Chapel's Batteries-Included Approach for Portable Parallel Programming

Engin Kayraklioglu

Advances in Applied Computer Science Invited Speaker Series
Los Alamos National Laboratory
June 18, 2025

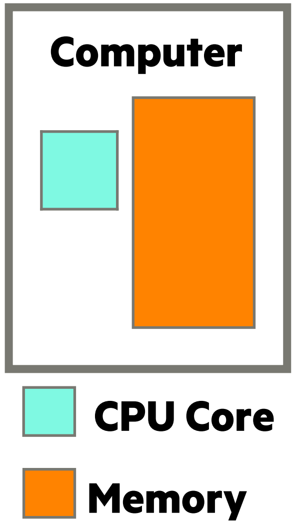
Computational Science

You start to learn programming…

- You have a processor (singular) and some memory
- You store your data in *the* memory
- You crunch your numbers in *the* processor
- Python is typically good enough

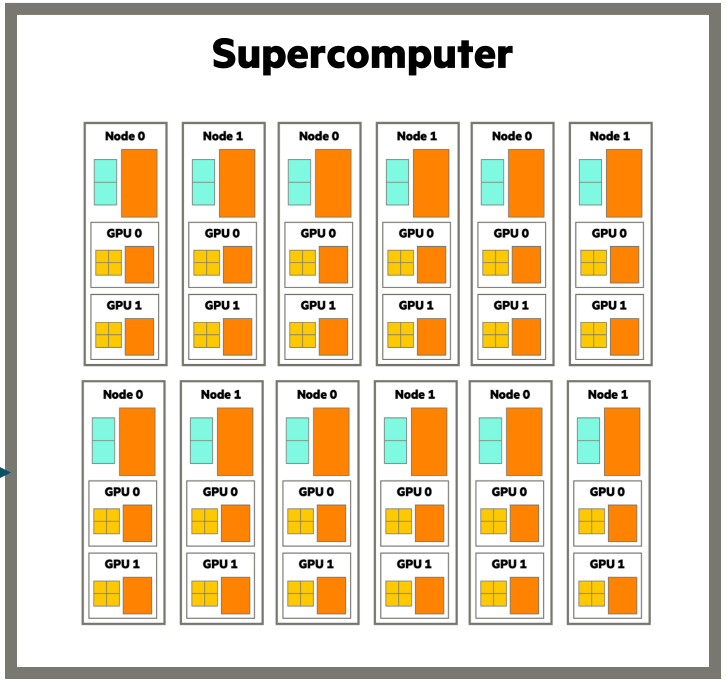
One day, you need to use a supercomputer…

- You have millions of processors
    - Some of them are GPUs
- Your memory is now distributed
    - Have to pay attention where your data is



You "just" learn some combination of

- C, C++, Fortran
- MPI
- OpenMP
- CUDA, HIP, SYCL, Kokkos
- SLURM



"Because there is no batteries-included programming abstraction to enable you to program both"

… or is there?

What is Chapel?

Chapel: A modern parallel programming language

- portable & scalable
- open-source & collaborative

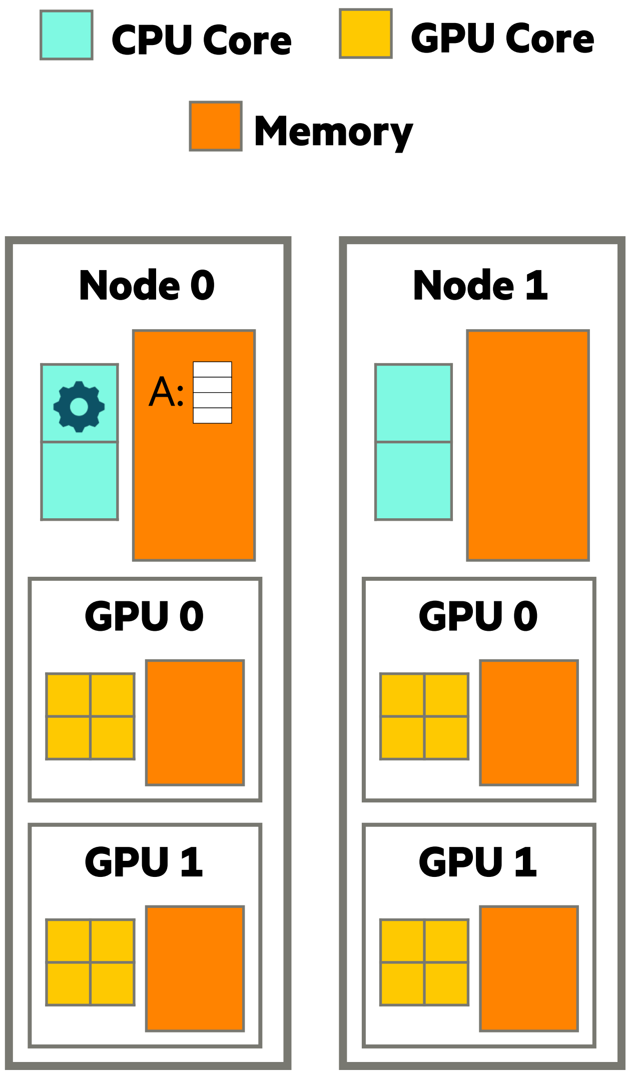
Goals:

- Support general parallel programming
- Make parallel programming at scale far more productive

chapel-lang.org

What does Chapel code look like?

Programming with Chapel: Fundamentals



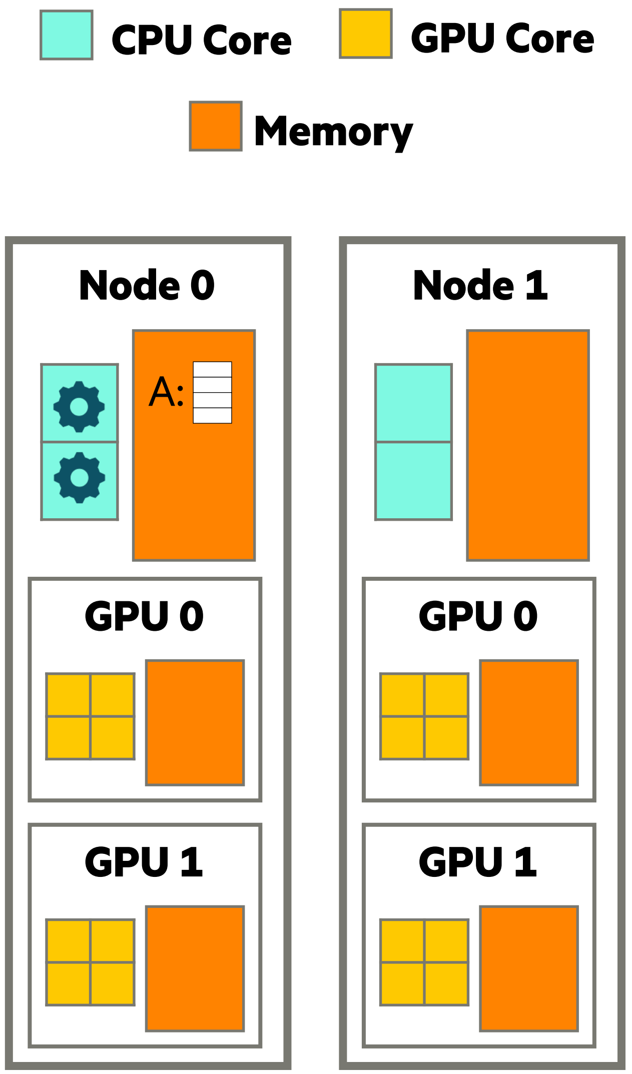

```
var A: [1..10] int;
```

Local, non-distributed array allocation

```
for elem in A do
  elem += 1;
```

Sequential iteration over the array

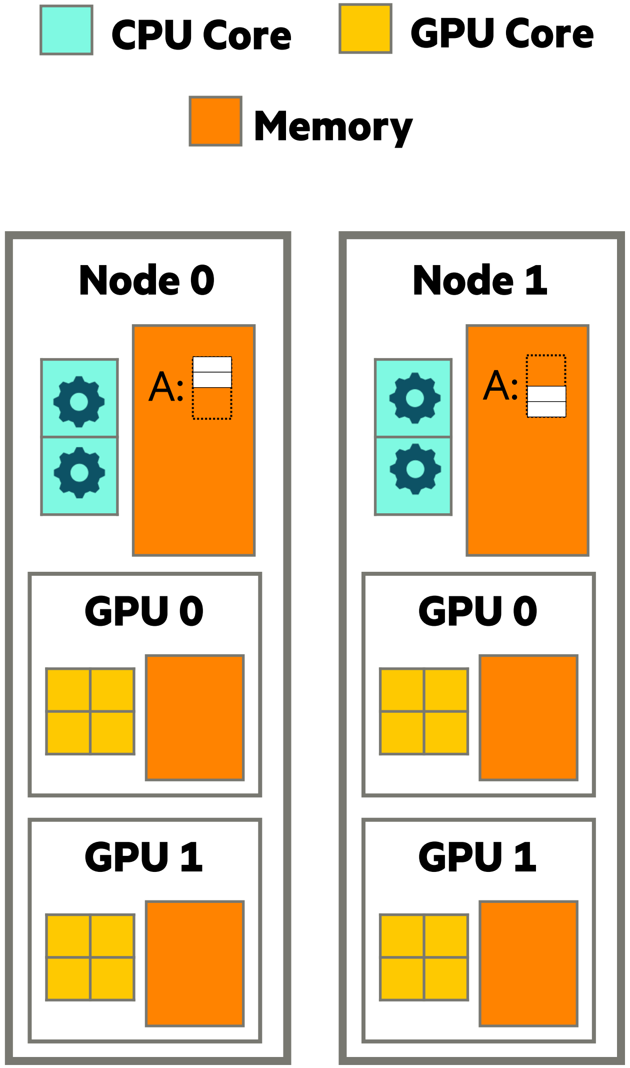
Programming with Chapel: Basic Data Parallelism




```
var A: [1..10] int;

forall elem in A do
  elem += 1;
```

