



**Hewlett Packard
Enterprise**

PARALLEL PROGRAMMING WITH CHAPEL

Brad Chamberlain
PNW PLSE, May 9, 2023



**Hewlett Packard
Enterprise**

**OR:
PERFORMANCE AT ANY COST?
HPC* AND 24H OF LE MANS**

Brad Chamberlain
PNW PLSE, May 9, 2023

HPC = High Performance Computing

PARALLEL COMPUTING HAS BECOME UBIQUITOUS

Traditional parallel computing:

- supercomputers
- commodity clusters

Today:

- multicore processors
- GPUs
- cloud computing



OAK RIDGE NATIONAL LABORATORY'S FRONTIER SUPERCOMPUTER



- 74 HPE Cray EX cabinets
- 9,408 AMD CPUs, 37,632 AMD GPUs
- 700 petabytes of storage capacity, peak write speeds of 5 terabytes per second using Cray ClusterStor storage system
- HPE Slingshot networking cables providing 100 GB/s network bandwidth.

TOP500

1

Built by HPE,
ORNL's Frontier
supercomputer
is #1 on the
TOP500.

1.1 exaflops of
performance.

TOP 500
The List.

GREEN500

2

Built by HPE,
ORNL's TDS and
full system are
ranked #2 & #6
on the Green500.

62.68 gigaflops/watt
power efficiency for
ORNL's TDS system,
52.23 gigaflops/watt
power efficiency for full
system.

The
GREEN
500

HPL-MxP

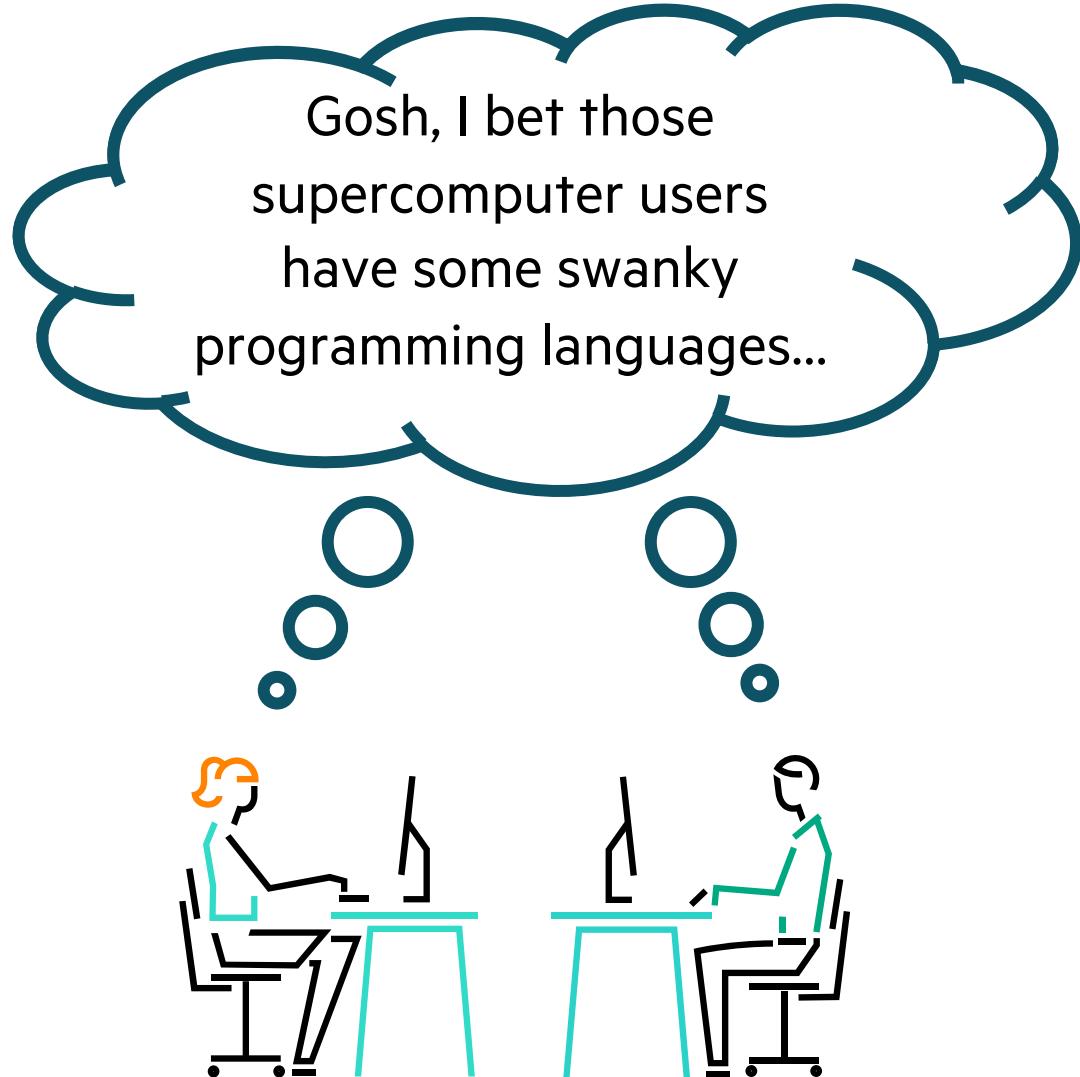
1

Built by HPE,
ORNL's Frontier
supercomputer
is #1 on the
HPL-MxP list.

7.9 exaflops on the
HPL-MxP benchmark
(formerly HPL-AI).

TOP 500
The List.

