Hewlett Packard Enterprise

CHAPEL OVERVIEW, AND FUTURE OPPORTUNITIES AND CHALLENGES FOR CHAPEL

Michelle Mills Strout, Ben McDonald,
Elliot Ronaghan, and Brad Chamberlain

SIAM PP22: Achieving Productivity at Scale with
Chapel in User Applications
February 24, 2022

CHAPEL TEAM

Chapel is truly a team effort—we’re currently at 19 employees (+ a director), and **we are hiring**

Chapel Development Team at HPE

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

TAKEAWAY FOR THIS TALK

Chapel is a parallel programming language that provides

ease of 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.

OUTLINE

Scientific Computing Challenges

Data Analysis Example

How Applications are Using Chapel

Future Opportunities and Challenges

SCIENTIFIC COMPUTING CHALLENGES

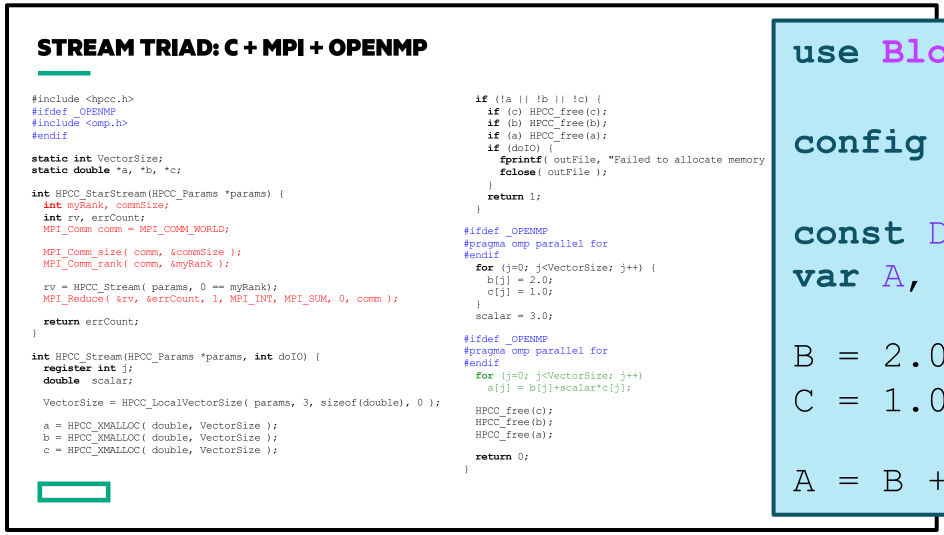
- **Steep learning curve to effectively achieve high performance**
  - Distributed-memory parallelism across nodes (MPI)
  - Parallelism within a node (OpenMP, Pthreads, CUDA, …)
  - Vectorization (intrinsics that are architecture specific)