Parallel iteration over the array

Programming with Chapel: Basic Data Parallelism



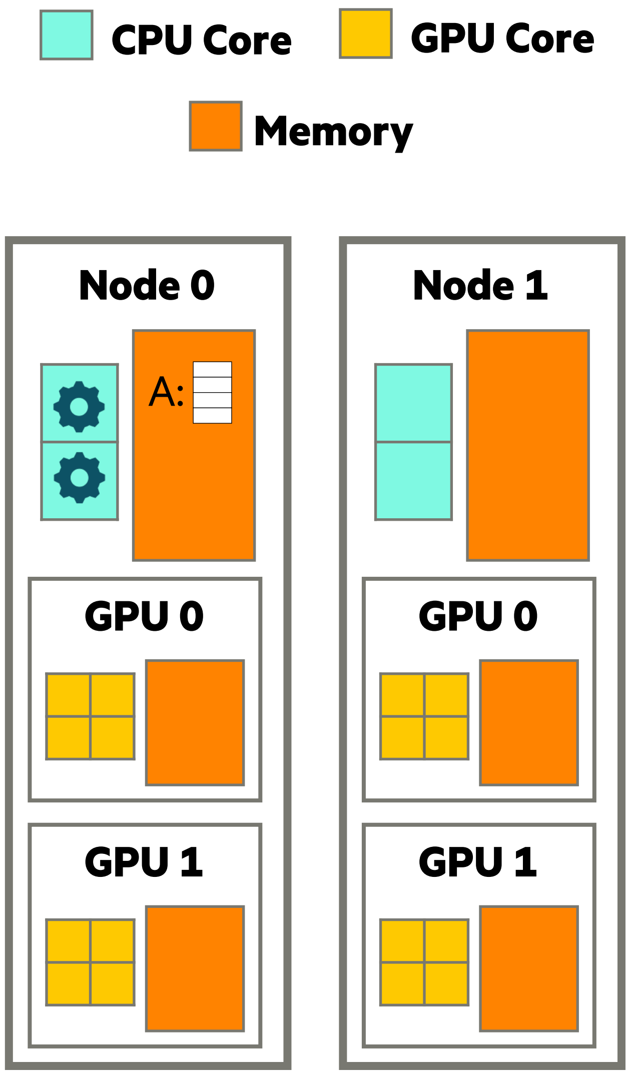

```
use BlockDist;
var Arr = blockDist.createArray(1..10, int);
```

Block-distributed array allocation

```
forall elem in Arr do
  elem += 1;
```

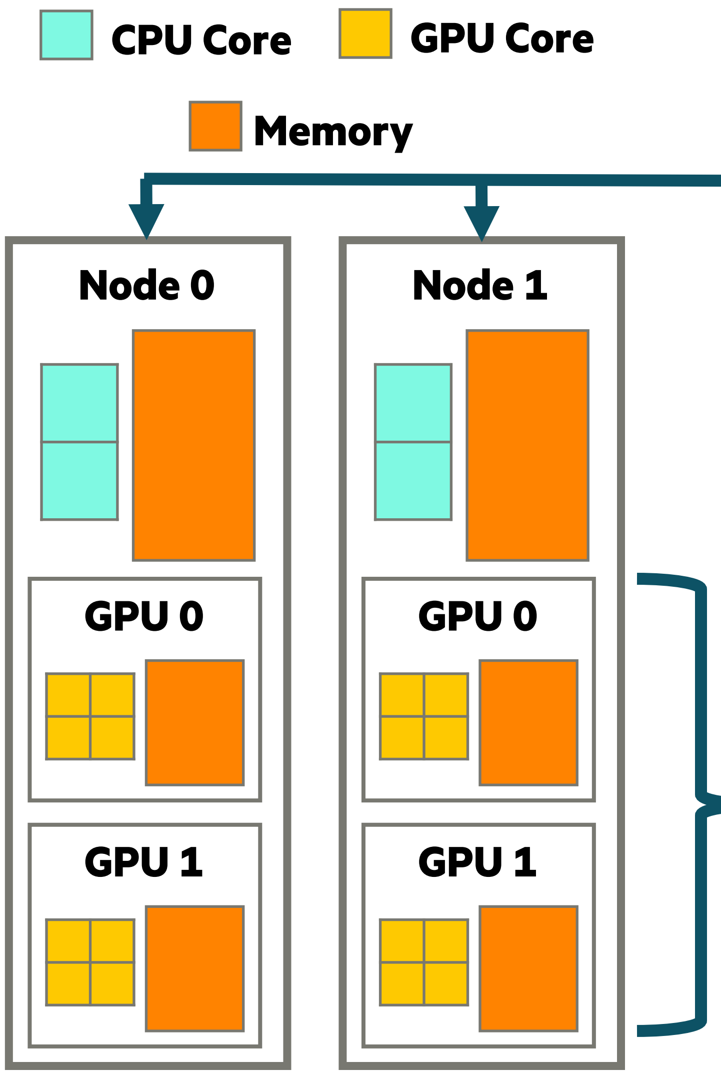
Distributed, parallel iteration over the array

Programming with Chapel: Basic Data Parallelism




```
var A: [1..10] int;

forall elem in A do
  elem += 1;
```

Programming with Chapel: Locality as a First-Class Citizen



- "locale" is a built-in Chapel type
- In the most common scenario: locale == node
- Under the hood, a locale is always a process
- Users can configure their applications to use:
  - a locale per socket
  - a locale per GPU
  - a locale per NUMA domain …

- GPUs are represented as "sublocales"

Predefined Variables for Locality

`here`

The current execution locale

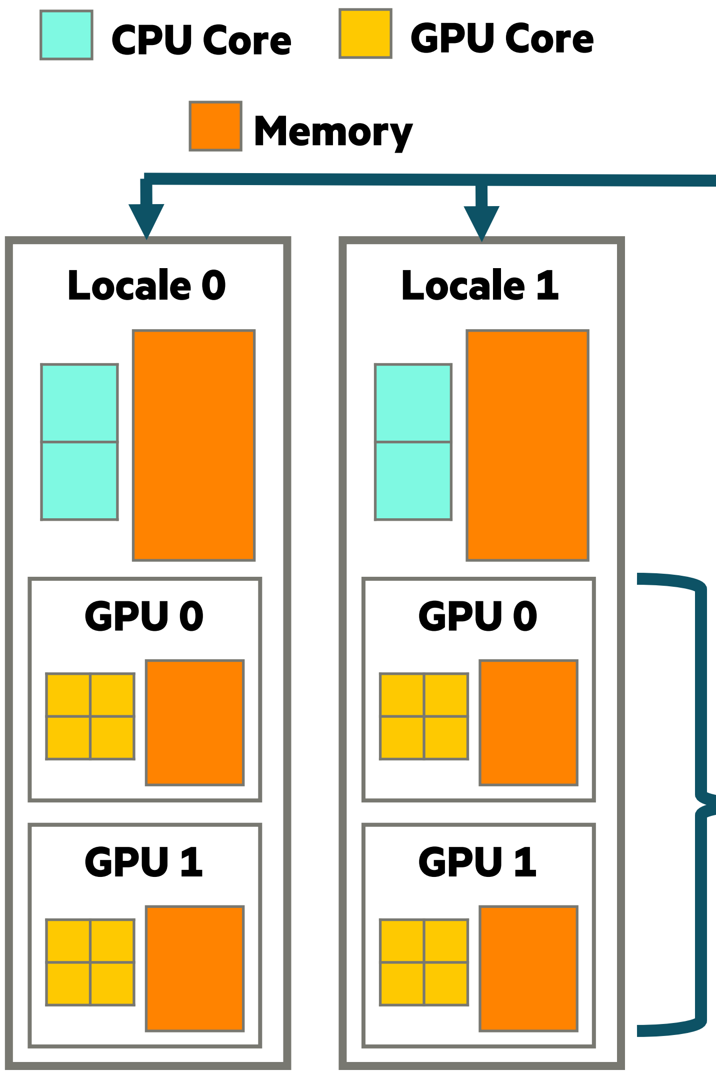
`Locales`

An array storing all locales

`locale.gpus`

An array storing GPU sublocales

Programming with Chapel: Locality as a First-Class Citizen



- "locale" is a built-in Chapel type
- In the most common scenario: locale == node
- Under the hood, a locale is always a process
- Users can configure their applications to use:
    - a locale per socket
    - a locale per GPU
    - a locale per NUMA domain …

- GPUs are represented as "sublocales"