A STRAINED(?) ANALOGY



HPC BENCHMARKS USING CONVENTIONAL PROGRAMMING APPROACHES

STREAM TRIAD: C + MPI + OPENMP

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

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

int HPCC_StarStream(HPCC_Parms *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_Parms *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 );

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

    return 0;
}
```

```
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 (%d).\n", VectorSize );
        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;
}
```

HPCC RA: MPI KERNEL

```
/* Perform updates to main table. The scalar equivalent is:
 * for (i=0;i<NUPDATE;i++) {
 *     Ran = (Ran << 1) ^ (((s64)4m) Ran < 0 ? POLY : 0);
 *     Table[Ran & (TABSIZ-1)] = Ran;
 * }
 */

MPI_Irecv(4LocalRecvBuffer, 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++) {
                    imsg = LocalRecvBuffer(bufferBase+j);
                    LocalOffset = (imsg & (tparams.TableSize - 1)) -
                                  tparams.GlobalStartMyProc;
                    HPCC_Table[LocalOffset] ^= imsg;
                }
            } else if (status.MPI_TAG == FINISHED_TAG) {
                NumberReceiving--;
            } else
                MPI_Abort( MPI_COMM_WORLD, -1 );
            MPI_Irecv(4LocalRecvBuffer, localBufferSize, tparams.dtype64,
                      MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
        }
    } while (have_done && NumberReceiving > 0);
    if (pendingUpdates < maxPendingUpdates) {
        Ran = (Ran << 1) ^ ((s64)1) ^ ((s64)1) Ran < ZERO64B ? POLY : ZERO64B;
        GlobalOffset = Ran & (tparams.TableSize-1);
        if ( GlobalOffset < tparams.Top )
            WhichPe = (GlobalOffset / (tparams.MiniLocalTableSize + 1));
        else
            WhichPe = (GlobalOffset - tparams.MiniLocalTableSize) /
                      tparams.MiniLocalTableSize;
        if (WhichPe == tparams.MyProc) {
            LocalOffset = (Ran & (tparams.TableSize - 1)) -
                          tparams.GlobalStartMyProc;
            HPCC_Table[LocalOffset] ^= Ran;
        }
    }
    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(4LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
                  UPDATE_TAG, MPI_COMM_WORLD, &outreq);
        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.dtype64, &recvUpdates);
                bufferBase = 0;
                for (j=0; j < recvUpdates; j++) {
                    imsg = LocalRecvBuffer(bufferBase+j);
                    LocalOffset = (imsg & (tparams.TableSize - 1)) -
                                  tparams.GlobalStartMyProc;
                    HPCC_Table[LocalOffset] ^= imsg;
                }
            } else if (status.MPI_TAG == FINISHED_TAG) {
                /* we got a done message. Thanks for playing.. */
                NumberReceiving--;
            } else
                MPI_Abort( MPI_COMM_WORLD, -1 );
            MPI_Irecv(4LocalRecvBuffer, localBufferSize, tparams.dtype64,
                      MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
        }
    } while (have_done && NumberReceiving > 0);
    MPI_Waitall( tparams.NumProcs, tparams.finish_red, tparams.finish_statuses);
}
```

WHAT IS CHAPEL?

Chapel: A modern parallel programming language

- portable & scalable
- open-source & collaborative
- pioneered and developed in Seattle (Cray Inc. / HPE)



Goals:

- Support general parallel programming
- Make parallel programming at scale far more productive



HPC BENCHMARKS: CONVENTIONAL APPROACHES VS. CHAPEL

STREAM TRIAD: C + MPI + OPENMP

```

#include <hpcc.h>
#include _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 );
    if (doIO) {
        #ifdef _OPENMP
        #pragma omp parallel
        #endif
        for (j=0; j < VectorSize; j++)
            a[j] = b[j];
    }
    scalar = 3.0;
    A = B = 2.0;
    C = 1.0;
    A = B + alpha * C;
}

```

```

use BlockDist;

config const m = 1000,
      alpha = 3.0;

const Dom = Block.createDomain({1..m});

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<RASize;i++) {
       Ra[i] = Ra[i] - 27.0*Ra[i]*Ra[i] + 0.7*POLY(i);
       Ra[i] = Ra[i]/(TANB2Z2-12) ~ Ra[i];
   }
   */

