APPLICATION EXAMPLES OF LEVERAGING TASK PARALLELISM WITH CHAPEL

Michelle Mills Strout with slides from Brad Chamberlain, Ben Albrecht, Elliot Ronaghan, Brandon Neth, and Scott Bachman

WAMTA: Workshop on Asynchronous Many-Task Systems and Applications
February 15, 2023
CURRENT CHAPEL DEVELOPMENT TEAM AT HPE

• **Lead:**
  • Michelle Strout

• **Tech Lead:**
  • Brad Chamberlain

• **Manager:**
  • Tim Zinsky

• **Subteam leads:**
  • Michael Ferguson
  • Elliot Ronaghan
  • Engin Kayraklioglu
  • Lydia Duncan

• **BTR/DevOps:**
  • Bhavani Jayakumaran

• **Visiting Scholar from NCAR:**
  • Scott Bachman

**Developers:**
Ahmad Rezaei
Andy Stone
Anna Rift
Ben Harshbarger
Ben McDonald
Daniel Fedorin
David Iten
David Longnecker
Jade Abraham
Jeremiah Corrado
John Hartman
Vass Litvinov
Chapel is a general-purpose programming language that provides ease of parallel programming, high performance, and portability, enabling development on laptops and execution on supercomputers.

Some history

2002: Design and development started with DARPA HPCS program,
2002-now: core development team continuously funded and grown from 5 to 21 FTEs,
2019: Arkouda data analytics package written in Chapel by others to provide interactive supercomputing (https://github.com/Bears-R-Us/arkouda), and
2019-now: Arkouda being used in production, and CHAMPs, ChplUltra, and ChOp being actively developed and used to do research.
Rewrite existing codes into Chapel (~100K lines of Chapel)

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

Writing code in Chapel (~10k lines of including parallel FFT)

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

Chapel server for a Python client (~30K lines of Chapel)

Arkouda: NumPy at Massive Scale
Mike Merrill, Bill Reus, et al.
US DoD

Calling out to Cuda (~1k lines of Chapel)

ChOp: Chapel-based Optimization
Tiago Carneiro, Nouredine Melab, et al.
INRIA Lille, France
MOTIVATION: APPS WRITTEN IN CHAPEL ARE FAST AND SCALABLE

Arkouda data analytics framework

- Sorting is within 2-3x of the world record
- Loading an HDF5 file, gather, scatter, and stream are all significantly faster than with Dask
- Arkouda can do groupby and argsort, which Dask can’t

CHAMPS Aeronautics Code

- Performance and scalability competitive with MPI + C++
- CHAMPS scales up to 256 nodes/locales on Cray XC. That experiment had 640 million cells in a 3D grid

Coral Reef Diversity Image Analysis

- Computation in Matlab took days, Chapel version with some algorithmic improvements takes seconds
TASKS IN CHAPEL

- A **task** is a unit of computation that can and should be executed in parallel with other tasks.
- Tasks can **share data** with other tasks through the lexical scoping of variables in the program's global namespace.
- The **mapping** of tasks to nodes is done either:
  - by the programmer **explicitly** by using the concept of locales or
  - **implicitly** through the 'forall' data parallelism abstraction when iterating over distributed data structures
    - 'forall' is explicit mapped to locales under the hood
    - can be user-defined with an iterator.
- Tasks can execute indefinitely until they yield control explicitly or through synchronization constructs.
TASK PARALLELISM SUPPORTED BY CHAPEL

- **Synchronous parallelism**
  - 'coforall', distributed memory parallelism across processes/locales with 'on' syntax
  - 'coforall', shared-memory parallelism over threads
  - 'cobegin', executes all statements in block in parallel

- **Asynchronous parallelism**
  - 'begin', creates an asynchronous task
  - 'sync' and 'atomic' vars for task coordination
  - spawning subprocesses

- **Higher-level parallelism abstractions**
  - 'forall', data parallelism and iterator abstraction
  - 'foreach', SIMD parallelism
  - 'scan', operations such as cumulative sums
  - 'reduce', operations such as summation
USE OF TASKS IN SOME APPLICATIONS AND BENCHMARKS