- **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

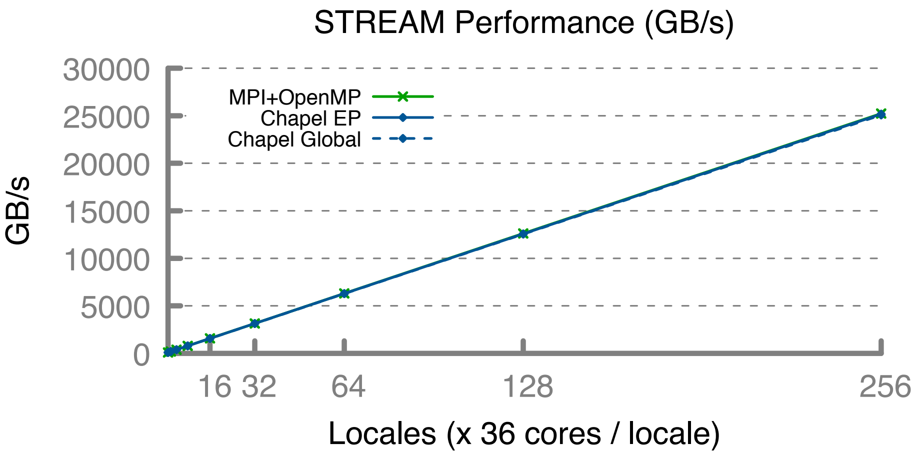


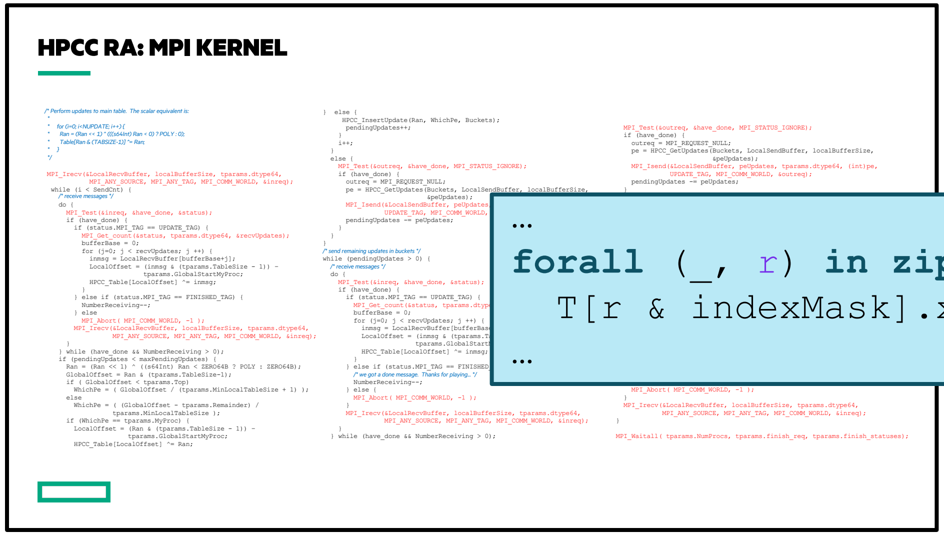
STREAM TRIAD: C + MPI + OPENMP

```c
#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 );
```

```c
  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;
}
```

```chapel
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;
```





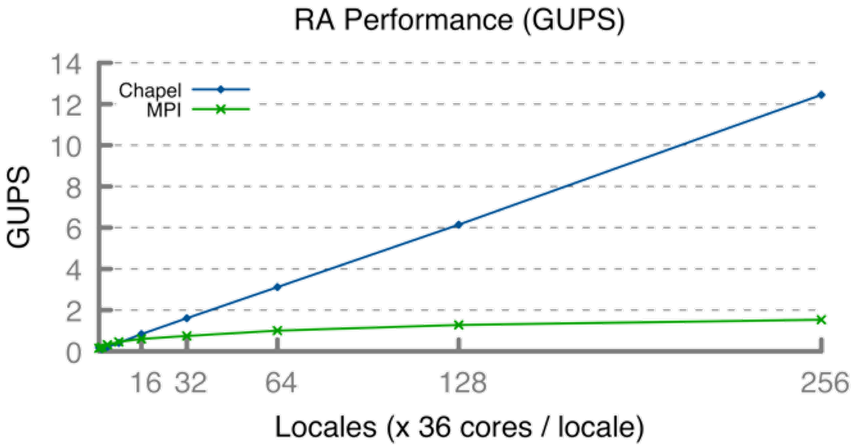
HPCC RA: MPI KERNEL

```c
/* 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;
```

```c
     }  else {
        HPCC_InsertUpdate(Ran, WhichPe, Buckets);
         pendingUpdates++;
       }
       i++;
     }
     else {
       MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
       if (have_done) {
         outreq = MPI_REQUEST_NULL;
         pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                              &peUpdates);
         MPI_Isend(&LocalSendBuffer, peUpdates,
                   UPDATE_TAG, MPI_COMM_WORLD,
         pendingUpdates -= peUpdates;
       }
     }
   }
/* send remaining updates in buckets */
while (pendingUpdates > 0) {
  /* receive messages */
  do {
    MPI_Test(&inreq, &have_done, &status);
    if (have_done) {
      if (status.MPI_TAG == UPDATE_TAG) {
        MPI_Get_count(&status, tparams.dtype
        bufferBase = 0;
        for (j=0; j < recvUpdates; j ++) {
          inmsg = LocalRecvBuffer[bufferBase
          LocalOffset = (inmsg & (tparams.Ta
                        tparams.GlobalStartM
          HPCC_Table[LocalOffset] ^= inmsg;
        }
      } else if (status.MPI_TAG == FINISHED_
        /* we got a done message.  Thanks for playing... */
        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);
```