MPI_Recv(iLocalRaBuffer, localBufferSize, tparams.dtyped4,
         MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, iinseq);
while (i < SendCount) {
    /* receive message */
    MPI_Datatype, ihaveDone, istatus);
    if (status.MPI_TAG == UPDATE_TAG) {
        if (i < tparams.RASize) {
            for (j=0;j<localBufferSize;j+=4)
                imsg = LocalRecvbuffer[iBuffBase+j];
            locOffSet = (i*tparams.RASize+1)-
                        tparams.RASize*globalStartIndex;
            HPMC_TableLocOffraSet(locOffSet, ~imsg);
        }
        else if (status.MPI_TAG == FINISHED_TAG) {
            if (i < tparams.RASize) {
                MPI_Irecv(iLocalRaBuffer, localBufferSize, tparams.dtyped4,
                         MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, iinseq);
                while (haveDone < NumberReceiving) {
                    if (pendingUpdates < maxPendingUpdates) {
                        Ra = Ra << 1; (readInt(Ra) < 0 ? 2 : ZERO48);
                        Ra = Ra >> 1; (readInt(Ra) < 0 ? 1 : 0);
                    }
                    if (GlobalOffset < tparams.Remander) /
                        tparams.RASize;
                    if (GlobalOffset == tparams.Remander) /
                        tparams.RASize;
                    if (GlobalOffset = (Ra + (tparams.RASize-1))-
                        tparams.RASize*globalStartIndex);
                    while (haveDone < NumberReceiving > 0);
                    HPMC_TableLocOffraSet(locOffSet, ~Ra);
                }
            }
            if (i < tparams.RASize) {
                MPI_Abort(MPI_COMM_WORLD, -1);
                MPI_Recv(iLocalRaBuffer, localBufferSize, tparams.dtyped4,
                        MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, iinseq);
                MPI_Waitall();
            }
        }
    }
}

```

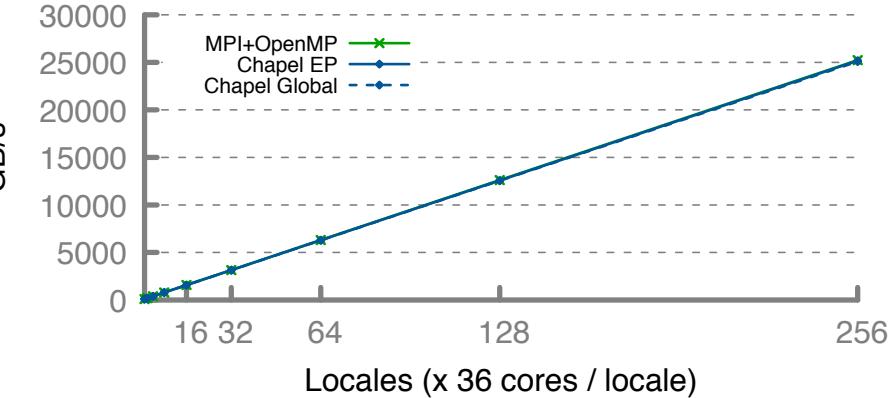
```

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

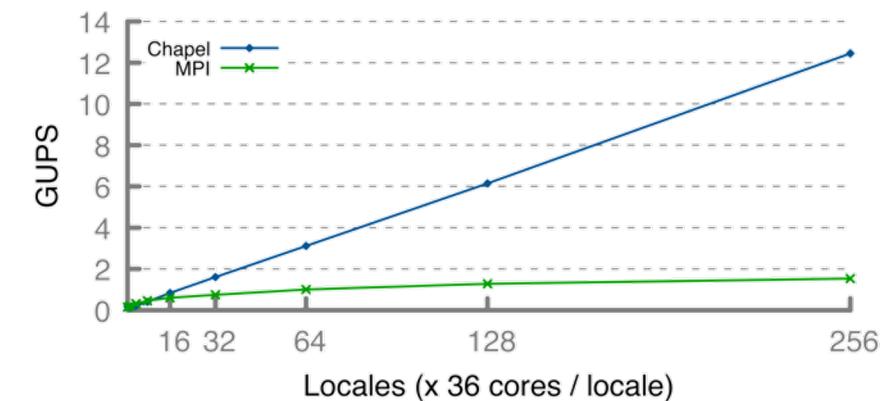
```

GB/s

STREAM Performance (GB/s)



RA Performance (GUPS)



HIGHLIGHT #1: CHAPEL SUPPORT FOR GPUS

Typical 2018-era Chapel Talk:

- **Me:** “Chapel’s generality goal is to support any parallel algorithm on any parallel architecture.”
- **Audience Q:** “So... does Chapel support GPUs?”
- **Me (with head bowed in shame):** “Only through interoperability with CUDA/OpenCL/OpenACC/OpenMP/...”

More recently:

- We’re targeting GPUs using Chapel’s traditional features for parallelism and locality

Let’s build up to a simple, “low-level” example using Stream Triad...



STREAM TRIAD: SHARED MEMORY

stream-ep.chpl

```
config var n = 1_000_000,  
       alpha = 0.01;
```

```
var A, B, C: [1..n] real;  
A = B + alpha * C;
```

Declare three arrays of size 'n'

Whole-array operations compute
Stream Triad in parallel

So far, this is simply a multi-core program

Nothing refers to remote locales (nodes),
explicitly or implicitly

STREAM TRIAD: DISTRIBUTED MEMORY

```
stream-ep.chpl
```

```
config var n = 1_000_000,  
        alpha = 0.01;  
  
coforall loc in Locales {  
    on loc {  
        var A, B, C: [1..n] real;  
        A = B + alpha * C;  
    }  
}
```

‘coforall’ loops execute each iteration as an independent task

the array of locales (nodes) on which this program is running

have each task run ‘on’ its locale

then run multi-core Stream, as before

This is a CPU-only program

Nothing refers to GPUs, explicitly or implicitly

STREAM TRIAD: DISTRIBUTED MEMORY, GPUS ONLY

stream-ep.chpl

```
config var n = 1_000_000,  
       alpha = 0.01;  
  
coforall loc in Locales {  
    on loc {  
  
        coforall gpu in here.gpus do on gpu {  
            var A, B, C: [1..n] real;  
            A = B + alpha * C;  
        }  
    }  
}
```

Use a similar ‘coforall’ + ‘on’ idiom
to run a Triad concurrently
on each of this locale’s GPUs

This is a GPU-only program

Nothing other than coordination code
runs on the CPUs

STREAM TRIAD: DISTRIBUTED MEMORY, GPUS AND CPUS

stream-ep.chpl

