

Arkouda: Terascale Data Science at Interactive Rates

Ben Albrecht, Mike Merrill,
Bill Reus, Brad Chamberlain

SciPy 2020

✉ chapel_info@cray.com
🌐 chapel-lang.org
🐦 [@ChapelLanguage](https://twitter.com/ChapelLanguage)

CRAY[®]
a Hewlett Packard Enterprise company



Motivation for Arkouda

Motivation: Say you've got...

...a bunch of Python programmers

...HPC-scale problems to solve

...access to HPC systems

<https://www.cscs.ch/computers/piz-daint/>



How will you leverage your Python programmers to get your work done?

Arkouda Design

Python3 Client

A screenshot of a Jupyter Notebook interface. The code cell In [1] imports the arkouda module. In [2] starts a server on localhost port 5555. In [3] creates arrays A and B of size 100W x 8, performs element-wise multiplication C = A*B, and prints array info. In [4] prints the sum of C, showing the result as 9999999900000000.0. In [5] shuts down the kernel.

```
In [1]: import arkouda as ak
In [2]: ak.v = False
ak.start(server="localhost",port=5555)
4.2.5
psp = topi/localhost:5555
In [3]: ak.v = False
k = 10**8 # 10**8 == 100W * 8 == 800MB # 2**25 * 8 == 256MB
A = ak.arange(0,k,1)
B = ak.arange(0,k,1)
C = A*B
print(ak.info(C),C)
name:id:3 dtype:int64 size:100000000 ndim:1 shape:(100000000) itemsize:8
[0 2 4 ... 199999994 199999996 199999998]
In [4]: S = (k*(k-1))/2
print(S.sum(C))
9999999900000000.0
9999999900000000
In [5]: ak.shutdown()
```



ZMQ
Socket

Code Modules

Chapel Server

Dispatcher

Indexing

Arithmetic

Sorting

Generation

I/O

...

Distributed
Object Store

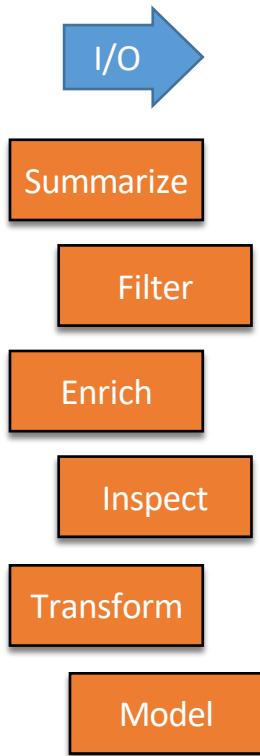
Platform

Meta

Distributed Array

MPP, SMP, Cluster, Laptop, etc.

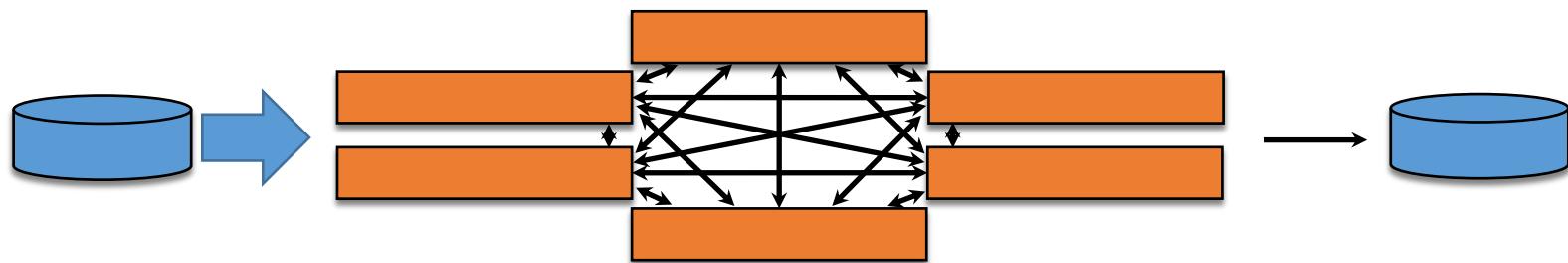
Data Science on 50 Billion Records



Operation	Example	Approx. Time (seconds)
Read from disk	<code>A = ak.read_hdf()</code>	30-60
Scalar Reduction	<code>A.sum()</code>	< 1
Histogram	<code>ak.histogram(A)</code>	< 1
Vector Ops	<code>A + B, A == B, A & B</code>	< 1
Logical Indexing	<code>A[B == val]</code>	1 - 10
Set Membership	<code>ak.in1d(A, set)</code>	1
Gather	<code>B = Table[A]</code>	4 - 120
Get Item	<code>print(A[42])</code>	< 1
Sort Indices by Value	<code>I = ak.argsort(A)</code>	15
Group by Key	<code>G = ak.GroupBy(A)</code>	30
Aggregate per Key	<code>G.aggregate(B, 'sum')</code>	10