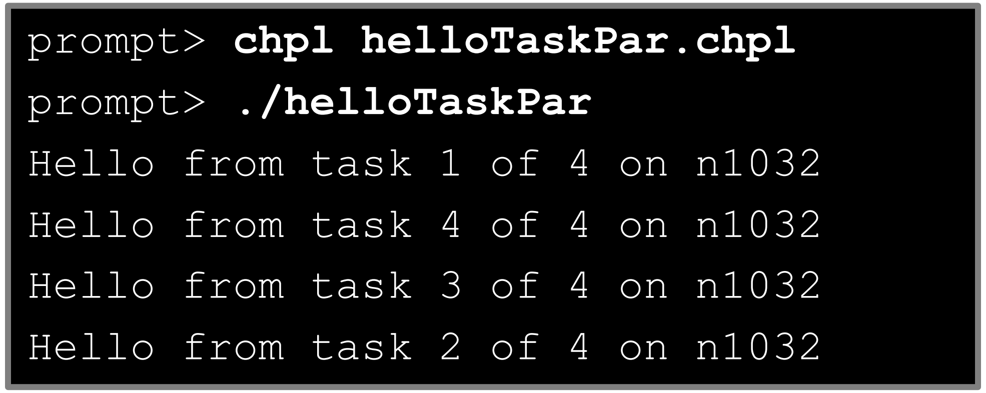
```c
  MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
  if (have_done) {
    outreq = MPI_REQUEST_NULL;
    pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                         &peUpdates);
    MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
              UPDATE_TAG, MPI_COMM_WORLD, &outreq);
    pendingUpdates -= peUpdates;
  }

      MPI_Abort( MPI_COMM_WORLD, -1 );
    }
    MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
              MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
  }

MPI_Waitall( tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);
```