```
config var n = 1_000_000,  
        alpha = 0.01;  
  
coforall loc in Locales {  
    on loc {  
        cobegin {  
            coforall gpu in here.gpus do on gpu {  
                var A, B, C: [1..n] real;  
                A = B + alpha * C;  
            }  
            {  
                var A, B, C: [1..n] real;  
                A = B + alpha * C;  
            }  
        }  
    }  
}
```

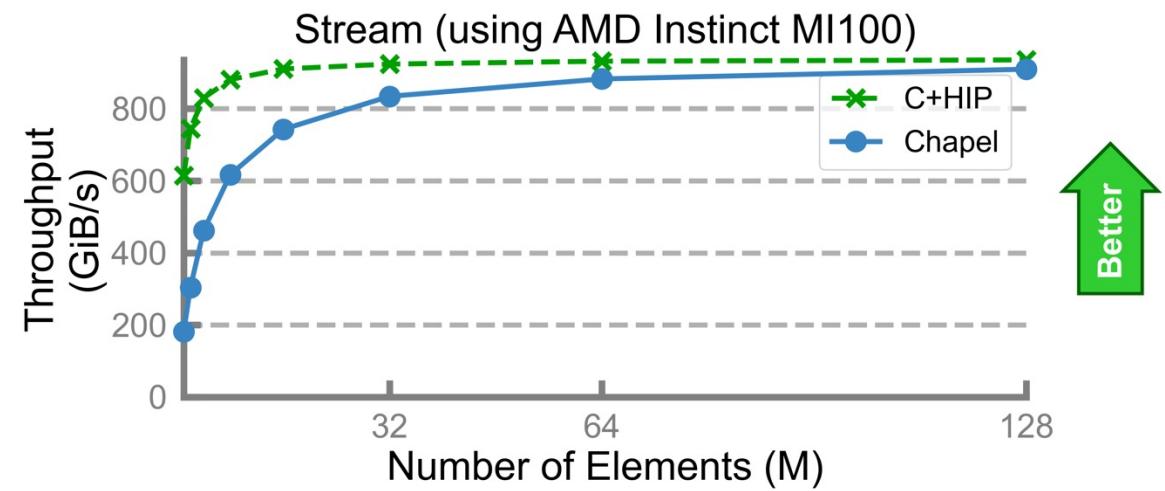
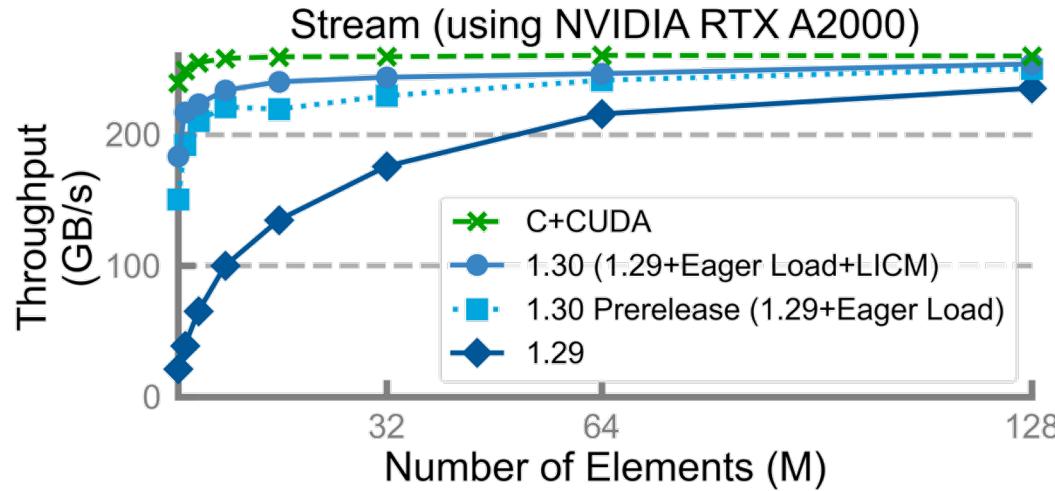
'cobegin { ... }' creates a task per child statement

one task runs our multi-GPU triad

the other runs the multi-CPU triad

This program uses all CPUs and GPUs across all of your compute nodes

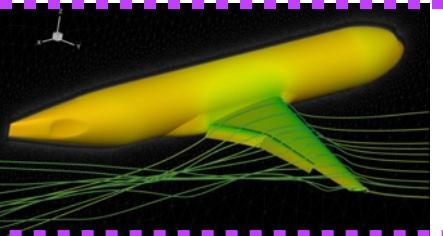
STREAM TRIAD: PERFORMANCE VS. REFERENCE VERSIONS



Performance vs. reference versions has become increasingly competitive over the past 4 months

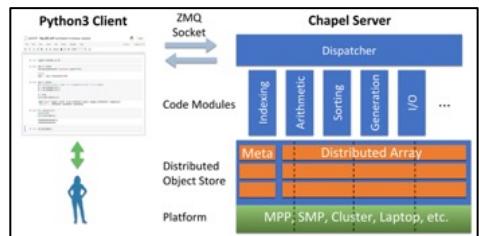


HIGHLIGHT #2: APPLICATIONS OF CHAPEL



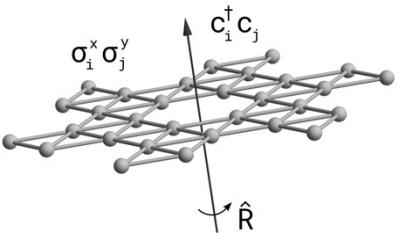
CHAMPS: 3D Unstructured CFD

Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



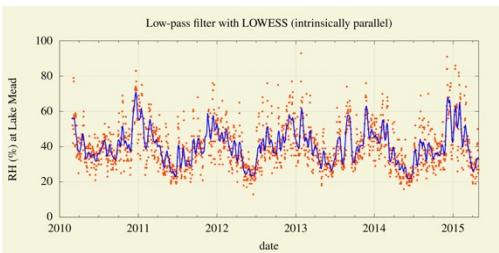
Arkouda: Interactive Data Science at Massive Scale

Mike Merrill, Bill Reus, et al.
U.S. DoD



Lattice-Symmetries: a Quantum Many-Body Toolbox

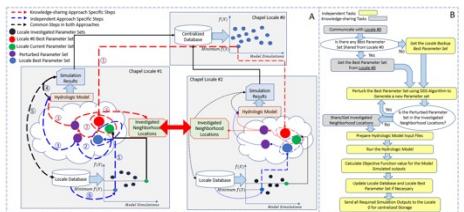
Tom Westerhout
Radboud University



Desk dot chpl: Utilities for Environmental Eng.

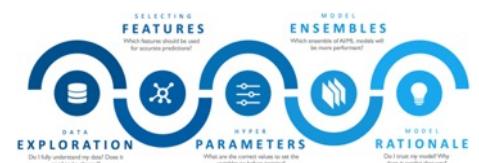
Nelson Luis Dias

The Federal University of Paraná, Brazil



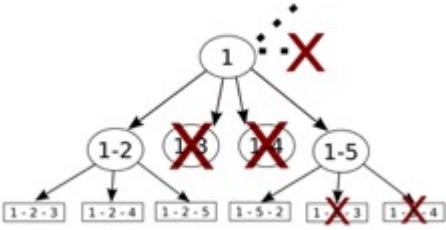
Chapel-based Hydrological Model Calibration

Marjan Asgari et al.
University of Guelph



