Productive Parallel Programming with the Chapel Language

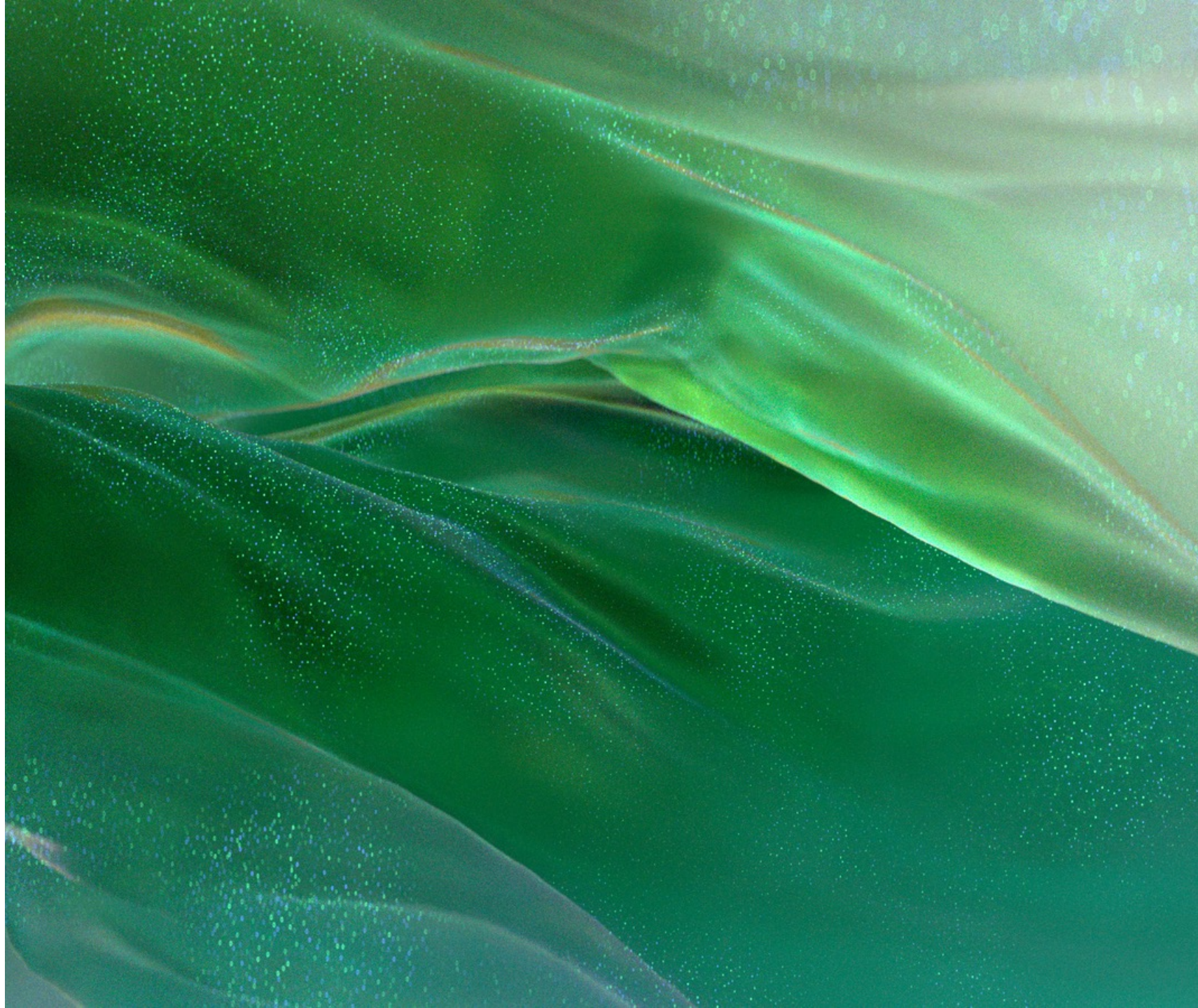
Michael Ferguson

October 11, 2024

Hewlett Packard Enterprise

Outline

- Motivating Example: Sorting
- Productivity
- Scalability
- GPU Computing
- Demos and Q&A



Motivating Example

Sorting in Standard Libraries

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

In Python

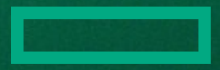
```
import random
import time

# 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")
```

