



**Hewlett Packard  
Enterprise**

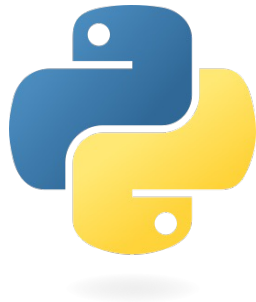
# **Arkouda and Chapel: Highlights Since CLSAC 2022**

**Brad Chamberlain**  
November 6, 2024



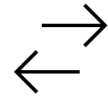
# What is Arkouda?

Q: “What is Arkouda?”

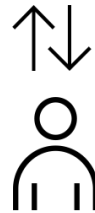
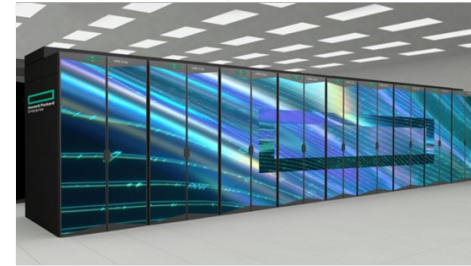


**Arkouda Client**  
(written in Python)

```
Jupyter big_add_sum Last Checkpoint: 18 minutes ago (Autosaved)
File Edit View Insert Cell Kernel Widgets Help Python 3.0
In [1]: import arkouda as ak
In [2]: ak.v = False
        ak.startup(server="localhost", port=5555)
        4.2.1
        SMP = tcp://localhost:5555
In [3]: ak.v = False
        N = 10**8 # 10**8 = 1000 * 10**5 = 1000000 # 2**22 = 4**11 = 250000
        A = ak.arange(N, 1)
        B = ak.arange(N, 1)
        C = A*B
        print(ak.sum(C, C))
name: "big_2" dtype: "int64" size:100000000 mem:1 shape:(100000000) int64:8
[0 2 4 ... 19999998 199999996 199999998]
In [4]: S = (B*(B-1))**2
        print(S)
        print(ak.sum(C))
99999990000000.0
99999990000000.0
In [5]: ak.shutdown()
```



**Arkouda Server**  
(written in Chapel)



**User writes Python code**



# What is Chapel?

---

**Chapel:** A modern parallel programming language

- Pythonic readability, writability, maintainability
- Performs/scales like Fortran, C/C++, MPI, SHMEM, OpenMP, CUDA, ...
- Safety features w.r.t. types, parallelism, memory



