# CHAPEL PROGRAMMING LANGUAGE
# GHC OPEN SOURCE DAY
# INTRODUCTION

FRIDAY SEPTEMBER 16TH, 2022

# ABOUT US

- **Michelle Strout** is the senior engineering manager of the Chapel programming language team since January of 2021. She has been a Computer Science professor since 2005 with expertise in compilers and high performance computing. She has also participated in GHC and OSD in the past as a mentor.

- **Lydia Duncan** is the sub-team lead for the 2.0 Library Stabilization effort. She's been on the Chapel team for 10 years and has worked on the compiler, libraries, 'chpldoc' documentation tool, and interoperability, among other things. She has helped with Chapel's participation in Google Summer of Code in the past.

# SCIENTIFIC COMPUTING CHALLENGES

- **Steep learning curve to effectively achieve high performance**
  - Distributed-memory parallelism across nodes (MPI)
  - Parallelism within a node (OpenMP, Pthreads, CUDA, …)
  - Vectorization (intrinsics that are architecture specific)

- **Preferred development model is on a laptop and then run on a cluster, cloud, or supercomputer**

- **Goal is to have …**
  - Ease of programming,
  - High performance, and
  - Portability

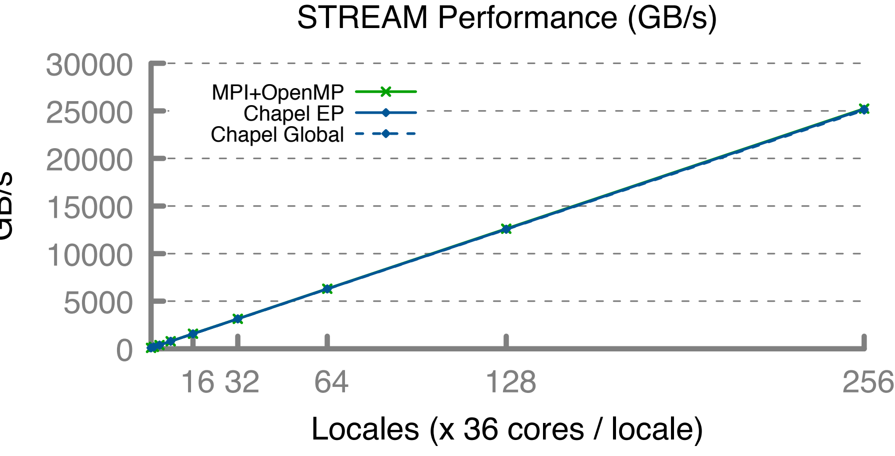
- **Chapel achieves all three of these goals**

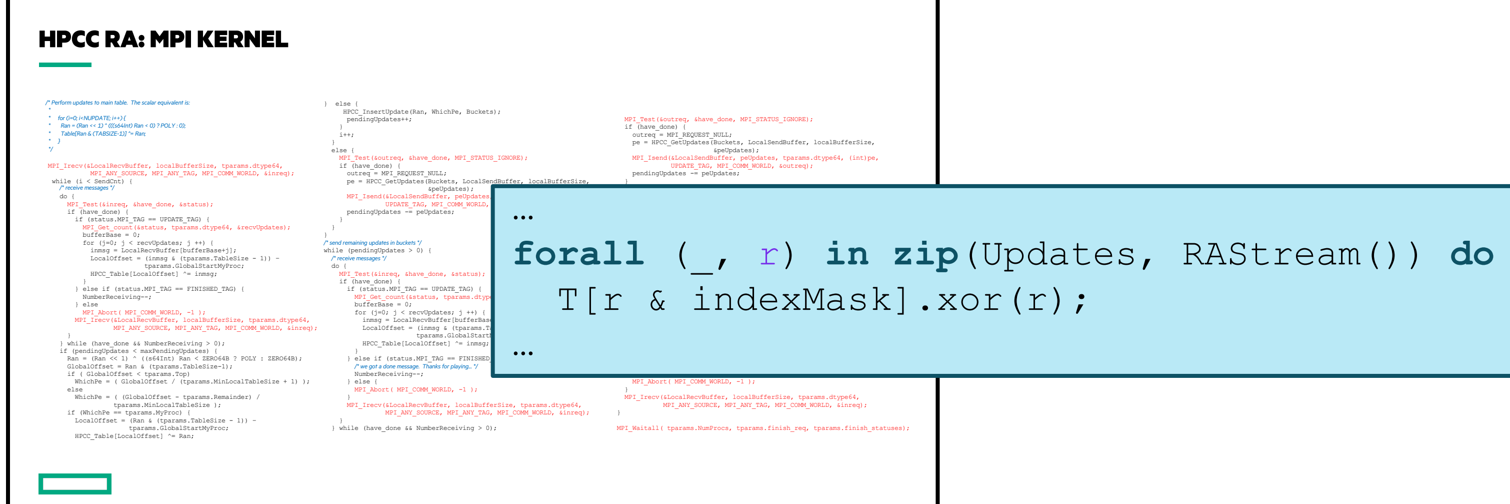
# EASE OF PROGRAMMING AND HIGH PERFORMANCE