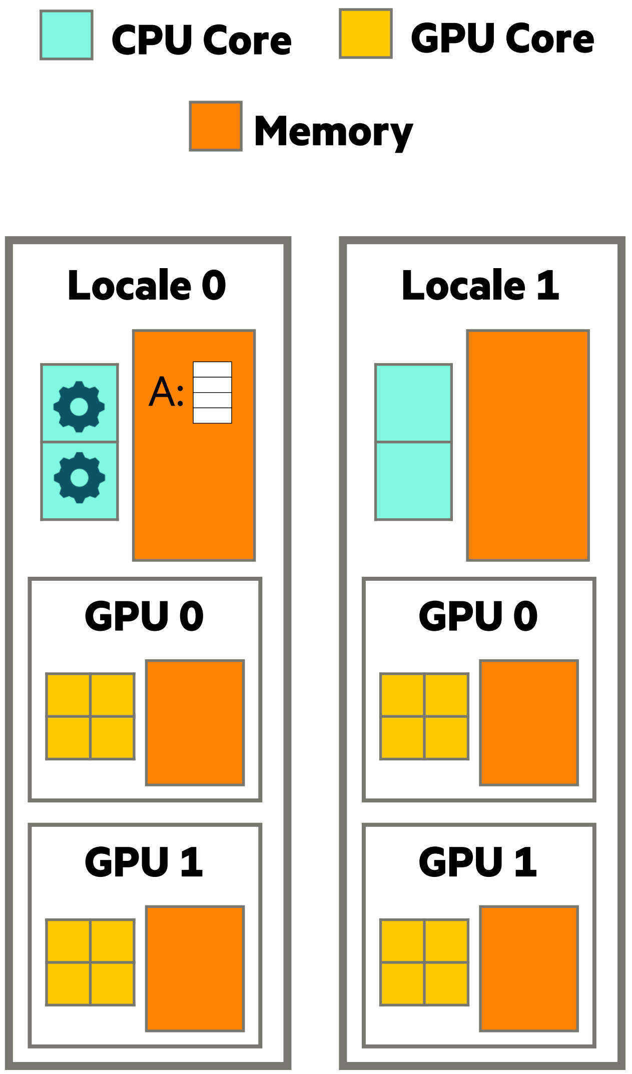
Predefined Variables for Locality

`here`
The current execution locale

`Locales`
An array storing all locales

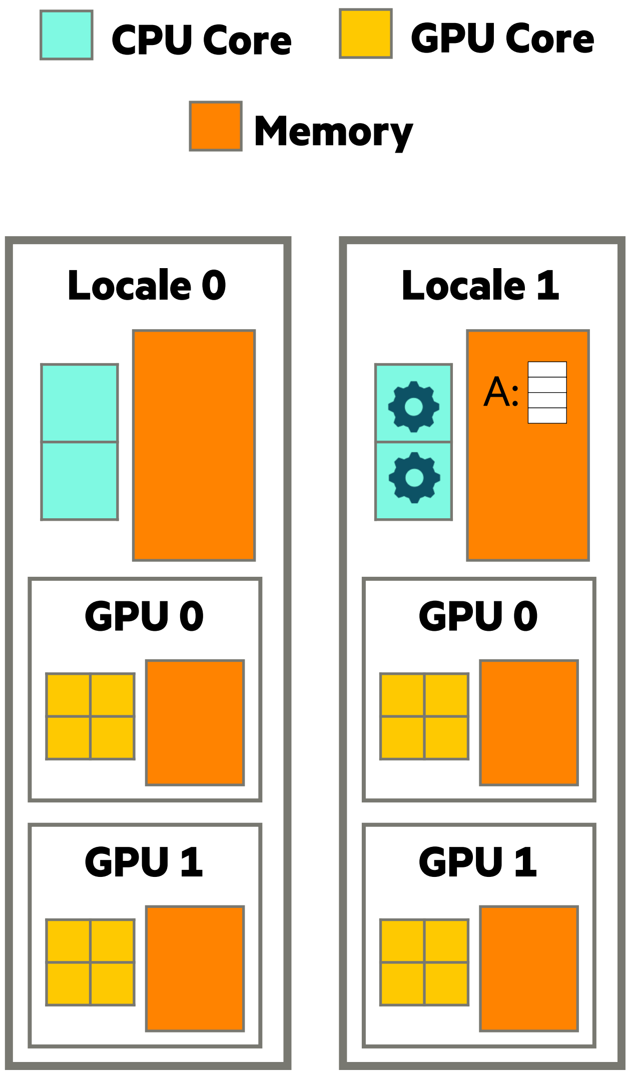
`locale.gpus`
An array storing GPU sublocales

Programming with Chapel: Basic Data Parallelism + Locality




```
var A: [1..10] int;

forall elem in A do
  elem += 1;
```

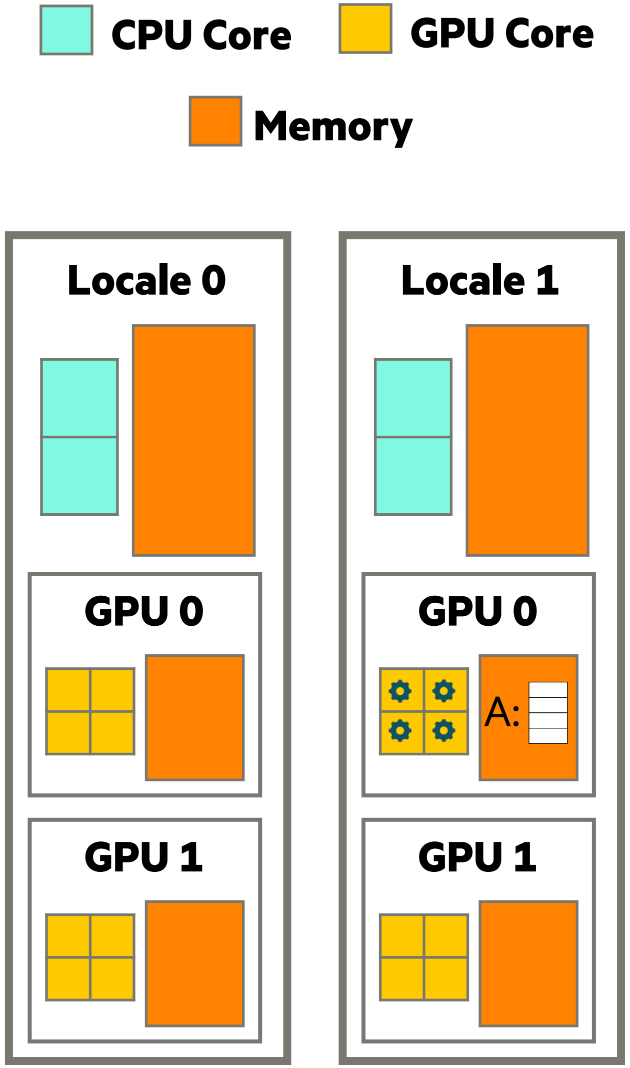
Programming with Chapel: Basic Data Parallelism + Locality




```
on Locales[1] {
  var A: [1..10] int;


  forall elem in A do
    elem += 1;
}
```

The 'on' statement moves the execution to a remote locale

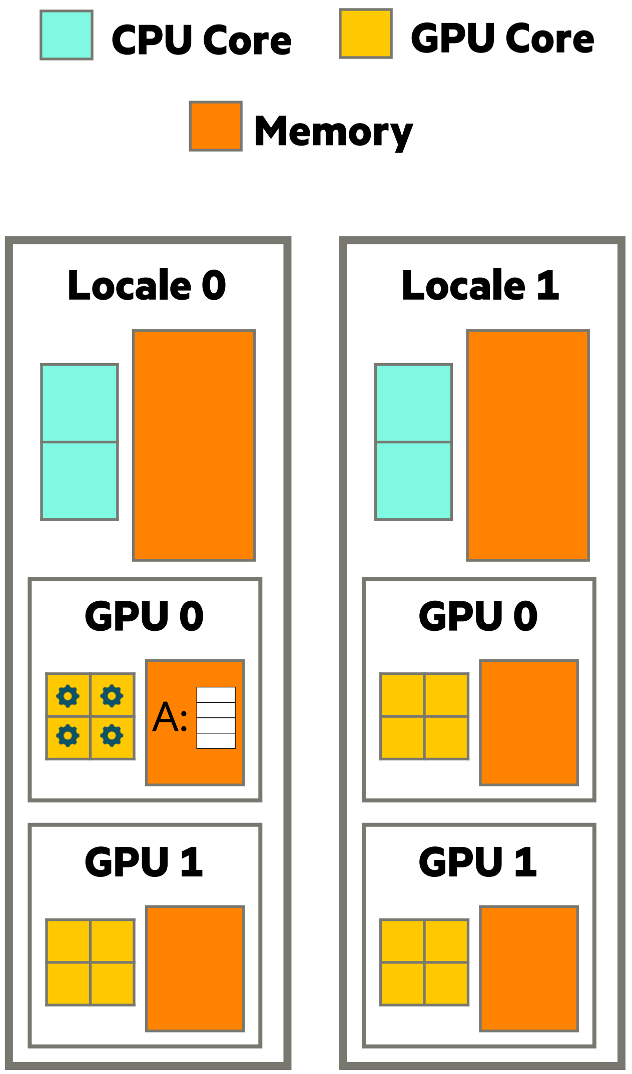
Programming with Chapel: Basic Data Parallelism + Locality




```
on Locales[1].gpus[0] {
  var A: [1..10] int;

  forall elem in A do
    elem += 1;
}
```

Each locale object has a 'gpus' array that store GPU sublocales

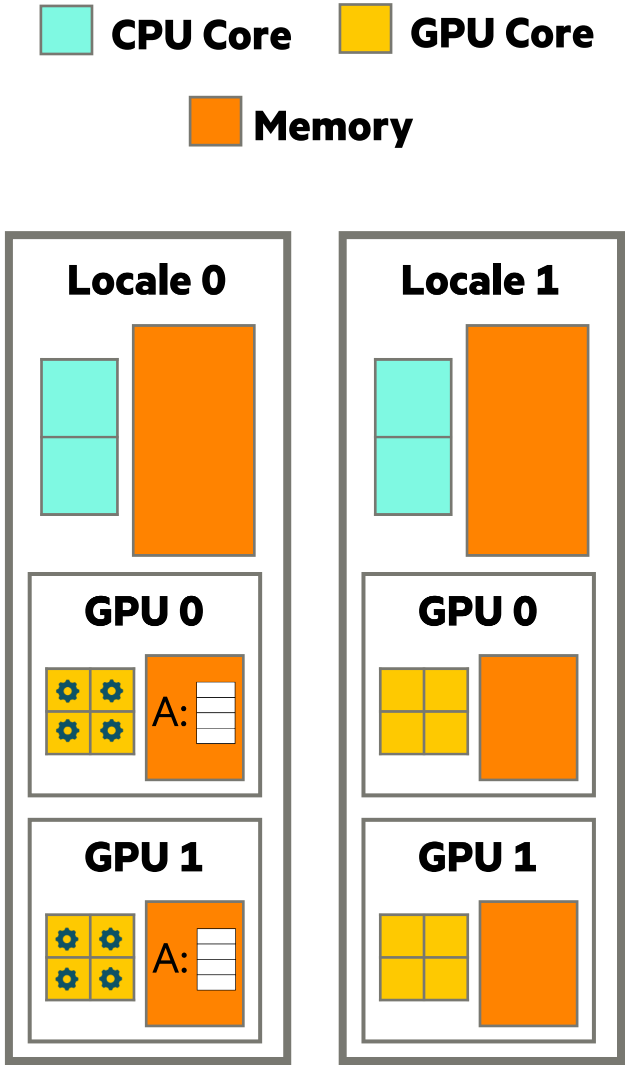
Programming with Chapel: Basic Data Parallelism + Locality