# Bale IndexGather in Chapel vs. SHMEM on HPE Cray EX (Slingshot-11)

## Chapel (Simple / Auto-Aggregated version)

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

## Chapel (Explicitly Aggregated version)

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

## SHMEM (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();
}
```

## SHMEM (Conveyors version)

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

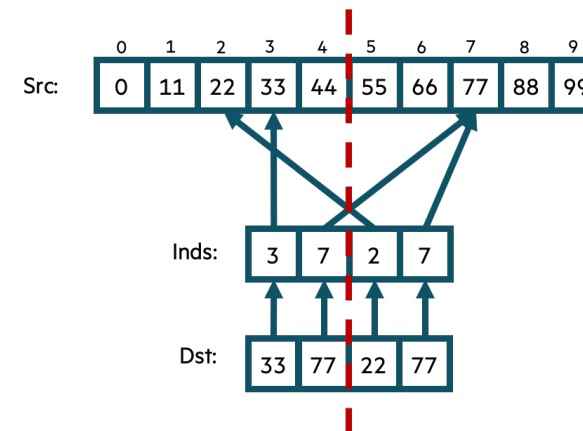
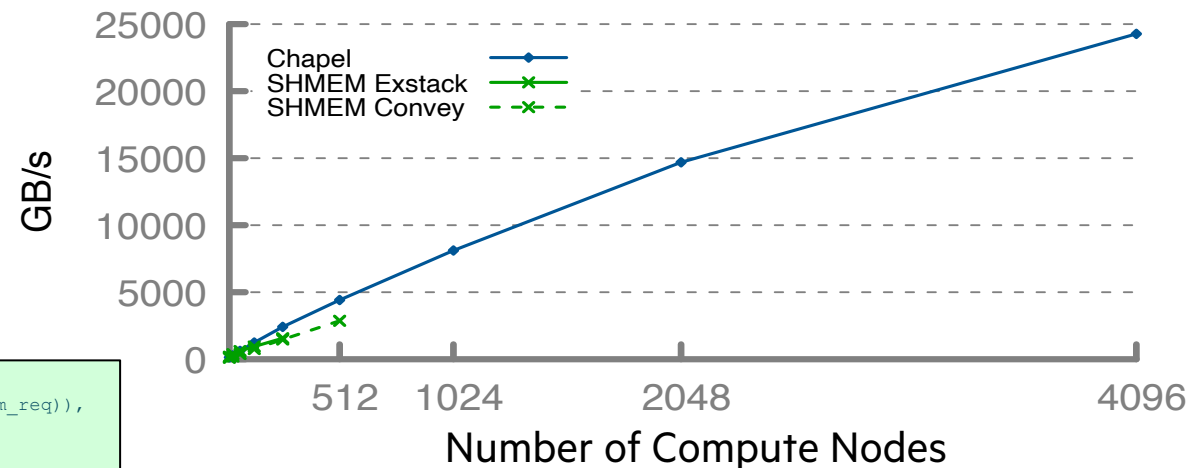
    for (; i < l_num_req; i++) {
        pkg.idx = i;
        pkg.val = pckindx[i] >> 16;
        pe = pckindx[i] & 0xffff;
        if (!convey_push(requests, &pkg, pe))
            break;
    }

    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;
        }
    }

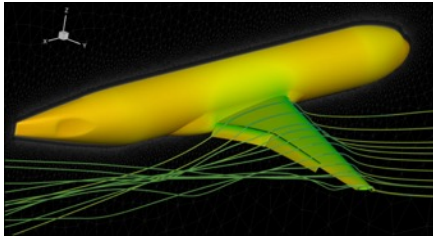
    while (convey_pull(replies, ptr, NULL) == convey_OK)
        tgt[ptr->idx] = ptr->val;
}
```

## Bale Indexgather Performance

HPE Cray EX (Slingshot-11)

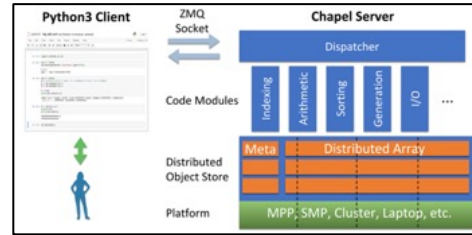


# Applications of Chapel



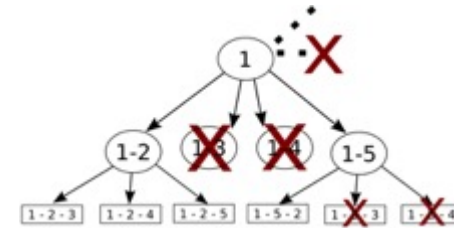
## CHAMPS: 3D Unstructured CFD

Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.  
École Polytechnique Montréal



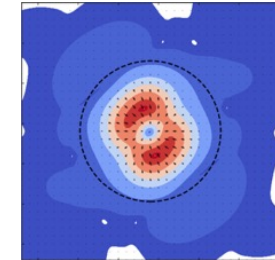
## Arkouda: Interactive Data Science at Massive Scale

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



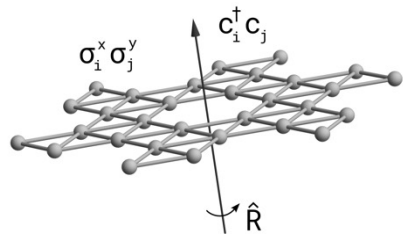
## ChOp: Chapel-based Optimization

T. Carneiro, G. Helbecque, N. Melab, et al.  
INRIA, IMEC, et al.



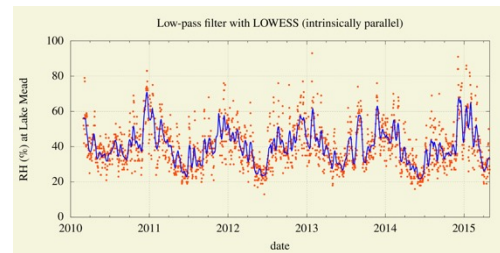
## ChpUltra: Simulating Ultralight Dark Matter

Nikhil Padmanabhan, J. Luna Zagorac, et al.  
Yale University et al.