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



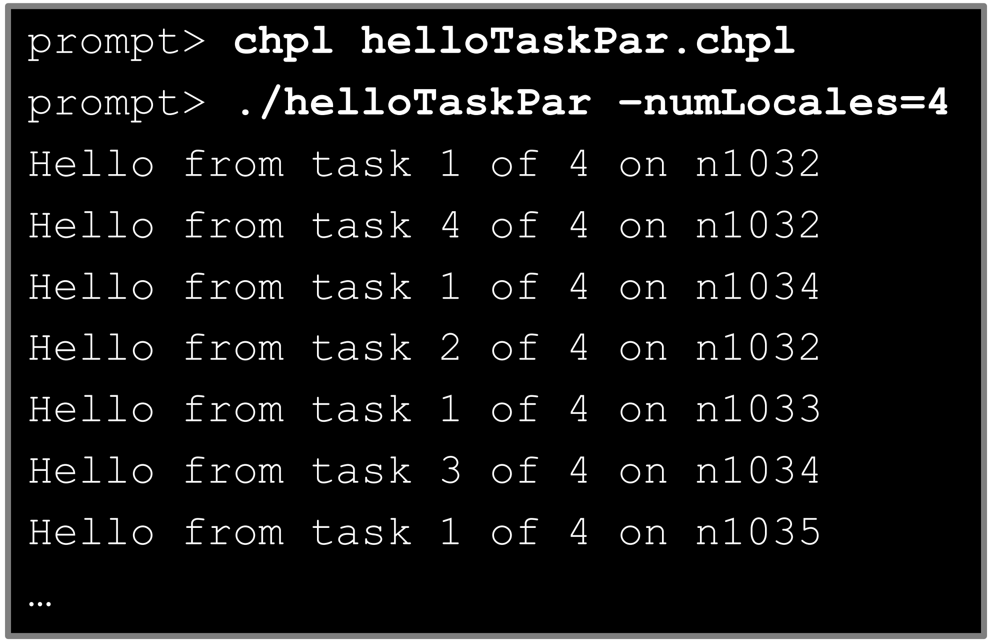
PORTABILITY

- **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)**

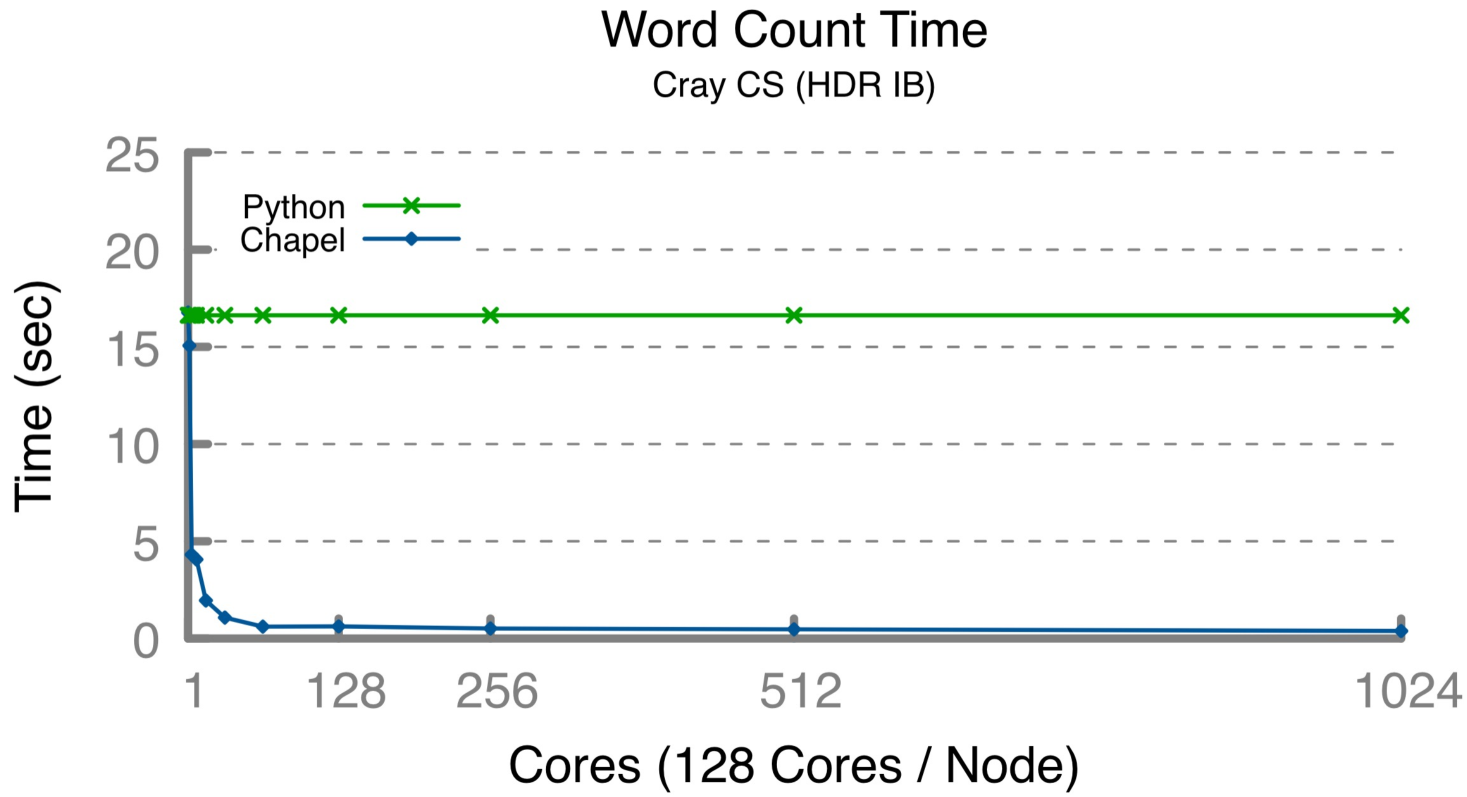


```
prompt> chpl helloTaskPar.chpl
prompt> ./helloTaskPar –numLocales=4
Hello from task 1 of 4 on n1032
Hello from task 4 of 4 on n1032
Hello from task 1 of 4 on n1034
Hello from task 2 of 4 on n1032
Hello from task 1 of 4 on n1033
Hello from task 3 of 4 on n1034
Hello from task 1 of 4 on n1035
…
```