'here' is a built-in representing the current execution locale

```
on here.gpus[0] {
  var A: [1..10] int;


  forall elem in A do
    elem += 1;
}
```

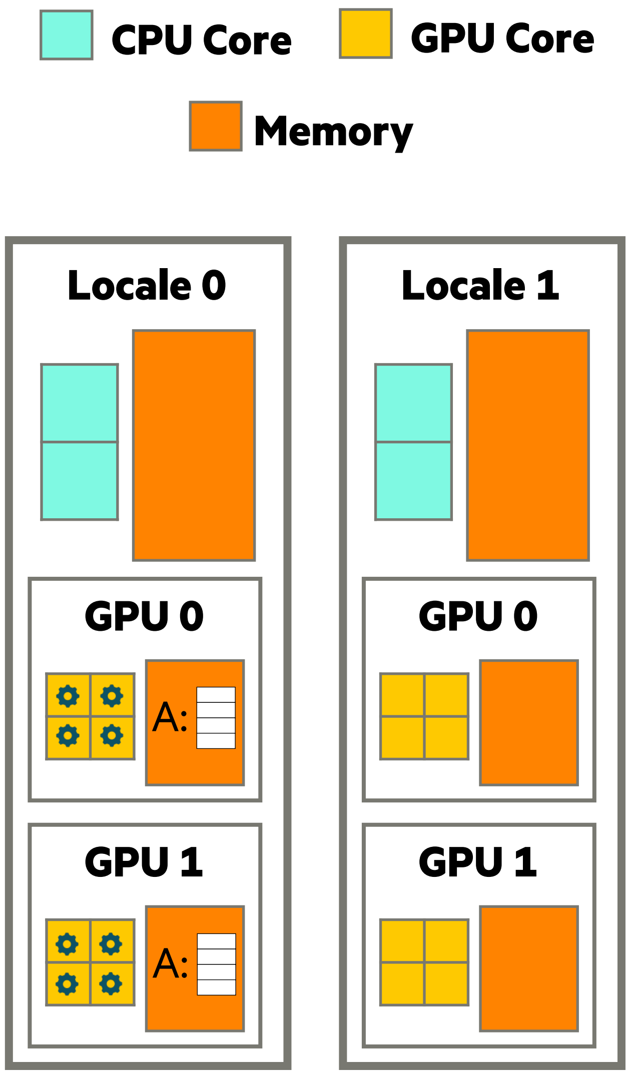
Programming with Chapel: Data + Task Par. + Locality



'coforall' loops run each iteration in a parallel task

```
coforall gpu in here.gpus {
  on gpu {
    var A: [1..10] int;


    forall elem in A do
      elem += 1;
  }
}
```

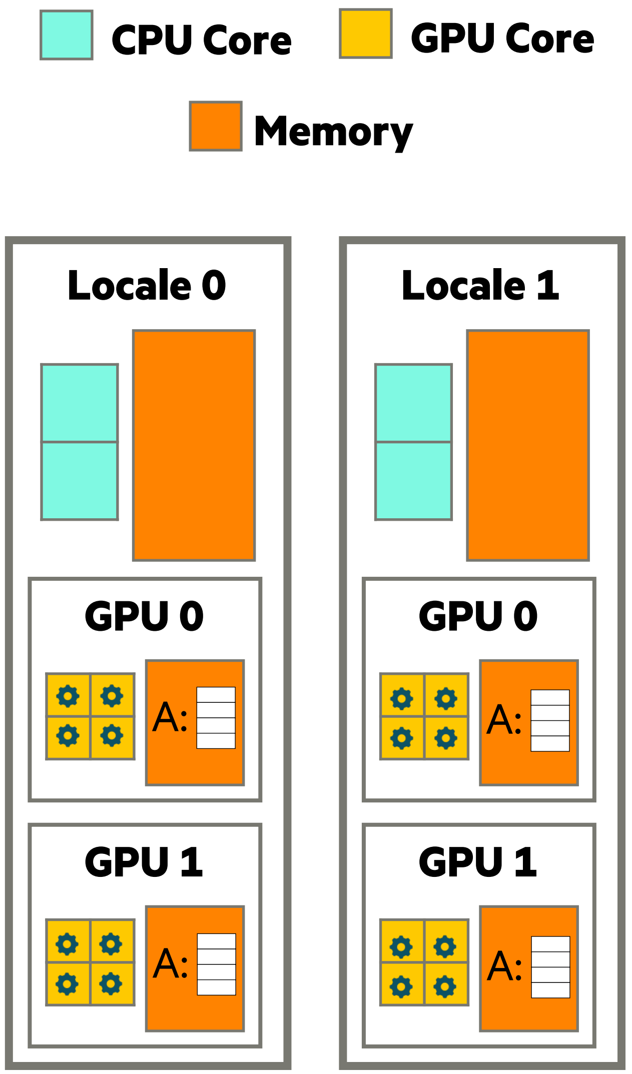
Programming with Chapel: Data + Task Par. + Locality



'do' can be used instead of curly braces for single-statement blocks (this is just a matter of style)

```
coforall gpu in here.gpus do on gpu {
  var A: [1..10] int;


  forall elem in A do
    elem += 1;
}
```

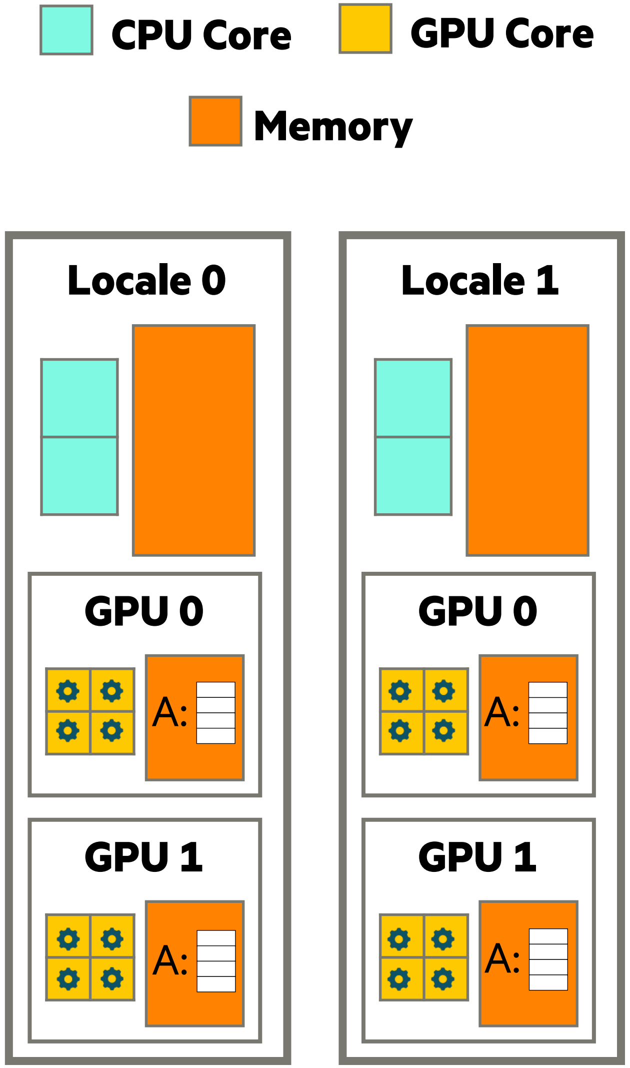
Programming with Chapel: Data + Task Par. + Locality



A very similar 'coforall' + 'on' for multilocale paralelleism

```
coforall loc in Locales do on loc {
  coforall gpu in here.gpus do on gpu {
    var A: [1..10] int;


    forall elem in A do
      elem += 1;
  }
}
```

Programming with Chapel: Data + Task Par. + Locality + Data Movement




```
var CpuArr: [1..10] int;
coforall loc in Locales do on loc {
  coforall gpu in here.gpus do on gpu {
    const myChunk = start..end; // math omitted
    var A = CpuArr[myChunk];


    forall elem in A do
      elem += 1;
  }
}
```

Arrays or slices can be copied across any locales, including GPU sublocale

Programming with Chapel: Honorable Mentions

'begin' statement

- Starts an asynchronous task

'cobegin' statement

- Every statement in the block becomes a parallel task

'sync' statement

- Synchronizes all asynchronous tasks at the end of the block

'atomic' variables

- All operations on the variable is performed atomically, potentially across the network

'sync' variables

- Atomic variables with empty/full semantics: e.g. it can't be read twice in a row

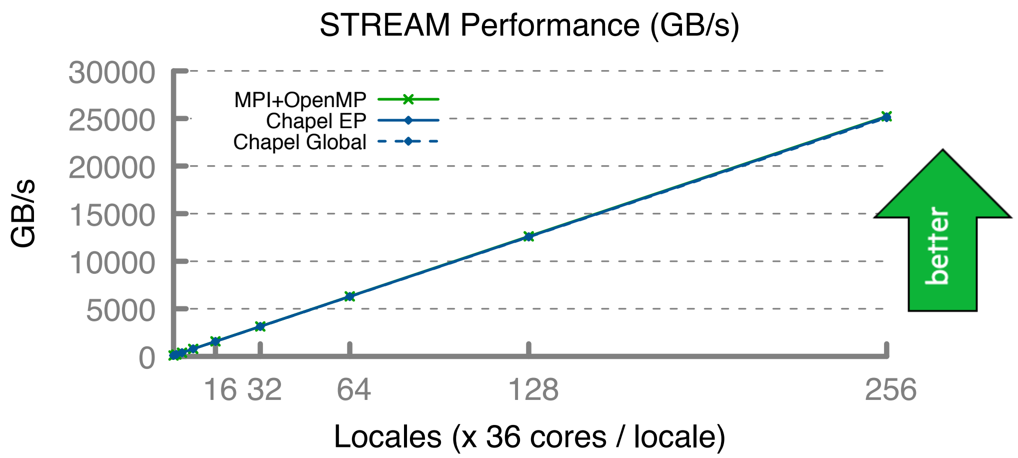
How does Chapel perform?

