CHAPEL PROGRAMMING LANGUAGE: OVERVIEW AND ROADMAP

Michelle Mills Strout
November 17, 2022

CHAPEL TEAM

Chapel is truly a team effort

see: https://chapel-lang.org/contributors.html

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:

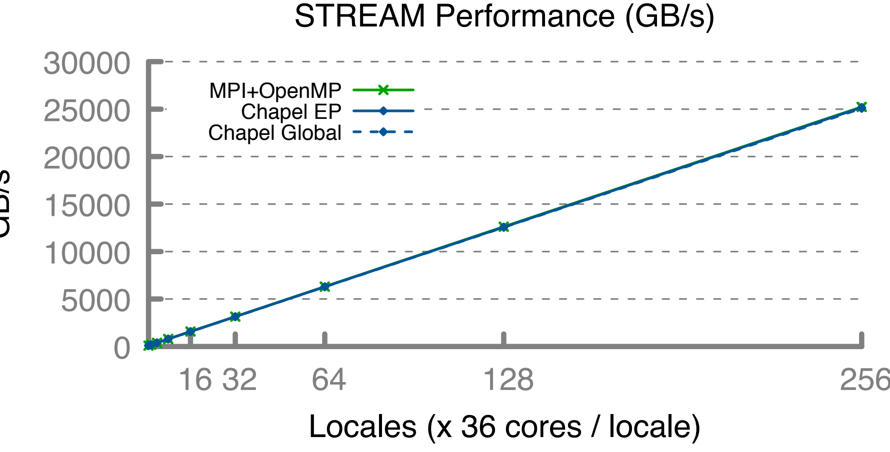
refactoring existing codes,
developing new codes,
serving high performance to Python codes **(Chapel server with Python client),** and
providing distributed and shared memory parallelism for existing codes.

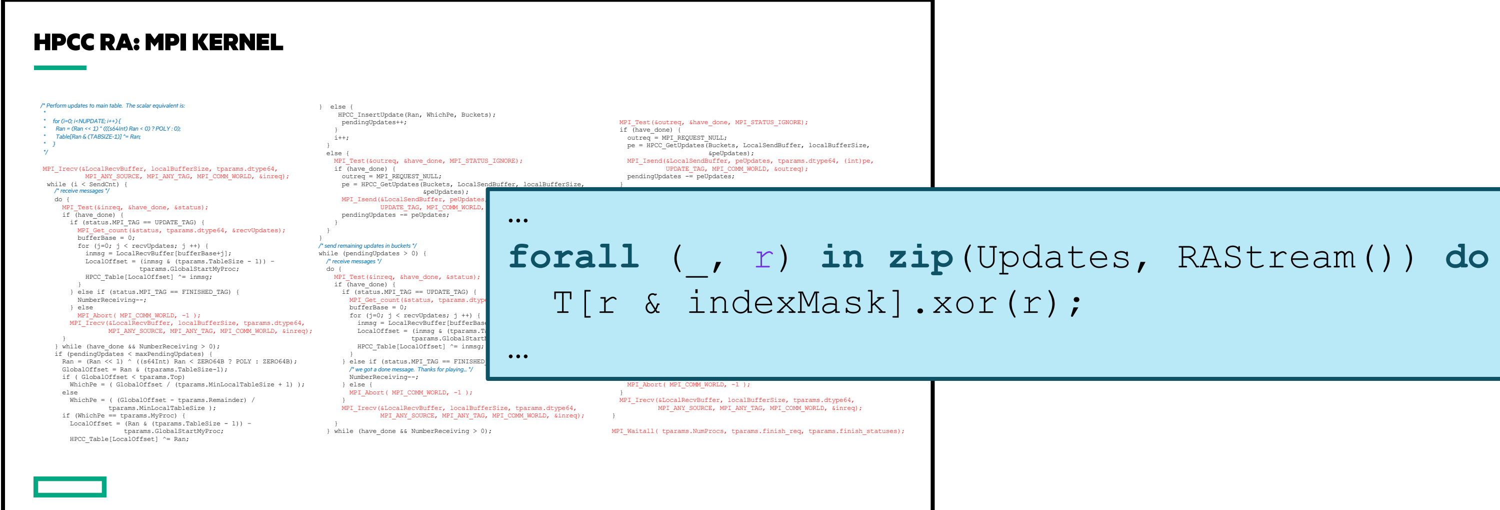
EASE OF PROGRAMMING AND HIGH PERFORMANCE

