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.

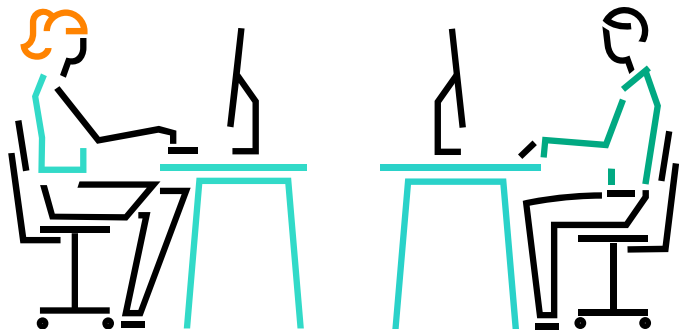
7.9 exaflops on the HPL-MxP benchmark (formerly HPL-AI).

TOP 500 The List.

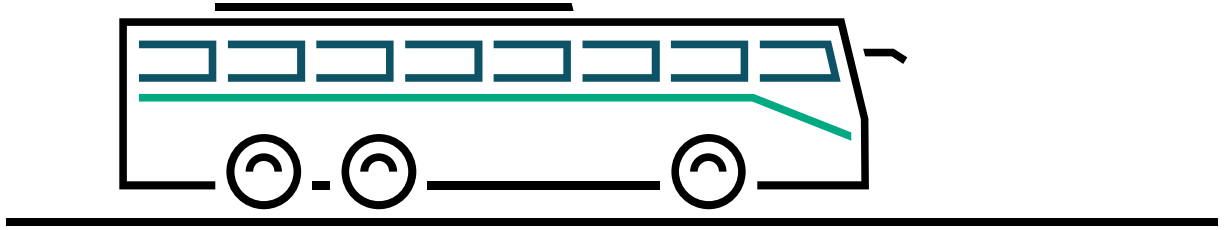
Source: May 30, 2022 Top500 release, HPL-MxP mixed-precision benchmark (formerly HPL-AI).

A STRAINED(?) ANALOGY

Gosh, I bet those supercomputer users have some swanky programming languages...



Gosh, those Le Mans racers must have an enviable driving experience...



HPC BENCHMARKS USING CONVENTIONAL PROGRAMMING APPROACHES

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 (%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) ^ ((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;
    } 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, 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 ++) {
          inmsg = LocalRecvBuffer[bufferBase+j];
          LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                        tparams.GlobalStartMyProc;
          HPCC_Table[LocalOffset] ^= inmsg;
        }
      } 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(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    }
  } while (have_done && NumberReceiving > 0);

  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;
  }
}
/* send our done messages */
for (proc_count = 0 ; proc_count < tparams.NumProcs ; ++proc_count) {
  if (proc_count == tparams.MyProc) { tparams.finish_req[tparams.MyProc] =
                                        MPI_REQUEST_NULL; continue; }
  /* send garbage - who cares, no one will look at it */
  MPI_Isend(&Ran, 0, tparams.dtype64, proc_count, FINISHED_TAG,
            MPI_COMM_WORLD, tparams.finish_req + proc_count);
}
/* Finish everyone else up... */
while (NumberReceiving > 0) {
  MPI_Wait(&inreq, &status);
  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) {
    /* 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);
}

MPI_Waitall( tparams.NumProcs, tparams.finish_req, 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:

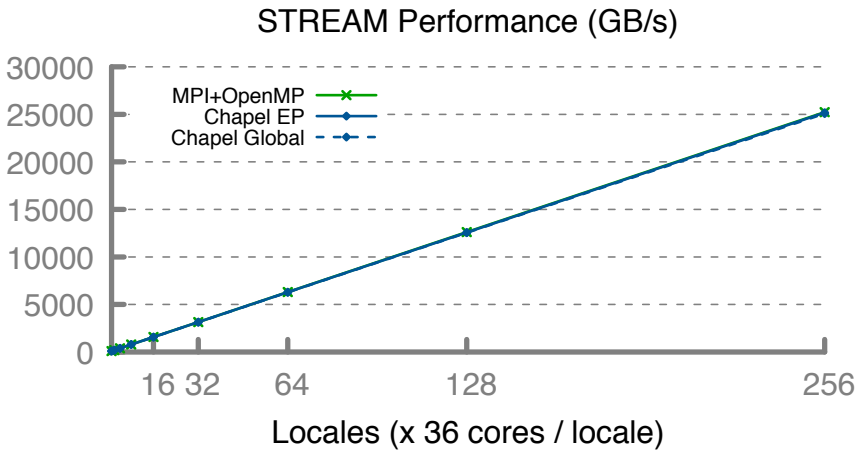
- Support general parallel programming
- Make parallel programming at scale far more productive