HPCC Stream Triad and RA in C + MPI + OpenMP vs. Chapel

STREAM TRIAD: C + MPI + OPENMP

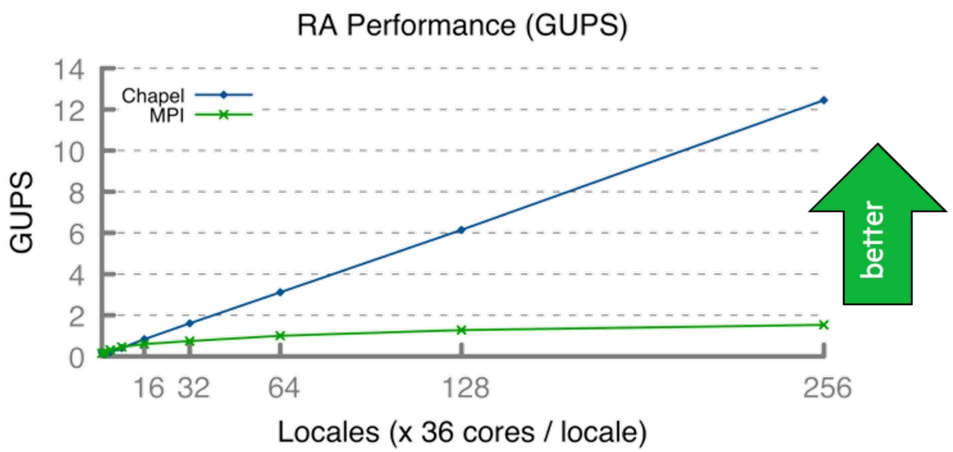
```c
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

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

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

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

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

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

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );
```

```chapel
use BlockDist;

config const n = 1_000_000,
             alpha = 0.01;
const Dom = blockDist.createDomain({1..n});
var A, B, C: [Dom] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```



HPCC RA: MPI KERNEL

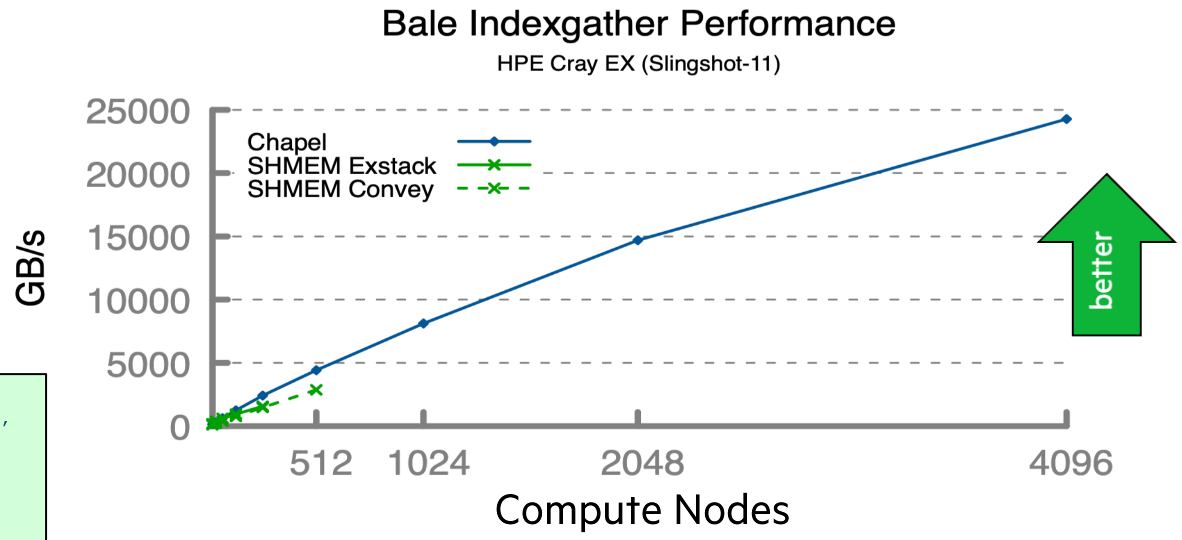
```chapel
…
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask].xor(r);
…
```



Bale IG 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];
```



SHMEM (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();
}
```

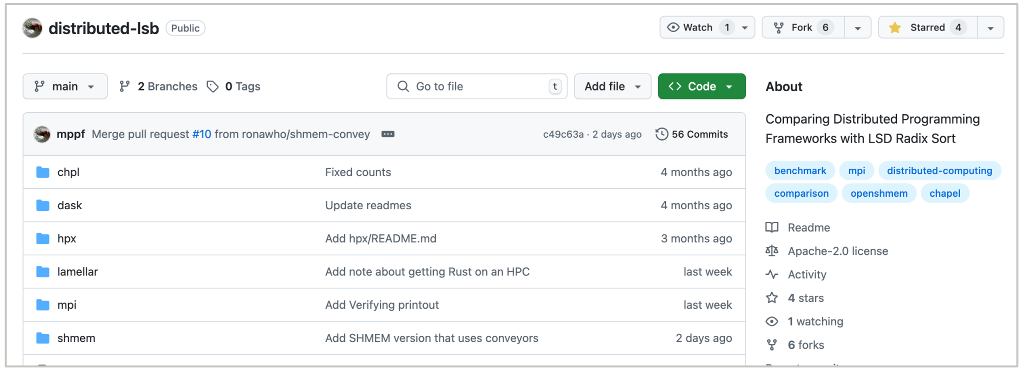
SHMEM (Conveyors version)

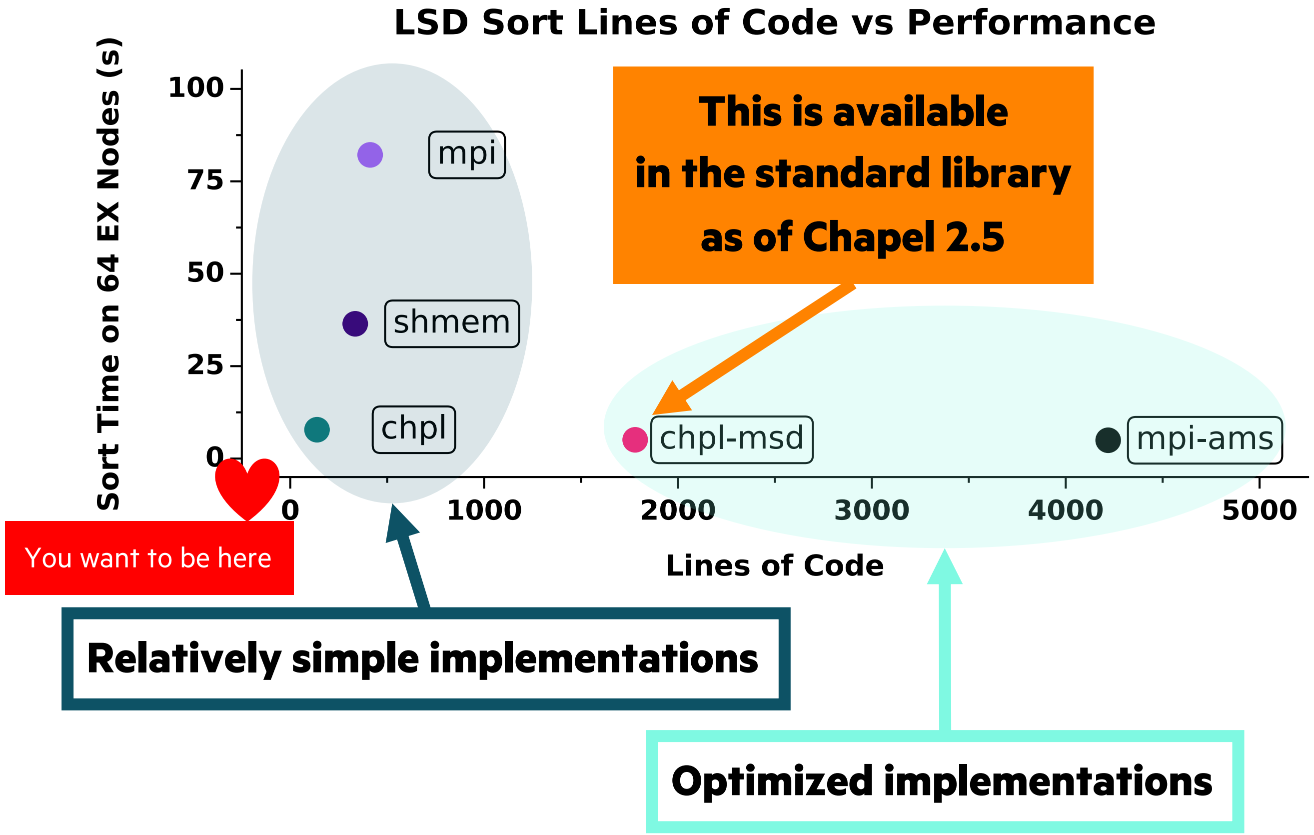
```
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
       more | convey_advance(replies, !more)) {