In Chapel

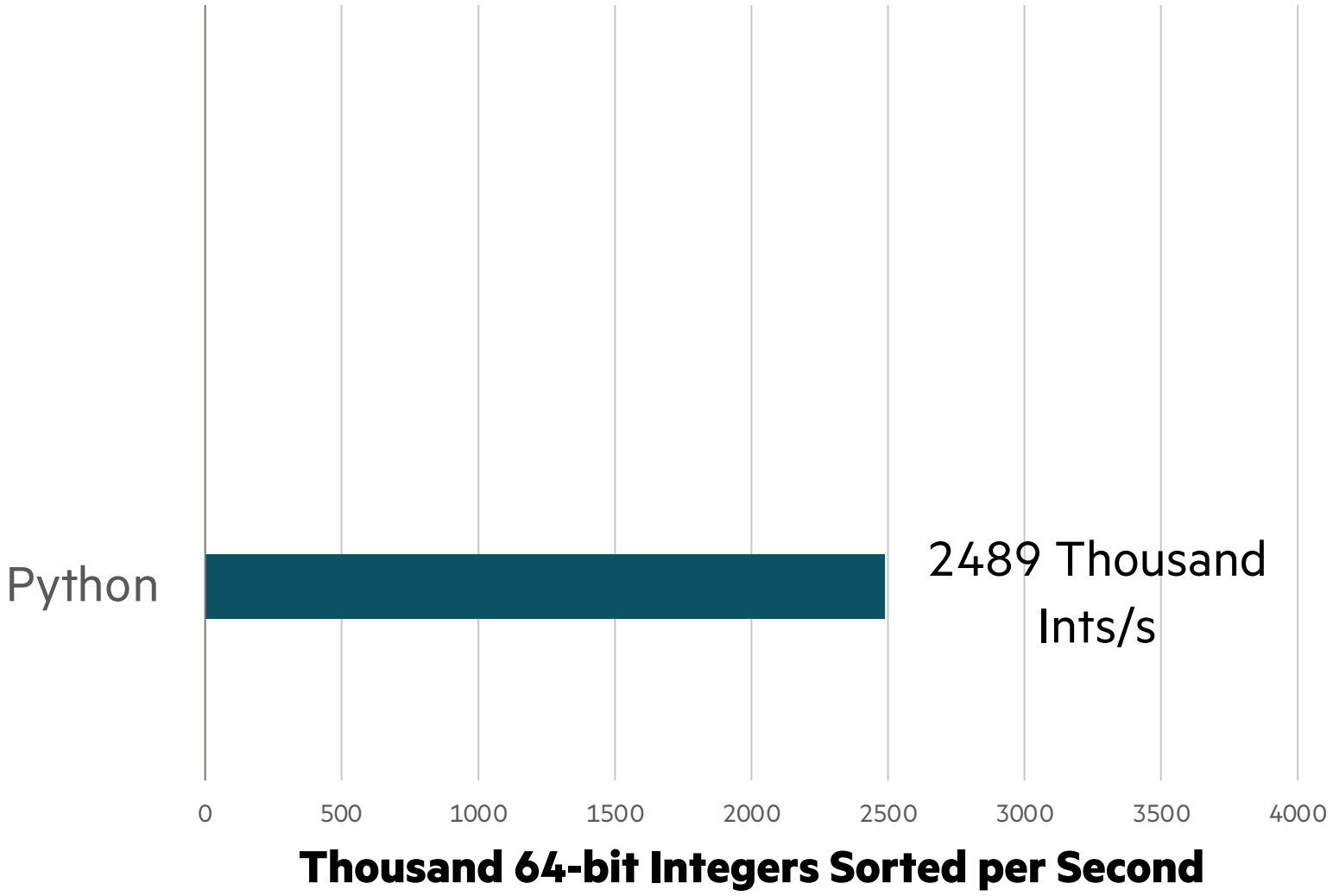
```
use Time, Sort, Random;

// 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?

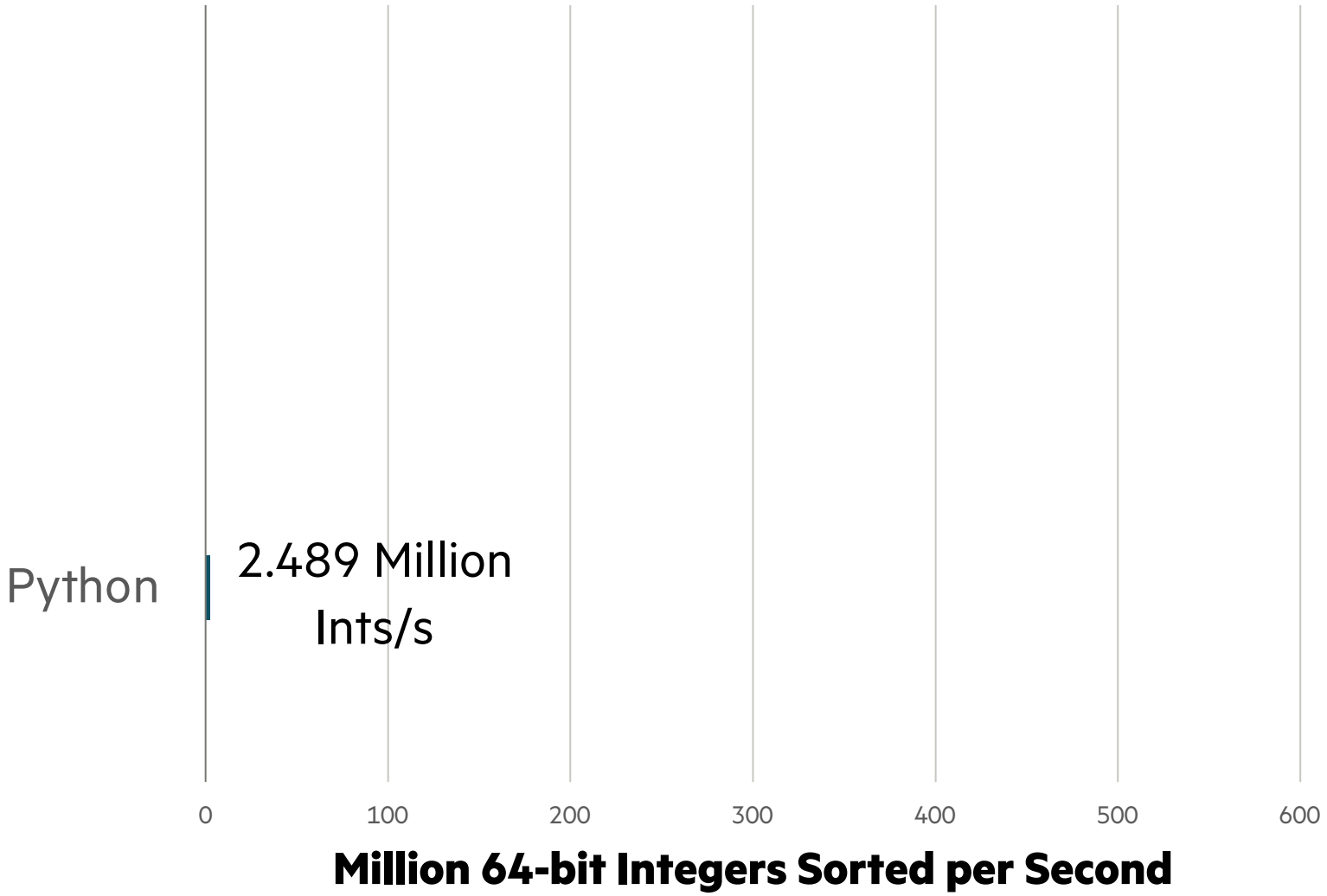


Results on the PC



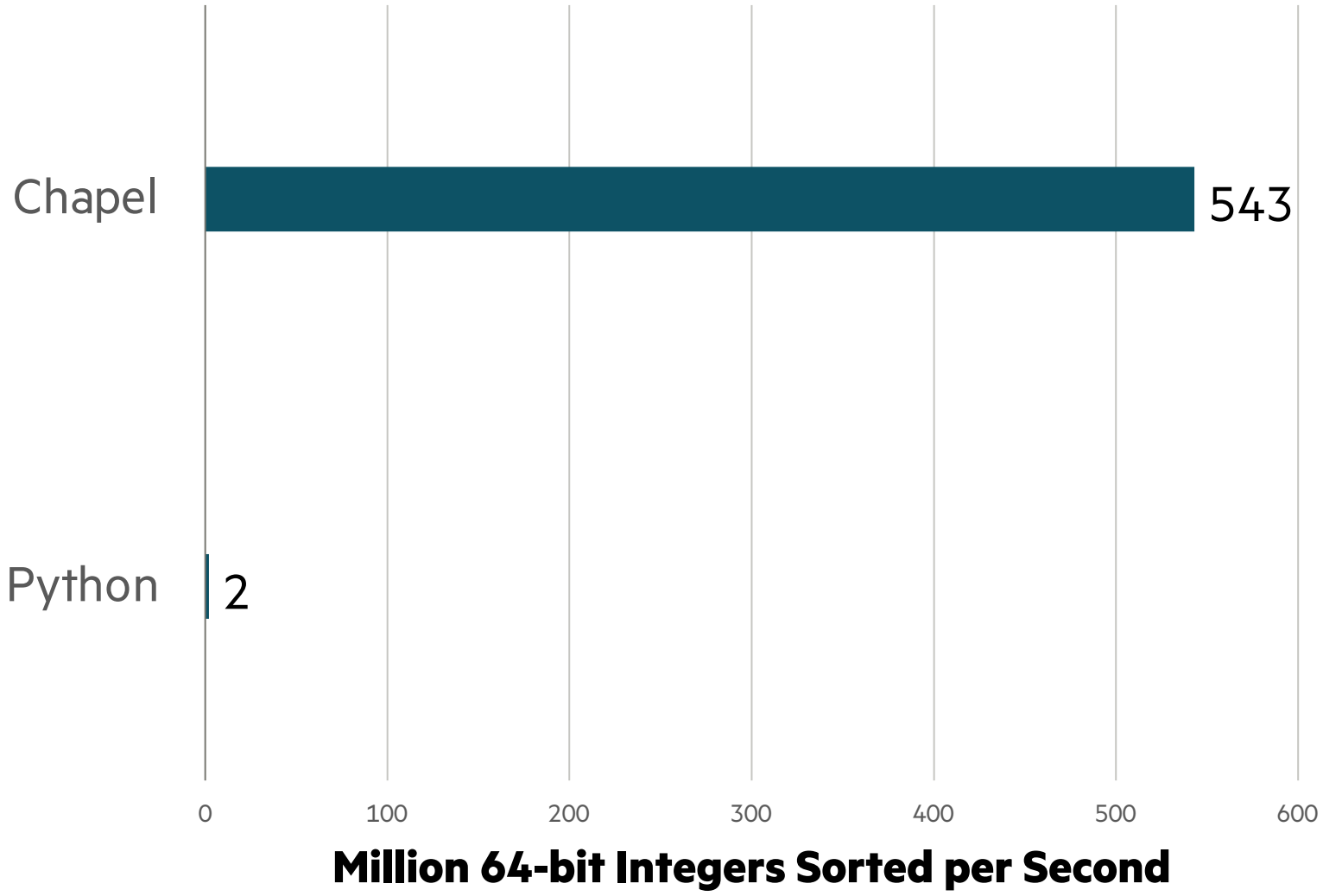
Here is the result for the Python program

Results on the PC



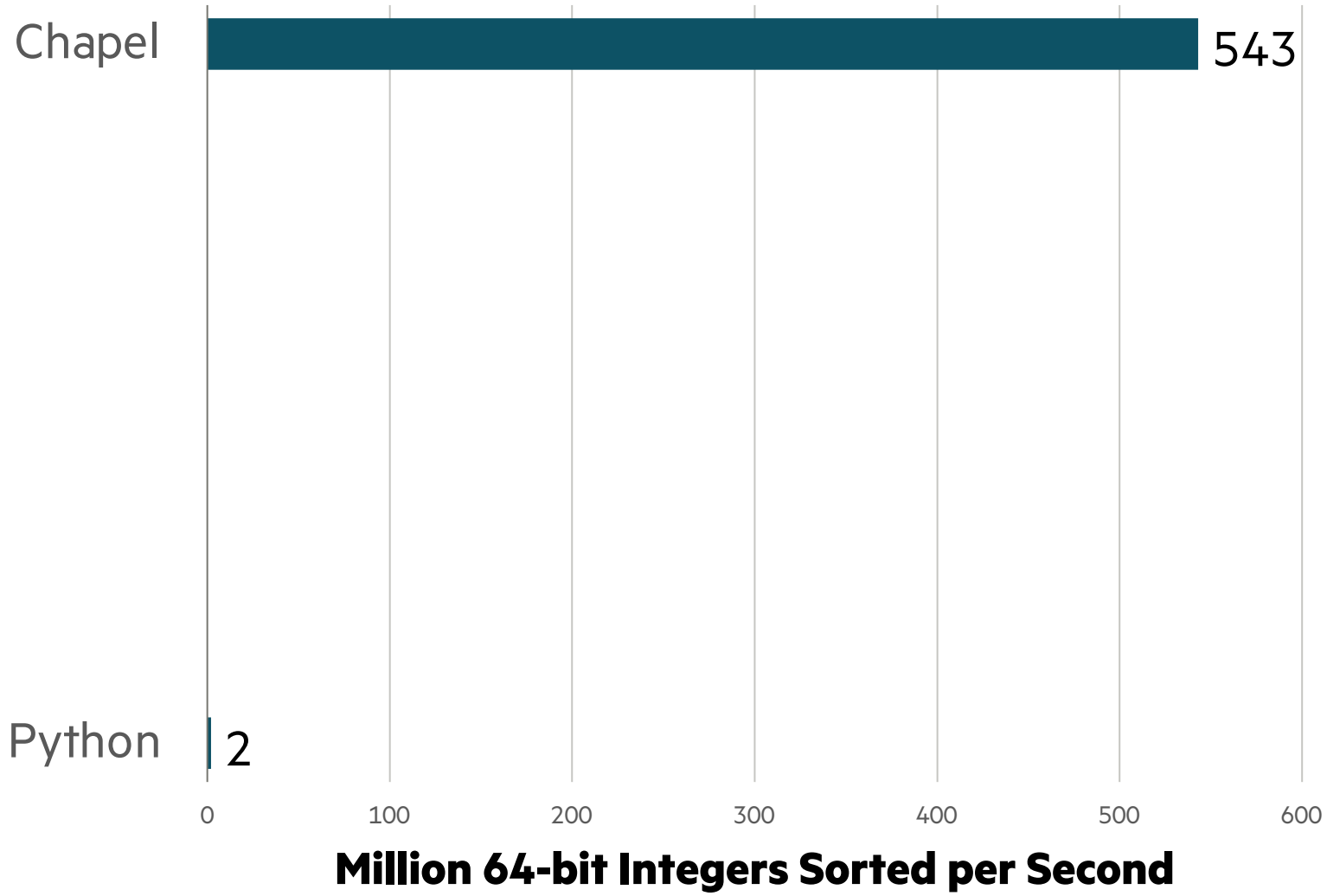
Let’s zoom out to millions

Results on the PC



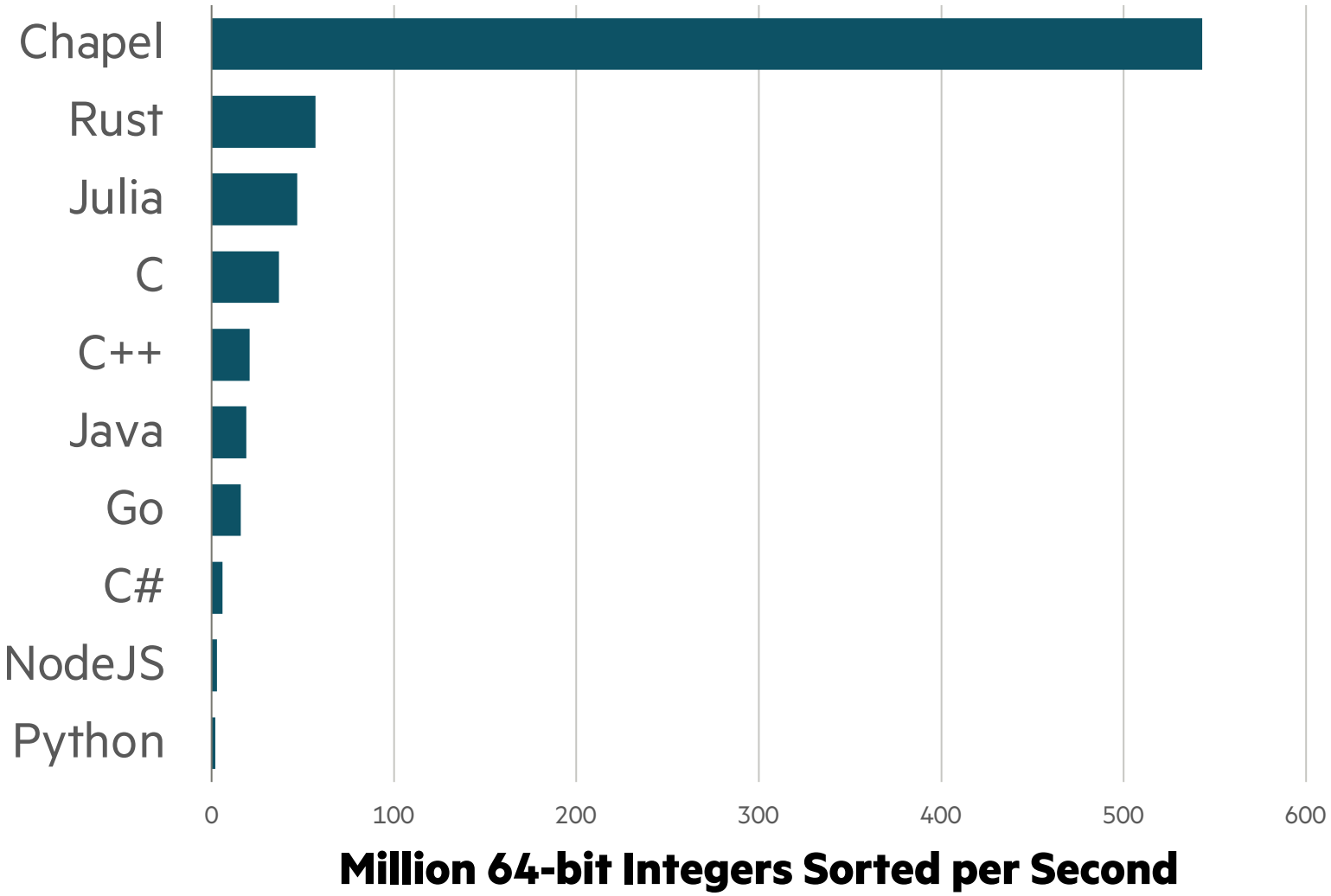
The Chapel sort is more than **200 times** faster!

Results on the PC



Let's make some room in the chart to consider other languages

Results on the PC



10 times faster

than the other languages measured in this experiment

15 times faster

than C with ‘qsort’