CrayAI HyperParameter Optimization (HPO)

Ben Albrecht et al.
Cray Inc. / HPE



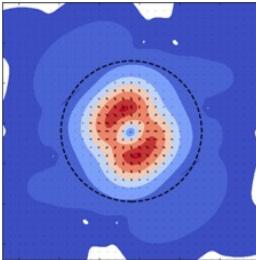
ChOp: Chapel-based Optimization

T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



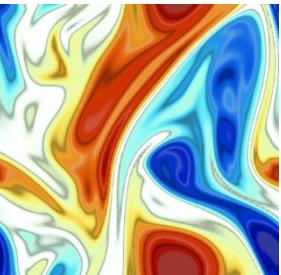
RapidQ: Mapping Coral Biodiversity

Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



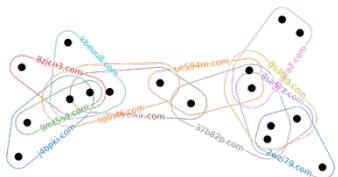
ChplUltra: Simulating Ultralight Dark Matter

Nikhil Padmanabhan, J. Luna Zagorac, et al.
Yale University et al.



ChapQG: Layered Quasigeostrophic CFD

Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



CHGL: Chapel Hypergraph Library

Louis Jenkins, Cliff Joslyn, Jesun Firoz, et al.
PNNL

?

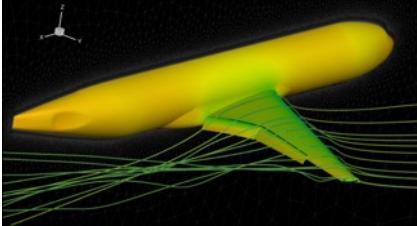
Your Application Here?

(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



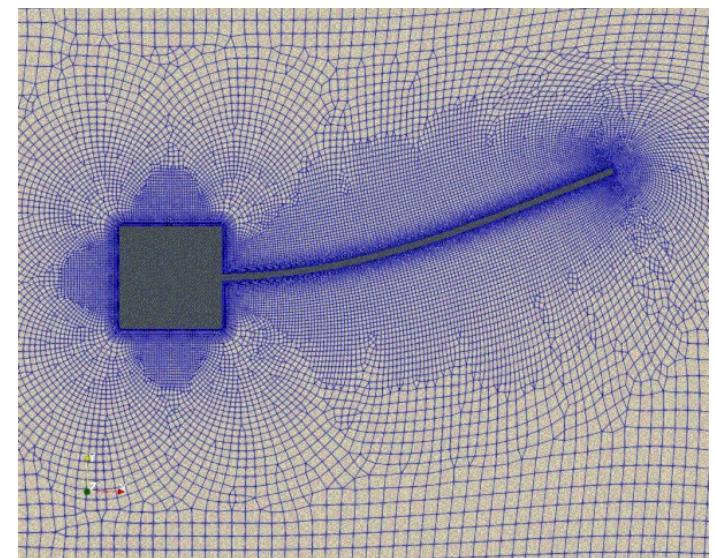
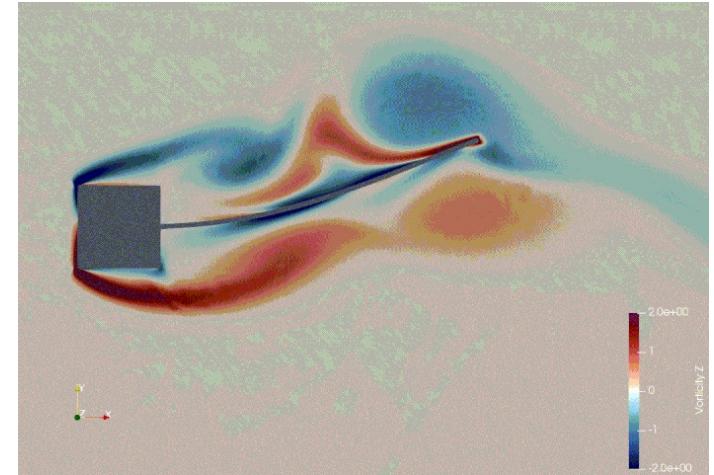
Who wrote it?

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



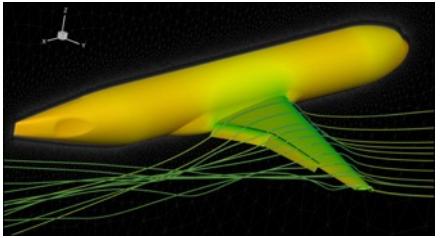
Why Chapel?

- students found it far more productive to use
- enabled them to compete with more established CFD centers



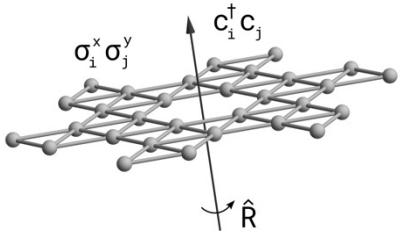
(images provided by the CHAMPS team and used with permission)

APPLICATIONS OF CHAPEL



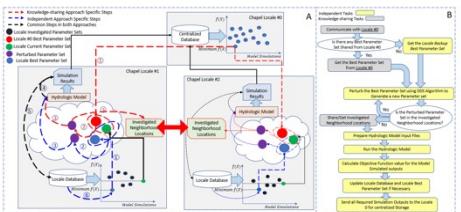
CHAMPS: 3D Unstructured CFD

Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



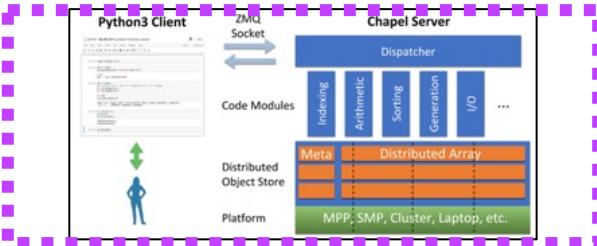
Lattice-Symmetries: a Quantum Many-Body Toolbox

Tom Westerhout
Radboud University



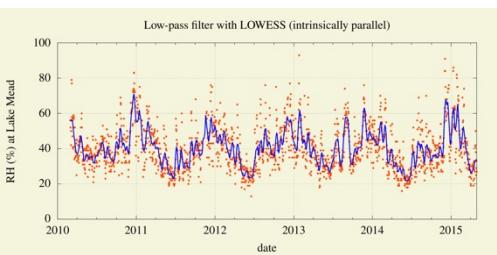
Chapel-based Hydrological Model Calibration

Marjan Asgari et al.
University of Guelph