## STREAM TRIAD: C + MPI + OPENMP

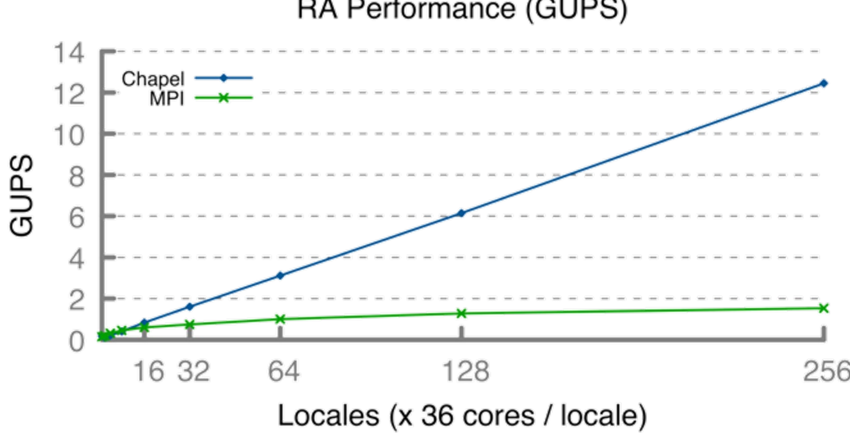
```
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

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 );

  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
      fprintf( outFile, "Failed to allocate memory
      fclose( outFile );
    }
    return 1;
  }

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 1.0;
  }
  scalar = 3.0;

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);

  return 0;
}
```

```
use BlockDist;

config const m = 1000,
             alpha = 3.0;
const Dom = {1..m} dmapped …;
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);
…
```



# PORTABILITY

- **On a laptop, cluster, or supercomputer (Shared-memory parallelism)**

```
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
```

- **On a cluster or supercomputer (Distributed-memory parallelism)**

```
prompt> chpl helloTaskPar.chpl
prompt> ./helloTaskPar –numLocales=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
…
```

# HOW TO GET STARTED

- Code of Conduct:
  - https://github.com/chapel-lang/chapel/blob/main/CODE_OF_CONDUCT.md
- Get set up on Github and try to run a sample program
  - Steps 1-5 of https://gracehopperosd.slack.com/archives/C03U95VF06N/p1663106760664899 in Slack
- Ensure you're set up for committing (step 6 in second link)
- Choose your task and get started! (remaining steps in second link)
- We're here to help, please ask questions!

# OTHER RESOURCES

- Commit signing:
  - https://chapel-lang.org/docs/developer/bestPractices/DCO.html#best-practices-dco
- Git tips:
  - https://chapel-lang.org/docs/developer/bestPractices/git.html
- Test system details:
  - https://chapel-lang.org/docs/developer/bestPractices/TestSystem.html
- How to deprecate symbols:
  - https://chapel-lang.org/docs/developer/bestPractices/Deprecation.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

## The Chapel Parallel Programming Language

Home

What is Chapel?
What's New?

Upcoming Events
Job Opportunities

How Can I Learn Chapel?
Contributing to Chapel

Download Chapel
Try Chapel Online

Documentation
Release Notes

Performance
Powered by Chapel

User Resources
Developer Resources

Social Media / Blog Posts
Press

Presentations
Papers / Publications

CHIUW
CHUG

Contributors / Credits

chapel_info@cray.com

### 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

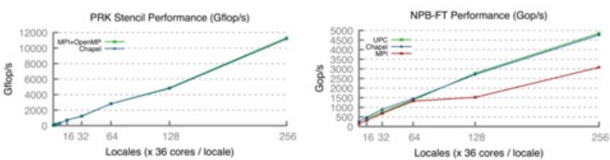
**Chapel Characteristics**

- **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:




- 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);
```