• "Hello world", shared and distributed-memory parallel: 'coforall'
• HPO, HyperParameter Optimization for ML training: subprocess spawning
• Arkouda, data analytics package: 'coforall', 'scan', 'forall'
• CHAMPS, 3D computational fluid dynamics for airplanes: 'coforall'
• ChOp, Chapel-based Optimization: 'forall', 'coforall', 'begin', 'atomic'
• ParFlow, C+MPI hydrodynamics code calling out to Chapel: 'forall'
• Coral reef, image analysis for eco diversity: 'coforall', 'forall', 'cobegin'
• Arbitrary task graphs: 'begin', 'atomic'
### USE OF TASKS IN SOME APPLICATIONS AND BENCHMARKS

<table>
<thead>
<tr>
<th>Application</th>
<th>Distributed 'coforall'</th>
<th>Threaded 'coforall'</th>
<th>Asynchronous 'begin'</th>
<th>'cobegin'</th>
<th>sync or atomic vars</th>
<th>subprocesses</th>
<th>forall</th>
<th>scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPO</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Arkouda</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>CHAMPS</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>ChOp</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>ParFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Coral Reef</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Task Graph</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

Poll Everywhere link: [pollev.com/michellestrout402](pollev.com/michellestrout402)

There will be fun questions throughout the talk
Which kind of task parallelism interests you the most?

- ‘coforall’ for distributed-memory parallelism
- ‘coforall’ for shared-memory parallelism
- ‘cobegin’ for executing statements in parallel
- ‘begin’ for asynchronous parallelism
- ‘forall’ abstraction
## USE OF TASKS IN SOME APPLICATIONS AND BENCHMARKS

<table>
<thead>
<tr>
<th>Application</th>
<th>Distributed 'coforall'</th>
<th>Threaded 'coforall'</th>
<th>Asynchronous 'begin'</th>
<th>'cobegin'</th>
<th>sync or atomic vars</th>
<th>subprocesses</th>
<th>forall</th>
<th>scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPO</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Arkouda</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CHAMPS</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChOp</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>ParFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Coral Reef</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Task Graph</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

Poll Everywhere link: [pollev.com/michellestrout402](pollev.com/michellestrout402)
There will be fun questions throughout the talk
**CHAPEL EXECUTION MODEL AND TERMINOLOGY: LOCALES**

- Locales can run tasks and store variables
  - Think “compute node” on a parallel system
  - User specifies number of locales on executable’s command-line

```bash
prompt> ./myChapelProgram --numLocales=4  # or --nl 4
```

**Locales array:**

- User’s code starts running as a single task on locale 0
  - Four nodes/CPUs

```c
locale 0
locale 1
locale 2
locale 3
```
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  writef("Hello from task %n of %n on %s\n", tid, numTasks, here.name);
TASK-PARALLEL “HELLO WORLD”

```chapel
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
    printf(“Hello from task %n of %n on %s\n”,
        tid, numTasks, here.name);
```

- ‘here’ refers to the locale on which we’re currently running
- how many processing units (think “cores”) does my locale have?
- what’s my locale’s name?
**TASK-PARALLEL “HELLO WORLD”**

```chpl
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
    writef("Hello from task %n of %n on %s\n", tid, numTasks, here.name);
```

A `coforall` loop executes each iteration as an independent task.

```chpl
prompt> chpl helloTaskPar.chpl
prompt> ./helloTaskPar
Hello from task 1 of 4 on n1032
Hello from task 4 of 4 on n1032
Hello from task 3 of 4 on n1032
Hello from task 2 of 4 on n1032
```
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  printf("Hello from task %n of %n on %s\n", tid, numTasks, here.name);