200 times faster

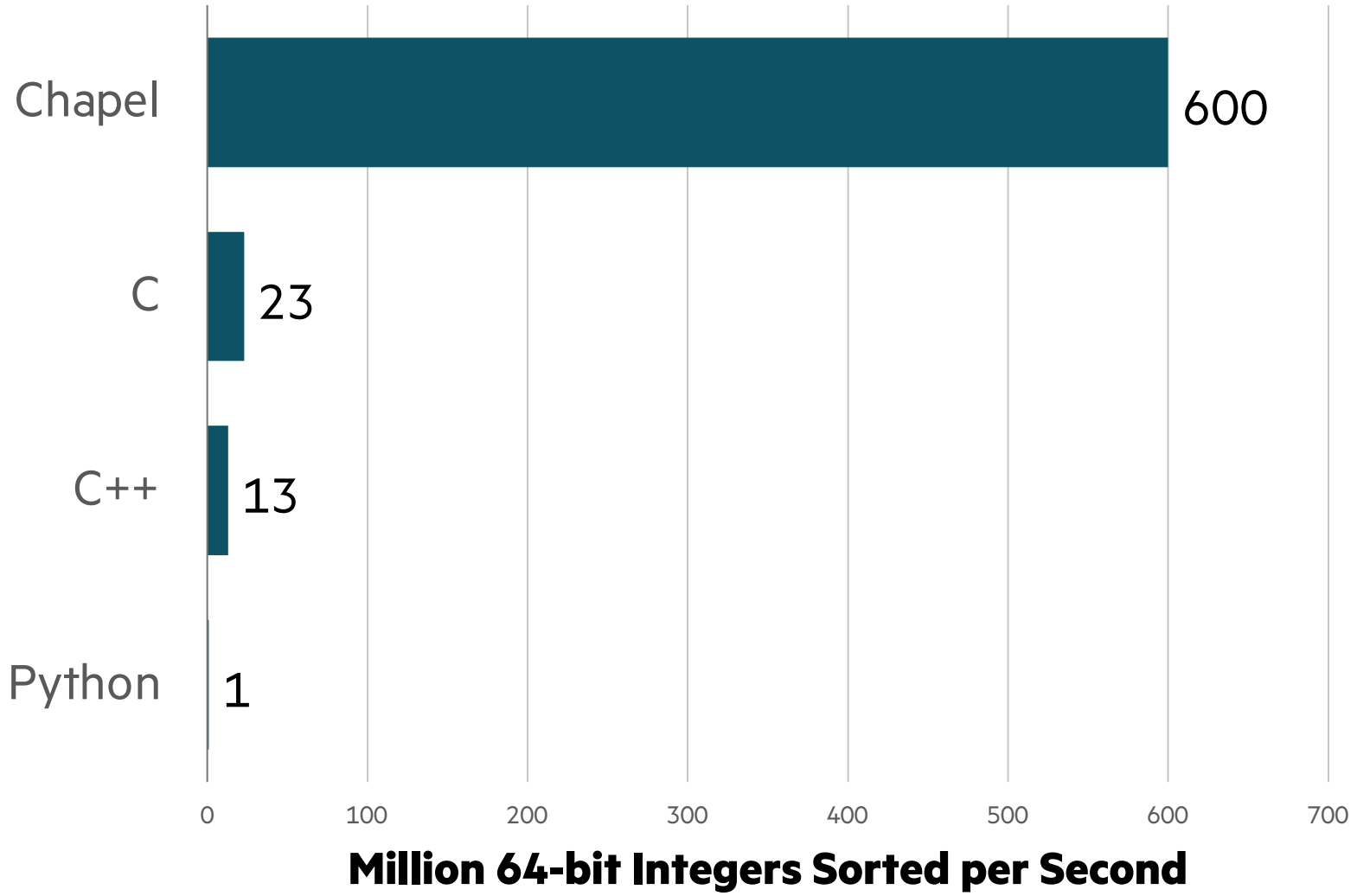
than Python’s ‘sort’

How about Server Hardware?

Server hardware is different.

How does that impact things?

Results on 1 Socket AMD EPYC 7543: 32 Cores



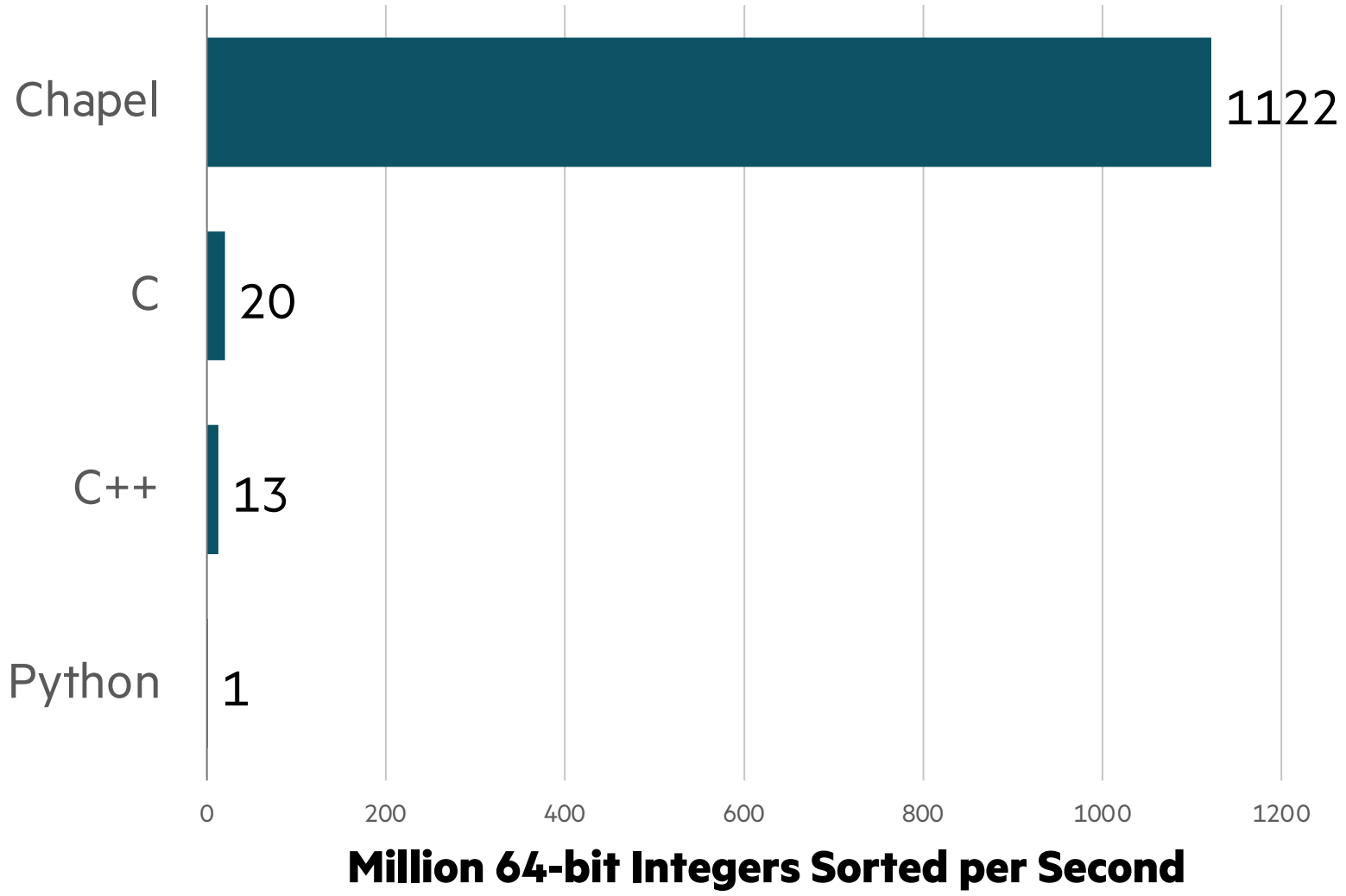
25 times faster

than C with ‘qsort’

400 times faster

than Python

Results on 2 Socket AMD EPYC 7763: 64 Cores



50 times faster

than C with ‘qsort’

1000 times faster

than Python

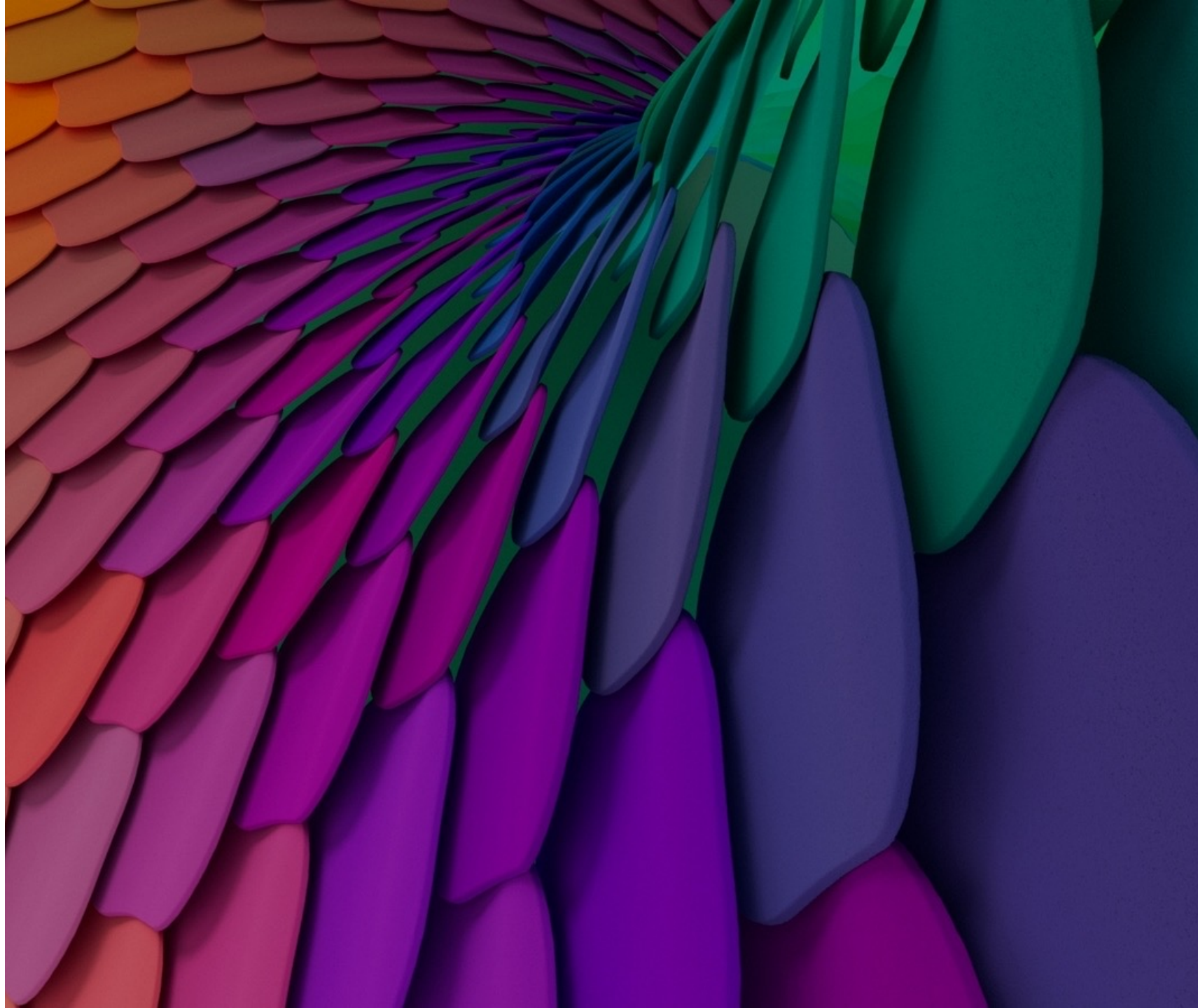
Why?

The main reason:

- Chapel used all the cores
- others used 1 core

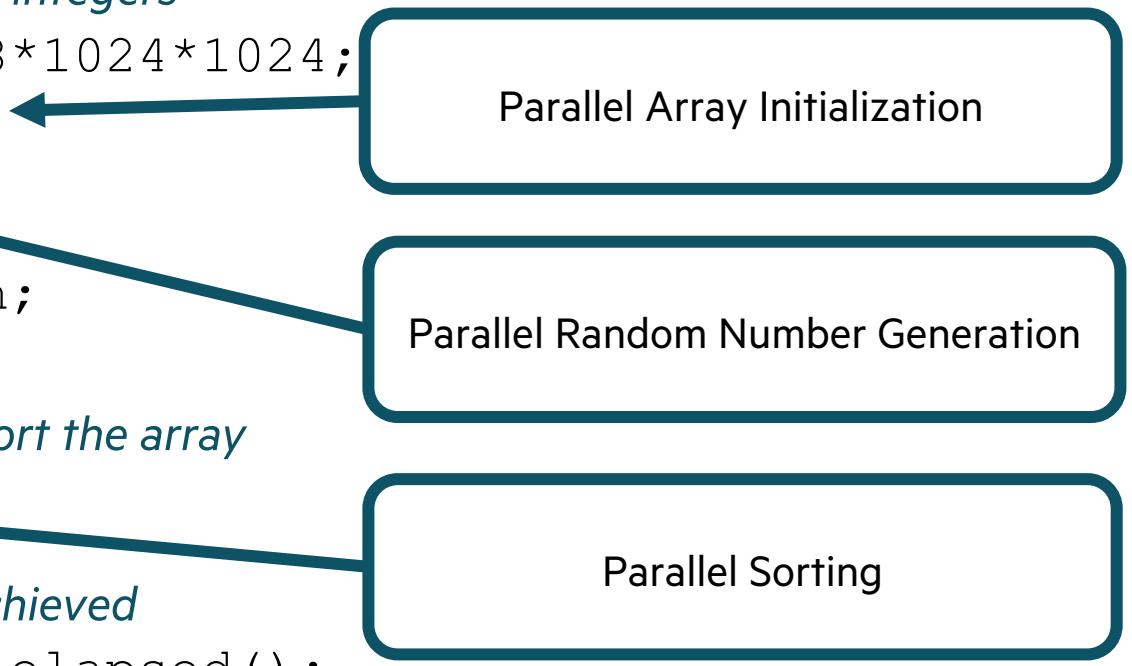
Easy Parallelism

- 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



Parallelism in the Benchmark

