or implicitly

# STREAM TRIAD: DISTRIBUTED MEMORY, GPUS ONLY

stream-ep.chpl

```
config var 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



# STREAM TRIAD: DISTRIBUTED MEMORY, GPUS AND CPUS

stream-ep.chpl

```
config var n = 1_000_000,  
          alpha = 0.01;  
  
coforall loc in Locales {  
  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

**This program uses all CPUs and GPUs  
across all of your compute nodes**

# STREAM TRIAD: DISTRIBUTED MEMORY, GPUS AND CPUS (REFACTOR)

stream-ep.chpl

```
config var 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;  
}
```

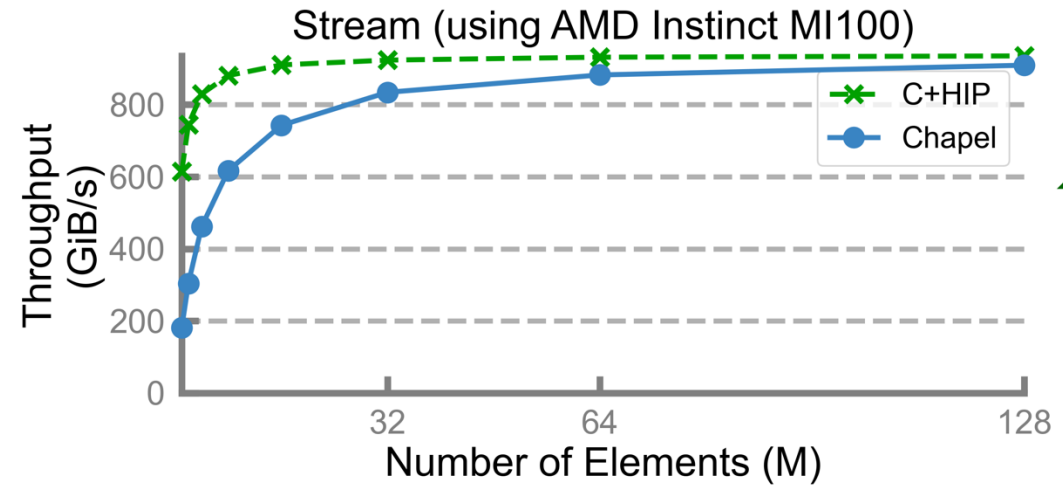
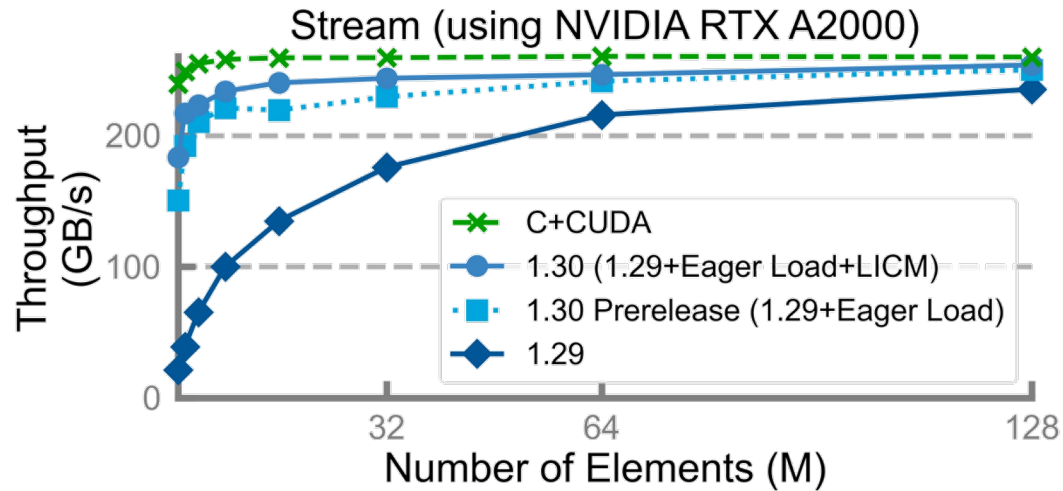
'cobegin { ... }' creates a task  
per child statement

one task runs our multi-GPU triad

the other runs the multi-CPU triad

the compiler creates CPU and GPU  
versions of this procedure

# STREAM TRIAD: PERFORMANCE VS. REFERENCE VERSIONS



**Performance vs. reference versions has become increasingly competitive over the past 4 months**



# OTHER CHAPEL EXAMPLES

---

- **Wavelet example by Jeremiah Corrado included in the github repository**
  - Slides and Code:  
[https://github.com/mstrout/ChapelForPythonProgrammersMay2023/tree/main/wavelet\\_example](https://github.com/mstrout/ChapelForPythonProgrammersMay2023/tree/main/wavelet_example)
- **Primers**
  - <https://chapel-lang.org/docs/primers/index.html>
- **Blog posts for Advent of Code**
  - <https://chapel-lang.org/blog/index.html>
- **Test directory in main repository**
  - <https://github.com/chapel-lang/chapel/tree/main/test>



# TUTORIAL SUMMARY

---

- **Takeaways**

- Chapel is a general-purpose programming language designed to leverage parallelism
- It is being used in some large production codes
- Our team is responsive to user questions and would enjoy having you participate in our community

- **How to get more help**

- Ask us questions on discourse, gitter, or stack overflow
- Also feel free to email me at [michelle.strout@hpe.com](mailto:michelle.strout@hpe.com)

- **Engaging with the community**

- Share your sample codes with us and your research community!
- Join us at our free, virtual workshop in June, <https://chapel-lang.org/CHI UW.html>



# CHAPEL RESOURCES

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


- (points to all other resources)

## Social Media:

- Twitter: [@ChapelLanguage](https://twitter.com/ChapelLanguage)
- Facebook: [@ChapelLanguage](https://facebook.com/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>



### 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

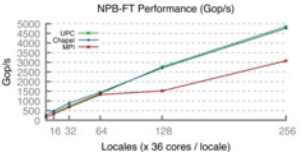
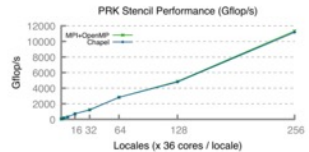
**Chapel Characteristics**

- **productive:** code tends to be similarly readable/writable as Python
- **scalable:** runs on laptops, clusters, the cloud, and HPC systems
- **fast:** performance *competes with or beats* C/C++ & MPI & OpenMP
- **portable:** 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 [blog-length](#) or [chapter-length](#) introduction to Chapel
- learn about [projects powered by Chapel](#)
- check out [performance highlights](#) like these:



PRK Stencil Performance (Gflop/s)

NPB-FT Performance (Gop/s)

- 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 default

forall i in {1..n} dmapped Cyclic(startIdx=1) do
  writeln("Hello from iteration ", i, " of ", n, " running on node ", here.id);
```