## Lattice-Symmetries: a Quantum Many-Body Toolbox

Tom Westerhout  
Radboud University



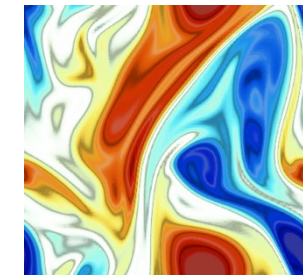
## Desk dot chpl: Utilities for Environmental Eng.

Nelson Luis Dias  
The Federal University of Paraná, Brazil



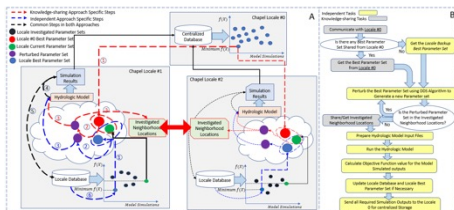
## RapidQ: Mapping Coral Biodiversity

Rebecca Green, Helen Fox, Scott Bachman, et al.  
The Coral Reef Alliance



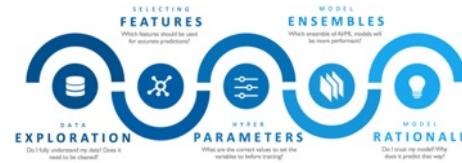
## ChapQG: Layered Quasigeostrophic CFD

Ian Grooms and Scott Bachman  
University of Colorado, Boulder et al.



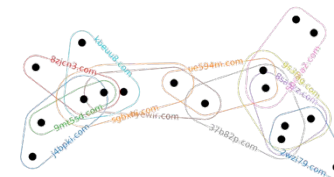
## Chapel-based Hydrological Model Calibration

Marjan Asgari et al.  
University of Guelph



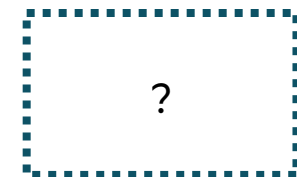
## CrayAI HyperParameter Optimization (HPO)

Ben Albrecht et al.  
Cray Inc. / HPE



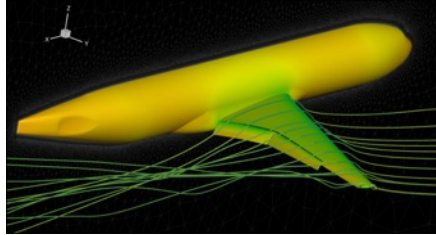
## CHGL: Chapel Hypergraph Library

Louis Jenkins, Cliff Joslyn, Jesun Firoz, et al.  
PNNL



## Your Application Here?

# Chapel's users reap its benefits at varying scales of systems and code sizes



**Computation:** aircraft simulation

**Code size:** 100,000+ lines

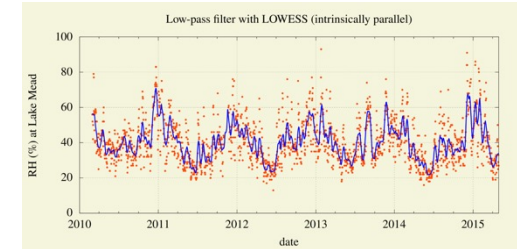
**Systems:** desktops, HPC systems



**Computation:** coral reef image analysis

**Code size:** ~300 lines

**Systems:** desktops, HPC systems w/ GPUs



**Computation:** ATTO data analysis

**Code size:** 5000+ lines

**Systems:** desktops w/ CPUs & GPUs (only)



# Chapel Blog Series: 7 Questions for Chapel Users

## Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts



### 7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

Posted on October 15, 2024.

Tags: [User Experiences](#)

By: [Eric Laurendeau](#)

This is the first in a new series of questions about the ways in which you are using Chapel and someone who is and ought to be using it.

