



Hewlett Packard
Enterprise

RAPID PROTOTYPING BY EXAMPLE: ARKOUDA ARGSORT IN CHAPEL

Brad Chamberlain

Rapid Prototyping for Exascale, ECP BoF Days

May 12, 2022

WHAT IS CHAPEL?

Chapel: A modern parallel programming language

- portable & scalable
- open-source & collaborative



Goals:

- Support general parallel programming
- Make parallel programming at scale far more productive
 - Python-like support for rapid prototyping
 - yet with the performance, scaling, GPU support of Fortran/C/C++, MPI, OpenMP, CUDA, ...



WHAT DO CHAPEL PROGRAMS LOOK LIKE?

helloTaskPar.chpl: print a message from each core in the system

```
coforall loc in Locales {
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n on %s\n",
            tid, numTasks, here.name);
  }
}
```

```
> chpl helloTaskPar.chpl
> ./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
...
```

fillArray.chpl: declare and initialize a distributed array

```
use CyclicDist;

config const n = 1000;

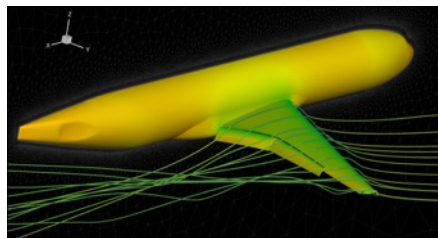
const D = {1..n, 1..n}
         dmapped Cyclic(startIdx = (1,1));
var A: [D] real;

forall (i,j) in D do
  A[i,j] = i*10 + j + (here.id+1)/10.0;

writeln(A);
```

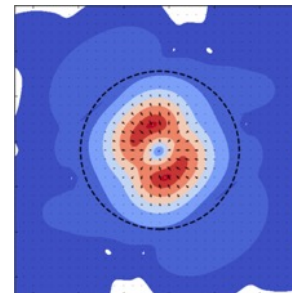
```
> chpl fillArray.chpl
> ./fillArray --n=5 --numLocales=4
11.1 12.2 13.1 14.2 15.1
21.3 22.4 23.3 24.4 25.3
31.1 32.2 33.1 34.2 35.1
41.3 42.4 43.3 44.4 45.3
51.1 52.2 53.1 54.2 55.1
```

FLAGSHIP CHAPEL APPLICATIONS



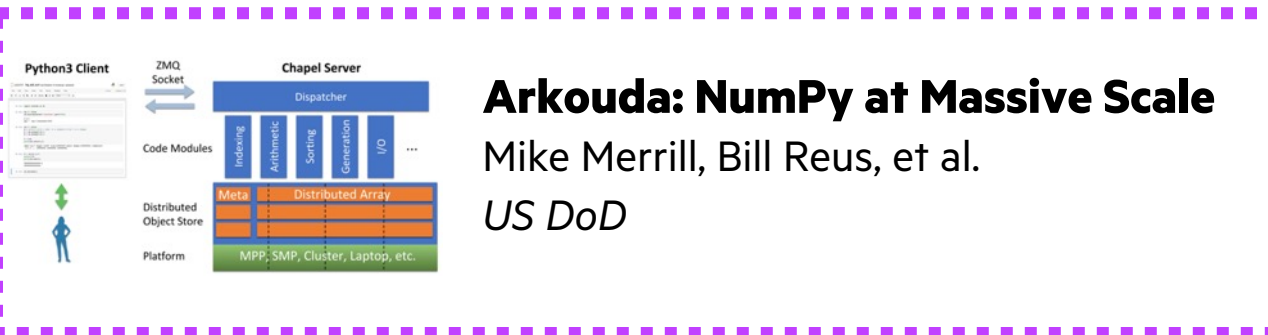
CHAMPS: 3D Unstructured CFD

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



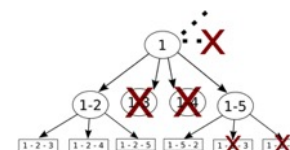
ChplUltra: Simulating Ultralight Dark Matter

Nikhil Padmanabhan, J. Luna Zagorac, et al.
Yale University / University of Auckland



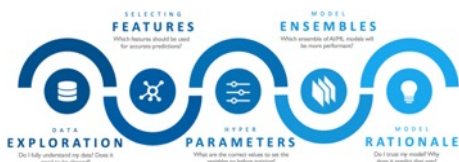
Arkouda: NumPy at Massive Scale

Mike Merrill, Bill Reus, et al.
US DoD



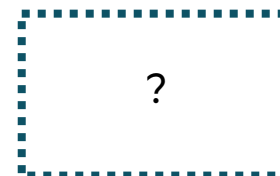
ChOp: Chapel-based Optimization

Tiago Carneiro, Nouredine Melab, et al.
INRIA Lille, France



Cray AI: Distributed Machine Learning

Hewlett Packard Enterprise



Your application here?