So far, this is a shared-memory program
Nothing refers to remote locales, explicitly or implicitly
### TASK-PARALLEL “HELLO WORLD” (DISTRIBUTED VERSION)

```
helloTaskPar.chpl

coforall loc in Locales {
    on loc {
        const numTasks = here.numPUs();
        coforall tid in 1..numTasks do
            writef("Hello from task %n of %n on %s\n", tid, numTasks, here.name);
    }
}
```
**TASK-PARALLEL “HELLO WORLD” (DISTRIBUTED VERSION)**

```chpl
helloTaskPar.chpl

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

- create a task per locale on which the program is running
- have each task run ‘on’ its locale
- then print a message per core, as before

```
prompt> chpl helloTaskPar.chpl
prompt> ./helloTaskPar -nl=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
Hello from task 1 of 4 on n1035
...
```
'coforall' has what semantics?

Starts an asynchronous task for each iteration

Starts a task for each iteration and waits until they all complete

Groups iterations so there is a task per available thread
## USE OF TASKS IN SOME APPLICATIONS AND BENCHMARKS

<table>
<thead>
<tr>
<th>Application</th>
<th>Distributed 'coforall'</th>
<th>Threaded 'coforall'</th>
<th>Asynchronous 'begin'</th>
<th>'cobegin'</th>
<th>sync or atomic vars</th>
<th>subprocesses</th>
<th>forall</th>
<th>scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPO</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Arkouda</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CHAMPS</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChOp</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>ParFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Coral Reef</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Task Graph</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Poll Everywhere link: [pollev.com/michellestrout402](pollev.com/michellestrout402)

There will be fun questions throughout the talk.
CRAY HYPERPARAMETER OPTIMIZATION (HPO)

- Python interface and Chapel backend
- Supported distributed optimization as well as distributed training
  - E.g., 20 nodes, 5 HPO instances each training on 4 nodes
- Chapel code includes spawning subprocesses, which is a non-blocking operation
  - Blocking on the completion of the subprocess can be done with a 'wait'

```plaintext
use Subprocess;
var process = spawn(['./train --params=...'],
                    stdout=pipeStyle.pipe);

//...
process.wait();
for line in process.stdout.readlines() {
    writeln(line);
}
```
### USE OF TASKS IN SOME APPLICATIONS AND BENCHMARKS

<table>
<thead>
<tr>
<th>Application</th>
<th>Distributed 'coforall'</th>
<th>Threaded 'coforall'</th>
<th>Asynchronous 'begin'</th>
<th>'cobegin'</th>
<th>sync or atomic vars</th>
<th>subprocesses</th>
<th>forall</th>
<th>scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPO</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Arkouda</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CHAMPS</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChOp</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>ParFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Coral Reef</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Graph</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Poll Everywhere link:** [pollev.com/michellestrout402](pollev.com/michellestrout402)
There will be fun questions throughout the talk
ARKOUDA IN ONE SLIDE

What is it?
- A Python library supporting a key subset of NumPy and Pandas for Data Science
- Implemented using a client-server model with Chapel as the server to support scalability
- Designed to compute results within the human thought loop (seconds to minutes on TB-scale arrays)
- ~30K lines of Chapel

Who did it?
- Mike Merrill, Bill Reus, et al., US DOD
- Open-source: https://github.com/Bears-R-Us/arkouda

Why Chapel?
- high-level language with C-comparable performance
- great distributed array support
- ports from laptop to supercomputer
- close to Pythonic—thus is readable for Python users who look under the hood
ARKOUDA ARGSORT: HERO RUN

- Recent hero run performed on large Apollo system
  - 72 TiB of 8-byte values
  - 480 GiB/s (2.5 minutes elapsed time)
  - used 73,728 cores of AMD Rome
  - ~100 lines of Chapel code
- Believed to be within 2-3x of world record
  - however, a bit apples-to-oranges:
    - they sort larger key values (to their benefit)
    - their data starts on disk (SSD)
• Lots of 'forall' used over distributed arrays
• Superfast radix sort uses 'coforall' and 'scan'

```c
// loop over digits
for rshift in {0..#nBits by bitsPerDigit} {
    coforall loc in Locales do on loc {
        coforall task in Tasks {
            // each task histograms the digit for the numbers it is responsible for
        }
    }
}