For our inaugural edition of the series, we turn to one of Chapel's most ambitious users: Éric and CHAMPS, but he

#### 1. Who are you?

My name is Éric Laurendeau and I'm currently completing my Bachelor's in Canada (the

## Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts



### 7 Questions for Scott Bachman: Analyzing Coral Reefs with Chapel

Posted on October 15, 2024.

Tags: [User Experiences](#)

User Experiences

By: [Brad Chamberlain](#)

In this second installment of our [Seven Questions for Chapel Users](#) series, we turn to Scott Bachman, who recently completed his analysis in coral reefs to study ocean acidification. Scott started as a visiting scholar with the University of Hawaii and during his time there he took on several projects he continued to apply Chapel in his work.

One noteworthy thing about the coral reefs is that the lines of Chapel code, yet can be used on supercomputers. This serves as a strong example of the framework covered in our [previous](#) edition. The code sizes that researchers are producing are growing.

This interview was conducted live (with the assistance of the Chapel team).

## Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts



### 7 Questions for Nelson Luís Dias: Atmospheric Turbulence in Chapel

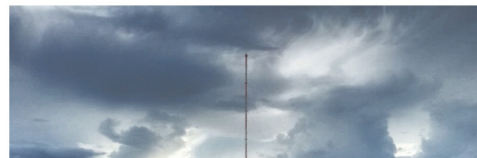
Posted on October 15, 2024.

Tags: [User Experiences](#) [Interviews](#) [Data Analysis](#)

[Computational Fluid Dynamics](#)

By: [Engin Kayraklioglu](#), [Brad Chamberlain](#)

In this edition of our [Seven Questions for Chapel Users](#) series, we turn to Dr. Nelson Luis Dias from Brazil who is using Chapel to analyze data generated by the [Amazon Tall Tower Observatory \(ATTO\)](#), a project dedicated to long-term, 24/7 monitoring of greenhouse gas fluctuations. Read on to learn more about his work and use of Chapel!



available at: <https://chapel-lang.org/blog/series/7-questions-for-chapel-users/>



# What is Arkouda?

Q: “What is Arkouda?”

**Arkouda Client**  
(written in Python)



```
Jupyter big_add_sum Last Checkpoint: 18 minutes ago (autosaved) Python 3.0
```

```
In [1]: import arkouda as ak
```

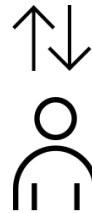
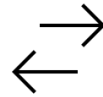
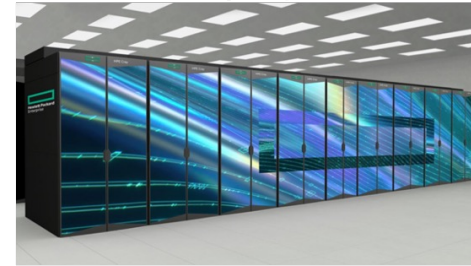
```
In [2]: ak.v = False
ak.startup(server="localhost", port=5555)
4.2.1
URL = http://localhost:5555
```

```
In [3]: ak.v = False
N = 10**4 # 10k = 1000 * 10 == 10000 # 2**22 = 4 == 250000
A = ak.arange(N, 1)
B = ak.arange(N, 1)
C = A*B
print(ak.sum(C), C)
name: "big_2" dtype: "int64" size:100000000 mem:1 shape:(100000000) int64:8
[0 2 4 ... 19999994 19999996 19999998]
```

```
In [4]: B = (B*(B-1))//2
print(C*B)
print(ak.sum(C))
99999990000000.0
999999900000000
```

```
In [5]: ak.shutdown()
```

**Arkouda Server**  
(written in Chapel)



**User writes Python code**

**A1:** “A scalable version of NumPy / Pandas for data scientists”

**A2:** “An extensible framework for arbitrary HPC computations”

**A3:** “A way to drive HPC systems interactively from Python on a laptop”





# In Memoriam

- Mike Merrill passed away two years ago this Friday
- Mike was the chief architect and developer of Arkouda, as well as a friend to many on the Chapel project