ARKOUDA ARGSORT: PROTOTYPE TO PRODUCTION

Arkouda:

- provides scalable NumPy / Pandas routines for use in data science
- supports massive data sets (multi-TB arrays)
- runs at interactive rates (seconds to a few minutes per operation)
- key, expensive operations: *groupBy* and *argSort*

Arkouda Argsort Milestones:

May 2019: first-draft counting sorts written and tuned

Sept 2019: looked at [NESL LSD radix sorts](#) and ~4 hours later had a ~100-line scalable sort
– achieved 80 GiB/s on 512 nodes of Cray XC

Nov 2019: changed ~12 lines of sort code to aggregate small messages
– 40% improvement on Cray XC, ~1000x improvement on InfiniBand

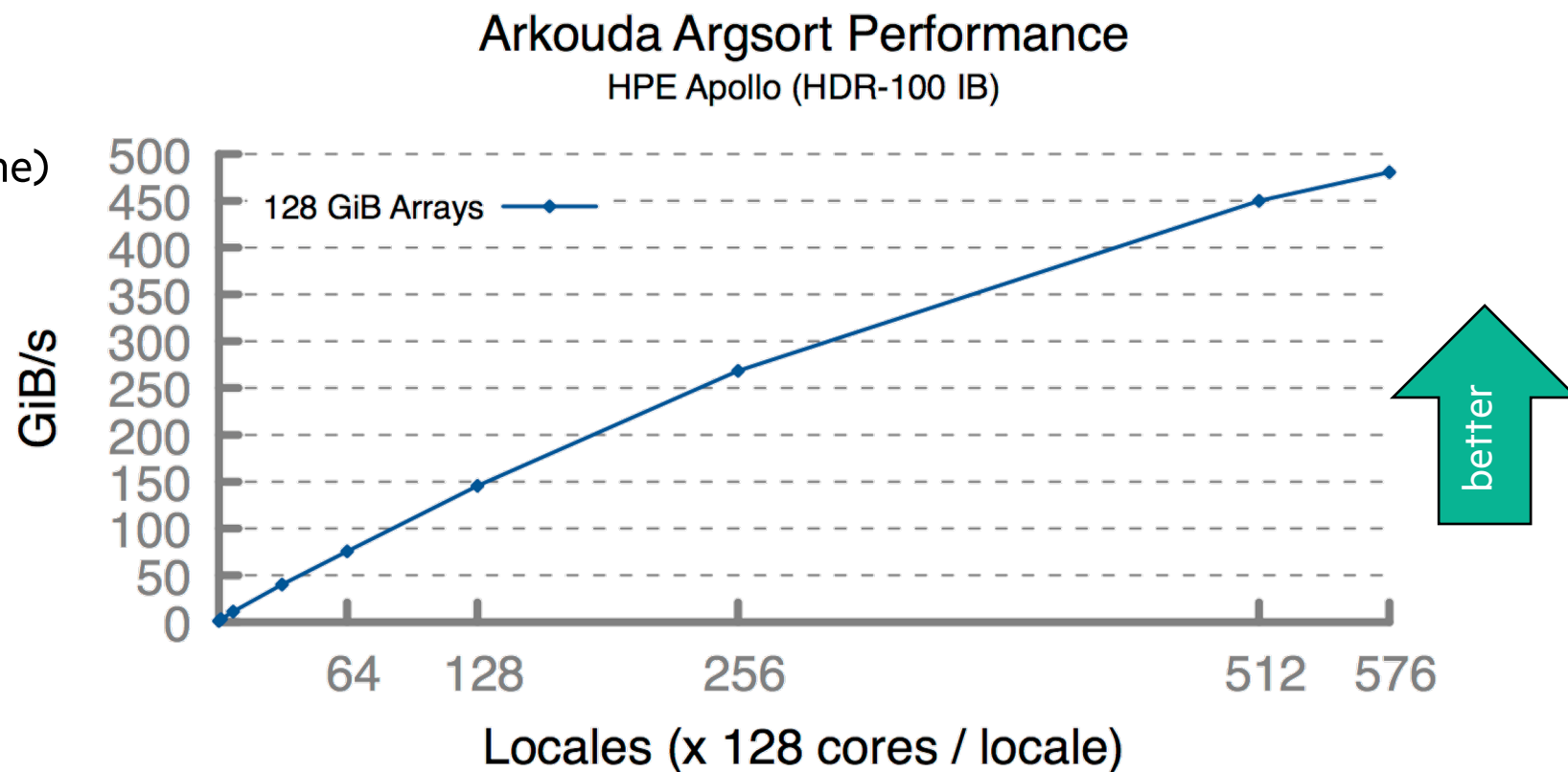
June 2021: did the following hero run



ARKOUDA ARGSORT AT MASSIVE SCALE

- Ran on a large Apollo system, summer 2022

- 73,728 cores of AMD Rome
- 72 TiB of 8-byte values
- 480 GiB/s (2.5 minutes elapsed time)
- ~100 lines of Chapel code



Close to world-record performance—quite likely a record for performance/SLOC

CHAPEL RESOURCES

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


- (points to all other resources)

Social Media:

- Twitter: [@ChapelLanguage](https://twitter.com/ChapelLanguage)
- Facebook: [@ChapelLanguage](https://facebook.com/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

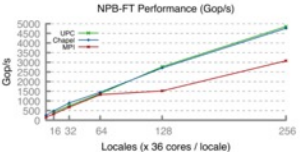
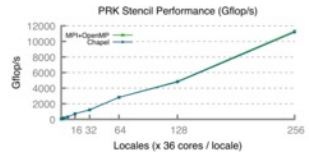
Chapel Characteristics

- **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:



PRK Stencil Performance (Gflop/s)

NPB-FT Performance (Gop/s)

