



Hewlett Packard
Enterprise

CHAPEL: FIVE HIGHLIGHTS SINCE CLSAC 2019

Brad Chamberlain

October 26, 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



CHAPEL TENDS TO BE COMPACT, CLEAN, AND FAST (BALE INDEX-GATHER)

Exstack version

```

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

```

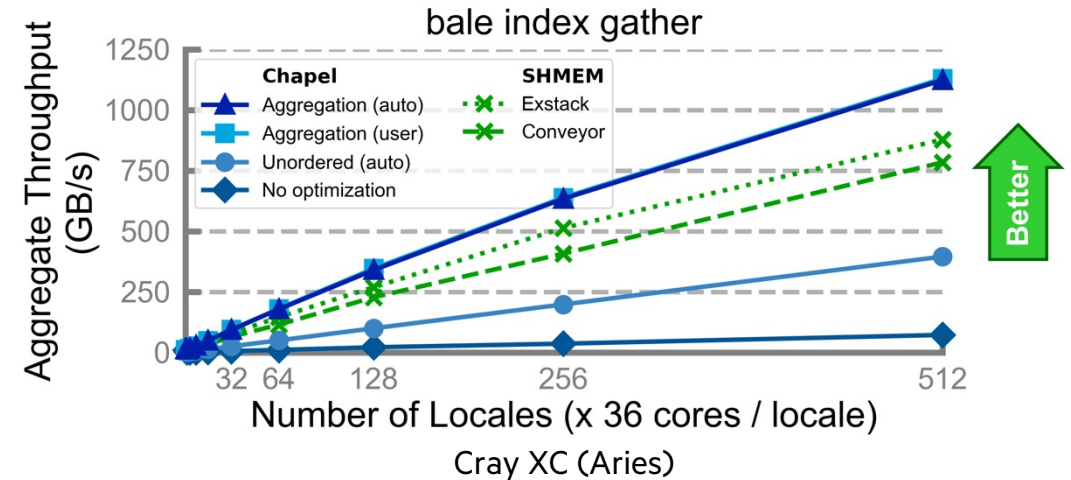
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
       more | convey_advance(replies, !more)) {

  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 explicit aggregator type)

```

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];

```

Highlight 1: Aggregators (explicit and compiler-added)

CHAPEL SUPPORT FOR GPUS

Typical 2019-era Chapel Talk:

- **Me:** “Chapel’s 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/...”



