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

Conveyors version

Manually Tuned Chapel version (using explicit aggregator type)

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

Highlight 1: Aggregators (explicit and compiler-added)
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

This is a GPU-only program
Nothing other than coordination code runs on the CPUs

```
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
 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;
    }
}
}
```

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

one task runs our GPU triad

the other runs the CPU triad

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

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

### 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 Highlights Since CLSAC 2019**
- Extensible, Modular Architecture
- Many, many New Features
- Performance and Scalability Improvements...
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
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](https://chapel-lang.org/contributors.html)
and [https://chapel-lang.org/jobs.html](https://chapel-lang.org/jobs.html)
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
• 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
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

SUMMARY

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

https://chapel-lang.org
@ChapelLanguage