- 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



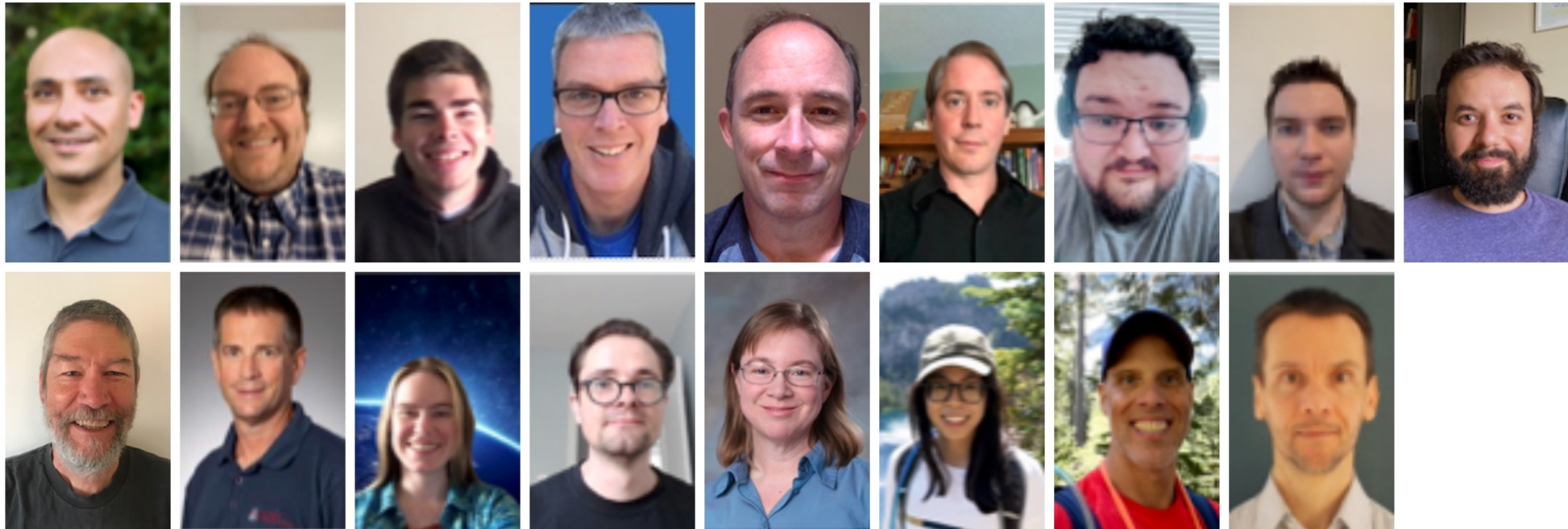
BACKUP SLIDES

THE CHAPEL TEAM

Chapel is a team effort—currently made up of 14 full-time employees, 2 part-time, and our director

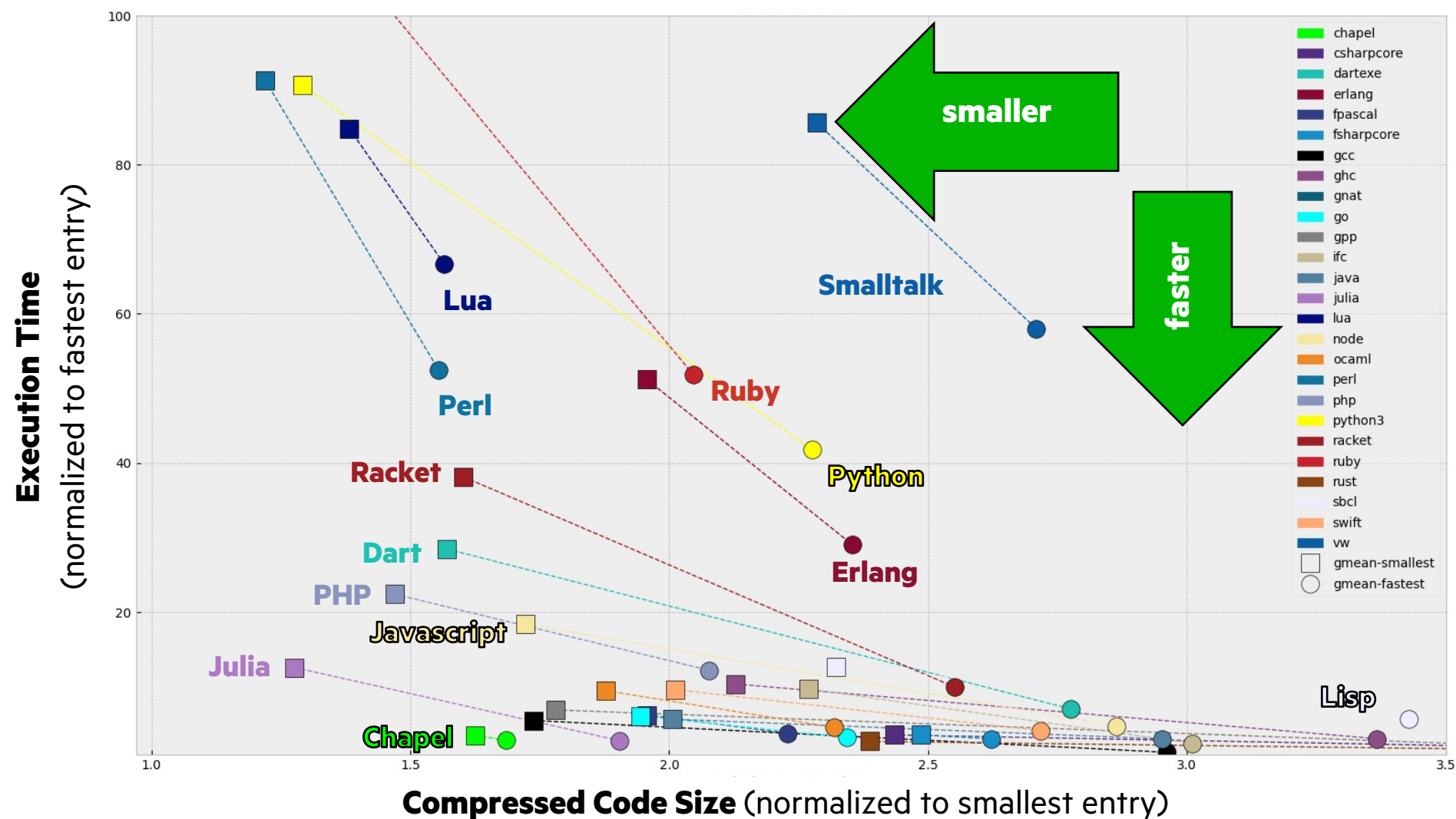
- we also have 3 more full-time engineers joining in the next few months, and 2 open positions