## Arkouda: NumPy-like arrays at massive scale backed by Chapel

Michael Merrill\*, William Reus<sup>†</sup>, and Timothy Neumann<sup>‡</sup>

U.S. Department of Defense Washington DC, USA

Email: \*mhmerrill@mac.com, <sup>†</sup>reus@post.harvard.edu, <sup>‡</sup>timothyneumann1@gmail.com



**Hewlett Packard  
Enterprise**

# **Arkouda and Chapel: Updates I'd want Mike to know**



**Brad Chamberlain**  
November 6, 2024

# Arkouda Argsort Scalability

## HPE Cray EX (May 2023)

- Slingshot-11 network (200 Gb/s)
- 8192 compute nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)

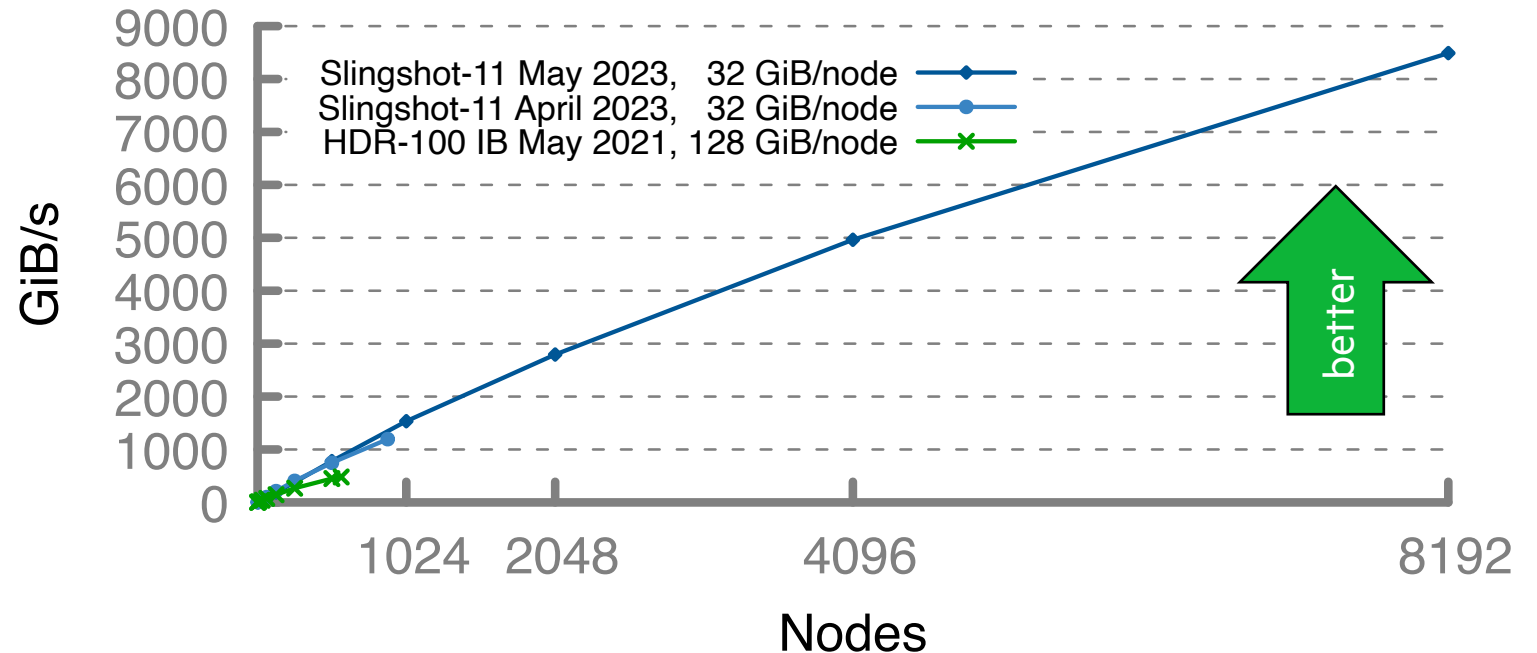
## HPE Cray EX (April 2023)

- Slingshot-11 network (200 Gb/s)
- 896 compute nodes
- 28 TiB of 8-byte values
- ~1200 GiB/s (~24 seconds)