STREAM TRIAD EP: 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 EP: 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 EP: 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 EP: 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;  
      }  
    }  
  }  
}
```

Highlight 2: Chapel now supports GPUs!
(via a work-in-progress prototype)

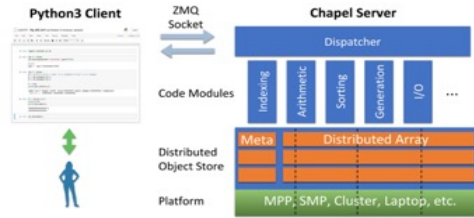
'cobegin { ... }' creates a task
per child statement

one task runs our GPU triad

the other runs the CPU triad

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

FLAGSHIP CHAPEL APPLICATIONS



Arkouda

What?

Interactive Data Analytics at Supercomputing Scale

Who?

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

How Much?

~25k lines of Chapel written since January 2019

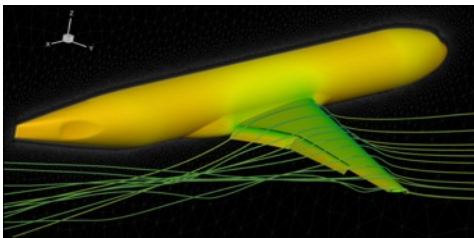
Why Chapel?

Scalability, supported rapid development, supports Pythonic code

Highlight 3: Both apps went into production & matured significantly

Arkouda Highlights Since CLSAC 2019

- Extensible, Modular Architecture
- Many, many New Features
- Performance and Scalability Improvements...



CHAMPS

What?

3D Unstructured CFD (Computational Fluid Dynamics)

Who?

Éric Laurendeau, et al., *Polytechnique Montreal*

How Much?

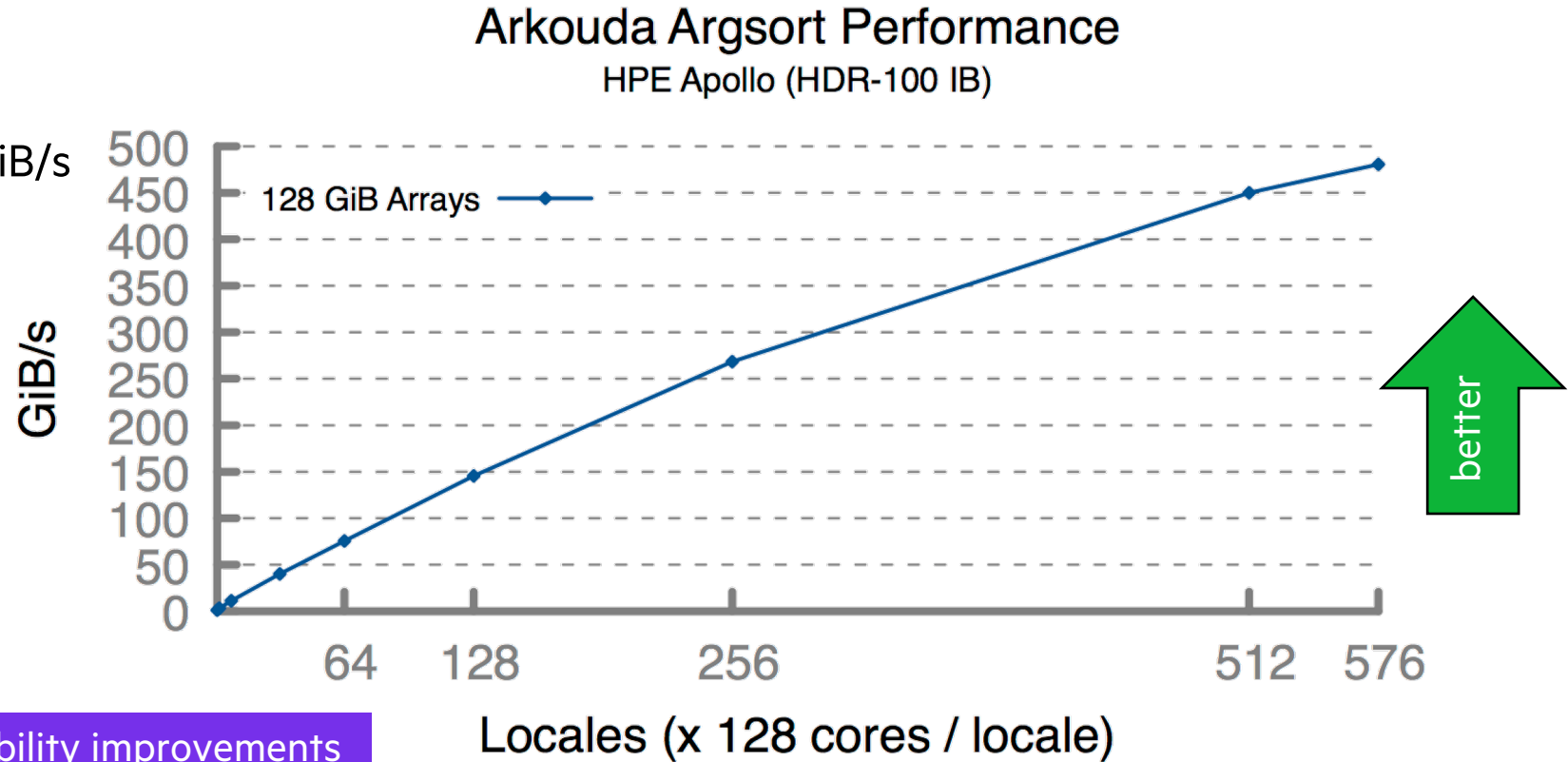
~100k lines of Chapel written since Spring 2019

Why Chapel?

Reduces time-to-science for junior and senior students while scalably generating world-class results

ARKOUDA ARGSORT AT MASSIVE SCALE

- Ran on a large HPE Apollo system, summer 2021
 - 73,728 cores of AMD Rome
 - 72 TiB of 8-byte values
 - 2.5 minutes elapsed time \Rightarrow 480 GiB/s
 - ~100 lines of Chapel code



Highlight 4: Major performance and scalability improvements

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

THE CHAPEL TEAM AT HPE



Our team now consists of:

- 19 full-time employees
- 1 visiting scholar (NCAR)
- our director

We also have:

- a new hire starting early 2023
- an open summer internship

Highlight 5: Team has grown from ~12 in 2019 to ~21 today

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



FUTURE WORK: CHAPEL AT THE EDGE?

- We have admittedly focused almost exclusively on “indoor” systems, from laptops to supercomputers
 - Though at times, this has included things like Raspberry Pi or AWS
- Potential future directions (up for grabs):
 - More **diverse accelerators** than typical GPUs (several talks)
 - Coordinating **loosely-coupled Chapel programs** from edge to cloud (Pete’s talk)
 - using ZeroMQ, Sockets, or something higher-level / more abstract?
 - **Jupyter notebook support** via interactive evaluation of Chapel (Sudip’s talk)
 - Your idea here...



SUMMARY

Chapel is unique among programming languages

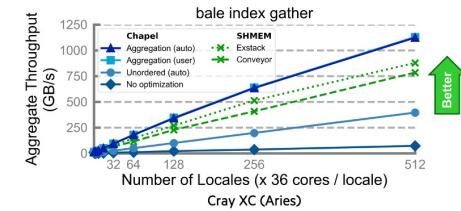
- built-in features for scalable parallel computing
- supports clean, concise code relative to conventional approaches
- ports and scales from laptops to supercomputers

Chapel is being used in production and at scale

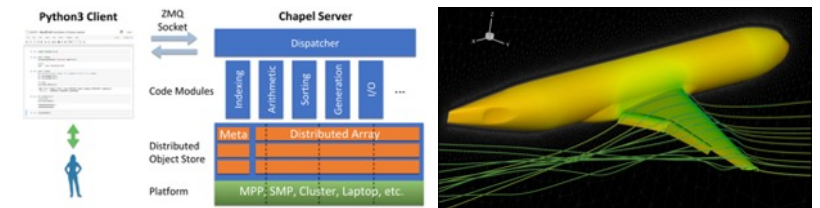
- users are reaping its benefits in applied, cutting-edge applications
- applicable to domains as diverse as data science and physical simulations

Progress over the past three years has been significant

- adding GPU support
- improving performance and scalability
- growing the team



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



CHAPEL RESOURCES

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


- (points to all other resources)

Social Media:

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

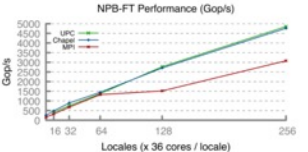
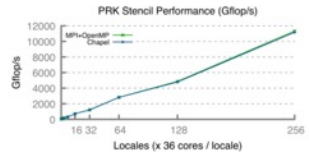
Chapel Characteristics

- **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:



The PRK Stencil Performance graph shows Chapel (green line) significantly outperforming OpenMP (red line) and MPI (blue line) as the number of locales increases from 16 to 256. The NPB-FT Performance graph shows Chapel (green line) also outperforming MPI (blue line) and OpenMP (red line) in terms of 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);
```