Chapel Development Team at HPE

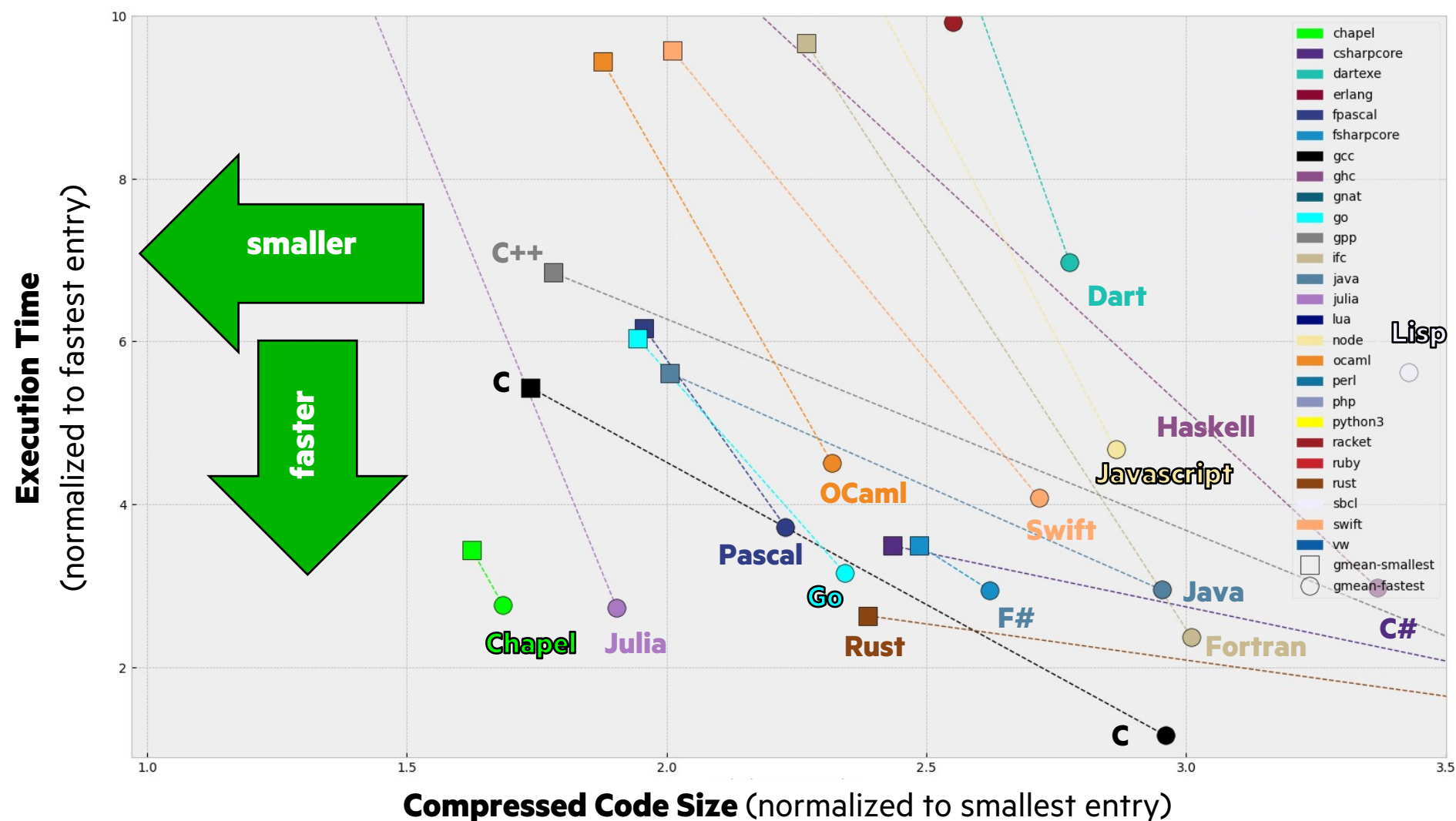


see: <https://chapel-lang.org/contributors.html>
and <https://chapel-lang.org/jobs.html>

CLBG: ALL-LANGUAGE SUMMARY (MAY 10, 2022)



CLBG: ALL-LANGUAGE SUMMARY (MAY 10, 2022, ZOOMED-IN)



FOR HPC BENCHMARKS, CHAPEL TENDS TO BE CONCISE, CLEAR, AND COMPETITIVE

STREAM TRIAD: C + MPI + OPENMP

```
#include <hpc.h>
#ifdef OPENMP
#include <omp.h>
#endif

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

int HPC_Stream(HPC_Parameters *params) {
    int myRank, commSize;
    int rv, errCount;
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Comm_size(comm, &commSize);
    MPI_Comm_rank(comm, &myRank);

    rv = HPC_Stream(params, 0 == myRank);
    MPI_Reduce(&rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm);

    return errCount;
}

int HPC_Stream(HPC_Parameters *params, int doIO) {
    register int j;
    double scalar;

    VectorSize = HPC_LocalVectorSize(params, 3, sizeof(double), 0);

    a = HPC_XMALLOC(double, VectorSize);
    b = HPC_XMALLOC(double, VectorSize);
    c = HPC_XMALLOC(double, VectorSize);

    if (!a || !b || !c) {
        if (c) HPC_free(c);
        if (b) HPC_free(b);
        if (a) HPC_free(a);
        if (doIO) {
            fprintf(stderr, "Failed to allocate memory\n");
            fclose(stderr);
        }
        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];

    HPC_free(c);
    HPC_free(b);
    HPC_free(a);

    return 0;
}
```