HPC BENCHMARKS: CONVENTIONAL APPROACHES VS. CHAPEL

STREAM TRIAD: C + MPI + OPENMP

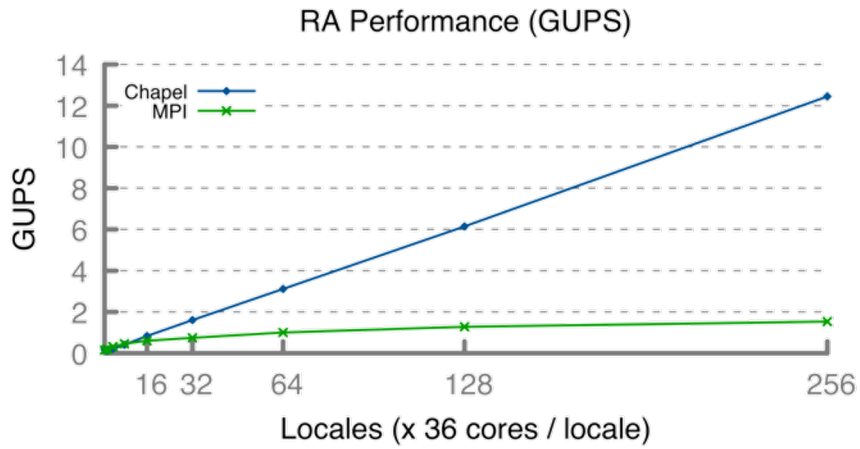
```
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

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 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

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



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;
      }
    }
  }
}
```

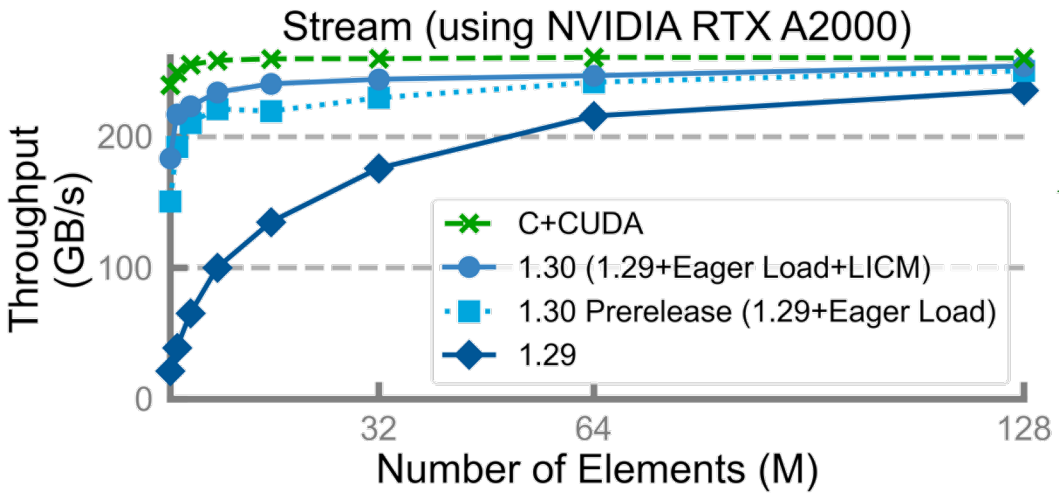
‘cobegin { … }’ creates a task per child statement