SUMMARY

Chapel is unique among programming languages

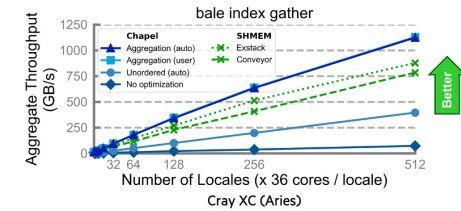
- built-in features for scalable parallel computing
- supports clean, concise code relative to conventional approaches
- ports and scales from laptops to supercomputers

Chapel is being used in production and at scale

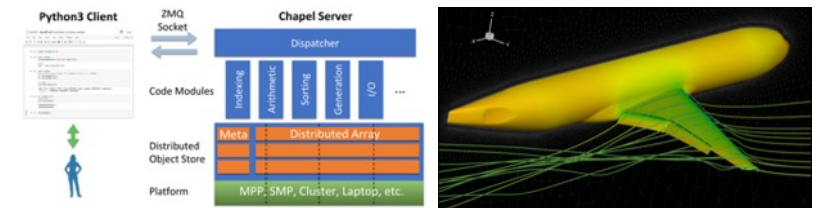
- users are reaping its benefits in applied, cutting-edge applications
- applicable to domains as diverse as data science and physical simulations

Progress over the past three years has been significant

- adding GPU support
- improving performance and scalability
- growing the team



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



THANK YOU

<https://chapel-lang.org>
@ChapelLanguage