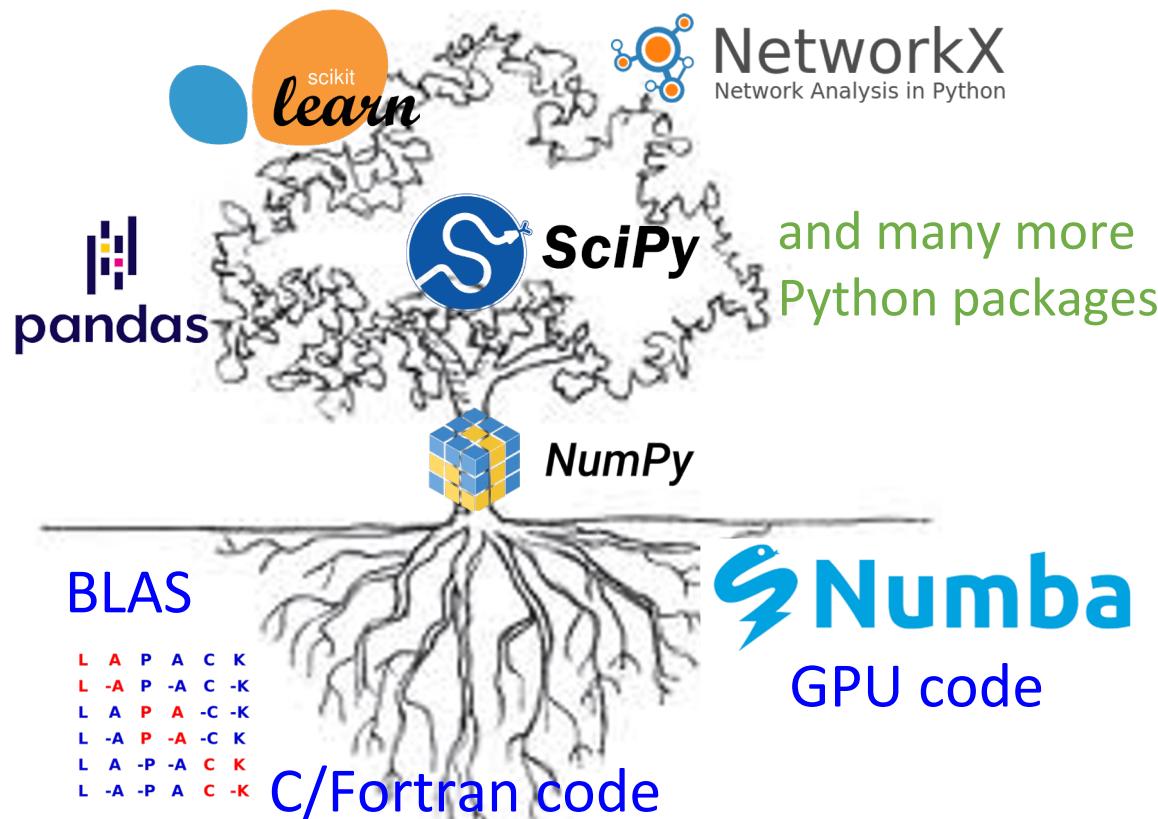
- A, B are 50 billion-element arrays of 32-bit values
- Timings measured on real data
- Hardware: Cray XC40
 - 96 nodes
 - 3072 cores
 - 24 TB
 - Lustre filesystem