```
use Time, Sort, Random;

// generate an array of random integers
config const n = 128*1024*1024;
var A: [0..<n] uint;
fillRandom(A);

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");
```

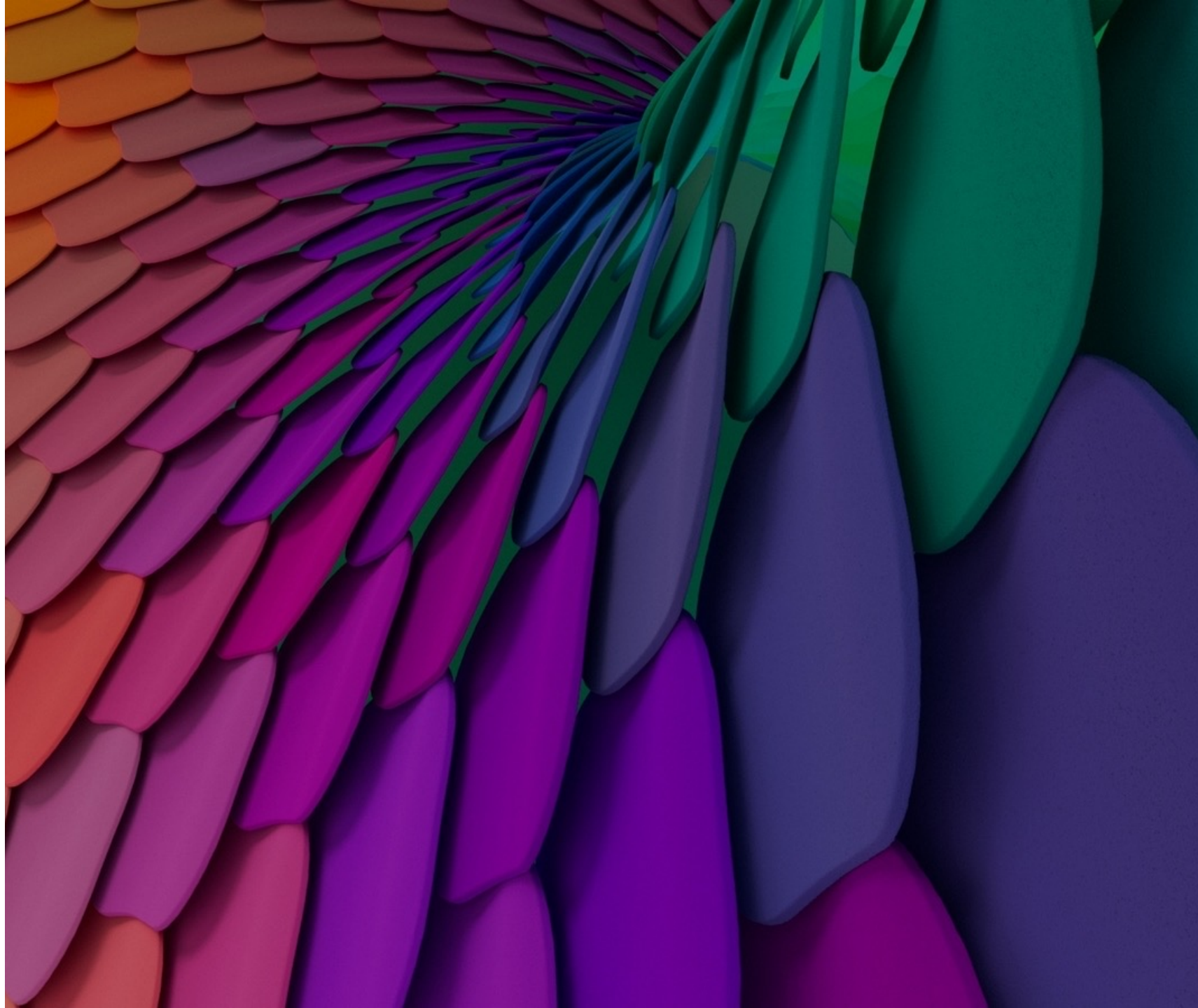


Key Aspects of the Chapel Programming Language

Productive Parallel Fast Scalable GPU-Enabled Open

Chapel is Productive

- Concise and readable
- Consistent concepts for parallel computing make it easier to learn
- Users can quickly attain parallel performance



Productive for Heat Diffusion Simulation

From a 2023 Benchmark Study

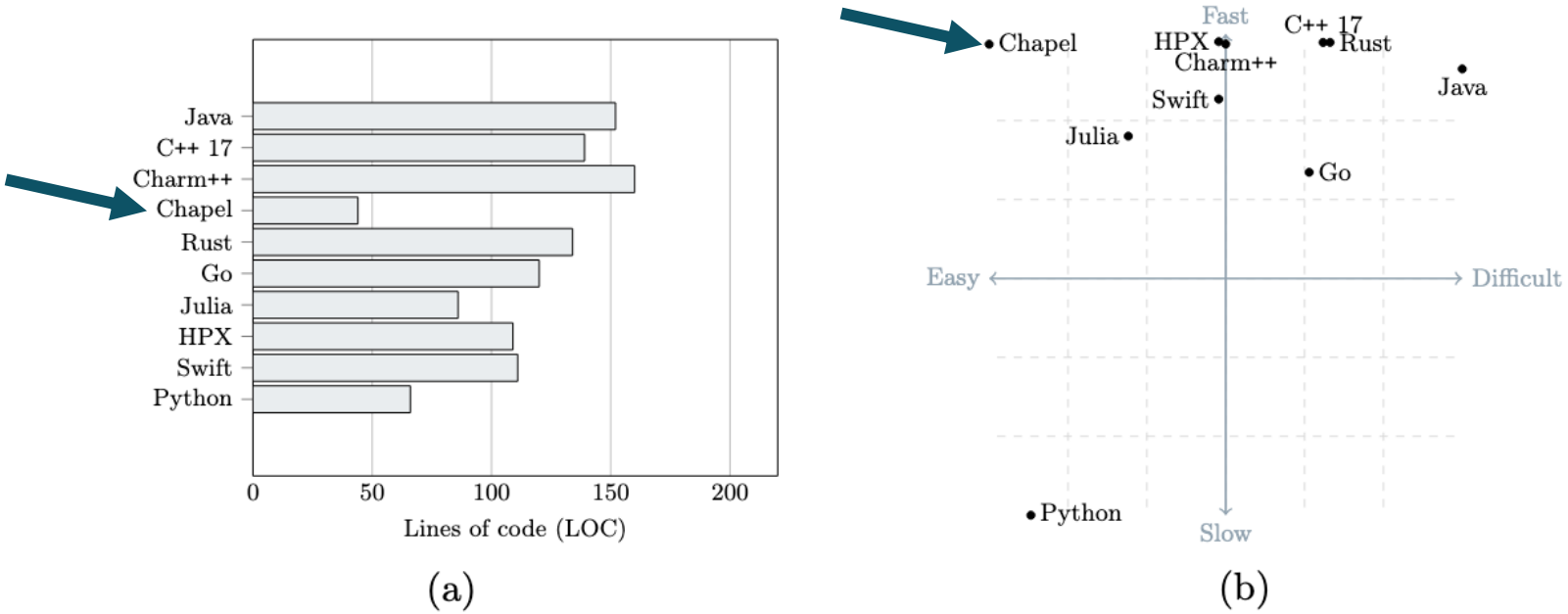


Fig. 1. Software engineering metrics: (a) Lines of codes for all implementations. The numbers were determined with the Linux tool *cloc* and (b) Two-dimensional classification using the computational time and the COCOMO model.

[1] Diehl, P., Morris, M., Brandt, S.R., Gupta, N., Kaiser, H. (2024). Benchmarking the Parallel 1D Heat Equation Solver in Chapel, Charm++, C++, HPX, Go, Julia, Python, Rust, Swift, and Java. In: Zeinalipour, D., *et al.* Euro-Par 2023: Parallel Processing Workshops. Euro-Par 2023. Lecture Notes in Computer Science, vol 14352. Springer, Cham. Available at https://arxiv.org/abs/2307.01117

Diehl et al. [1] studied the productivity of writing a parallel heat equation solver.

They found the Chapel implementation:

- Significantly shorter
- Easier to develop
- Among the fastest

Productive for Parallel Metaheuristics

From a 2020 Comparative Study

- 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

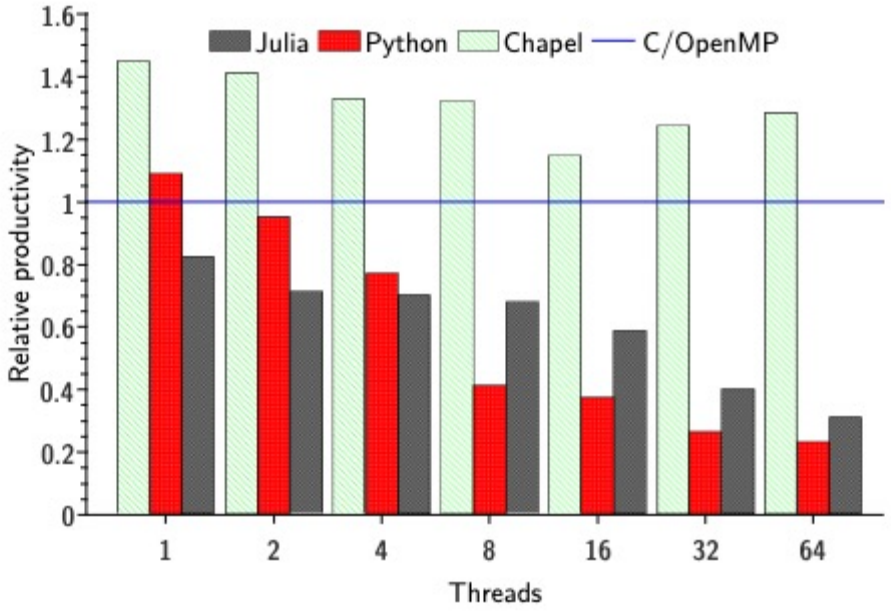


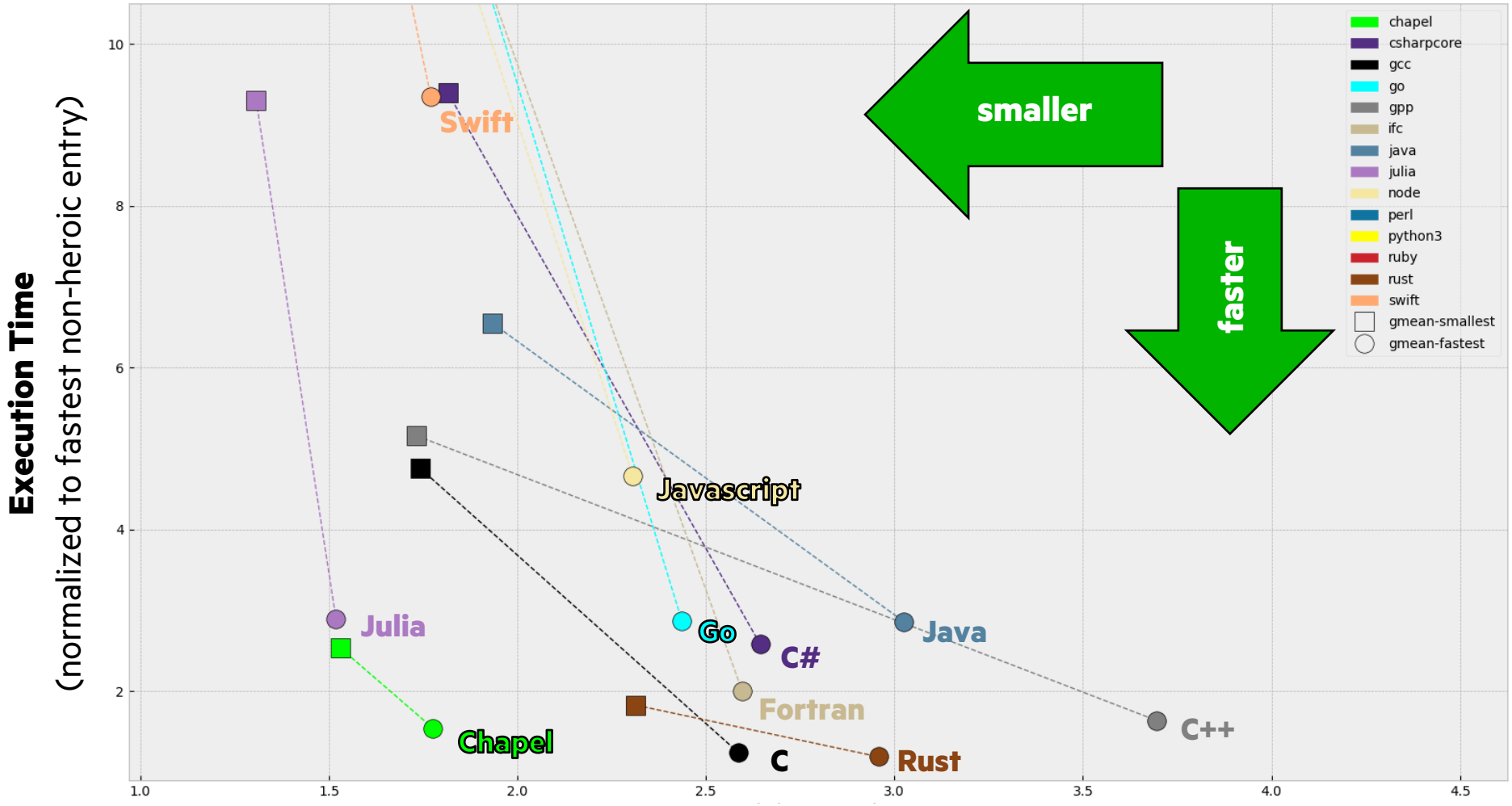
Figure 7: Relative productivity achieved by Chapel, Julia, and Python compared to the C/OpenMP reference. Results are given for the instance *nug22* and execution on 1 to 64 threads.

A figure from [1]

[1] Jan Gmys, Tiago Carneiro, Nouredine Melab, El-Ghazali Talbi, Daniel Tuyttens. A comparative study of high-productivity high-performance programming languages for parallel metaheuristics. Swarm and Evolutionary Computation, 2020, 57, 10.1016/j.swevo.2020.100720 . Available at https://inria.hal.science/hal-02879767

Good Performance with Small Source Source Code size

CLBG Summary, Oct 6, 2024 (selected languages, no Heroic versions, zoomed-in)



Chapel is Compact, clear, and competitive

STREAM TRIAD: C + MPI + OPENMP

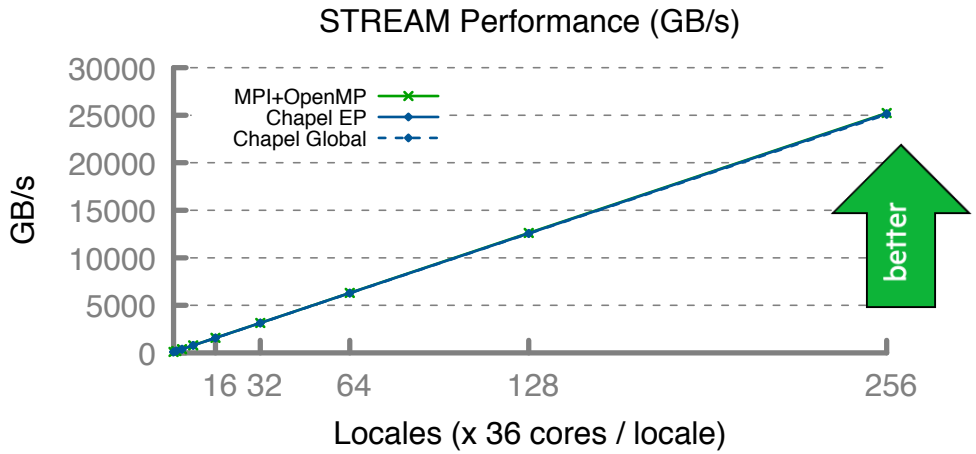
```
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