## HPE Apollo (May 2021)

- HDR-100 Infiniband network (100 Gb/s)
- 576 compute nodes
- 72 TiB of 8-byte values
- ~480 GiB/s (~150 seconds)

Arkouda Argsort Performance



Achieved using ~100 lines of Chapel



# Arkouda's Modularity Is Being Leveraged

- Arkouda servers can be built by mixing and matching various modules
  - Initial modules supported NumPy and Pandas
- Key examples include:
  - Graph Processing
  - Multidimensional Arrays supporting the Python Array API
  - Sparse Matrix Computations

**Jeremiah Corrado will demo this for the Pangeo community, Nov 20th**



# New Tooling that Simplifies Adding New Arkouda Operations

## Was:

```
/* intIndex "a[int]" response to __getitem__(int) */
@arkouda.registerND(cmd_prefix="[int]")
proc intIndexMsg(cmd: string, msgArgs: borrowed MessageArgs, st: borrowed SymTab, param nd: int): MsgTuple throws {
  param pn = Reflection.getRoutineName();
  var repMsg: string; // response message
  var idx = msgArgs.get("idx").getTuple(nd);
  const name = msgArgs.getValueOf("array");
  imLogger.debug(getModuleName(),getRoutineName(),getLineNumber(),
    "%s %s %?" format(cmd, name, idx));
  var gEnt: borrowed GenSymEntry = getGenericTypedArrayEntry(name, st);

  select (gEnt.dtype) {
    when (DType.Int64) {
      var e = toSymEntry(gEnt, int, nd);
      repMsg = "item %s %?" format(dtype2str(e.dtype), e.a[...idx]);

      imLogger.debug(getModuleName(),getRoutineName(),getLineNumber(),repMsg);
      return new MsgTuple(repMsg, MsgType.NORMAL);
    }
    when (DType.UInt64) {
      var e = toSymEntry(gEnt, uint, nd);
      repMsg = "item %s %?" format(dtype2str(e.dtype), e.a[...idx]);

      imLogger.debug(getModuleName(),getRoutineName(),getLineNumber(),repMsg);
      return new MsgTuple(repMsg, MsgType.NORMAL);
    }
    when (DType.Float64) {
      var e = toSymEntry(gEnt, real, nd);
      repMsg = "item %s %.17r" format(dtype2str(e.dtype), e.a[...idx]);

      imLogger.debug(getModuleName(),getRoutineName(),getLineNumber(),repMsg);
      return new MsgTuple(repMsg, MsgType.NORMAL);
    }
    when (DType.Bool) {
      var e = toSymEntry(gEnt, bool, nd);
      repMsg = "item %s %?" format(dtype2str(e.dtype), e.a[...idx]);
      repMsg = repMsg.replace("true", "True"); // chapel to python bool
      repMsg = repMsg.replace("false", "False"); // chapel to python bool

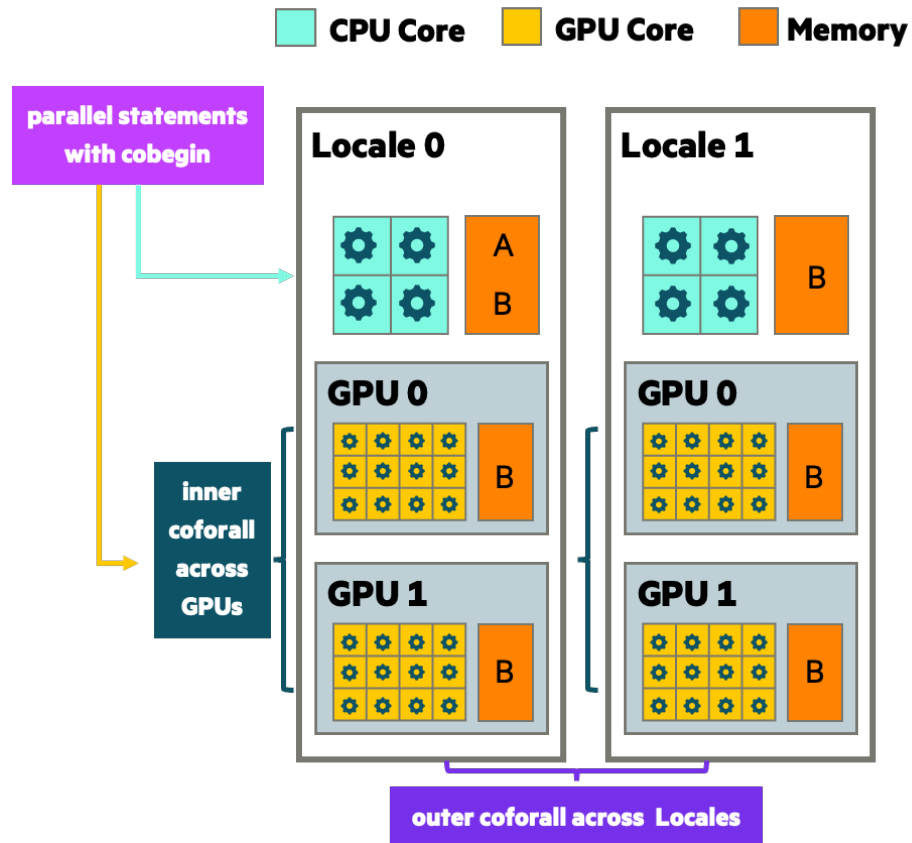
      imLogger.debug(getModuleName(),getRoutineName(),getLineNumber(),repMsg);
      return new MsgTuple(repMsg, MsgType.NORMAL);
    }
    when (DType.BigInt) {
      var e = toSymEntry(gEnt, bigint, nd);
      repMsg = "item %s %?" format(dtype2str(e.dtype), e.a[...idx]);
      imLogger.debug(getModuleName(),getRoutineName(),getLineNumber(),repMsg);
      return new MsgTuple(repMsg, MsgType.NORMAL);
    }
    otherwise {
      var errorMsg = notImplementedError(pn, dtype2str(gEnt.dtype));
      imLogger.error(getModuleName(),getRoutineName(),getLineNumber(),errorMsg);
      return new MsgTuple(errorMsg, MsgType.ERROR);
    }
  }
}
```