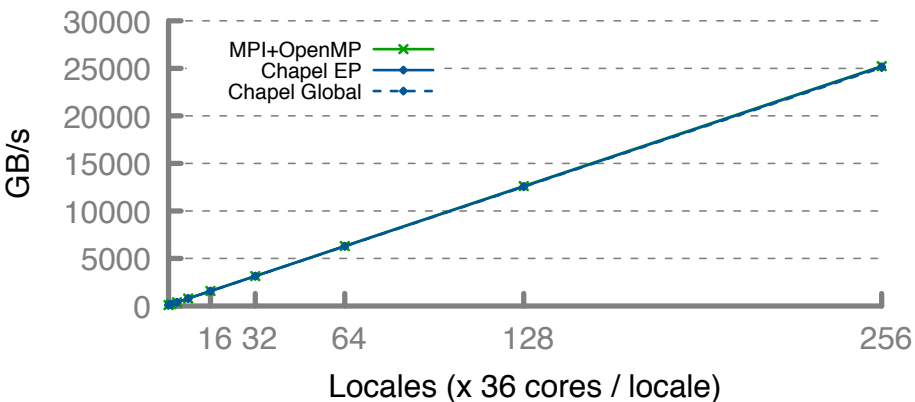
use **BlockDist**;

```
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;
```

STREAM Performance (GB/s)



HPCC RA: MPI KERNEL

```
/* Perform update to maintain the scalar equivalent to
 * the (a+b)*c*(a+b) operation.
 * The scalar equivalent is:
 * a = (a+b)*c*(a+b)
 * b = (a+b)*c*(a+b)
 * c = (a+b)*c*(a+b)
 */

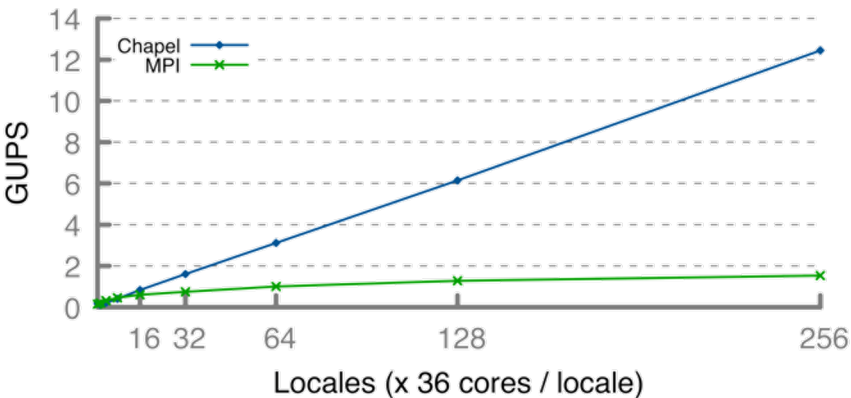
MPI_Irecv(localBuffer, localBufferSize, MPI_COMM_WORLD, &myRank);
while (i < GlobalSize) {
    /* Receive message */
    do {
        MPI_Test(&status, &have_data, &status);
        if (status.MPI_TAG == UPDATE_TAG) {
            MPI_Get_count(&status, MPI_COMM_WORLD, &count);
            bufferBase = 0;
            for (j=0; j<count; j++) {
                long = LocalBuffer[bufferBase+j];
                localOffset = (long * (params.TabSize - 1)) +
                    (params.GlobalOffset);
                HPC_Tab(localOffset) = long;
            }
        } else if (status.MPI_TAG == FINISHED_TAG) {
            MPI_Send(&count, 1, MPI_COMM_WORLD, &myRank);
            MPI_Irecv(localBuffer, localBufferSize, MPI_COMM_WORLD, &myRank);
        }
    } while (have_data && !have_finished);

    if (pendingUpdates < maxPendingUpdates) {
        long = (GlobalOffset + params.TabSize - 1) *
            (GlobalOffset + params.TabSize - 1) *
            (GlobalOffset + params.TabSize - 1);
        HPC_Tab(localOffset) = long;
    }
}

MPI_Finalize();
```

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

RA Performance (GUPS)



BALE INDEXGATHER

Exstack version

```
i=0;
while( exstack_proceed(ex, (i==l_num_req)) ) {
  i0 = i;
  while(i < l_num_req) {
    l_idx = pckindx[i] >> 16;
    pe = pckindx[i] & 0xffff;
    if(!exstack_push(ex, &l_idx, pe))
      break;
    i++;
  }

  exstack_exchange(ex);

  while(exstack_pop(ex, &idx, &fromth)) {
    idx = ltable[idx];
    exstack_push(ex, &idx, fromth);
  }
  lgp_barrier();
  exstack_exchange(ex);

  for(j=i0; j<i; j++) {
    fromth = pckindx[j] & 0xffff;
    exstack_pop_thread(ex, &idx, (uint64_t)fromth);
    tgt[j] = idx;
  }
  lgp_barrier();
}
```