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

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );
```

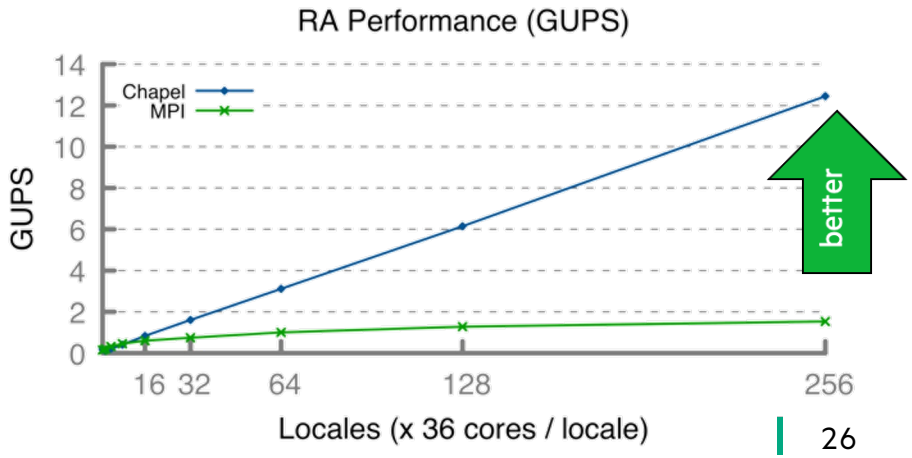
```
use BlockDist;

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

```
/* Perform updates to main table.  The scalar equivalent is:
 *
 *     for (i=0; i<NUPDATE; i++){
 *       Ran = (Ran << 1) ^ ((s64Int) Ran < 0) ? POLY : 0;
 *       Table[Ran & (TABSIZE-1)] ^= Ran;
 *     }
 */

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
while (i < SendCnt) {
  /* receive messages */
  do {
    MPI_Test(&inreq, &have_done, &status);
    if (have_done) {
      if (status.MPI_TAG == UPDATE_TAG) {
        MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
        bufferBase = 0;
        for (j=0; j < recvUpdates; j ++) {
          inmsg = LocalRecvBuffer[bufferBase+j];
          LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                        tparams.GlobalStartMyProc;
          HPCC_Table[LocalOffset] ^= inmsg;
        }
      } else if (status.MPI_TAG == FINISHED_TAG) {
        NumberReceiving--;
      } else
        MPI_Abort( MPI_COMM_WORLD, -1 );
      MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    }
  } while (have_done && NumberReceiving > 0);
  if (pendingUpdates < maxPendingUpdates) {
    Ran = (Ran << 1) ^ (s64Int) Ran < ZERO64B ? POLY : ZERO64B;
    GlobalOffset = Ran & (tparams.TableSize-1);
    if ( GlobalOffset < tparams.Top)
      WhichPe = ( GlobalOffset / (tparams.MinLocalTableSize + 1) );
    else
      WhichPe = ( (GlobalOffset - tparams.Remainder) /
                  tparams.MinLocalTableSize );
    if (WhichPe == tparams.MyProc) {
      LocalOffset = (Ran & (tparams.TableSize - 1)) -
                    tparams.GlobalStartMyProc;
      HPCC_Table[LocalOffset] ^= Ran;
```

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





Doing the Impossible

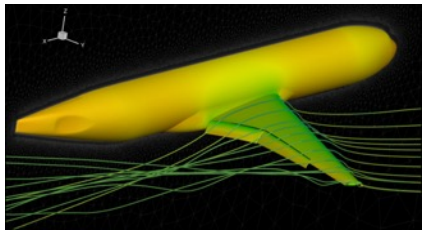
> *[Chapel] promotes programming efficiency … We ask students at the master's degree to do stuff that would take 2 years and they do it in 3 months.*

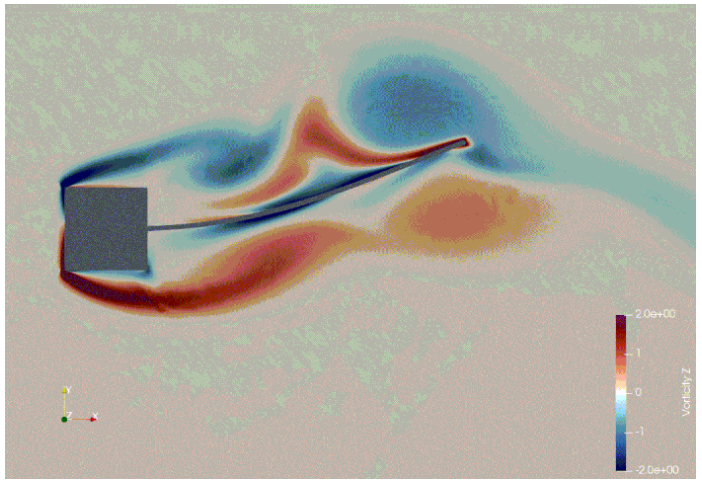
Éric Laurendeau
Professor, Department of Mechanical Engineering, Polytechnique Montréal
quote from his 2021 CHIUW Keynote [video]

CHAMPS Summary

What is it?

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

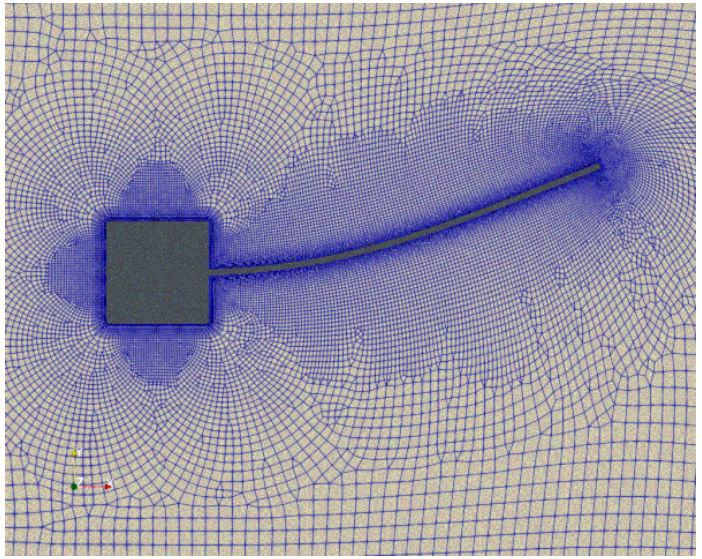




Who wrote it?

- Professor Éric Laurendeau's students + postdocs at Polytechnique Montreal

POLYTECHNIQUE MONTRÉAL



Why Chapel?

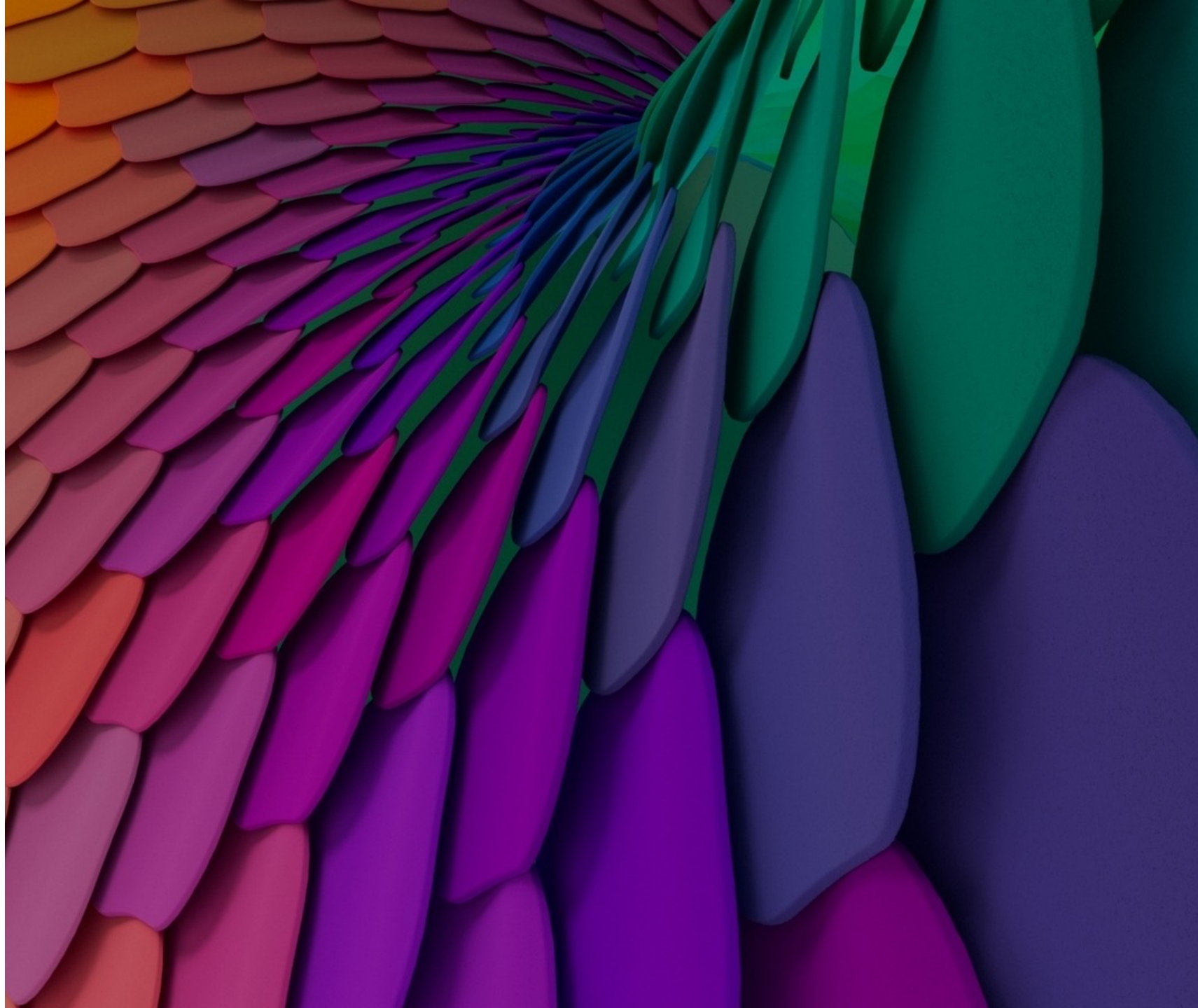
- Performance and scalability competitive with MPI + C++
- Students found it far more productive to use
- Enabled them to compete with more established CFD centers

(images provided by the CHAMPS team and used with permission)

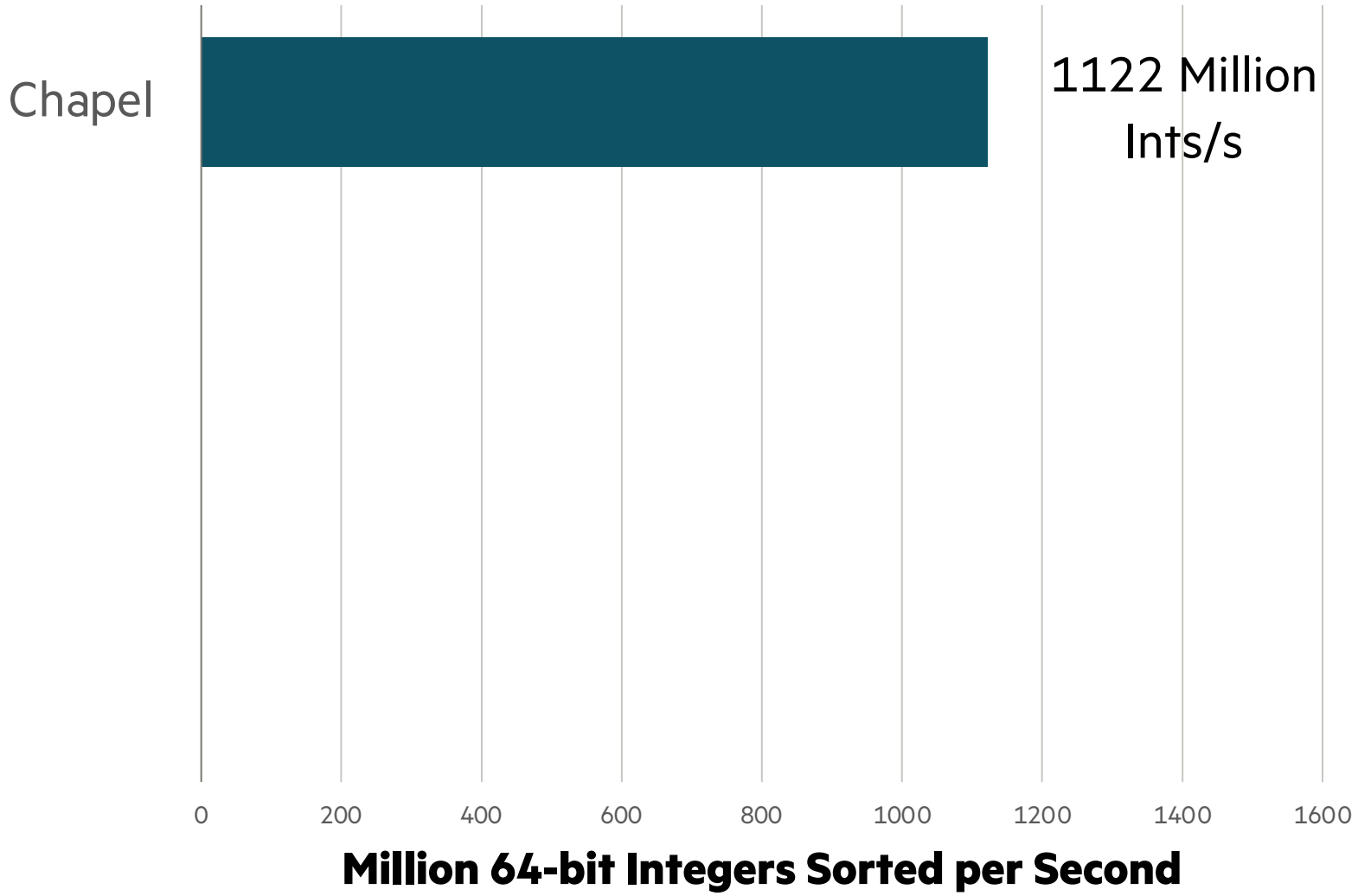
Chapel is Scalable

- Adding more cores and/or more nodes can improve performance!