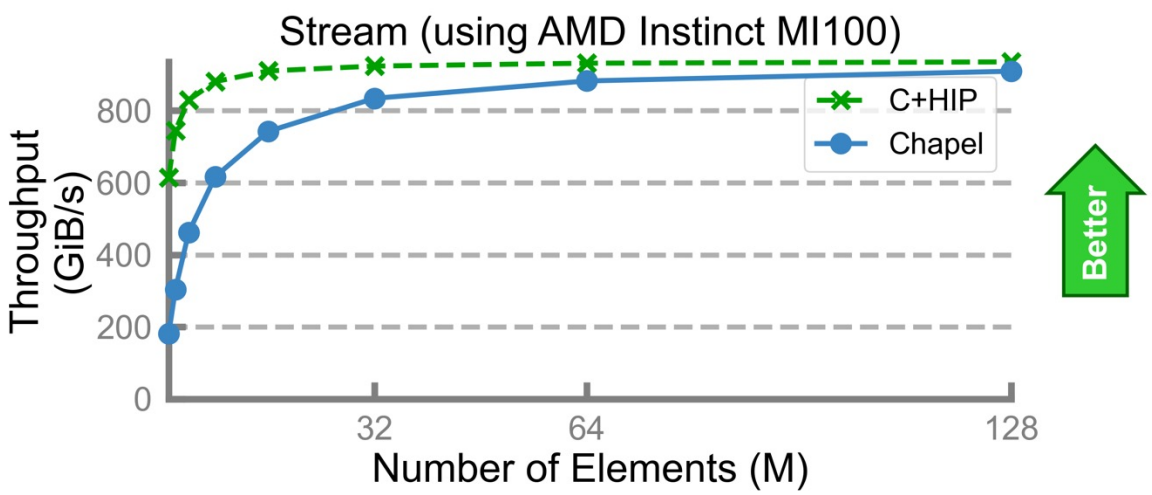
one task runs our multi-GPU triad

the other runs the multi-CPU triad

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

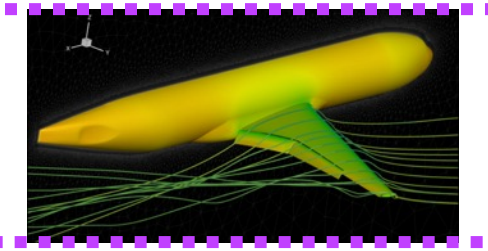
STREAM TRIAD: PERFORMANCE VS. REFERENCE VERSIONS



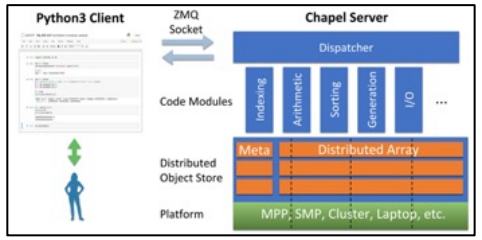


Performance vs. reference versions has become increasingly competitive over the past 4 months

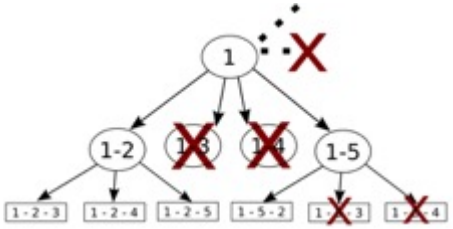
HIGHLIGHT #2: APPLICATIONS OF CHAPEL



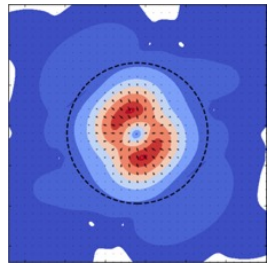
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



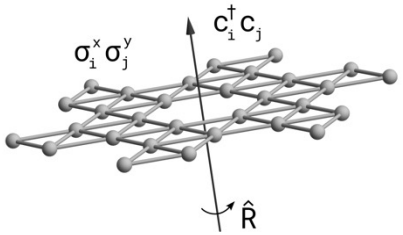
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



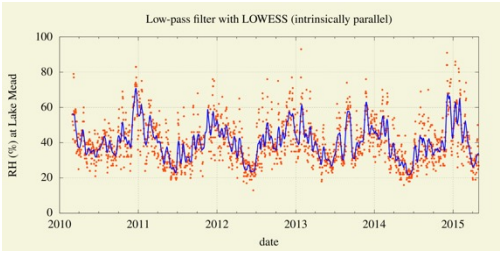
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



ChplUltra: Simulating Ultralight Dark Matter
Nikhil Padmanabhan, J. Luna Zagorac, et al.
Yale University et al.