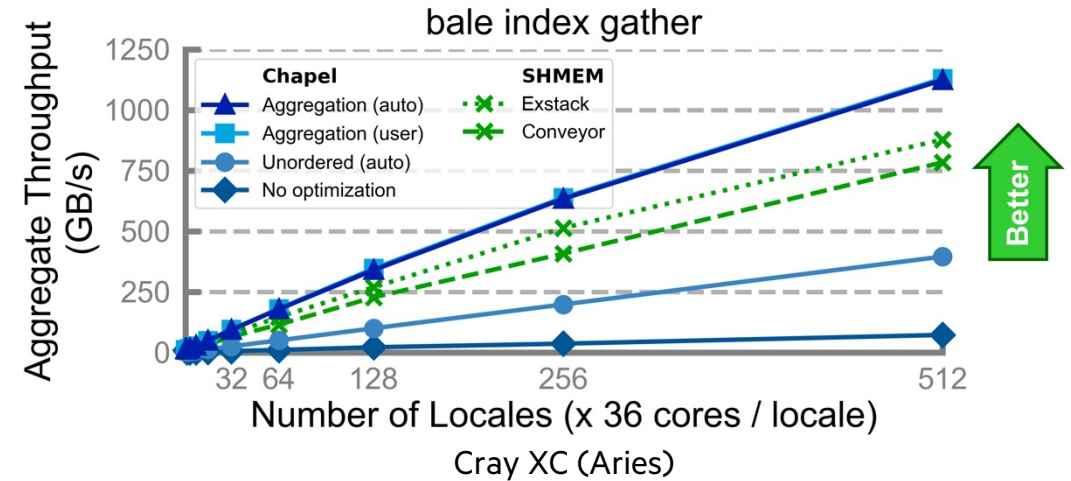
Conveyors version

```
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
       more | convey_advance(replies, !more)) {

  for (; i < l_num_req; i++) {
    pkg.idx = i;
    pkg.val = pckindx[i] >> 16;
    pe = pckindx[i] & 0xffff;
    if (!convey_push(requests, &pkg, pe))
      break;
  }

  while (convey_pull(requests, ptr, &from) == convey_OK) {
    pkg.idx = ptr->idx;
    pkg.val = ltable[ptr->val];
    if (!convey_push(replies, &pkg, from)) {
      convey_unpull(requests);
      break;
    }
  }

  while (convey_pull(replies, ptr, NULL) == convey_OK)
    tgt[ptr->idx] = ptr->val;
}
```



Manually Tuned Chapel version (using aggregator abstraction)

```
forall (d, i) in zip(Dst, Inds) with (var agg = new SrcAggregator(int)) do
  agg.copy(d, Src[i]);
```

Elegant Chapel version (compiler-optimized w/ '--auto-aggregation')

```
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```


PARALLEL COMPUTING IN PYTHON?

Motivation: Say you've got...

...HPC-scale data science problems to solve

...a bunch of Python programmers

...access to HPC systems

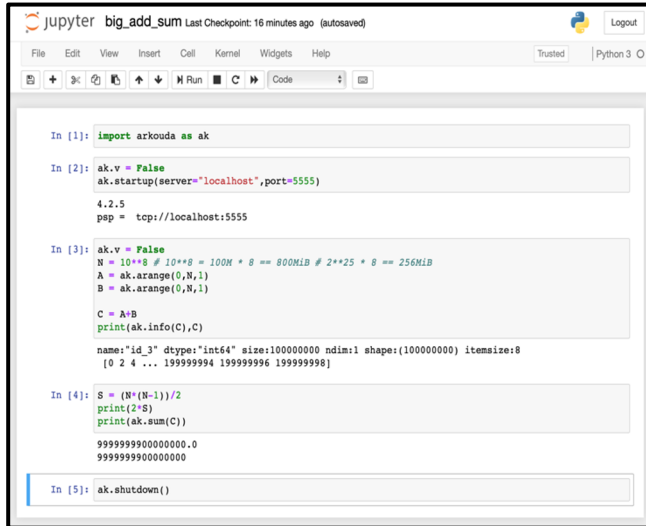


How will you leverage your Python programmers to get your work done?



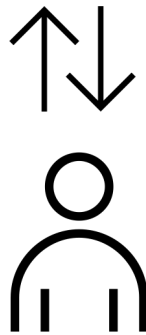
ARKOUDA'S HIGH-LEVEL APPROACH

Arkouda Client (written in Python)



```
big_add_sum Last Checkpoint: 16 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
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 = False
         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 NumPy/Pandas calls

ARKOUDA SUMMARY

What is it?

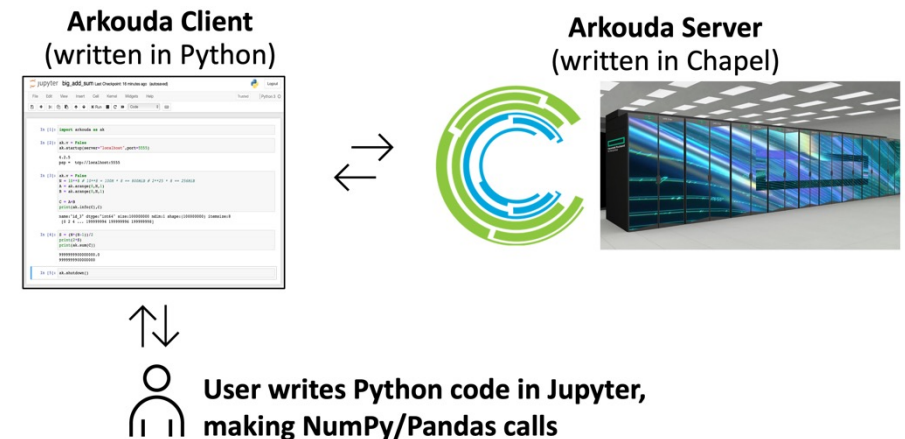
- A Python library supporting a key subset of NumPy and Pandas for Data Science
 - Uses a Python-client/Chapel-server model to get scalability and performance
 - Computes massive-scale results (multi-TB-scale arrays) within the human thought loop (seconds to a few minutes)
- ~20k lines of Chapel, largely written in 2019, continually improved since then

Who wrote it?