- Chapel enables application performance at any scale
  - laptops
  - workstations
  - cloud systems
  - clusters
  - the largest supercomputers

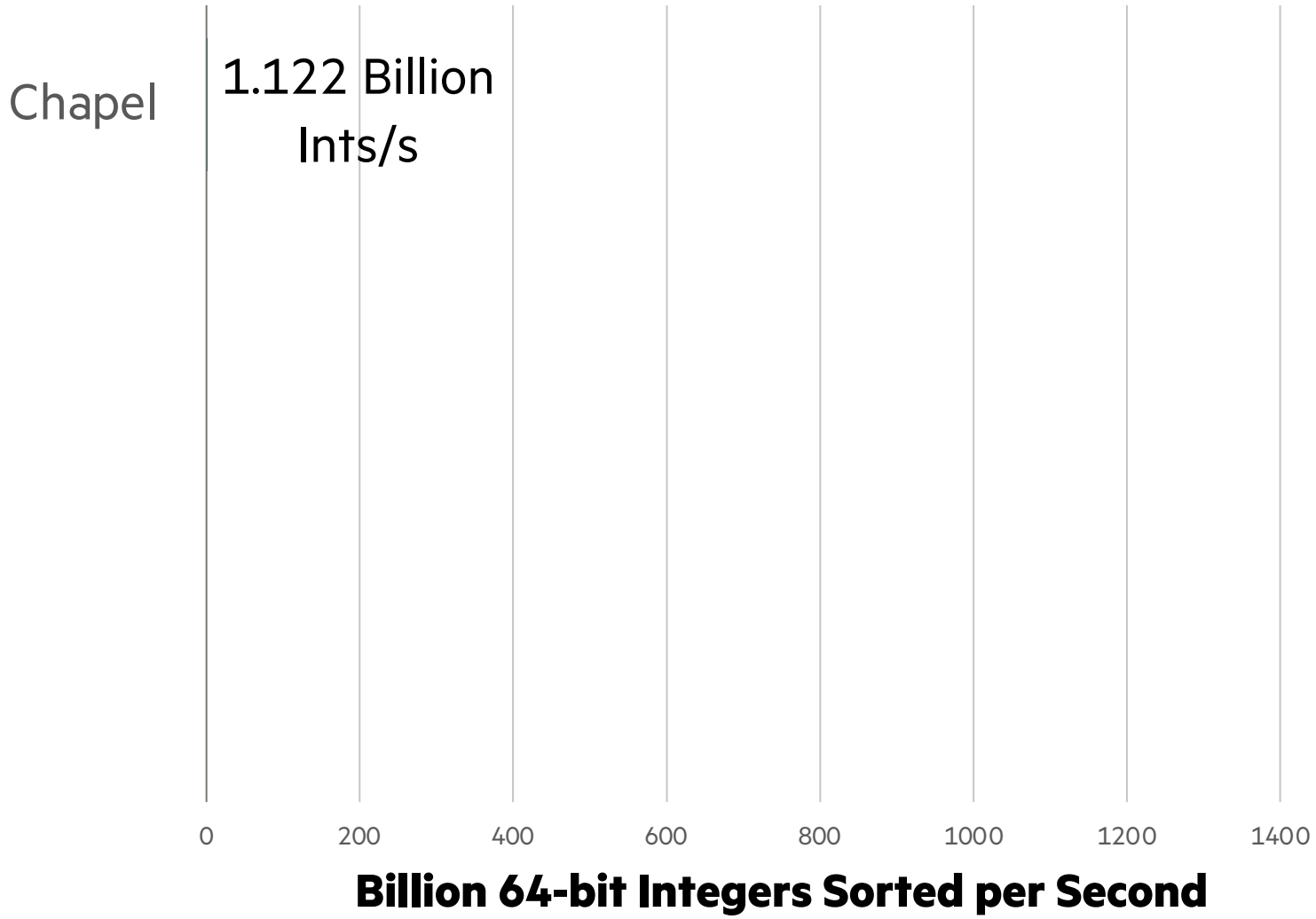


Can we use Chapel to sort on a supercomputer?



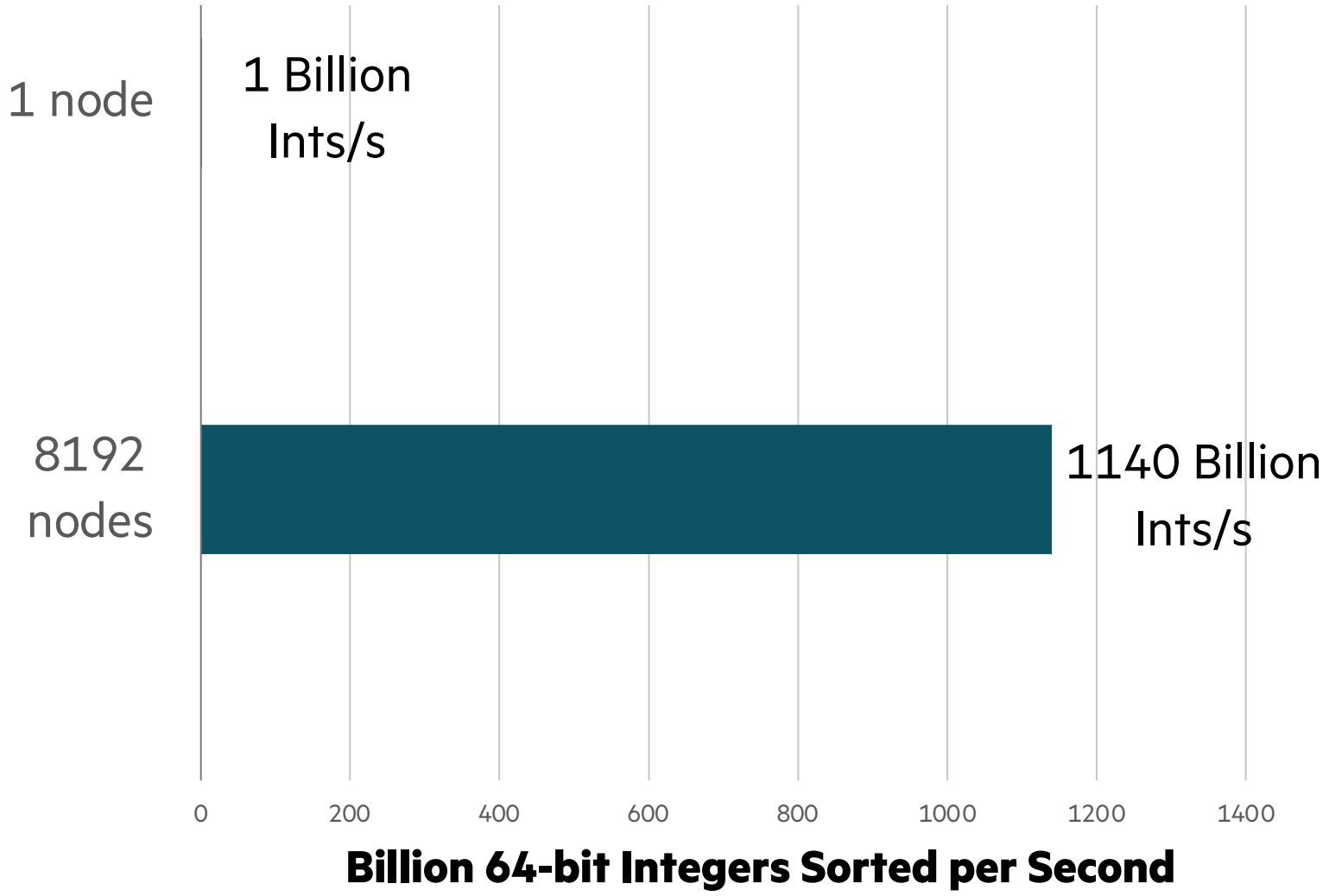
This line is the best result from earlier, on a 2 Socket AMD EPYC 7763: 64 Cores

Can we use Chapel to sort on a supercomputer?



Zooming out to billions

Can we use Chapel to sort on a supercomputer?

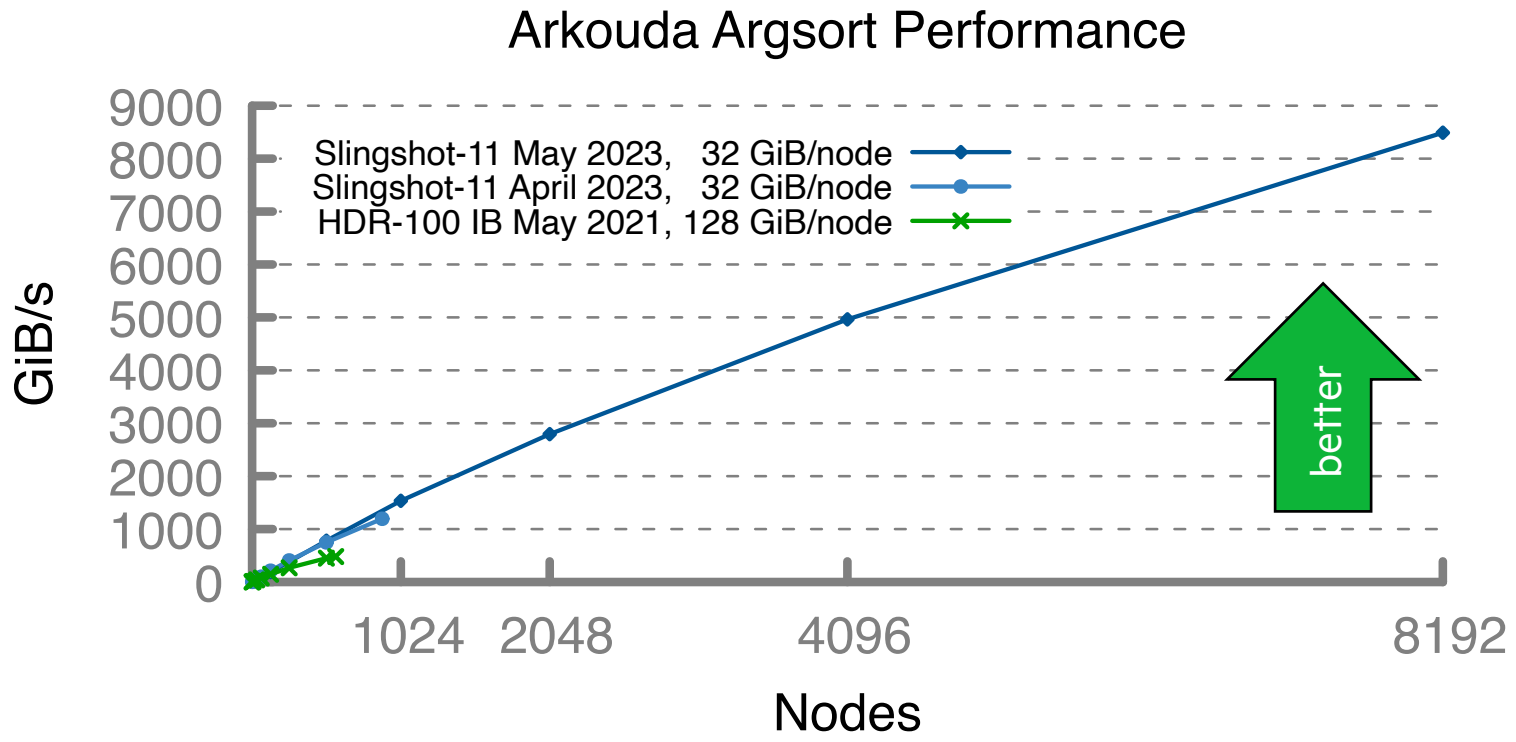


This result is from Arkouda's scalable radix sort, written in Chapel

- sorted 256 TiB
- in about 30 seconds
- on 8192 nodes
- of an HPE Cray EX

1000x faster than the single node result!

Radix Sort in Arkouda/Chapel Scaling to ~9TB/s on >8K Nodes



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

Arkouda Summary

What is it?