Data Science Demands Interactivity

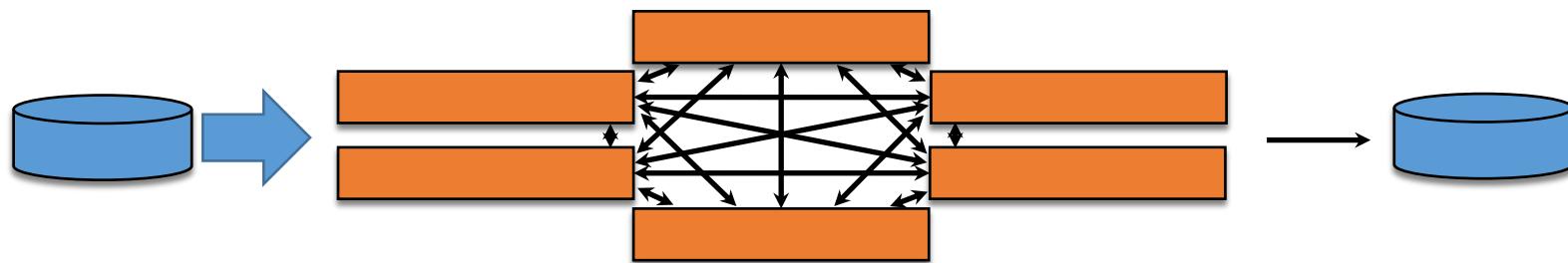


- Productivity with just enough performance
 - No compilation
 - No intermediate I/O
 - No writing boilerplate code
 - *Fast enough* to stay within thought loop
- Interactive Python on a large server satisfies these criteria for datasets up to 10-100 GB

Python Is Not Really Python



Data Science Demands Scaling



- Must use the whole dataset
 - Unbiased sampling of large datasets is difficult
 - Even unbiased sampling eliminates rare and high-order effects
 - Physics of most datasets are global, not local
- Datasets have outgrown (normal) computers
 - Server memory: ~ 1 TB
 - Many datasets > 10 TB

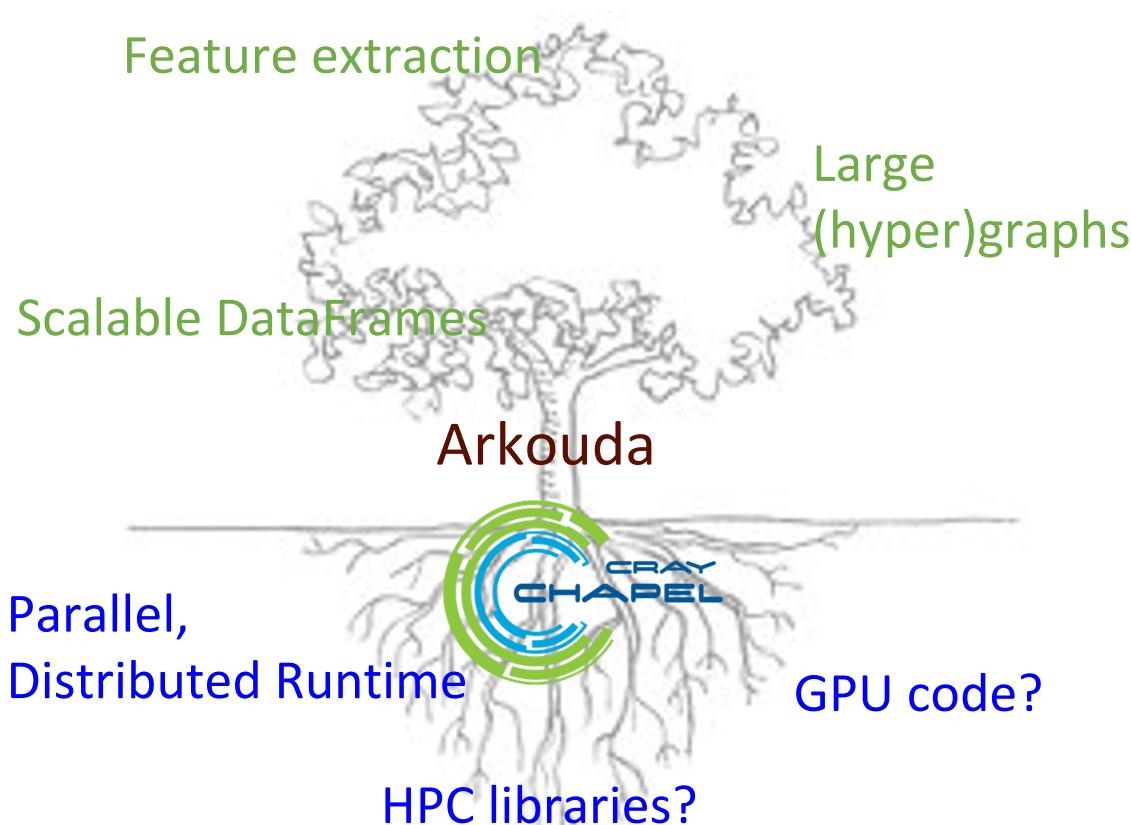
Arkouda

Load Terabytes of data...
... into a familiar, interactive UI ...
... where standard data science operations ...
... execute within the human thought loop ...
... and interoperate with optimized libraries.