## Now:

```
@arkouda.registerCommand("[int]")
proc intIndex(const ref array: [?d] ?t, idx: d.rank*int): t {
  return array[idx];
}
```

# Chapel and GPUs

- Chapel now supports vendor-neutral GPU programming (NVIDIA and AMD)
  - Uses the identical features as for programming multi-core CPUs and HPCs



```
var A: [1..n, 1..n] real;
coforall l in Locales do on l {
  cobegin {
    coforall g in here.gpus do on g {
      var B: [1..n, 1..n] real;
      B = 2;
      A = B;
    }
  }
}
writeln(A);
```



# GPU Highlights: Paul Sathre's ChapelCon 2024 Keynote



## A case for parallel-first languages in a post-serial, accelerated world

Paul Sathre  
Research Software Engineer  
Synergy Lab & NSF Center for Space, High-Performance and Resilient Computing  
Virginia Tech



Intro

What I really care about:

Closing the gaps between the **parallel hardware** we already have, and **the people** who could benefit from it

So how do we **enable** them?  
(Conversely, what are the **barriers** to use?)

4



Sathre, P. "A case for parallel-first languages in a post-serial, accelerated world"  
ChapelCon'24 -- June 7, 2024



Act I: Parallelism is everywhere ...

## Post-serial or "serial with sprinkles"

- Dominant programming models are still **post-serial**
  - "Sprinkles": **optional** libraries, pragmas, language extensions
- Chapel presents a different option: **parallel-first**
  - A non-separable part of the **keywords**, **data abstractions**, and **semantics** of the language (i.e. **promotion**)



# ChAI: Chapel AI

- A native PyTorch-esque Chapel module
  - Supports tensors, training, and inference
  - Runs locally, distributed, and/or on GPUs
  - Can be integrated into other HPC codes – e.g., Arkouda
- Implemented by Iain Moncrief
  - junior at Oregon State University
  - notably, written in one summer internship