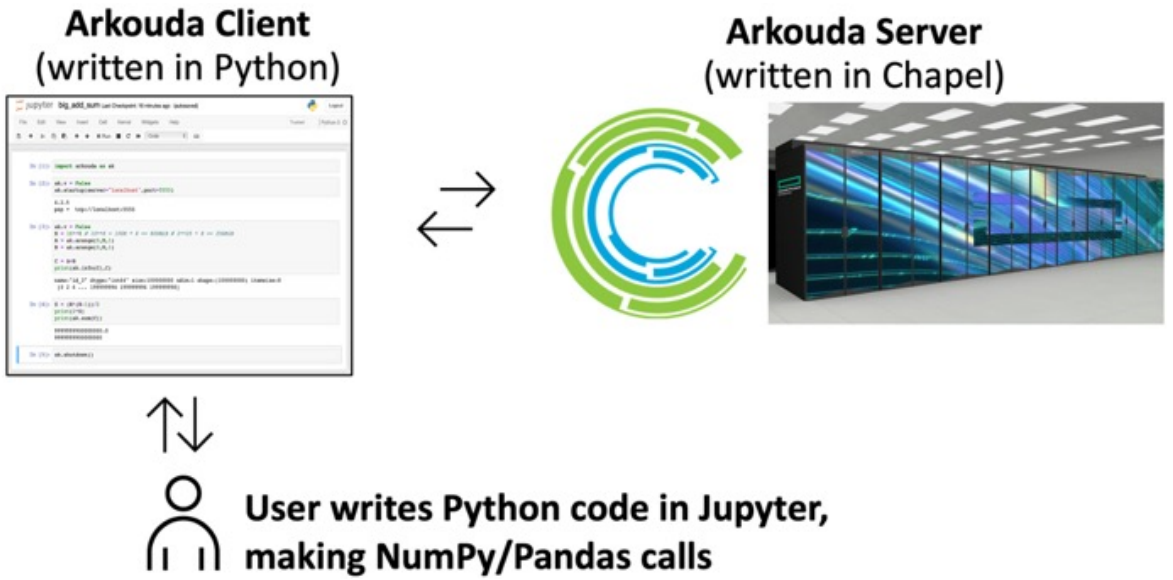
- A framework for interactive, high performance data analytics
- Computes massive-scale results (TB-scale arrays) within the human thought loop (seconds to a few minutes)
- User observation: *No other tool provides Exploratory Data Analysis (EDA) at these scales*
- ~30k lines of Chapel + ~25k lines of 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?

- 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

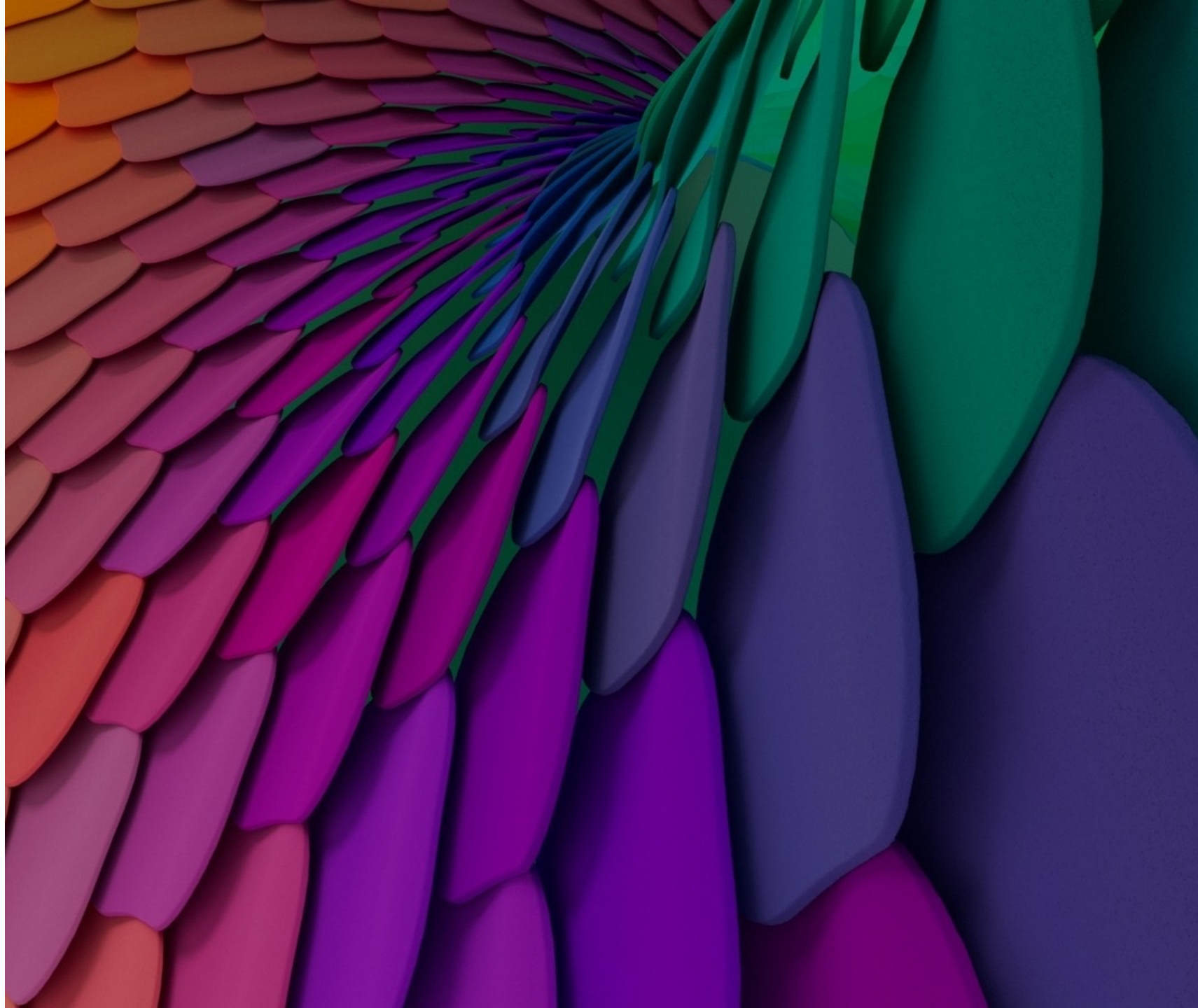


Chapel is Gpu-Enabled

- GPUs have a lot of capability
- Can be challenging to program

- Chapel helps programmability!
  - Use it to program 1 GPU
  - Or many GPUs in a big system

- Let’s look at an example



1D Heat Equation Example

This is the 1-D heat diffusion simulation from the ChapelCon'24 tutorial

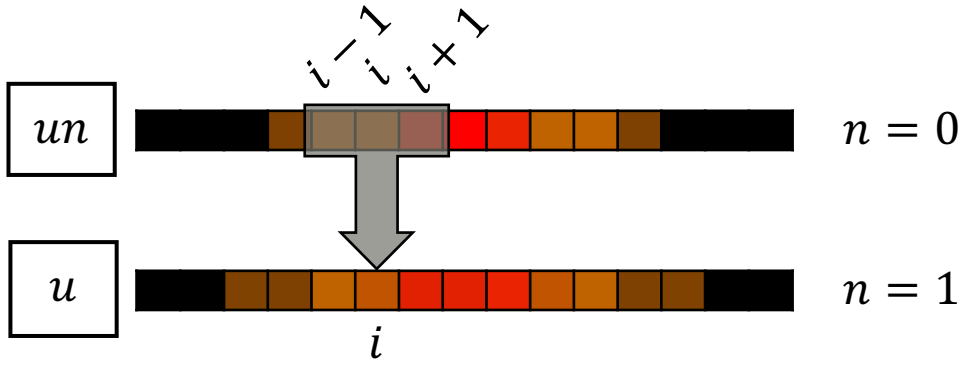
This version is serial and roughly matches what one might write in Python

```
1  const omega = {0..<nx},
2        omegaHat = omega.expand(-1);
3  var u: [omega] real = 1.0;
4  u[nx/4..3*nx/4] = 2.0;
5  var un = u;
6  for 1..N {
7    un <=> u;
8    for i in omegaHat do
9      u[i] = un[i] + alpha *
10            (un[i-1] - 2*un[i] + un[i+1]);
11 }
```

1D Heat Equation Example: Parallel on Multiple Cores

Changing the ‘for’ to a ‘forall’ makes this program parallel on multiple cores

```
 1  const omega = {0..<nx},
 2        omegaHat = omega.expand(-1);
 3  var u: [omega] real = 1.0;
 4  u[nx/4..3*nx/4] = 2.0;
 5  var un = u;
 6  for 1..N {
 7    un <=> u;
★8    forall i in omegaHat do
 9      u[i] = un[i] + alpha *
10               (un[i-1] - 2*un[i] + un[i+1]);
11  }
```



Switched the inner ‘for’ loop to a ‘forall’, which automatically runs the loop in parallel when possible.

The rest of the code is unchanged!

1D Heat Equation Example: Parallel o a GPU

We can use the ‘on’ statement to request the same program run on a GPU!

```
1   on here.gpus[0] {
2     const omega = {0..<nx},
3           omegaHat = omega.expand(-1);
4     var u: [omega] real = 1.0;
5     u[nx/4..3*nx/4] = 2.0;
6     var un = u;
7     for 1..N {
8       un <=> u;
9       forall i in omegaHat do
10        u[i] = un[i] + alpha *
11             (un[i-1] - 2*un[i] + un[i+1]);
12    }
13  }
```

This ‘on’ statement requests GPU execution

The rest of the code is unchanged!

Chapel GPU Code is Compact and Competitive

Simple STREAM Triad in CUDA

```
#include <string>
#include <vector>

#include <stdio.h>
#include <float.h>
#include <limits.h>
#include <unistd.h>
#include <sys/time.h>

typedef double real;

template <typename T>
__global__ void set_array(T * __restrict__ const a, T value, int len)
{
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    if (idx < len)
        a[idx] = value;
}

template <typename T>
__global__ void STREAM_Triad(T const * __restrict__ a, T const * __restrict__ b,
__restrict__ const c, T scalar, int len)
{
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    if (idx < len)
        c[idx] = a[idx] + scalar * b[idx];
}
```

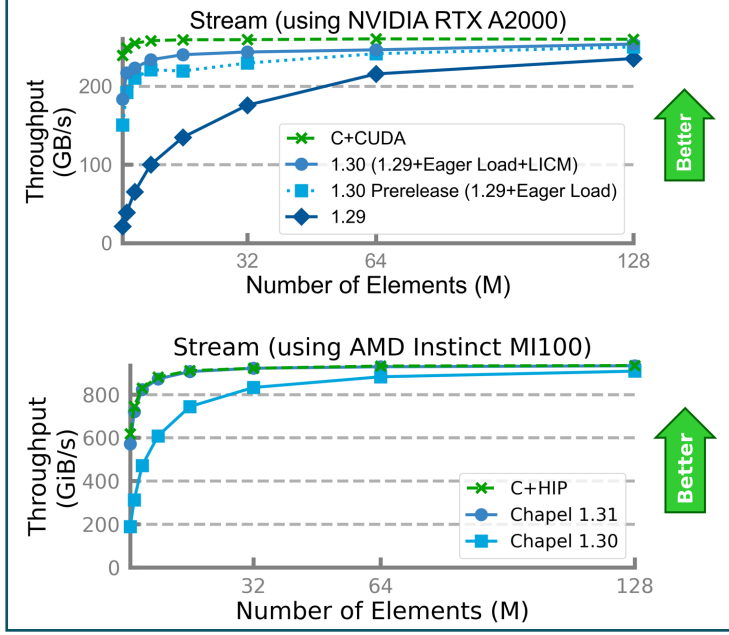
```
int main(int argc, char** argv)
{
    real *d_a, *d_b, *d_c;

    cudaFree(d_b);
    cudaFree(d_c);
}
```

```
config const m = 1<<26,
             alpha = 3.0;

on here.gpus[0] {
  var A, B, C: [1..m] real;

  B = 0.5;
  C = 0.5;

  A = B + alpha*C;
}
```