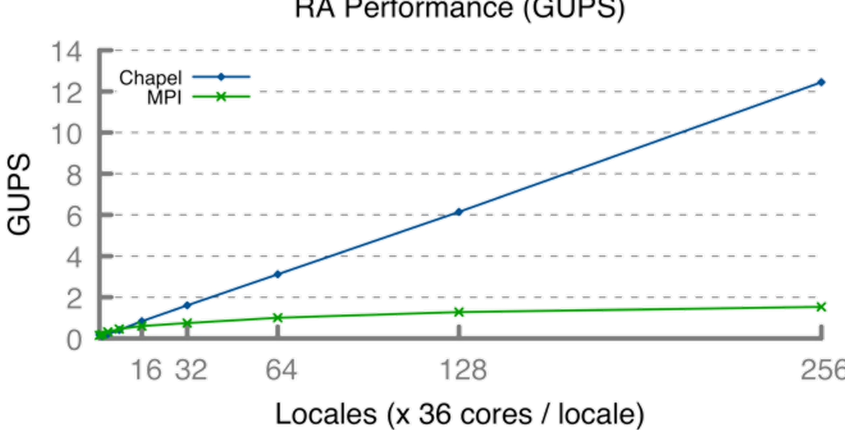
STREAM TRIAD: C + MPI + OPENMP

```
#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;
```







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
…
```

EXAMPLE CODE: ANALYZING MULTIPLE FILES USING PARALLELISM

word-count.chpl

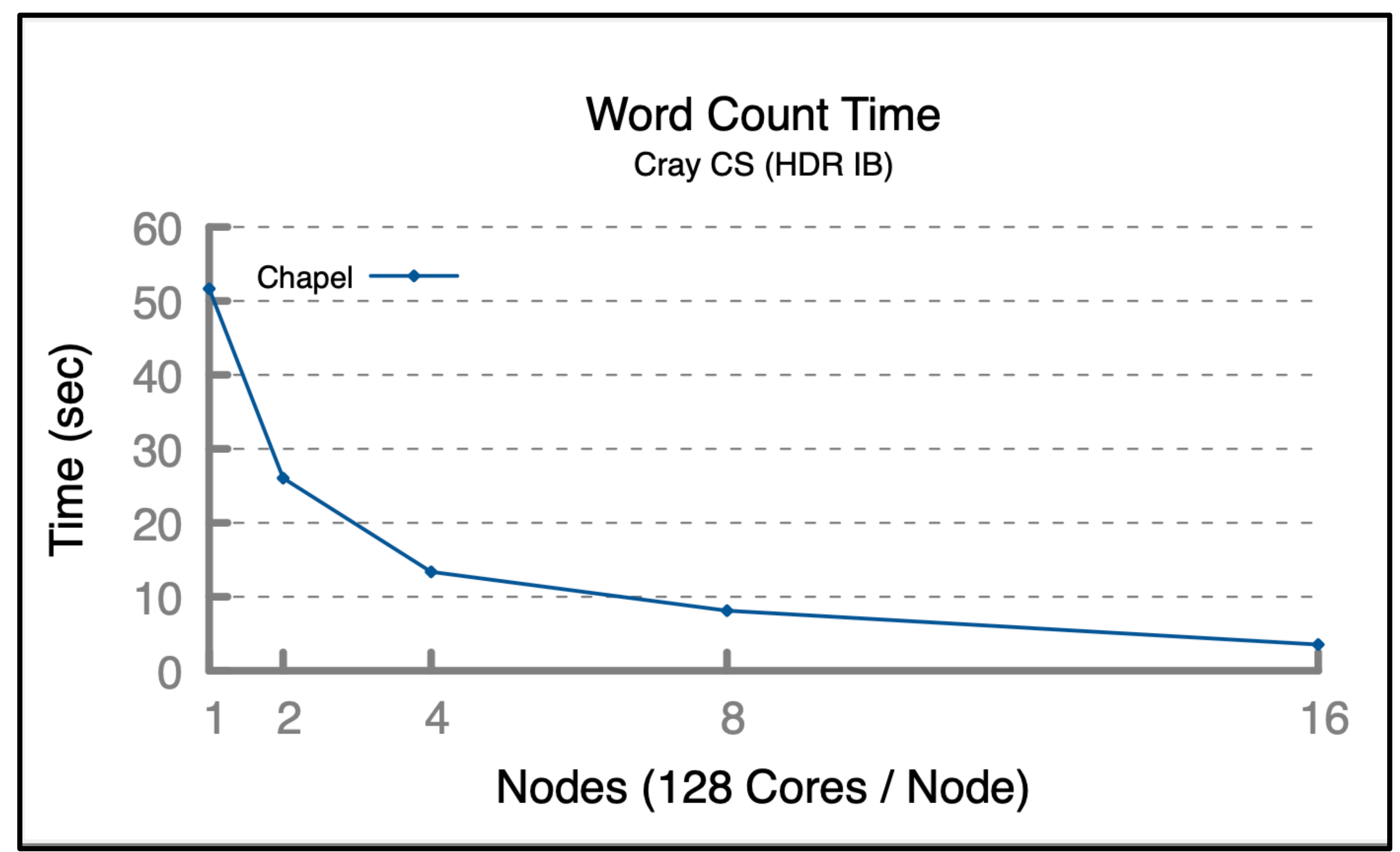
```
use FileSystem;
config const dir = "DataDir";
var fList = findfiles(dir);
var filenames
  = newBlockArr(0..<fList.size,string);
filenames = fList;

// per file word count
forall f in filenames {
  ...
  while reader.readline(line) {
    for word in line.split(" ") {
      wordCount[word] += 1;
    }
  }
 ...
}
```