The screenshot shows the GitHub repository page for 'ChAI' by user 'Iain Moncrief'. The repository is public and has 4 forks and 3 stars. The main branch is 'main' with 5 branches and 1 tag. A pull request #12 is merged from chapel-lang/main. The repository contains several folders and files, including 'docs', 'examples', 'learning', 'lib', 'presentation', 'scripts', 'src', 'test', '.gitignore', 'Mason.toml', and 'README.md'. The README is expanded to show the title 'ChAI: Chapel Artificial Intelligence' and a description: 'ChAI is a library for AI/ML in Chapel. Due to Chapel's highly parallel nature, it is well-suited for AI/ML tasks; the goal of the library is to provide a foundation for such tasks, enabling local, distributed, and GPU-enabled computations.'

File/Folder	Description	Last Commit
docs	Rebuild docs.	3 months ago
examples	Update MultiLocaleInference.chpl to have better comments	2 months ago
learning	Add example file.	3 months ago
lib	minor updates for compatability with chapel 2.2	2 months ago
presentation	Add data for presentation	3 months ago
scripts	Added pretrained model	3 months ago
src	Add tests and fix some functions.	3 months ago
test	make model directory configurable in loadFromSpec test, ...	2 months ago
.gitignore	Add tensors for examples.	3 months ago
Mason.toml	Changed name to ChAI.	4 months ago
README.md	Add an initial README file	last month





# But wait, there's more!

- **AWS/EFA:** Now supported by Chapel and Arkouda
- **Spack/E4S:** Now support Chapel (Arkouda in-progress)
- **Chapel 2.0:** released this year, providing forward-compatibility
- **Parquet:** improved and optimized support
- **New websites:** Arkouda's is online, Chapel's will launch soon
- ...



The screenshot shows the Arkouda website homepage. At the top, there is a navigation bar with the Arkouda logo, the name 'Arkouda', and links for 'github', 'documentation', and 'gitter'. The main heading reads 'Massive-scale data science, from the comfort of your laptop'. Below this, there are two radio buttons: 'Arkouda Ready for supercomputers' (selected) and 'NumPy Industry standard'. A code block contains the following Python code:

```
# Launch an Arkouda server: ./arkouda_server -nl <number-of-locates>
import arkouda as ak
# connect to the server
ak.connect('localhost', 5555)
# Generate two large arrays
a = ak.random.randint(0, 2**32, 2**38) # ----> Won't fit on a single machine!
b = ak.random.randint(0, 2**32, 2**38) # 1TB of random integers.
# add them
c = a + b
# Sort the array and print first 10 elements
c = ak.sort(c)
print(c[0:10])
```

At the bottom, there are three buttons: 'Try it Out', 'Tutorial Video', and 'Chat on Gitter'.

The screenshot shows the Chapel Programming Language website. At the top, there is a navigation bar with the Chapel logo and links for 'DOWNLOAD', 'DOCS', 'LEARN', 'RESOURCES', 'COMMUNITY', and 'BLOG'. The main heading reads 'The Chapel Programming Language' with the tagline 'Productive parallel computing at every scale.' Below this, there is a list of examples with radio buttons: 'Hello World' (selected), 'Distributed Hello World', 'Parallel File IO', '1D Heat Diffusion', and 'GPU Kernel'. A code block contains the following Chapel code:

```
writeln("Hello, world!");
// create a parallel task per processor core
coforall tid in 0..here.maxTaskPar do
  writeln("Hello from task ", tid);
// print these 1,000 messages in parallel using all cores
forall i in 1..1000 do
  writeln("Hello from iteration ", i);
```

Below the code block, there are three buttons: 'TRY CHAPEL', 'GET CHAPEL', and 'LEARN CHAPEL'. At the bottom, there are three columns with the following text:

- PRODUCTIVE**: Concise and readable without compromising speed or expressive power. Consistent concepts for parallel computing make it easier to learn.
- PARALLEL**: Built from the ground up to implement parallel algorithms at your desired level of abstraction. No need to trade low-level control for convenience.
- FAST**: Chapel is a compiled language, generating efficient machine code that meets or beats the performance of other languages.

# Thank you

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