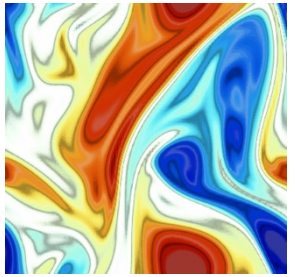


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

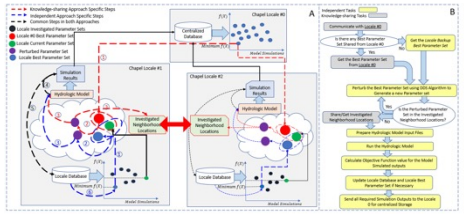


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

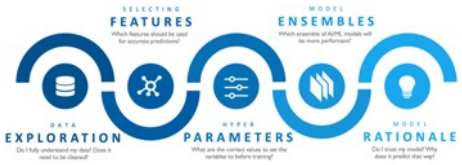
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



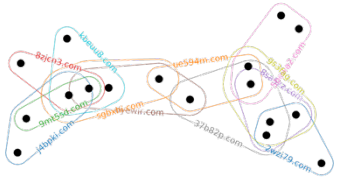
ChapQG: Layered Quasigeostrophic CFD
Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



CrayAI HyperParameter Optimization (HPO)
Ben Albrecht et al.
Cray Inc. / HPE



CHGL: Chapel Hypergraph Library
Louis Jenkins, Cliff Joslyn, Jesun Firoz, et al.
PNNL

?

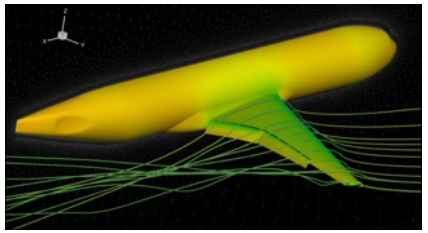
Your Application Here?

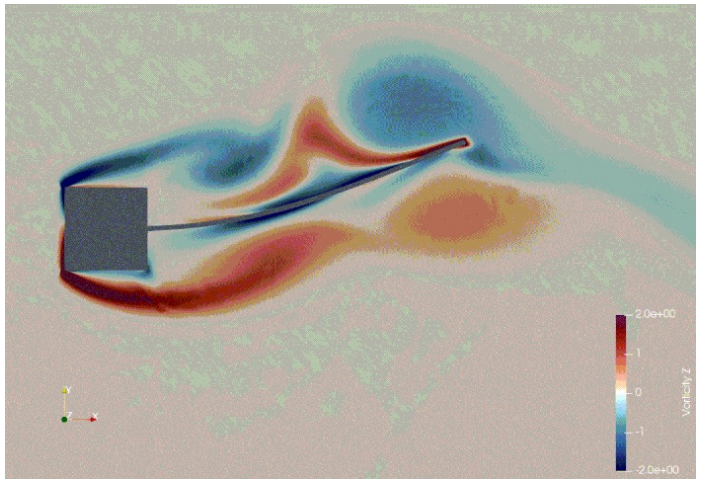
(images provided by their respective teams and used with permission)

CHAMPS SUMMARY

What is it?

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

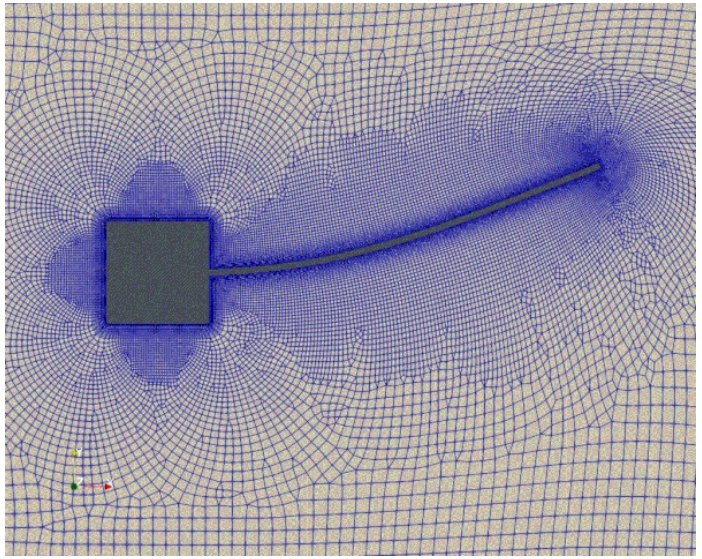




Who wrote it?