```
prompt> chpl --fast word-count.chpl
prompt> ./word-count
prompt> ./word-count -nl 4
```

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

SCALING FROM LAPTOP TO SUPERCOMPUTER

- **Data Analysis Example**
  - Per file word count on all the files in a directory
  - Serial to threaded and distributed by using a forall over a parallel distributed array
  - Good scaling even for file I/O (below is for 10K files at 3MB each)



LAPTOP TO SUPERCOMPUTERS BASED ON ARRAY DISTRIBUTION

for loop: each iteration is executed serially by the current task

- predictable execution order, similar to conventional languages

forall loop: all iterations are executed by one or more tasks in no specific order

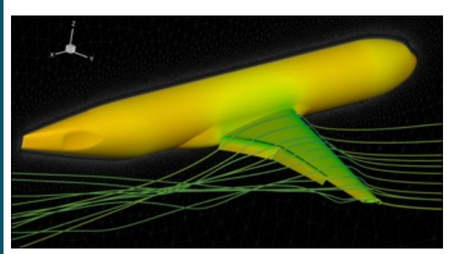
- implemented using one or more tasks, locally or distributed, as determined by the iterand expression

```
forall elem in myLocArr do …        // task-level parallelism over local arrays

forall elem in myDistArr do …       // distributed arrays use tasks on each locale owning part of the array
```

Version of Parquet reader	1 Node/Locale Performance	16 Node/Locale Performance
for loop	0.85 GiB/s	10.75 GiB/s
forall loop	7.46 GiB/s	23.26 GiB/s

benchmark uses 400 Parquet files of size 0.25 GiB each

HOW APPLICATIONS ARE USING CHAPEL

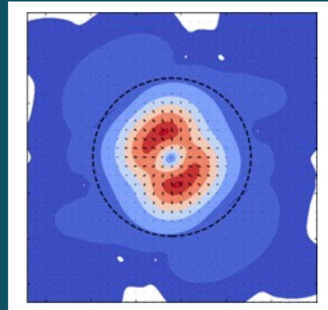


Refactoring existing codes into Chapel (~100K lines of Chapel)

CHAMPS: 3D Unstructured CFD

Éric Laurendeau, Simon Bourgault-Côté, Matthieu Parenteau, et al.

École Polytechnique Montréal



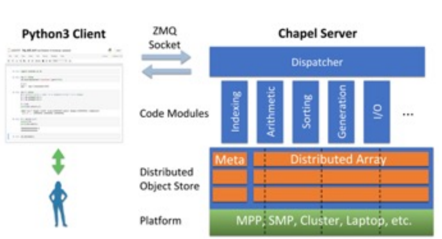
Writing code in Chapel

(~10k lines of including parallel FFT)

ChplUltra: Simulating Ultralight Dark Matter

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

Yale University / University of Auckland



Chapel server for a Python client (~25K lines of Chapel)

Arkouda: NumPy at Massive Scale

Mike Merrill, Bill Reus, et al.

US DoD

IN MEMORIAM

- In Memoriam: Mike Merrill passed last week, and he will be greatly missed.
- Mike was the originator and main developer of Arkouda.

CHAPEL ROADMAP

- **Generate code for GPUs**
  - Nascent support for NVIDIA
  - Exploring AMD and Intel support
- **Rearchitect the compiler**
  - Shed cruft from research prototype days to harden the compiler
  - Reduce compile times
    – potentially via separate compilation / incremental recompilation?
  - Support interpreted / interactive Chapel programming
- **Continue to optimize performance**
- **Release Chapel 2.0**
  - guarantee backwards-compatibility for core language and library
- **Foster a growing Chapel community**

```
// Stream
// Variables stored on GPU
// vector operations executed on GPU
config var n = 1_000_000, alpha = 0.01;

coforall loc in Locales on loc {
  coforall gpu in loc.gpus do on gpu {
    var A, B, C: [1..n] real;

    B = ...;
    C = ...;

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

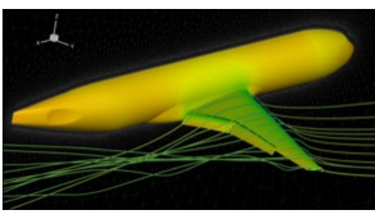
SUMMARY

Chapel cleanly supports...

ease of programming,
high performance, and
portability

Chapel is being used for productive parallel applications at scale

- recent users have reaped its benefits in 10k–100k-line applications



Chapel provides clean ways to transition from laptop development to a cluster/supercomputer

The Chapel Development Team is

- ... working on a number of exciting initiatives!
- ... looking forward to hearing from you!

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

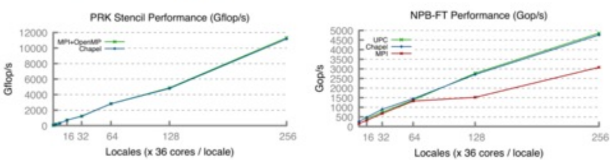
Chapel Characteristics

- **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);
```

THANK YOU

https://chapel-lang.org
@ChapelLanguage