Arkouda: Interactive Data Science at Massive Scale

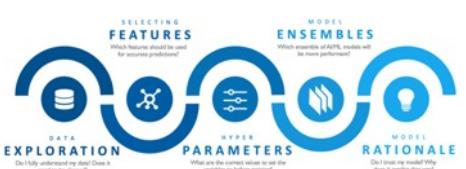
Mike Merrill, Bill Reus, et al.
U.S. DoD



Desk dot chpl: Utilities for Environmental Eng.

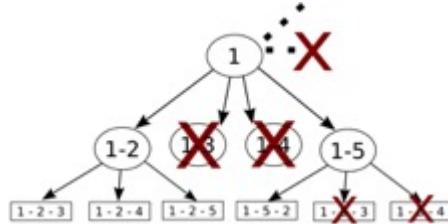
Nelson Luis Dias

The Federal University of Paraná, Brazil



CrayAI HyperParameter Optimization (HPO)

Ben Albrecht et al.
Cray Inc. / HPE



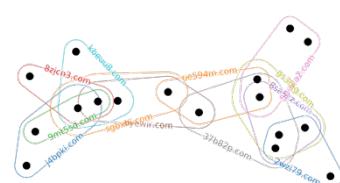
ChOp: Chapel-based Optimization

T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



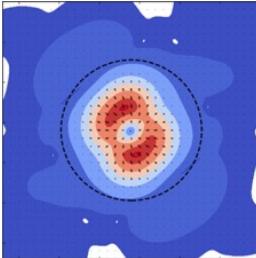
RapidQ: Mapping Coral Biodiversity

Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



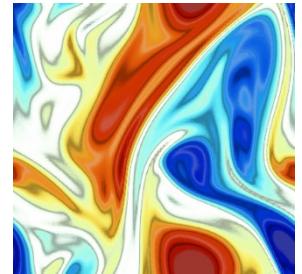
CHGL: Chapel Hypergraph Library

Louis Jenkins, Cliff Joslyn, Jesun Firoz, et al.
PNNL



ChPlUltra: Simulating Ultralight Dark Matter

Nikhil Padmanabhan, J. Luna Zagorac, et al.
Yale University et al.



ChapQG: Layered Quasigeostrophic CFD

Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.

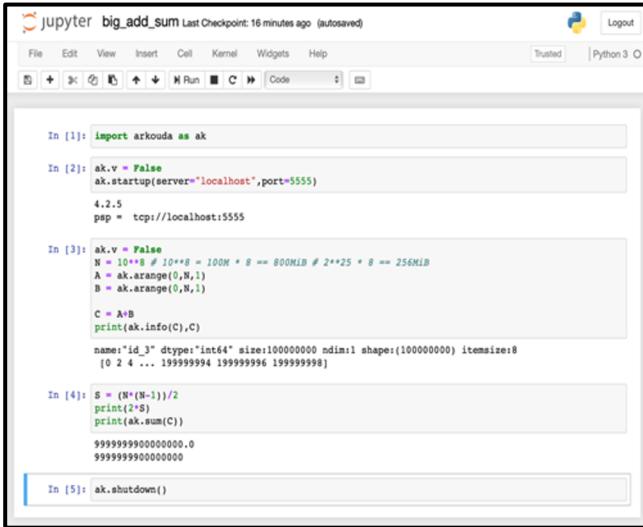


Your Application Here?

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

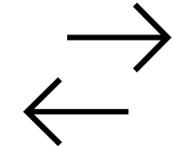
ARKOUDA: A PYTHON LIBRARY AND FRAMEWORK FOR INTERACTIVE HPC

Arkouda Client (written in Python)



A screenshot of a Jupyter Notebook interface. The title bar says "jupyter big_add_sum Last Checkpoint: 16 minutes ago (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The code cell contains Python code for initializing the Arkouda library, connecting to a local server, creating large arrays, performing matrix multiplication, printing results, and shutting down. The output cell shows the resulting array and its sum.