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

POLYTECHNIQUE MONTRÉAL

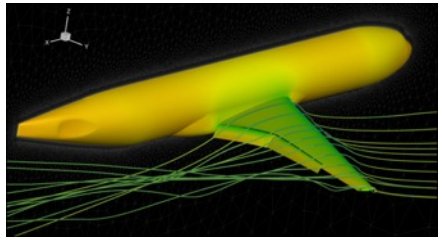


Why Chapel?

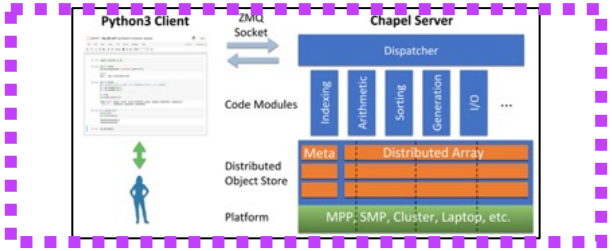
- students found it far more productive to use
- enabled them to compete with more established CFD centers

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

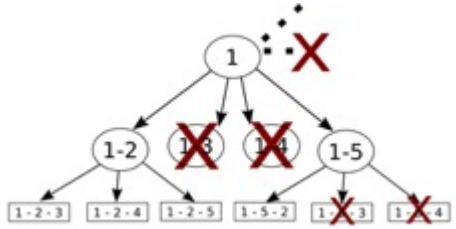
APPLICATIONS OF CHAPEL



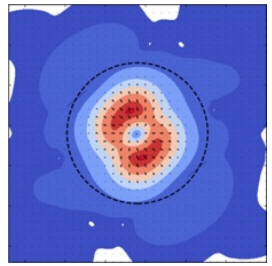
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



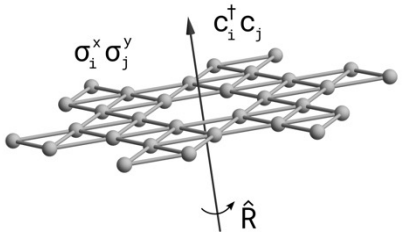
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



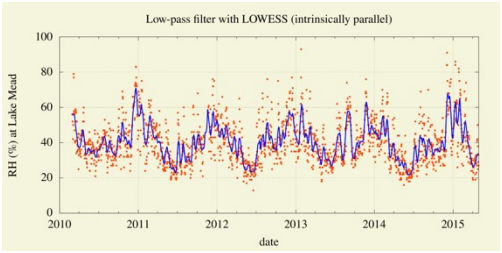
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



ChplUltra: Simulating Ultralight Dark Matter
Nikhil Padmanabhan, J. Luna Zagorac, et al.
Yale University et al.

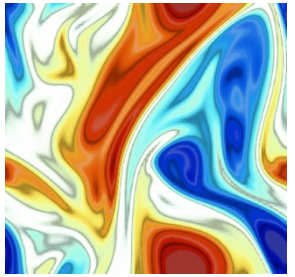


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

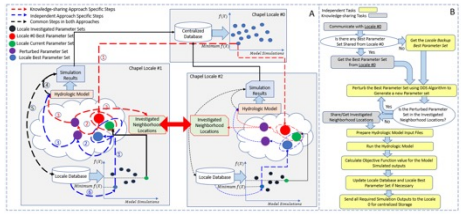


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

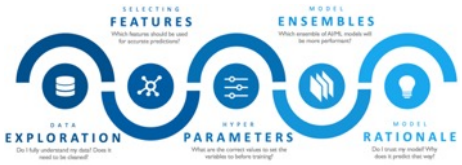
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