// scan globalCounts to get bucket ends on each locale/task
coforall loc in Locales do on loc {
    coforall task in Tasks {
        // move all of the numbers where they need to go
    }
}
```
### USE OF TASKS IN SOME APPLICATIONS AND BENCHMARKS

<table>
<thead>
<tr>
<th>Application</th>
<th>Distributed 'coforall'</th>
<th>Threaded 'coforall'</th>
<th>Asynchronous 'begin'</th>
<th>'cobegin'</th>
<th>sync or atomic vars</th>
<th>subprocesses</th>
<th>forall</th>
<th>scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPO</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Arkouda</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CHAMPS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChOp</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coral Reef</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Graph</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Poll Everywhere link: [pollev.com/michellestrout402](pollev.com/michellestrout402)

There will be fun questions throughout the talk.
What is it?
- Computational Fluid Dynamics framework for airplane simulation written from scratch
- Modular design, permitting various computational modules to be integrated (or not)
- First ~48k lines written in ~2 years, now up to over 100k lines

Who did it?
- Professor Eric Laurendeau’s team at Polytechnique Montreal
- not open-source (yet), but available by request to researchers

Task Parallel Patterns
- SPMD-like parallelism with 'coforall's
- Threaded parallelism with 'coforall's
CHAMPS: QUOTE AND STATUS FROM THE PI

• Eric Laurendeau (PI) gave our CHIUW 2021 keynote
  • title: HPC Lessons From 30 Years of Practice in CFD Towards Aircraft Design and Analysis
  • students also gave talks on their individual efforts
  • key excerpt:
    "So CHAMPS, that's the new solver that has been made, and all made by the students... So, [Chapel] promotes the programming efficiency. It was easy for them to learn. ...I see the end result. We ask students at the master's degree to do stuff that would take 2 years and they do it in 3 months. And I'm not joking, this is from 2 years to 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."

• CHAMPS participating in 4th CFD High Lift Prediction Workshop and 1st Icing Prediction Workshop
  • teams compete against one another to do the same massive simulations
    – entries compared in terms of model accuracy, performance, practicality
  • sponsored by AIAA and NASA
  • initial results are looking competitive to longer-lived / more established codes from Stanford, MIT, etc.
### USE OF TASKS IN SOME APPLICATIONS AND BENCHMARKS

<table>
<thead>
<tr>
<th>Application</th>
<th>Distributed 'coforall'</th>
<th>Threaded 'coforall'</th>
<th>Asynchronous 'begin'</th>
<th>'cobegin'</th>
<th>sync or atomic vars</th>
<th>subprocesses</th>
<th>forall</th>
<th>scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPO</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Arkouda</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CHAMPS</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChOp</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ParFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Coral Reef</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Task Graph</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

**Poll Everywhere link:** pollev.com/michellestrout402
There will be fun questions throughout the talk
**CHOP IN ONE SLIDE**

**What is it?**
- Tree-based, branch and bound optimization algorithms
- Irregular tree, lots of pruning

**Who did it?**
- Tiago Carneiro and Nouredine Melab at the Imec - Belgium and INRIA Lille
- Open-source: [https://github.com/tcarneirop/ChOp](https://github.com/tcarneirop/ChOp)

**Chapel Task Parallel Constructs**
- Heavy use of 'forall' to implement a distributed task load balancer
- Using 'begin' and 'atomic' variables in checkpointing code

*from slides for "Towards Ultra-scale Optimization Using Chapel" by Tiago Carneiro (University of Luxembourg) and Nouredine Melab (INRIA Lille), CHIUW 2021*
### USE OF TASKS IN SOME APPLICATIONS AND BENCHMARKS

<table>
<thead>
<tr>
<th>Application</th>
<th>Distributed 'coforall'</th>
<th>Threaded 'coforall'</th>
<th>Asynchronous 'begin'</th>
<th>'cobegin'</th>
<th>sync or atomic vars</th>
<th>subprocesses</th>
<th>forall</th>
<th>scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPO</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Arkouda</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CHAMPS</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChOp</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>ParFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Coral Reef</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Task Graph</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

**Poll Everywhere link:** [pollev.com/michellestrout402](pollev.com/michellestrout402)

There will be fun questions throughout the talk.
PARFLOW

• Watershed hydrology model
• Bottom of the bedrock to the top of the canopy
• Began in 1998
• > 1,000,000 lines of C code

• Chapel piece
  • Summer intern project
  • MPI+C code calls out to Chapel
  • Uses ‘forall’ for ...
    – Shared-memory parallelism
    – User-defined iterators over boundary conditions
SURFACE TRAVERSAL, C VS CHAPEL

Surface Traversal Macro, C

Surface Traversal, Chapel