DATA ANALYSIS EXAMPLE

TRANSITIONING 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 100 files at 3MB each)



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

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

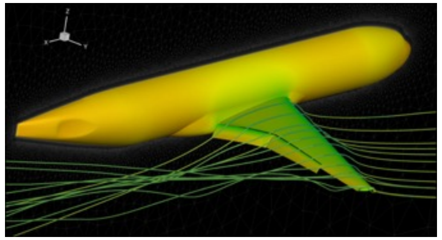
- implemented using one or more tasks, locally or distributed, as determined by the iterand expression

```
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 Locale Performance	16 Locale Performance
Original	0.85 GiB/s	10.75 GiB/s
Parallel+Batch	7.46 GiB/s	23.26 GiB/s

benchmark uses 400 files of size 0.25 GiB each

HOW APPLICATIONS ARE USING CHAPEL

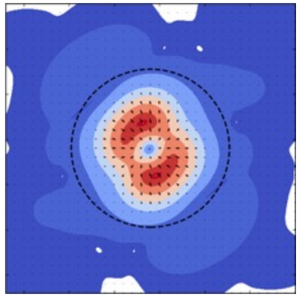


Refactoring existing codes into Chapel (~48K lines of Chapel)

CHAMPS: 3D Unstructured CFD

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

École Polytechnique Montréal

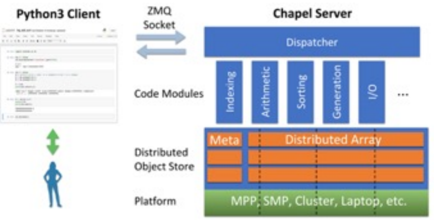


Writing code in Chapel (~10k lines of including parallel FFT)

ChplUltra: Simulating Ultralight Dark Matter

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

Yale University / University of Auckland

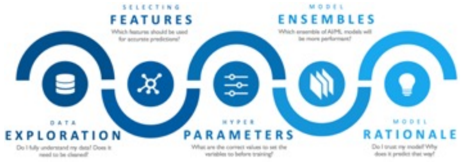


Chapel server for a Python client (~16K lines of Chapel)

Arkouda: NumPy at Massive Scale

Mike Merrill, Bill Reus, et al.

US DoD


SELECTING FEATURES
Which features should be used for accurate predictions?
MODEL ENSEMBLES
Which ensemble of AI/ML models will be more performant?
DATA EXPLORATION
Do I fully understand my data? Does it need to be cleaned?
HYPER PARAMETERS
What are the correct values to set the variables to before training?
MODEL RATIONALE
Do I trust my model? Why does it predict that way?

Chapel providing distributed parallelism for ML training (~8k lines of Chapel)

CrayAI: Distributed Machine Learning

Hewlett Packard Enterprise

FUTURE OPPORTUNITIES AND CHALLENGES FOR CHAPEL

- **Generate code for GPUs (see Engin talk in MS79)**
  - How will the compiler need to evolve? Will the language need to?
- **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**