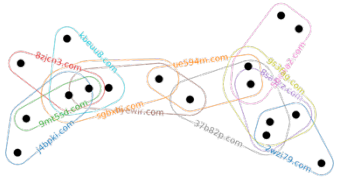
ChapQG: Layered Quasigeostrophic CFD
Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



CrayAI HyperParameter Optimization (HPO)
Ben Albrecht et al.
Cray Inc. / HPE



CHGL: Chapel Hypergraph Library
Louis Jenkins, Cliff Joslyn, Jesun Firoz, et al.
PNNL

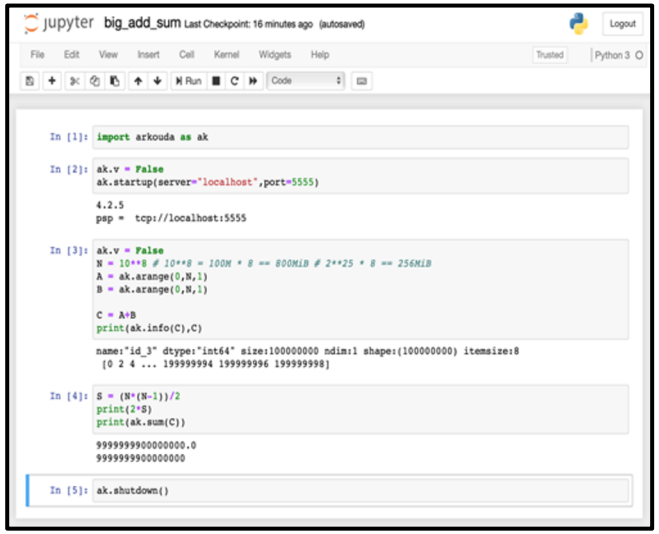
?

Your Application Here?

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

ARKOUDA: A PYTHON LIBRARY AND FRAMEWORK FOR INTERACTIVE HPC

Arkouda Client
(written in Python)



Arkouda Server
(written in Chapel)

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

ARKOUDA SUMMARY

What is it?

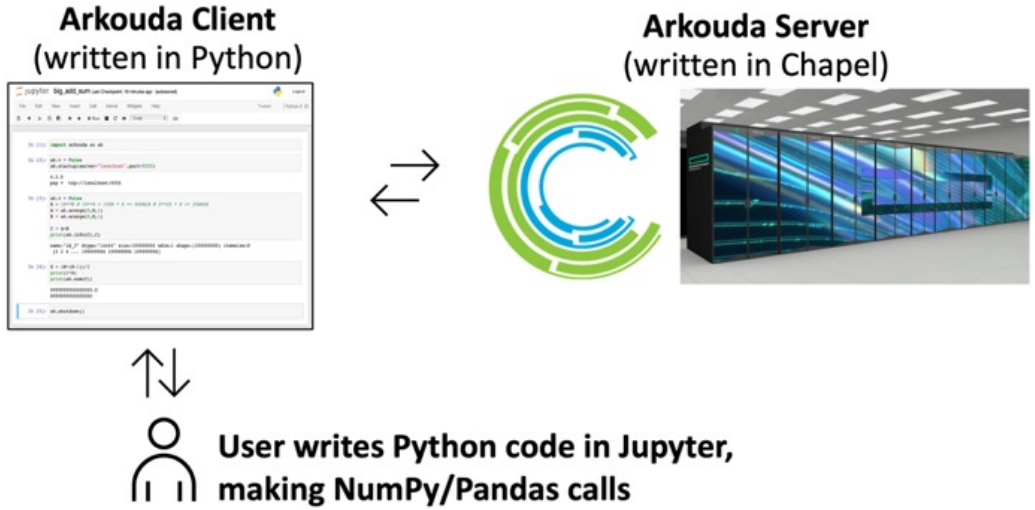
- A client-server framework for interactive supercomputing in Python
- ~30k lines of Chapel and ~25k lines of Python, written since 2019
- Open-source: https://github.com/Bears-R-Us/arkouda

Who wrote it?

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

Why Chapel?