Arkouda: an HPC shell for data science

- Chapel backend (server)
- Jupyter/Python frontend (client)
- NumPy-like API

Arkouda: NumPy for HPC



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



Why Chapel?

- From the lead Arkouda developers:
 - High-level language with C-comparable performance
 - Parallelism is a first-class citizen
 - Portable code: from laptop up to supercomputer
 - Integrates with (distributed) numerical libraries (e.g., FFTW, FFTW-MPI)
 - Close to “Pythonic” (for a statically typed language)
 - Lowers the barrier for users to modify the backend implementation

```
var D = {1..1000, 1..1000} dmapped Block(...),  
       A: [D] real;  
  
forall (i,j) in D do  
  A[i,j] = i + (j - 0.5)/1000;
```



Arkouda Design

Python3 Client

A screenshot of a Jupyter Notebook interface. The code cell In [1] imports `arkouda` as `ak`. Cell In [2] starts a server on localhost port 5555. Cells In [3] and In [4] demonstrate matrix multiplication and summation of large arrays (1000x1000). Cell In [5] shuts down the client. The interface includes a toolbar with file operations, a code editor, and a help menu.

```
In [1]: import arkouda as ak
In [2]: ak.v = False
ak.start(server="localhost",port=5555)
4.2.5
psp = top/localhost:5555
In [3]: ak.v = False
n = 10**3 # 10**3 * 10**3 * 8 == 800MB * 8 == 2560MB
A = ak.arange(0,N,1)
B = ak.arange(0,N,1)
C = A*B
print(ak.info(C),C)
name:id:3 dtype:int64 size:1000000000 ndim:1 shape:(100000000) itemsize:8
[0 2 4 ... 199999994 199999996 199999998]
In [4]: S = (N*(N-1))/2
print(S.sum(C))
9999999900000000.0
9999999900000000
In [5]: ak.shutdown()
```



ZMQ
Socket

Code Modules

Chapel Server

Dispatcher

Indexing

Arithmetic

Sorting

Generation

I/O

...