  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;
}
```

Distributed Sorting Performance and Productivity Survey

- Using distributed sort as proxy to compare
  - Chapel, MPI, SHMEM and others

- All implementations are available
  - github.com/mppf/distributed-lsb





Further Reading on Sorting Capabilities of Standard Libraries

- Chapel's standard library leverages parallelism out-of-the-box

Read more on Chapel blog

chapel-lang.org/blog/posts/std-sort-performance/

Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts

Comparing Standard Library Sorts: The Impact of Parallelism

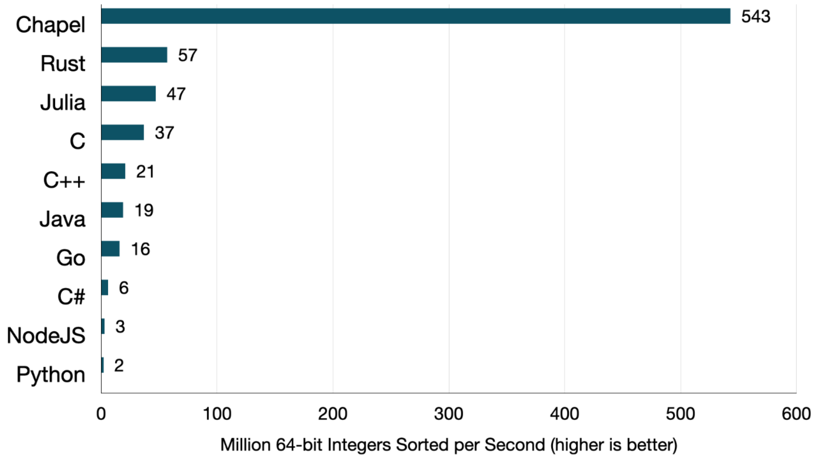
Posted on January 30, 2024.

Tags: Sorting Performance Language Comparison

By: Michael Ferguson

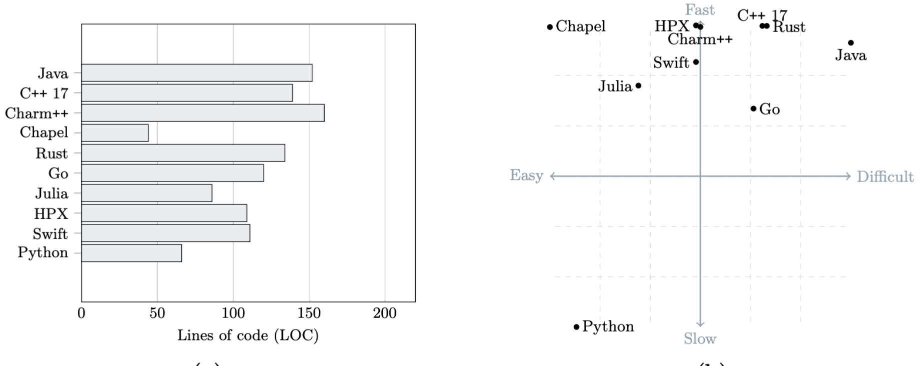
Computing hardware is parallel. Everything from the Raspberry Pi to a supercomputer uses parallelism. The Chapel language and standard library make it easy to use that parallel hardware effectively.

The Chapel standard library `sort` routine is at least **10 times** faster than any other standard library `sort` measured in this benchmarking experiment. The reason: Chapel's standard library `sort` routine is parallel but the others are not. Chapel is designed for parallel computing and its standard library is built to leverage that parallelism.



Sorting 1 GiB of random 64-bit ints on a PC

Other Comparisons



Diehl et al. "Benchmarking the Parallel 1D Heat Equation Solver in Chapel, Charm++, C++, HPX, Go, Julia, Python, Rust, Swift, and Java"

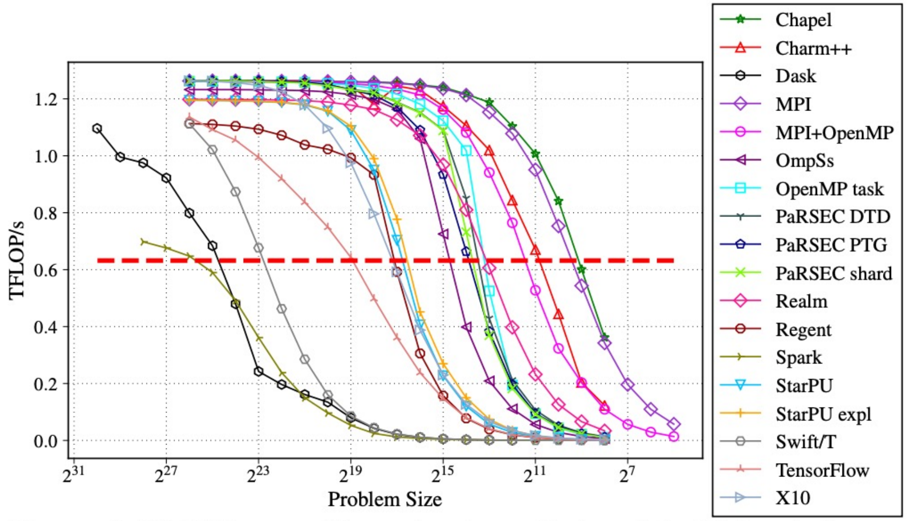


Figure 6: FLOPS vs problem size (stencil, 1 node). Higher is better.

Slaughter et al. "Task Bench: A Parameterized Benchmark for Evaluating Parallel Runtime Performance"

	OpenMP	Kokkos	CUDA	HIP	Chapel
Skylake	457	444	-	-	488
Cascade Lake	2491	2538	-	-	1571
Sapphire Rapids	6710	7009	-	-	3815
Rome	3484	3471	-	-	3140
Milan	2687	2652	-	-	2771
ThunderX2	427	432	-	-	246
POWER9	33	33	-	-	494
P100	1798	2692	2729	-	2725
V100	3857	5360	5772	-	5582
A100	3639	4537	5087	-	5329
MI60	4436	3554	-	3739	3610
MI100	4288	3554	-	3761	3659

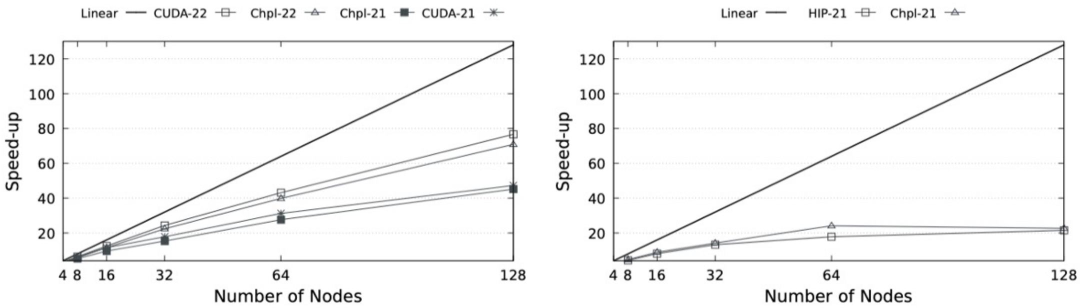
(a) Effective GFLOP/s, higher is better

	OpenMP	Kokkos	CUDA	HIP	Chapel
Skylake	24.1%	23.5%	-	-	25.7%
Cascade Lake	40.5%	41.3%	-	-	25.6%
Sapphire Rapids	53.1%	55.4%	-	-	30.2%
Rome	56.7%	56.5%	-	-	51.1%
Milan	72.6%	71.7%	-	-	74.9%
ThunderX2	43.3%	43.8%	-	-	25.0%
POWER9	2.8%	2.8%	-	-	41.9%
P100	37.8%	56.5%	57.3%	-	57.2%
V100	49.2%	68.4%	73.7%	-	71.3%
A100	37.3%	46.5%	52.2%	-	54.7%
MI60	60.2%	48.2%	-	50.7%	49.0%
MI100	37.2%	30.8%	-	32.6%	31.7%

(b) Architectural efficiency, higher is better

Fig. 2: miniBUDE results for small deck bm1

Milthorpe et al. "Performance Portability of the Chapel Language on Heterogeneous Architectures"



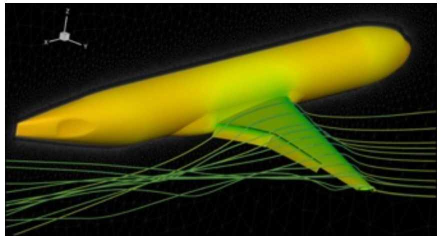
(a) NVIDIA-based System

(b) AMD-based system

Carneiro et al. "Investigating Portability in Chapel for Tree-Based Optimization on GPU-Powered Clusters"

All great… but what about real-world usage?

Productivity Across Diverse Application Scales (code and system size)



Computation: Aircraft simulation / CFD
Code size: 100,000+ lines
Systems: Desktops, HPC systems

7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

Posted on September 17, 2024.

Tags: Computational Fluid Dynamics | User Experiences | Interviews

By: Engin Kayraklioglu, Brad Chamberlain

“Chapel worked as intended: the code maintenance is very much reduced, and its readability is astonishing. This enables undergraduate students to contribute, something almost impossible to think of when using very complex software.”