```plaintext
iter surfacePoints(ground, outerDomain) {
  for f in 0..<NumFaces do
    for box in ground.surfaceBoxes(f) do
      for (i,j,k) in box do
        yield (i,j,k,create_fdir(f));
    }
```

CHAPEL NEEDS 40% LESS CODE
## USE OF TASKS IN SOME APPLICATIONS AND BENCHMARKS

<table>
<thead>
<tr>
<th>Application</th>
<th>Distributed 'coforall'</th>
<th>Threaded 'coforall'</th>
<th>Asynchronous 'begin'</th>
<th>'cobegin'</th>
<th>sync or atomic vars</th>
<th>subprocesses</th>
<th>forall</th>
<th>scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPO</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Arkouda</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CHAMPS</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChOp</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>ParFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Coral Reef</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Task Graph</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

**Poll Everywhere link:** [pollev.com/michellestrout402](pollev.com/michellestrout402)

There will be fun questions throughout the talk
Analyzing images for coral reef diversity

Less than 300 lines of code scales out to 100s of processors

Full maps calculated in seconds, rather than days

Task parallel patterns

'forall' in convolve_and_calculate doing shared memory, thread-level parallelism per node

'coforall' in main doing distributed memory parallelism over swaths of the image

'cobegin' could be used to input different file formats in parallel
Create a \((P \times P)\) mask to find all points within a given radius.
We convolve this mask over the entire domain and perform a weighted reduce at each location.

(Add up weighted values of all points in neighborhood)
We convolve this mask over the entire domain and perform a weighted reduce at each location.
We convolve this mask over the entire domain and perform a weighted reduce at each location.
We convolve this mask over the entire domain and perform a weighted reduce at each location.
We convolve this mask over the entire domain and perform a weighted reduce at each location.
We convolve this mask over the entire domain and perform a weighted reduce at each location.
We convolve this mask over the entire domain and perform a weighted reduce at each location.

etc.
ALGORITHM: Divide the domain into “strips” and allocate a task for each strip.

Task 1

Task 2

...

Task (n-1)

Task n
### USE OF TASKS IN SOME APPLICATIONS AND BENCHMARKS

<table>
<thead>
<tr>
<th>Application</th>
<th>Distributed 'coforall'</th>
<th>Threaded 'coforall'</th>
<th>Asynchronous 'begin'</th>
<th>'cobegin'</th>
<th>sync or atomic vars</th>
<th>subprocesses</th>
<th>forall</th>
<th>scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPO</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Arkouda</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CHAMPS</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChOp</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Coral Reef</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Task Graph</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

**Poll Everywhere link:** pollev.com/michellestrout402

There will be fun questions throughout the talk.
ARBITRARY TASK GRAPHS

- **Encoding dependencies**
  - Array of 'atomic int'
  - 'numToWaitFor[i]' is the number of tasks that task i depends on

- **Starting asynchronous tasks**
  - 'for' loop
  - 'begin' block statement defines each task i

- **Waiting for dependencies to resolve**
  - Each task waits for the number of tasks it depends on to go to zero
  - 'numToWaitFor[i].waitFor(0)'

- **Tell dependent tasks when done**
  - 'numToWaitFor[j].fetchadd(-1)'

---