Distributed
Object Store

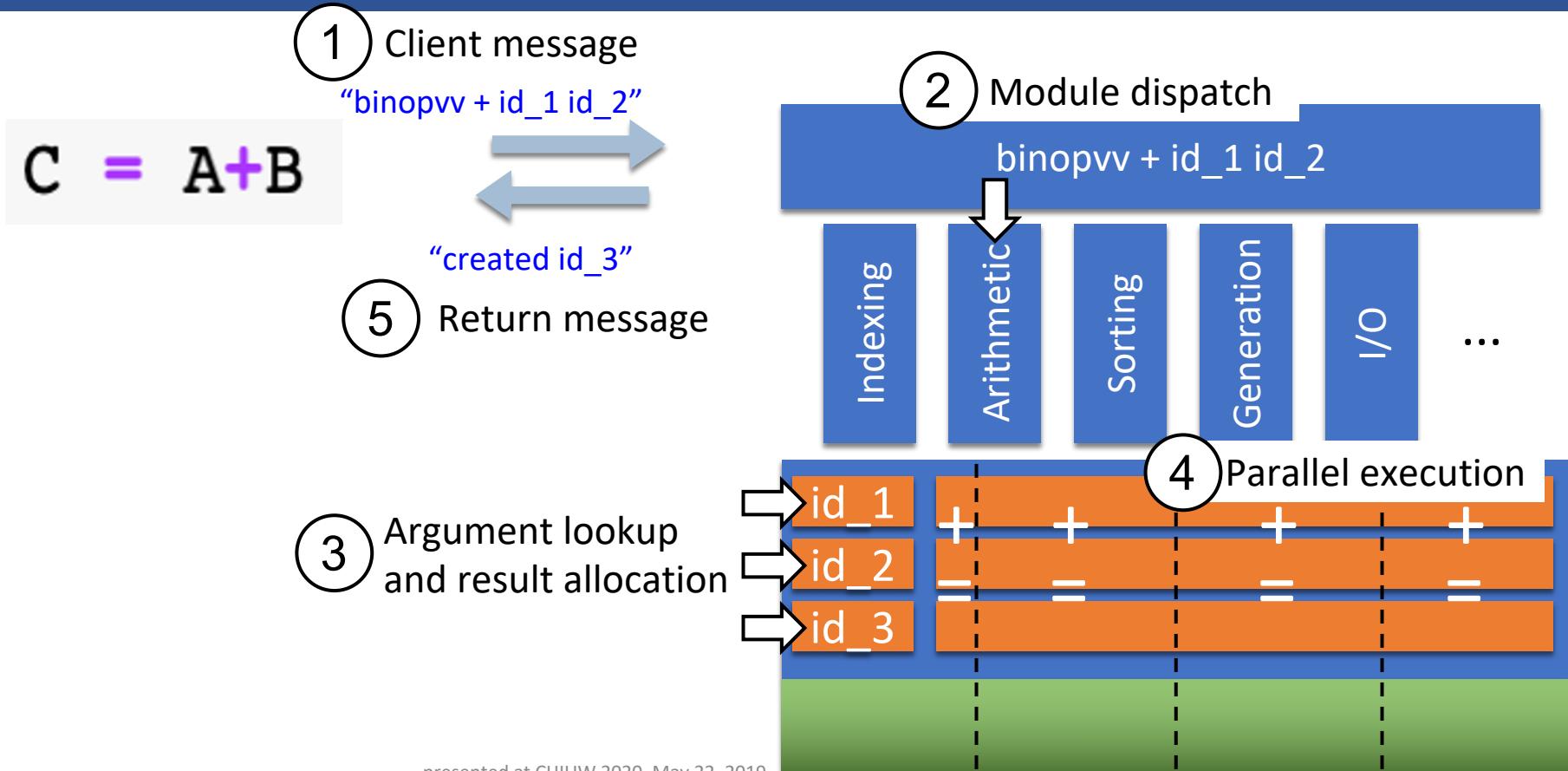
Platform

Meta

Distributed Array

MPP, SMP, Cluster, Laptop, etc.

A Chapel Interpreter



Using Arkouda: Startup

1) Initialize server in terminal

```
> arkouda_server -nl 96
```

```
server listening on hostname:port
```

2) Connect to server in Jupyter

```
import arkouda as ak  
ak.connect(hostname, port)
```

4.2.5

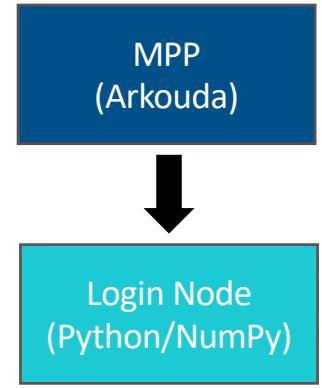
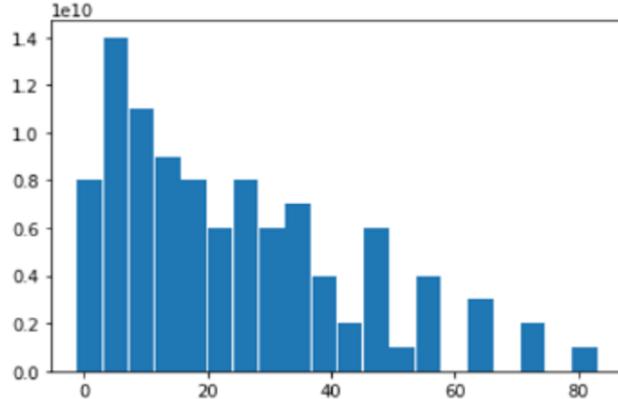
```
psp = tcp://nid00104:5555  
connected to tcp://nid00104:5555
```



Using Arkouda: Workflow

```
In [9]: A = ak.randint(0, 10, 10**11)
B = ak.randint(0, 10, 10**11)
C = A * B
hist = ak.histogram(C, 20)
Cmax = C.max()
Cmin = C.min()
executed in 3.96s, finished 13:45:28 2019-09-12
```

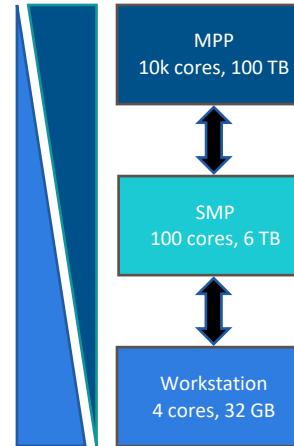
```
In [10]: bins = np.linspace(Cmin, Cmax, 20)
         = plt.bar(bins, hist.to_ndarray(), width=(Cmax-Cmin)/20)
executed in 193ms, finished 13:45:28 2019-09-12
```