```
In [1]: import arkouda as ak
In [2]: ak.v = False
ak.startup(server="localhost", port=5555)
4.2.5
psp = tcp://localhost:5555
In [3]: ak.v = True
N = 10**8 # 10**8 = 100M * 8 == 800MB # 2**25 * 8 == 256MB
A = ak.arange(0,N,1)
B = ak.arange(0,N,1)
C = A*B
print(ak.info(C),C)
name:id_3 dtype:int64 size:100000000 ndim:1 shape:(100000000) itemsize:8
[0 2 4 ... 199999994 199999996 199999998]
In [4]: S = (N*(N-1))/2
print(2*S)
print(ak.sum(C))
9999999900000000.0
9999999900000000.
In [5]: ak.shutdown()
```



Arkouda Server (written in Chapel)



User writes Python code in Jupyter,
making familiar NumPy/Pandas calls



ARKOUDA SUMMARY

What is it?

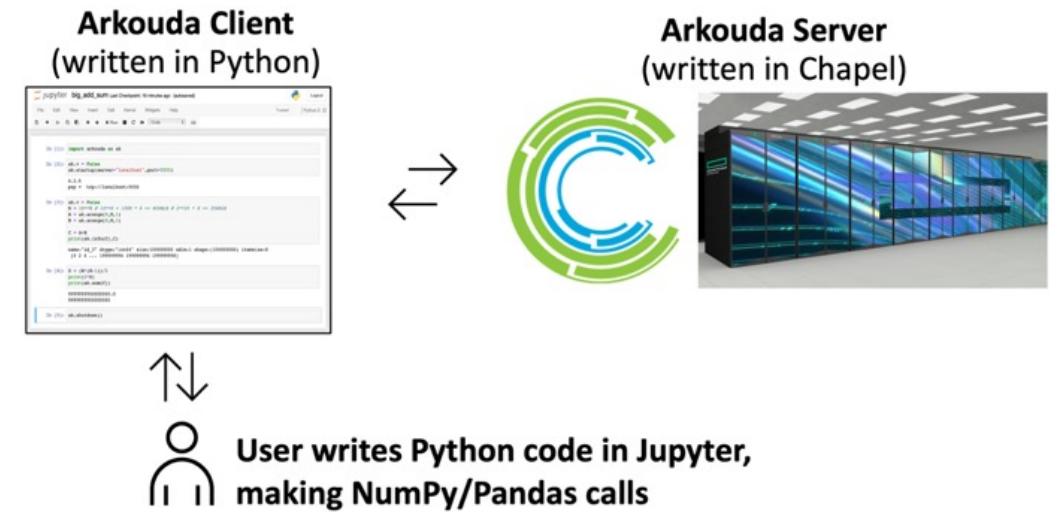
- A client-server framework for interactive supercomputing in Python
- ~30k lines of Chapel and ~25k lines of Python, written since 2019
- Open-source: <https://github.com/Bears-R-Us/arkouda>

Who wrote it?

- Mike Merrill, Bill Reus, *et al.*, US DoD

Why Chapel?

- ability to develop on laptop, deploy on supercomputer
- close to Pythonic



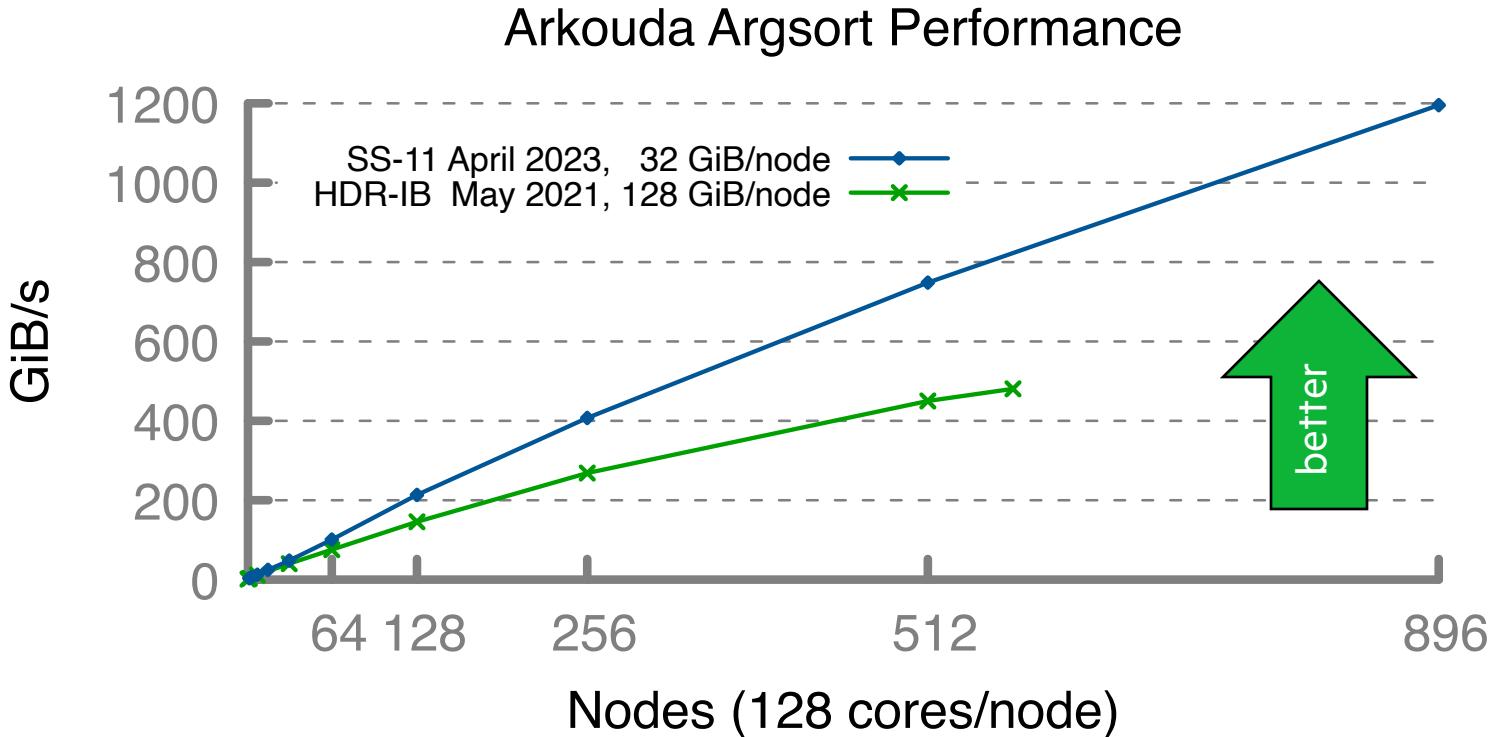
SCALABILITY OF ARKOUDA'S ARG SORT ROUTINE

HPE Cray EX (spring 2023)

- 114,688 cores of AMD Rome
- Slingshot-11 network (200 Gb/s)
- 28 TiB of 8-byte values
- 1200 GiB/s (24 seconds elapsed time)