```
// HPCC Stream
// Variables stored on GPU
// forall’s are executed on GPU
on here.getChild(1) {
  var A, B, C: [1..n] real;
  const alpha = 2.0;

  forall b in B do b = 1.0;
  forall c in C do c = 2.0;

  forall a, b, c in zip(A, B, C) do
    a = b + alpha * c;
}
```

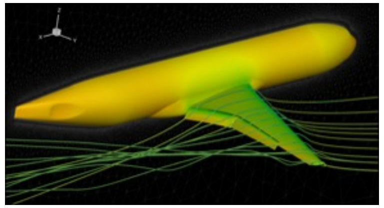
SUMMARY

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–48k-line applications



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

The Chapel Development Team is

- … at 19 people and is hiring!
- … working on a number of exciting initiatives!
- … looking forward to hearing from you!

CHIUW 2022 SUBMISSIONS DUE APRIL 15TH

The Chapel Parallel Programming Language

Home

What is Chapel?
What's New?

Upcoming Events
Job Opportunities

How Can I Learn Chapel?

Contributing to Chapel
Community

Download Chapel
Try Chapel Online

CHIUW 2022

The 9th Annual
Chapel Implementers and Users Workshop

June 10, 2022
free and online in a virtual format

Call For Papers and Talks

CHAPEL RESOURCES

Chapel homepage: https://chapel-lang.org

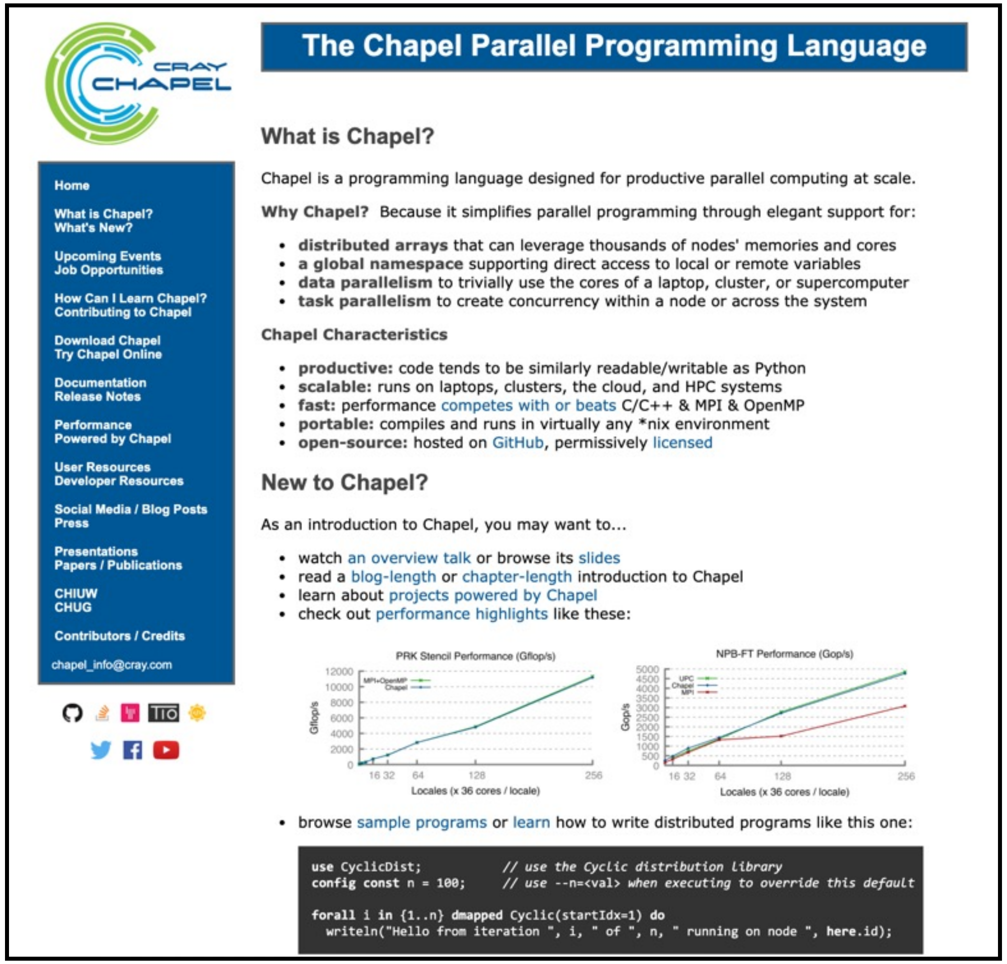
- (points to all other resources)

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