Michael Ferguson
February 7, 2024

CHAPEL AND OPEN
PRODUCTIVE PARALLEL
COMPUTING AT SCALE
OUTLINE

- Motivation: Sorting
- What is Chapel?
- Comparing to Other Languages
- What do Chapel users say?
- Applications written in Chapel
- Demos and Q&A
- Wrap-Up
SORTING IN STANDARD LIBRARIES

Parallelism is Essential to Performance
SORTING IN STANDARD LIBRARIES

• Most standard libraries include a ‘sort’ routine

• It’s an essential building block
  • supports GroupBy in data analysis tools such as Arkouda or Pandas
  • supports indexing, searching, many other algorithms

• Let’s investigate the performance of standard library ‘sort’ routines

• Why focus on standard libraries? They
  • are more likely to be used in practice than other implementations
  • show what a programming language has to offer
  • set an example for libraries
  • form a common language for programmers
THE BENCHMARK

- Sort 1GiB of 64-bit integers
  - i.e. 128*1024*1024 integers

- Use random values
THE TEST SYSTEM

My PC!

CPU: AMD Ryzen 9 7950X
• 4.5GHz, 16 cores, 32 threads

Memory: 64 GiB of DDR5 memory
• 5200MT/s CL40

Motherboard:
• Gigabyte X670 Aorus Elite AX

OS: Ubuntu 23.10

Total Cost: ~ $1500
import random
import time

# generate an array of random integers
n = 128*1024*1024
array = [random.randint(0, 0xffffffffffffffff) for _ in range(n)]

start = time.time()
# use the standard library to sort the array
array.sort()
stop = time.time()

# print out the performance achieved
elapsed = stop-start
print("Sorted", n, "elements in", elapsed, "seconds")
print(n/elapsed/1_000_000, "million elements sorted per second")
use Time, Sort, Random;

// generate an array of random integers
config const n = 128*1024*1024;

var A: [0..<n] uint; // note: int, uint default to 64 bits
fillRandom(A);  // set the elements to random values

var timer: stopwatch;
timer.start();

// use the standard library to sort the array
sort(A);

// print out the performance achieved
var elapsed = timer.elapsed();
writeln("Sorted ", n, " elements in ", elapsed, " seconds");
writeln(n/elapsed/1_000_000, " million elements sorted per second");
BOTH PROGRAMS ARE SIMPLE

How do they perform?
10 times faster than the other languages measured in this experiment.

15 times faster than C with ‘qsort’.

200 times faster than Python’s ‘sort’.
BUT I HAVE A SERVER

How does that impact things?
RESULTS ON 1 SOCKET AMD EPYC 7543: 32 CORES

- **Chapel**: 25 times faster than C with ‘qsort’
- **C**
- **C++**: 400 times faster than Python

Million 64-bit Integers Sorted per Second
RESULTS ON 2 SOCKET AMD EPYC 7763: 64 CORES

- **Chapel**: 50 times faster than C with ‘qsort’
- **C**: 1000 times faster than Python

![Bar graph showing sorting speed (Million 64-bit Integers Sorted per Second)]
WHY?

The main reason:

• Chapel used all the cores
• others used 1 core
EASY PARALLELISM

• A parallel programming language can make it easy to use parallel hardware

• A parallel standard library brings additional productivity

• Chapel is a language built for parallelism & includes a parallel standard library
WHAT IS CHAPEL?

Productive Parallel Programming
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
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);
PERFORMANCE AND PRODUCTIVITY

How does Chapel compare to other languages?
For Desktop Benchmarks

CHAPEL IS COMPACT AND FAST

Execution Time (normalized to fastest entry)
Compressed Code Size (normalized to smallest entry)

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

[Perl, PHP, Dart, Racket, Erlang, Julia, Lua, Chapel, Python, Smalltalk]
CHAPEL IS COMPACT AND FAST (ZOOMED)
For Desktop Benchmarks

Compressed Code Size (normalized to smallest entry)
[plot generated by summarizing data from https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html as of Feb 8, 2023]
Gmys et al. [1] compared productivity and performance of several programming languages when implementing parallel metaheuristics for optimization problems.

Evaluated with a dual-socket, 32-core machine.

Result: Chapel more productive in terms of performance achieved vs. lines of code vs Julia and Python+Numba.

Figure 7: Relative productivity achieved by Chapel, Julia, and Python compared to the C/OpenMP reference. Results are given for the instance nps22 and execution on 1 to 64 threads.

CHAPEL USERS

What do they say about it?
FROM OUR COMMUNITY

A Programming Language For Everybody

“It’s fast. Parallelization is really easy! I didn’t know I could get so much from my desktop until I used it [Chapel].”

Nelson Luís Dias
Professor, Environmental Engineering Department, Federal University of Paraná (Brazil)
quote from his CHIUW 2022 talk [video]
Éric Laurendeau
Professor, Department of Mechanical Engineering, Polytechnique Montréal
quote from his 2021 CHIUW Keynote [video]
APPLICATIONS OF CHAPEL

Scaling to Solve Real Problems
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)
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 Luís 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)
CHOP SUMMARY

What is it?
- Tree-based, branch and bound optimization algorithms
- Irregular tree, lots of pruning

Who did it?
- Tiago Carneiro and Nouredine Melab at the Imec - Belgium and INRIA Lille
- Open-source: [https://github.com/tcarneirop/ChOp](https://github.com/tcarneirop/ChOp)

Why Chapel?
- Found Chapel to be more productive than alternatives
  - In the 2020 publication mentioned earlier
  - And in subsequent work

From slides for "Towards Ultra-scale Optimization Using Chapel" by Tiago Carneiro (University of Luxembourg) and Nouredine Melab (INRIA Lille), CHIUW 2021
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

ChapiGQ: 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)
DATA SCIENCE IN PYTHON AT SCALE?

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
APPLICATIONS OF CHAPEL: LINKS TO USERS' TALKS (SLIDES + VIDEO)

CHAMPs: 3D Unstructured CFD
Arkouda: Interactive Data Science at Massive Scale
ChOp: Chapel-based Optimization
ChplUltra: Simulating Ultralight Dark Matter

Lattice-Symmetries: a Quantum Many-Body Toolbox
RapidQ: Mapping Coral Biodiversity
ChapQG: Layered Quasigeostrophic CFD

Your Application Here?

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

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)
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
THANK YOU

https://chapel-lang.org
@ChapelLanguage