```
config const N = 4;
var A : [0..<N,0..<N] bool;

// hardcoding a task graph
// 0 and 1 can execute right away
// 2 and 3 have to wait for
A[2,0] = true;
A[2,1] = true;
A[3,1] = true;

var numToWaitFor : [0..<N] atomic int;

// encoding how many each task needs to wait for
forall (i,j) in A.domain {
  if A[i,j] { numToWaitFor[i].fetchAdd(1); }
}

// start all of the tasks and have them wait as needed and
// have them decrement the appropriate numToWaitFor
for i in 0..<N {
  begin {
    numToWaitFor[i].waitFor(0);

    // do task i work
    writeln("Task: ", i, " is working away");

    for j in i+1..<N { // should only be later tasks
      if A[j,i] { numToWaitFor[j].fetchAdd(-1); }
    }
  }
}
**TASKS ON GPUS? GPU SUPPORT IN CHAPEL**

- Chapel compiler generating code for GPUs
  - Nascent support for NVIDIA
  - Exploring AMD and Intel support
- Chapel code calling CUDA examples
- Key concepts
  - Using the 'locale' concept to indicate execution and data allocation on GPUs
  - 'forall' and 'foreach' loops will be converted to kernels
  - Arrays declared in 'on here.gpus[0]' blocks are allocated on the GPU
- For more info...
  - [https://chapel-lang.org/docs/technotes/gpu.html](https://chapel-lang.org/docs/technotes/gpu.html)

```chapel
use GPUDiagnostics;
startGPUDiagnostics();

var operateOn = if here.gpus.size > 0
  then here.gpus else [here,];

// Same code can run on GPU or CPU
coforall loc in operateOn do on loc {
  var A: [1..10] int;
  foreach a in A do a+=1;
  writeln(A);
}

stopGPUDiagnostics();
writeln(getGPUDiagnostics());
```
TALK TAKEAWAYS

- Chapel is a general-purpose programming language designed to leverage parallelism
- It is being used in some large production codes
  - **Performance**: The resulting applications are fast and scalable
  - **Programmability**: The code is relatively easy to write and maintain
- Chapel supports many different task parallelism patterns

<table>
<thead>
<tr>
<th>Application</th>
<th>Distributed 'coforall'</th>
<th>Threaded 'coforall'</th>
<th>Asynchronous 'begin'</th>
<th>'cobegin'</th>
<th>sync or atomic vars</th>
<th>subprocesses</th>
<th>forall</th>
<th>scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
<td>✓</td>
<td>□</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPO</td>
<td>✓</td>
<td>□</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Arkouda</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>CHAMPS</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>ChOp</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>ParFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coral Reef</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Task Graph</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Poll Everywhere link: pollev.com/michellestrout402
Let me know your thoughts in a short survey
USE A CONTAINER TO CHECKOUT SOME CHAPEL EXAMPLES

• Links for some Chapel examples
    – Especially check out days 11 and 12 since they cover sync variables and atomics
  • Wavelet example by Jeremiah Corrado, Slides and Code at
    https://github.com/mstrout/ChapelForPythonProgrammersFeb2023/tree/main/wavelet_example
  • Primers, https://chapel-lang.org/docs/primers/index.html
  • Test directory in main repository, https://github.com/chapel-lang/chapel/tree/main/test

• Using a container on your laptop
  • See https://github.com/mstrout/ChapelForPythonProgrammersFeb2023 for some example code
  • First, install podman or docker for your machine and then start them up
  • Then, the below commands work with podman or docker

    podman pull docker.io/chapel/chapel    # takes about 3 minutes
    cd ChapelForPythonProgrammersFeb2023    # assuming git clone has happened
    podman run --rm -v "$PWD":/myapp -w /myapp chapel/chapel chpl hello.chpl
    podman run --rm -v "$PWD":/myapp -w /myapp chapel/chapel ./hello
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

Figure 7: Efficiency vs task granularity (stencil, 1 node). Higher is better.

Figure 10: METG vs deps/task (nearest, 1 node). Lower is better.