- Mike Merrill, Bill Reus, *et al.*, US DoD
- Open-source: <https://github.com/Bears-R-Us/arkouda>

Why Chapel?

- high-level language with performance and scalability
- close to Pythonic
 - enabled writing Arkouda rapidly
 - doesn't repel Python users who look under the hood
- ports from laptop to supercomputer



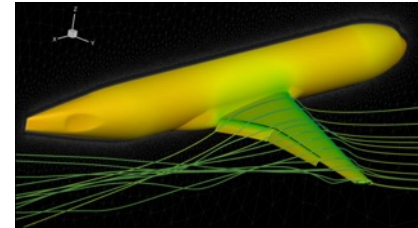
ARKOUDA PERFORMANCE COMPARED TO NUMPY

benchmark	NumPy 0.75 GB	Arkouda (serial) 0.75 GB 1 core, 1 node	Arkouda (parallel) 0.75 GB 36 cores x 1 node	Arkouda (distributed) 384 GB 36 cores x 512 nodes
argsort	0.03 GiB/s --	0.05 GiB/s 1.66x	0.50 GiB/s 16.7x	55.12 GiB/s 1837.3x
coargsort	0.03 GiB/s --	0.07 GiB/s 2.3x	0.50 GiB/s 16.7x	29.54 GiB/s 984.7x
gather	1.15 GiB/s --	0.45 GiB/s 0.4x	13.45 GiB/s 11.7x	539.52 GiB/s 469.1x
reduce	9.90 GiB/s --	11.66 GiB/s 1.2x	118.57 GiB/s 12.0x	43683.00 GiB/s 4412.4x
scan	2.78 GiB/s --	2.12 GiB/s 0.8x	8.90 GiB/s 3.2x	741.14 GiB/s 266.6x
scatter	1.17 GiB/s --	1.12 GiB/s 1.0x	13.77 GiB/s 11.8x	914.67 GiB/s 781.8x
stream	3.94 GiB/s --	2.92 GiB/s 0.7x	24.58 GiB/s 6.2x	6266.22 GiB/s 1590.4x

CHAMPS SUMMARY

What is it?

- 3D unstructured CFD framework for airplane simulation
- ~100k lines of Chapel written from scratch in ~3 years



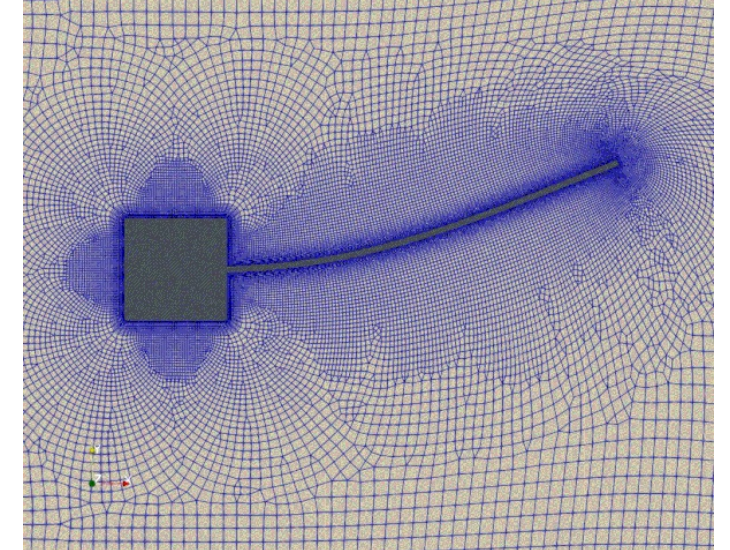
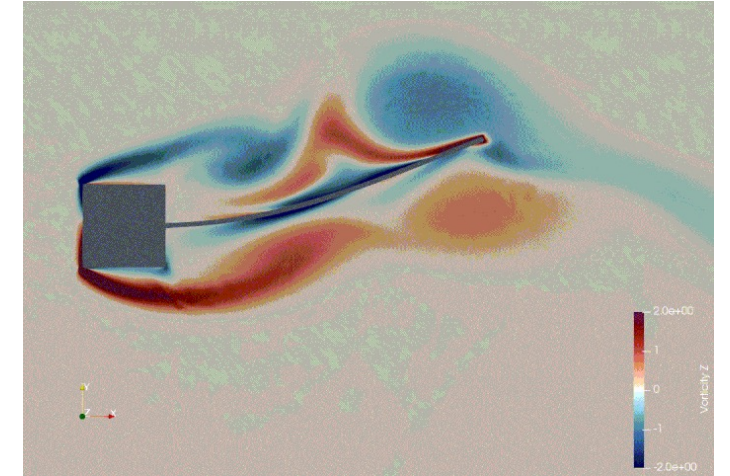
Who wrote it?

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



Why Chapel?