Computation: Coral reef image analysis
Code size: ~300 lines
Systems: Desktops, HPC systems w/ GPUs

7 Questions for Scott Bachman: Analyzing Coral Reefs with Chapel

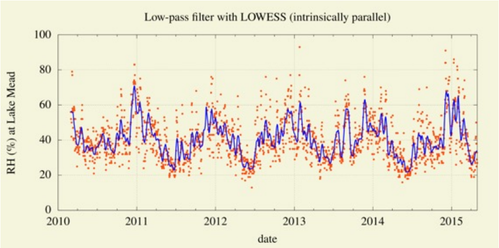
Posted on October 1, 2024.

Tags: Earth Sciences | Image Analysis | GPU Programming | User Experiences | Interviews

By: Brad Chamberlain, Engin Kayraklioglu

In this second installment of our Seven Questions for Chapel Users series, we're looking at a recent success story in which Scott Bachman used Chapel to unlock new scales of biodiversity analysis in coral reefs to study ocean health using satellite image processing. This is work that

“With the coral reef program, I was able to speed it up by a factor of 10,000. Some of that was algorithmic, but Chapel had the features that allowed me to do it.”



Computation: Atmospheric data analysis
Code size: 5000+ lines
Systems: Desktops, sometimes w/ GPUs

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 Luís 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

“Chapel allows me to use the available CPU and GPU power efficiently without low-level programming of data synchronization, managing threads, etc.”

[read this interview series at: https://chapel-lang.org/blog/series/7-questions-for-chapel-users/]

More on CHAMPS & CFD: Previous Talk at NASA Ames is Available

Check out

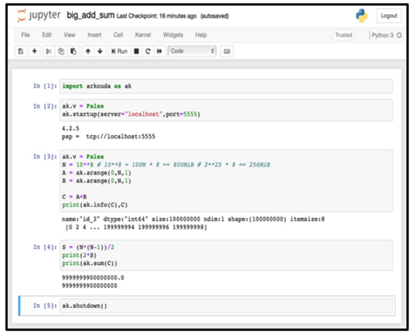
www.nas.nasa.gov/pubs/ams/2025/02-20-25.html

Arkouda

What is Arkouda?

Q: “What is Arkouda?”

Arkouda Client
(written in Python)

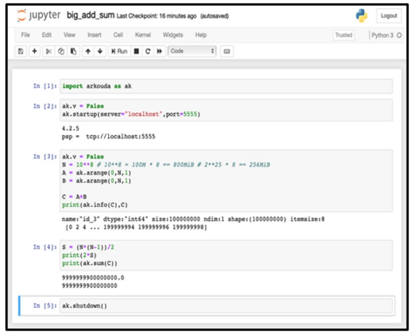


User writes Python code making familiar NumPy/Pandas calls

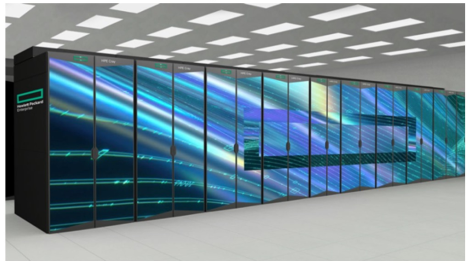
What is Arkouda?

Q: "What is Arkouda?"

Arkouda Client
(written in Python)



Arkouda Server
(written in Chapel)



User writes Python code
making familiar NumPy/Pandas calls

A: "A scalable version of NumPy / Pandas for data scientists"

Performance and Productivity: Sorting with Arkouda

HPE Cray EX

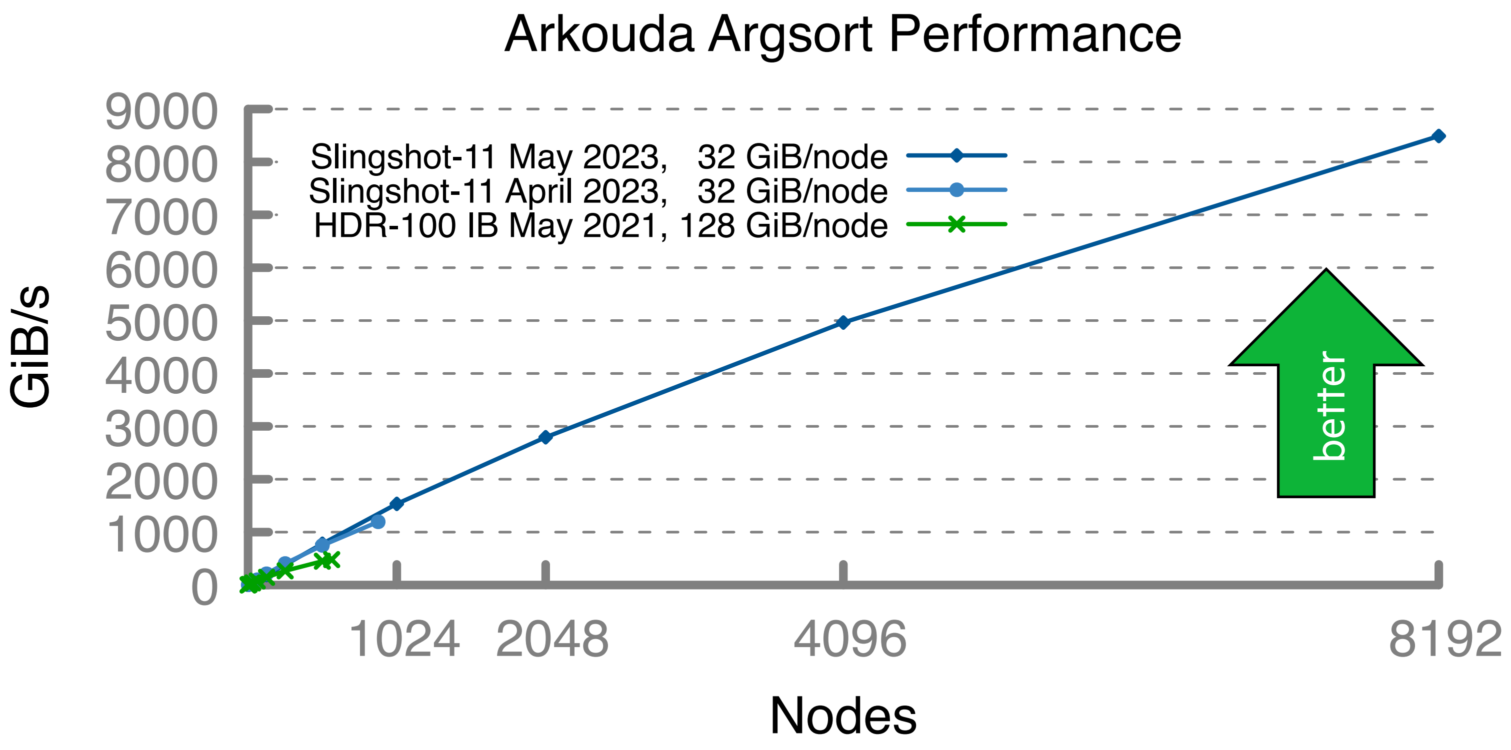
- Slingshot-11 network (200 Gb/s)
- 8192 compute nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)

HPE Cray EX

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

HPE Apollo

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

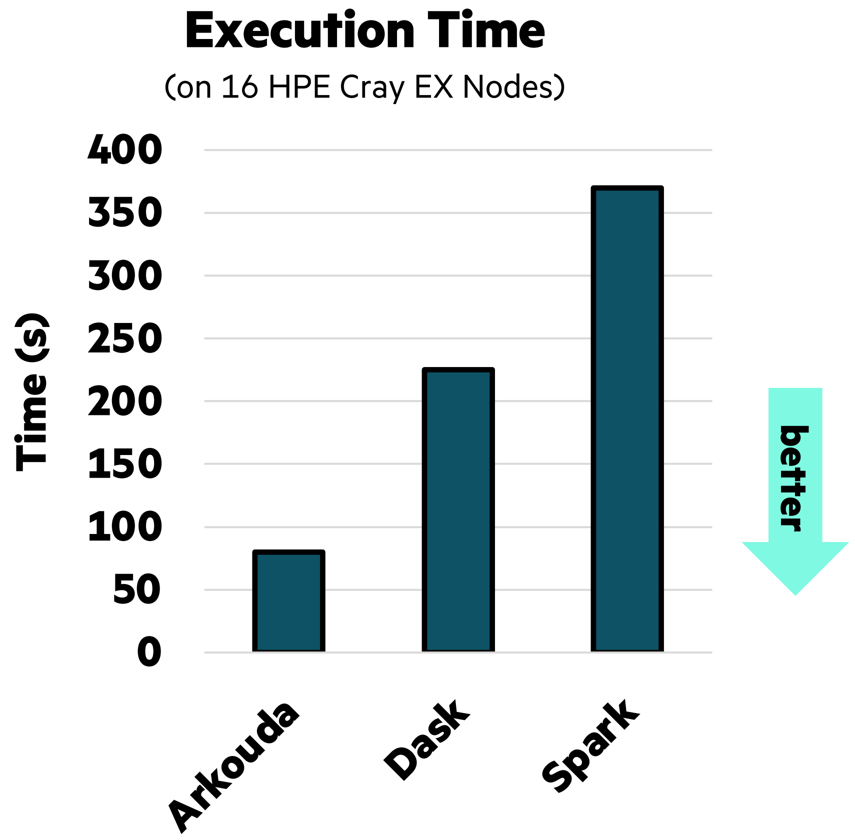


Implemented using ~100 lines of Chapel

Performance and Productivity: Telemetry Analysis with Arkouda

- ~500 GB of server telemetry data
  - Stored in Parquet files
    - Loaded in dataframes
  - Measured time includes:
    - IO
    - Histogram, mean, max, covariance

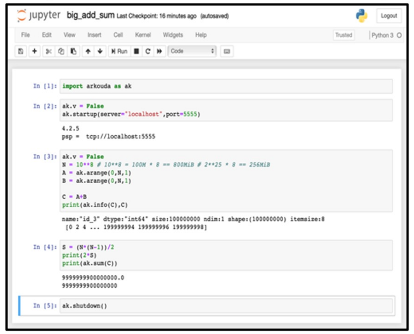
Arkouda performs significantly better than Dask and Spark



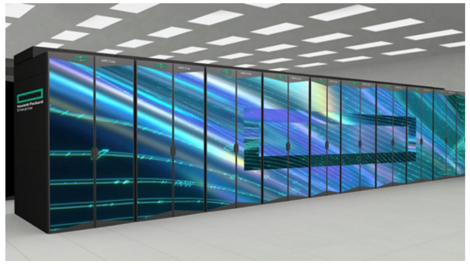
What is Arkouda?

Q: “What is Arkouda?”

Arkouda Client
(written in Python)



Arkouda Server
(written in Chapel)



User writes Python code
making familiar NumPy/Pandas calls

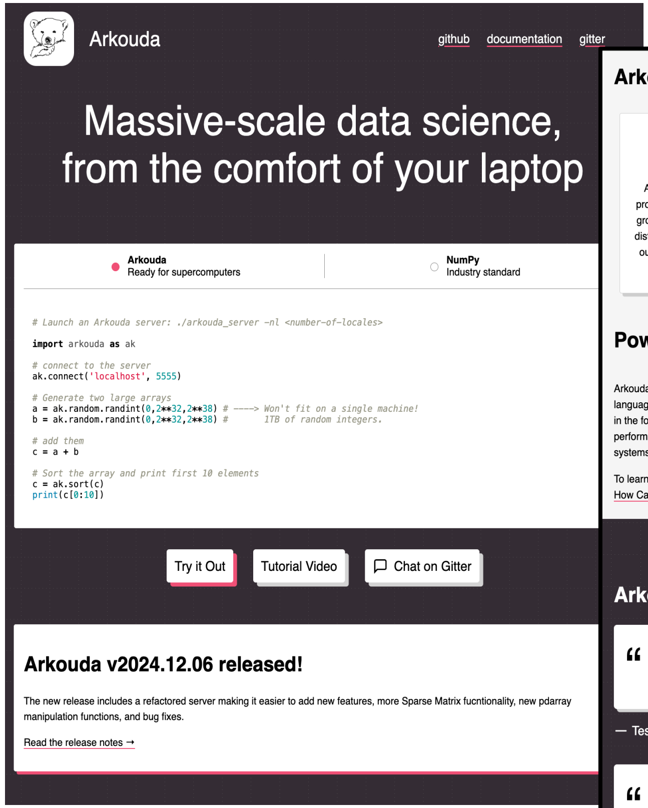
A: “A scalable version of NumPy / Pandas for data scientists”

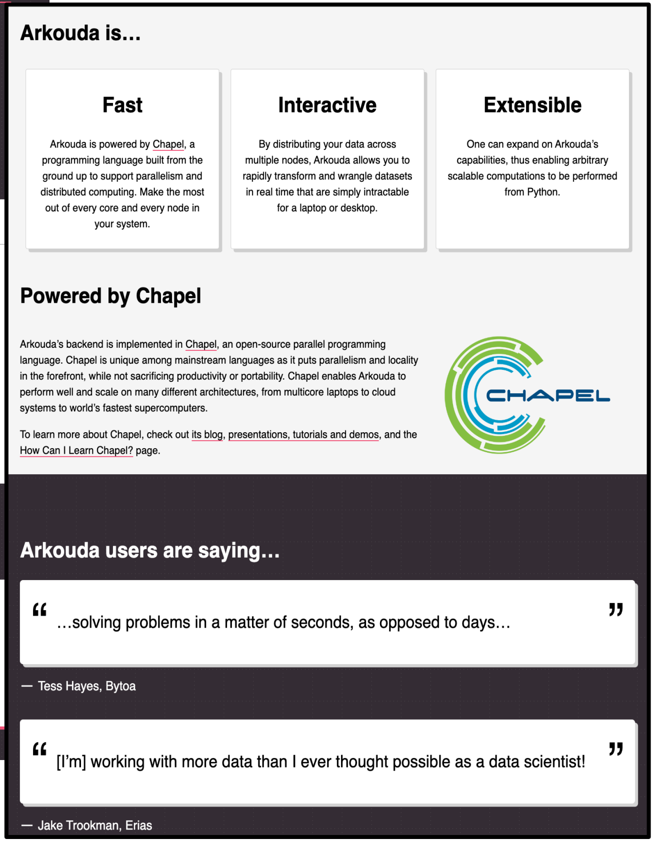
A’: “An extensible framework for arbitrary HPC computations”

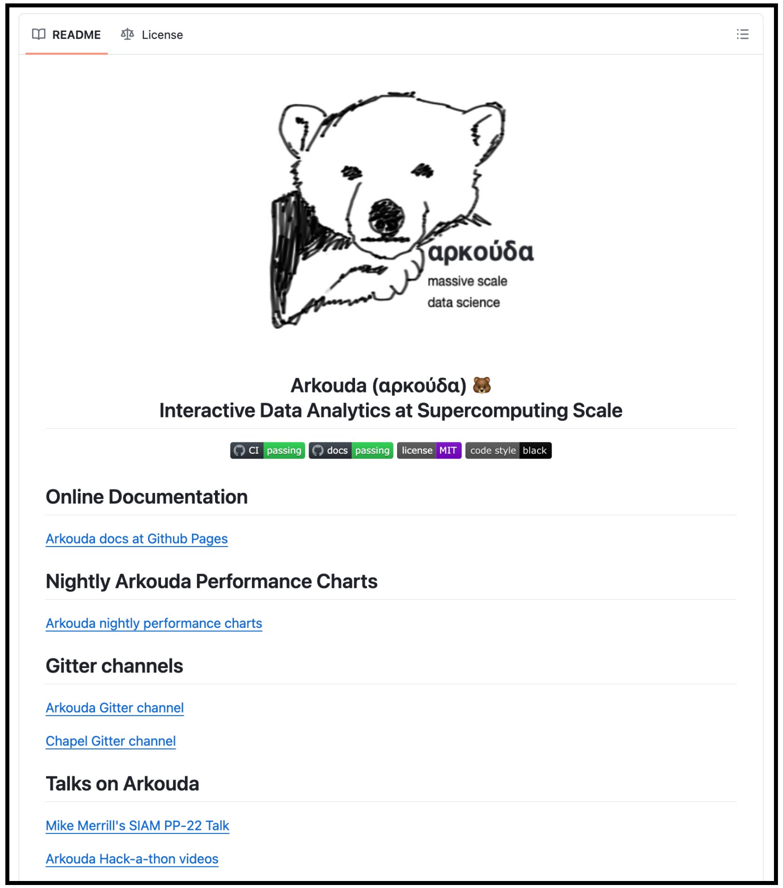
A”: “A way to drive HPC systems interactively from Python on a laptop”

Arkouda Resources

Website: https://arkouda-www.github.io/ **GitHub:** https://github.com/Bears-R-Us/arkouda







Arkouda Interview

Blog: Interview with founding co-developer, Bill Reus: https://chapel-lang.org/blog/posts/7qs-reus/

Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts

7 Questions for Bill Reus: Interactive Supercomputing with Chapel for Cybersecurity

Posted on February 12, 2025.

Tags: User Experiences | Interviews | Data Analysis | Arkouda

By: Engin Kayraklioglu, Brad Chamberlain

Table of Contents

We're very excited to kick off the 2025 edition of our Seven Questions for Chapel Users series with the following interview with Bill Reus. Bill is one of the co-creators of Arkouda, which is one of Chapel's flagship applications. To learn more about Arkouda and its support for interactive data analysis at massive scales, read on!

1. Who are you?

My name is Bill Reus, and I live near Annapolis, MD and the beautiful Chesapeake Bay. I am currently a data scientist doing statistical modeling and simulation for the United States government, but I began my career as an experimental chemist. In graduate school, I measured electron transport through thin films of organic molecules using an apparatus that our group invented to collect large volumes of noisy data quickly and with low cost. This approach contrasted with the typical means of studying molecular electronics, which was to spend weeks or months collecting a small number of exquisite measurements in ultra-high vacuum and at ultra-low temperature.

"I was on the verge of resigning myself to learning MPI when I first encountered Chapel. After writing my first Chapel program, I knew I had found something much more appealing."

…

"Chapel's separation of concerns immediately felt like the most natural way to think about large-scale computing. I would highly encourage anyone wanting to get into HPC programming to start with Chapel."

How do I learn more about Chapel?

Ways to Engage with the Chapel Community

“Live” Virtual Events

- ChapelCon (formerly CHIUW), annually
- Project Meetings, weekly
- Deep Dive / Demo Sessions, weekly timeslot

Community / User Forums

- Discord
- Discourse
- Email Contact Alias `chapel+qs@discoursemail.com`
- GitHub Issues
- Gitter
- Reddit
- Stack Overflow

Electronic Communications

- Chapel Blog, ~biweekly
- Community Newsletter, quarterly
- Announcement Emails, around big events

Social Media

- Bluesky
- Facebook
- LinkedIn
- Mastodon
- X / Twitter
- YouTube

Chapel has been accepted to HPSF

Timeline:

- **May 2024:** HPSF launched at ISC
- **September 2024:** Began accepting applications for member projects
- **January 2025:** Chapel accepted to HPSF at the “established” project level
- **May 2025:** First-ever HPSFcon

Resources:

- **Website:** https://hpsf.io/
- **Blog:** https://hpsf.io/blog/
- **YouTube channel:** https://www.youtube.com/@HPSF-community
- **GitHub org:** https://github.com/hpsfoundation

HPSF
HIGH PERFORMANCE
SOFTWARE FOUNDATION

Closing Remarks

Chapel allows programmers to leverage most common parallel hardware

- Multicore, multinode, including cloud resources
- NVIDIA and AMD GPUs are supported with vendor-neutral code

Same set of programming abstractions are used to achieve this portability

- No need to add things on, Chapel comes batteries-included
- No need to paradigm-shift when going from a single node to scaling on a supercomputer

Chapel is being used in many different fields, and in a wide range of institutions

- Some application fields are CFD, data analytics, graph processing, ecological research, astrophysics
- Have been used by academia, industry, and government
- From desktops to supercomputers

Thank you