48

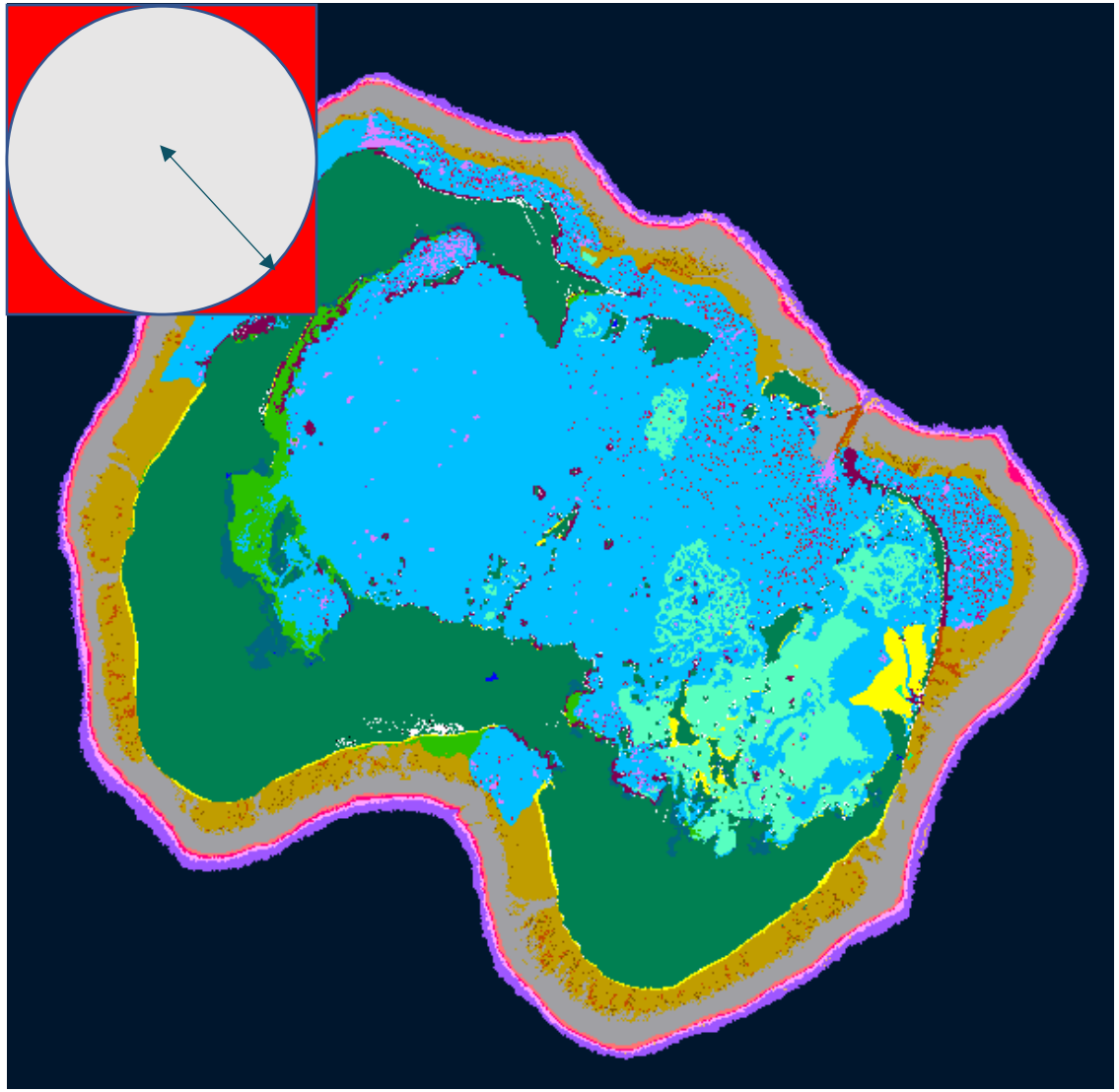
Performance

- On par with HIP, very close to CUDA



Use Case: Image Processing for Coral Reef Biodiversity

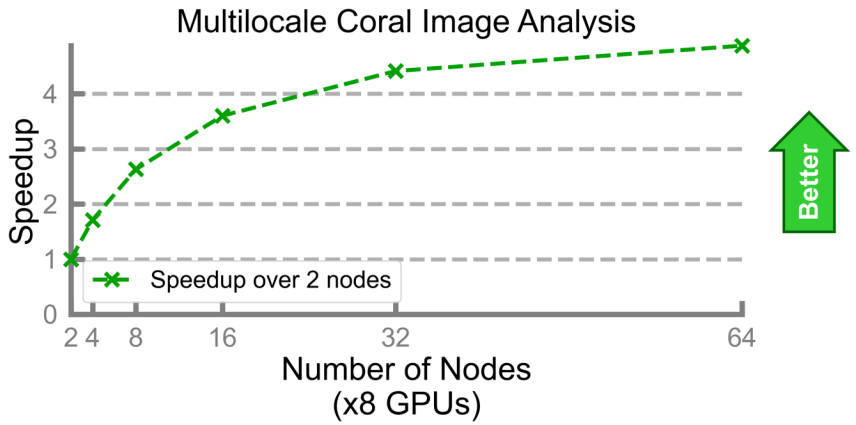
- **Analyzing images for coral reef biodiversity**
  - Important for prioritizing interventions
- **Algorithm implemented productively**
  - Add up weighted values of all points in a neighborhood, i.e., convolution over image
  - Developed by Scott Bachman, NCAR scientist who was a visiting scholar on the Chapel team at HPE
  - Scott started learning Chapel in Sept 2022, started Coral Reef app in Dec 2022, already had collaborators presenting results in Feb 2023
  - In July, changed ~5 lines in a variant to run on GPU
- **Performance**
  - Less than 300 lines of Chapel code scales out to 100s of processors on Cheyenne (NCAR)
  - Full maps calculated in seconds, rather than days
    - **10,000 times faster**



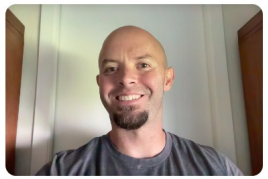
Use Case: Image Processing for Coral Reef Biodiversity

Runs on Frontier!

- At 64 nodes, takes 20 minutes
  - As opposed to ~27 days on a laptop
- Straightforward code changes:
  - from sequential Chapel code
  - to GPU-enabled
  - to multi-node, multi-GPU, multi-thread



Read a recent interview with Scott Bachman on Chapel Blog



7 Questions for Scott Bachman: Analyzing Coral Reefs with Chapel

Posted on October 1, 2024.

Tags: Earth Sciences | Image Analysis | GPU Programming | User Experiences | Interviews

By: Brad Chamberlain, Engin Kayraklioglu

In this second installment of our Seven Questions for Chapel Users series, we're looking at a recent success story in which Scott Bachman used Chapel to unlock new scales of biodiversity analysis in coral reefs to study ocean health using satellite image processing. This is work that Scott started as a visiting scholar with the Chapel team at HPE, and it is just one of several projects he took on during his time with us. Since wrapping up his visit at HPE, Scott has continued to apply Chapel in his work, which he describes below.

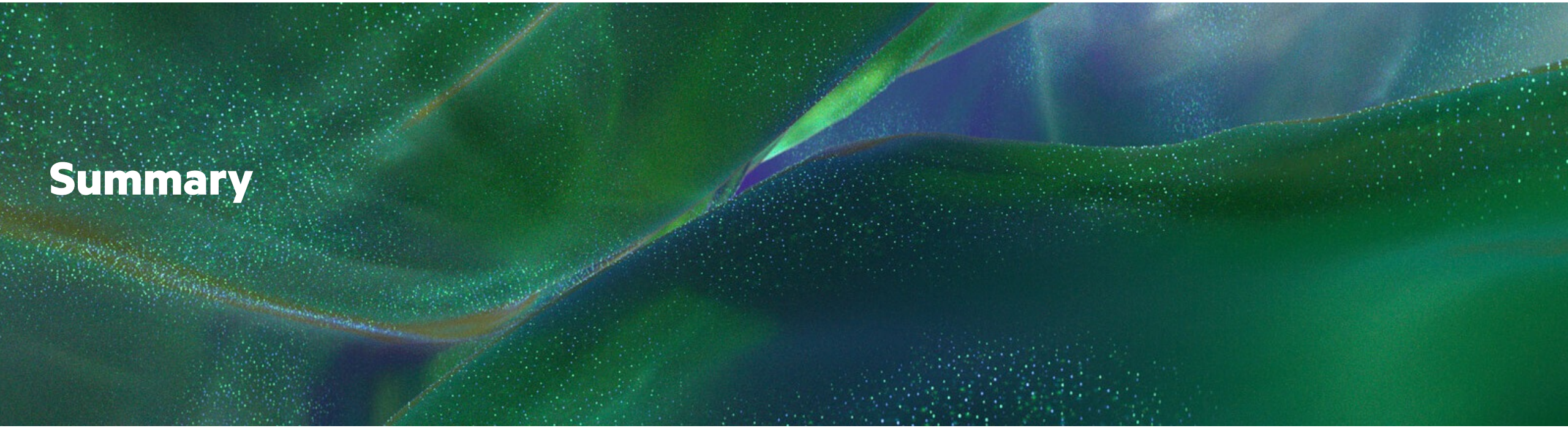
One noteworthy thing about the computation Scott describes here is that it is just a few hundred lines of Chapel code, yet can be used to drive the CPUs and GPUs of the world's largest supercomputers. This serves as a sharp contrast with the 100+k lines that make up the CHAMPS framework covered in our previous interview. Together, the two demonstrate the vast spectrum of code sizes that researchers are productively writing in Chapel.

Helping Scientists Do Science

> *I told them "Don't hire software engineers. I'll do it, and I'll write it in Chapel because I can do it by myself, and I can stand this thing up really fast." And that is exactly what happened.*

Scott Bachman
Oceanographer, National Center for Atmospheric Research (NCAR) and Technical Modeling Lead at [C]Worthy
quote from his interview on the Chapel Blog

Summary



Institutions and Application Domains Using Chapel

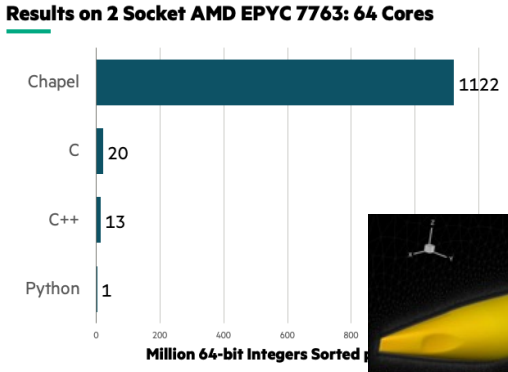
Institutions	Application Domains
École Polytechnique Montréal in Canada	3D Unstructured CFD for Airplane simulation
U.S. Govt	Arkouda: Exploratory Data Analysis at Scale
[C]Worthy	Oceanographic Modeling
Coral Reef Alliance	Image Analysis
New Jersey Institute of Technology in USA	Distributed Graph Analytics in Arachne/Arkouda
Radboud University in The Netherlands	Quantum Simulations
INRIA in France and IMEC in Belgium	Branch and Bound Optimization (e.g., N-Queens)
The Federal University of Paraná in Brazil	Environmental Engineering
University of Guelph in Canada	Hydrological model
University of Colorado, Boulder in USA	Structured CFD for climate science
PNNL in USA	Hypergraph Library
Yale University in USA	Distributed FFTs
Cray/AI at HPE	Hyper Parameter Optimization

HPE provides paid Chapel support for some organizations

Chapel is Fast, Productive, Scalable, GPU-Enabled, and Open Source

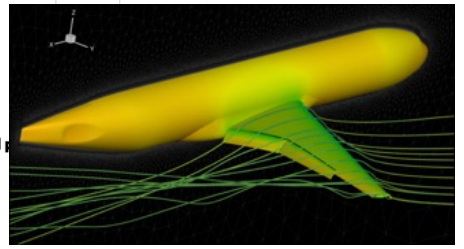
Fast and Scalable

- Easier parallelism allows your program to use more of your hardware



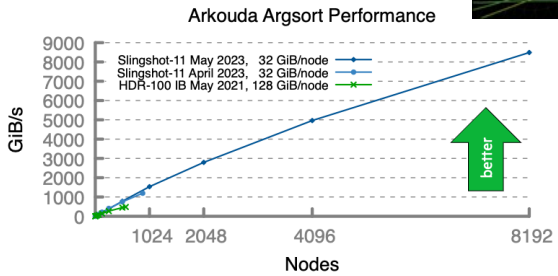
Productive

- On the CHAMPS team, students could complete projects in 8x less time



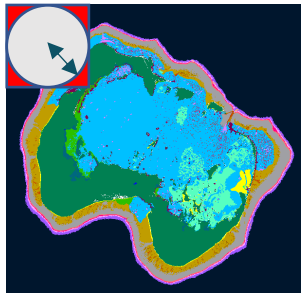
Scalable Across Nodes

- Arkouda sort has scaled to 8192 nodes to achieve 1000x speedup



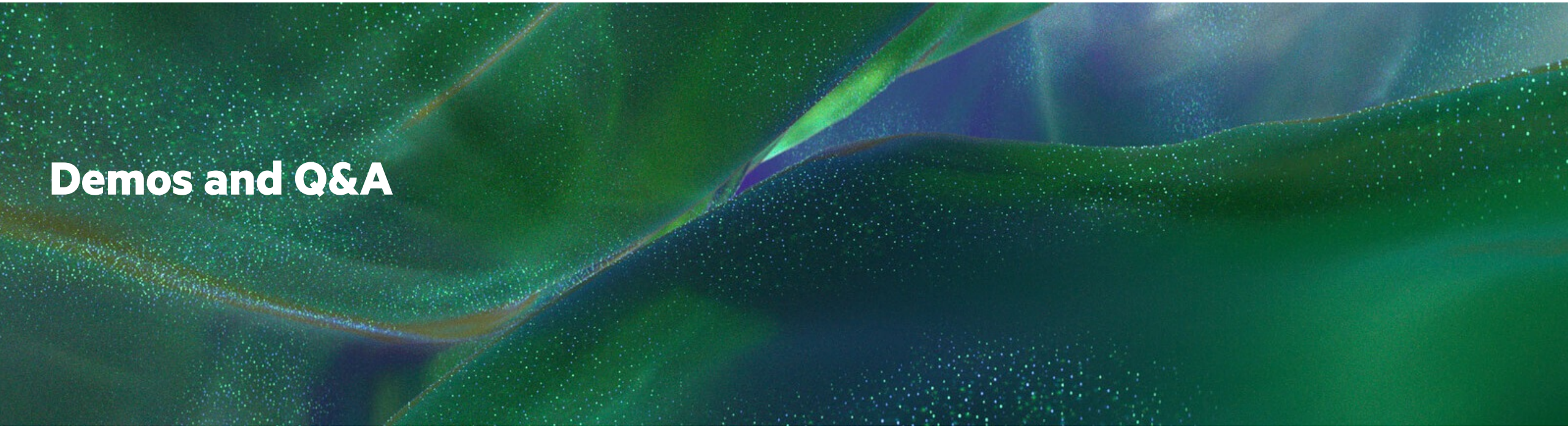
GPU-Enabled

- Coral Reef Biodiversity application ran on GPUs changing only ~5 lines



We welcome you to participate in our open-source community!

Demos and Q&A



Demos

- Available Demos
  - Installing and Compiling Hello World
  - Calling Chapel code from Python
  - ‘on’ and multi-node execution on a supercomputer
  - Game of Life