- performance and scalability competitive with MPI + C++
- students found it far more productive to use



CHAMPS: EXCERPT FROM ÉRIC'S CHIUW 2021 KEYNOTE

HPC Lessons From 30 Years of Practice in CFD Towards Aircraft Design and Analysis (June 4, 2021)

*“To show you what Chapel did in our lab... [our previous framework] ended up 120k lines. And my students said, ‘We can’t handle it anymore. It’s too complex, we lost track of everything.’ And today, they went **from 120k lines to 48k lines, so 3x less.***

*But the code is not 2D, it’s 3D. And it’s not structured, it’s unstructured, which is way more complex. And it’s multi-physics... **So, I’ve got industrial-type code in 48k lines.**”*

*“[Chapel] promotes the programming efficiency ... **We ask students at the master’s degree to do stuff that would take 2 years and they do it in 3 months.** So, if you want to take a summer internship and you say, ‘program a new turbulence model,’ well they manage. And before, it was impossible to do.”*

*“So, for me, this is like the proof of the benefit of Chapel, **plus the smiles I have on my students everyday in the lab because they love Chapel as well.** So that’s the key, that’s the takeaway.”*

- Talk available online: https://youtu.be/wD-a_KyB8aI?t=1904 (hyperlink jumps to the section quoted here)

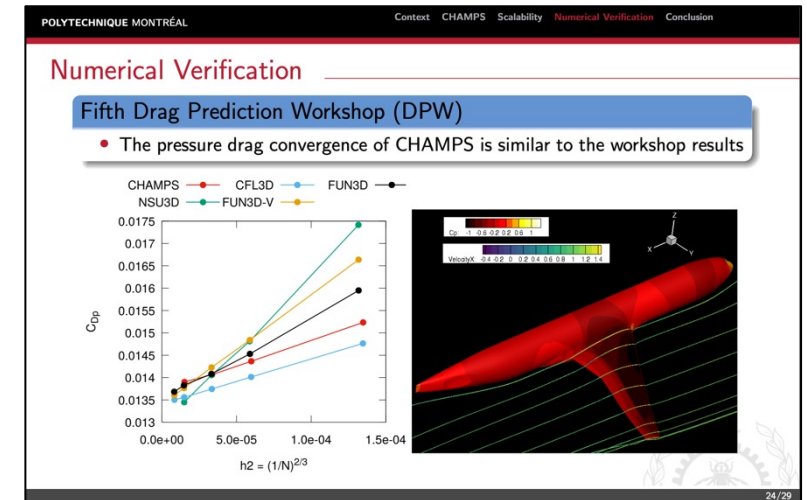
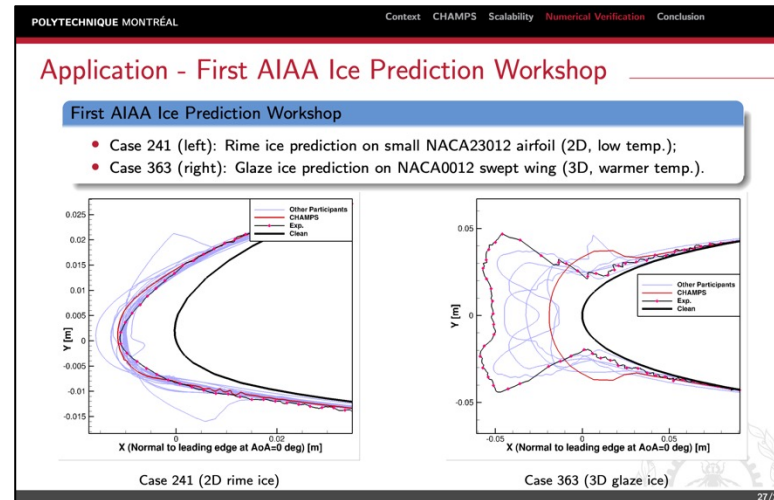
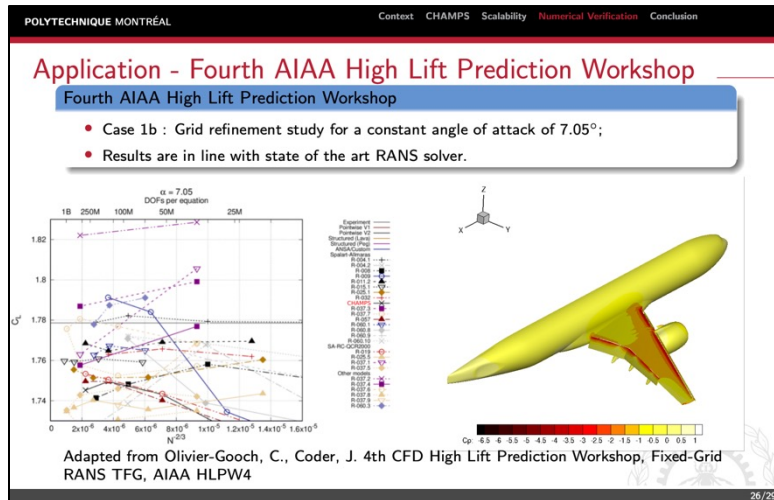


**POLYTECHNIQUE
MONTRÉAL**



CHAMPS HIGHLIGHTS IN 2021

- Presented at CASI/IASC Aero 21 Conference
- Presented to CFD Society of Canada (CFDSC)
- Participated in 4th AIAA High-lift Prediction Workshops, 1st AIAA Ice Prediction Workshop
- Reproduced results from 5th AIAA Drag Prediction Workshop



- Generating results comparable to high-profile sites: Boeing, Lockheed Martin, NASA, JAXA, Georgia Tech, ...

Looking ahead:

- giving 6–7 presentations at AIAA Aviation Forum and Exposition, June 2022
- participating in 7th AIAA Drag Prediction Workshop