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



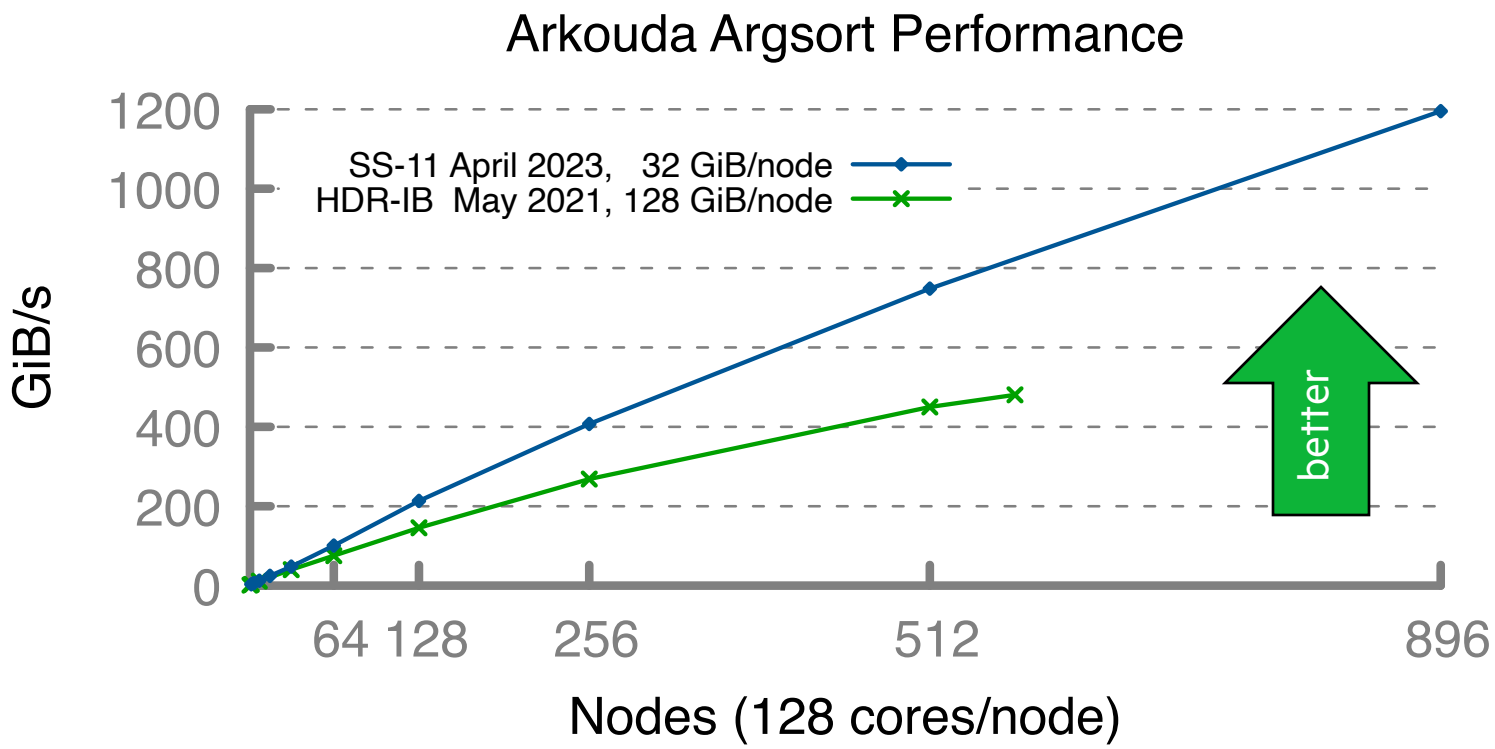
SCALABILITY OF ARKOUDA'S ARGSORT ROUTINE

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

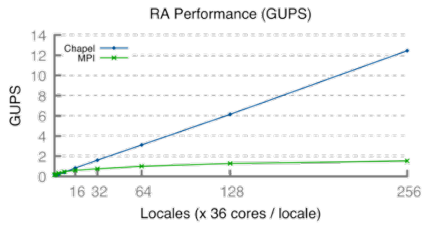
HIGHLIGHT #3: THE CHAPEL TEAM AT HPE HAS GROWN