HPE Apollo (summer 2021)

- 73,728 cores of AMD Rome
- HDR Infiniband network (100 Gb/s)
- 72 TiB of 8-byte values
- 480 GiB/s (2.5 minutes elapsed time)



A notable performance achievement in ~100 lines of Chapel



HIGHLIGHT #3: THE CHAPEL TEAM AT HPE HAS GROWN

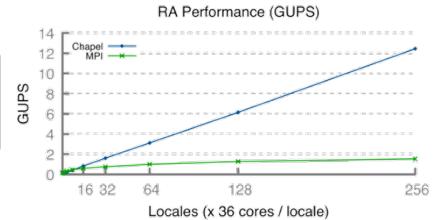


SUMMARY

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

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



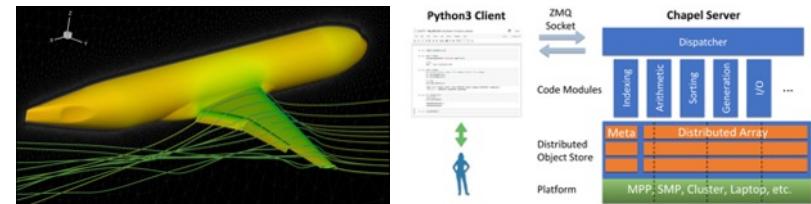
Vendor-neutral GPU support is maturing rapidly

- fleshes out an overdue aspect of “any parallel hardware”

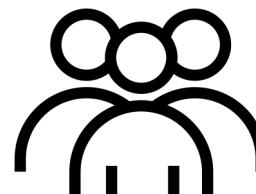
```
coforall gpu in here.gpus do on gpu {  
    var A, B, C: [1..n] real;  
    A = B + alpha * C;  
}
```

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



We're interested in helping new users and fostering new collaborations



CHAPEL RESOURCES

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

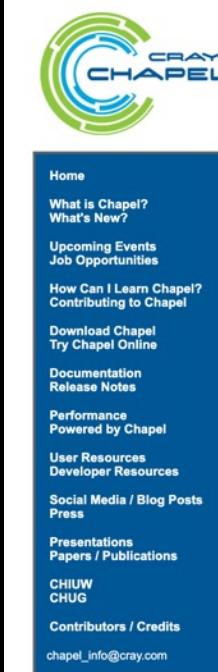
- (points to all other resources)

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

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

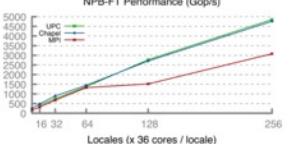
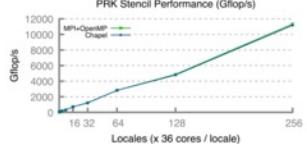
Chapel Characteristics

- 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