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);
…
```



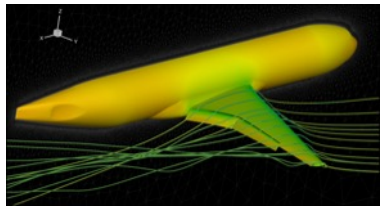
Vendor-neutral GPU support is maturing rapidly

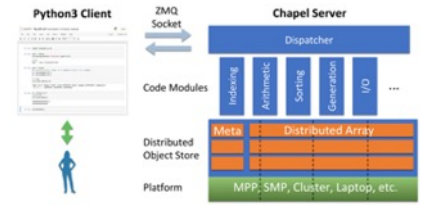
- fleshes out an overdue aspect of “any parallel hardware”

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

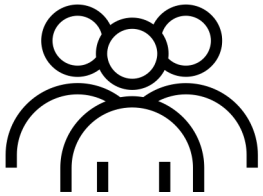
Chapel is being used for productive parallel computing at scale

- 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

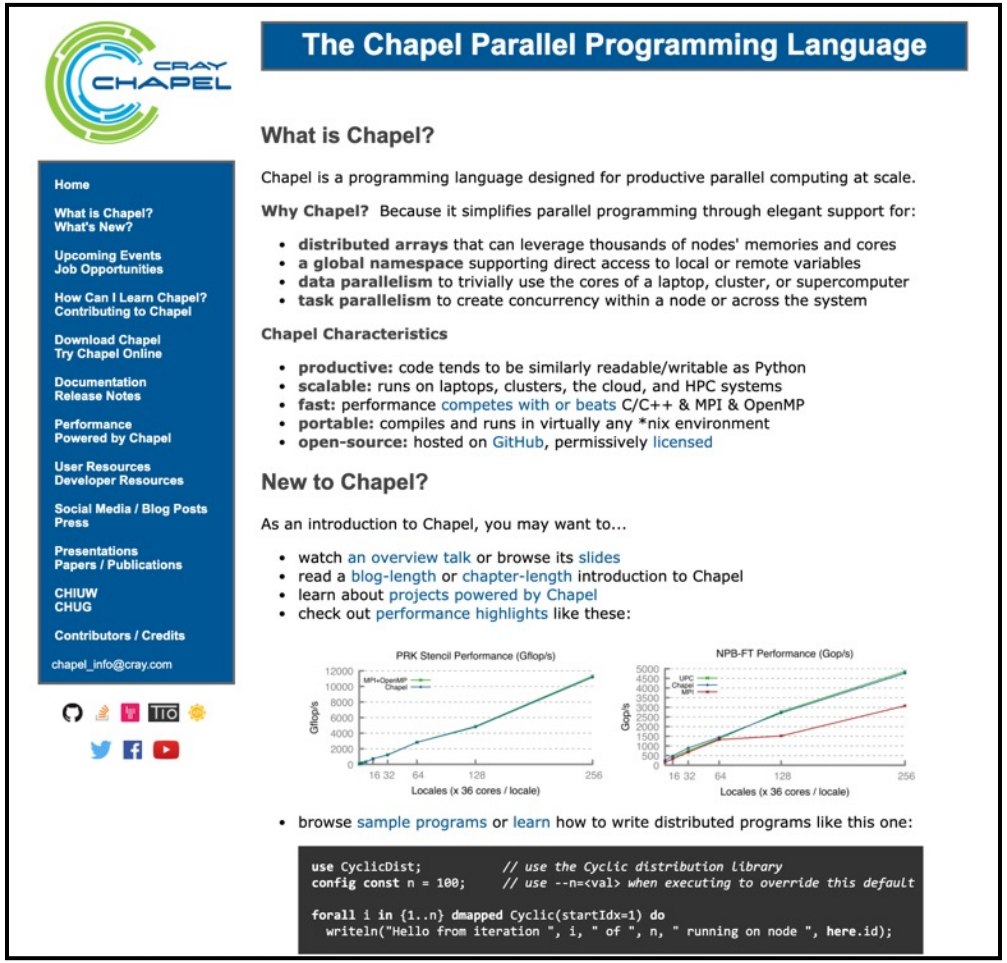
- (points to all other resources)

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



THANK YOU

https://chapel-lang.org
@ChapelLanguage