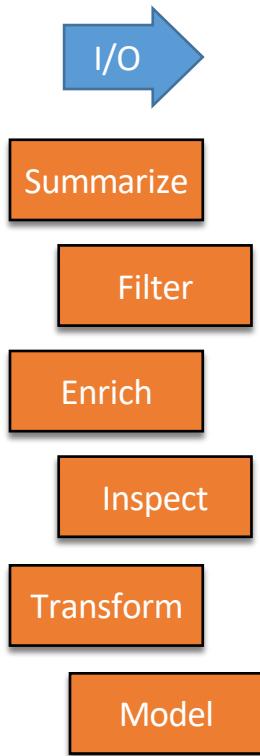
Arkouda Accomplishments

By taking this approach, these users were able to:

- interact with a running Chapel program from Python within Jupyter
- run the same back-end program on...
 - ...a Mac laptop
 - ...an Infiniband cluster
 - ...an HPE Superdome X
 - ...a Cray XC
- compute on TB-sized arrays in seconds
- with 1-2 person-months of effort



Data Science on 50 Billion Records



Operation	Example	Approx. Time (seconds)
Read from disk	<code>A = ak.read_hdf()</code>	30-60
Scalar Reduction	<code>A.sum()</code>	< 1
Histogram	<code>ak.histogram(A)</code>	< 1
Vector Ops	<code>A + B, A == B, A & B</code>	< 1
Logical Indexing	<code>A[B == val]</code>	1 - 10
Set Membership	<code>ak.in1d(A, set)</code>	1
Gather	<code>B = Table[A]</code>	4 - 120
Get Item	<code>print(A[42])</code>	< 1
Sort Indices by Value	<code>I = ak.argsort(A)</code>	15
Group by Key	<code>G = ak.GroupBy(A)</code>	30
Aggregate per Key	<code>G.aggregate(B, 'sum')</code>	10

- A, B are 50 billion-element arrays of 32-bit values
- Timings measured on real data
- Hardware: Cray XC40
 - 96 nodes
 - 3072 cores
 - 24 TB
 - Lustre filesystem

NumPy vs. Arkouda Performance

CRAY
a Hewlett Packard Enterprise company

Performance normalized to NumPy

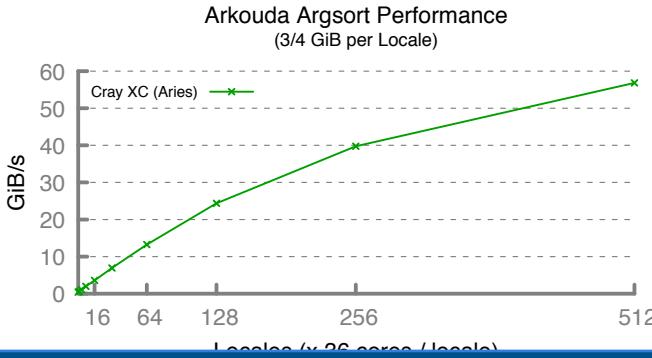
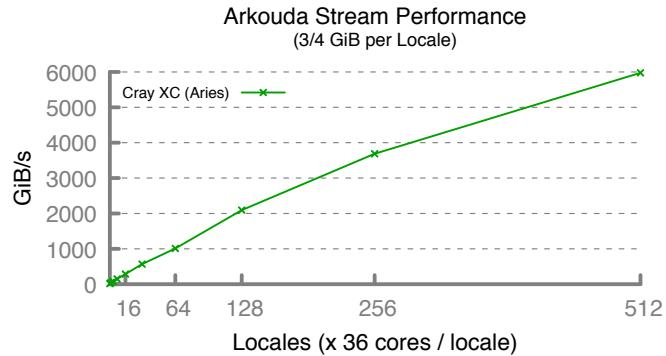
Cray XC (Aries network)
36-core (72 HT), 128 GB RAM
dual 18-core “Broadwell”, 2.1 GHz

benchmark	NumPy	Arkouda (serial)	Arkouda (1-node/36-core)	Arkouda (512-node)
argsort	1.0	2.0	16.7	1837.3
coargsort	1.0	2.3	16.7	984.7
gather	1.0	0.4	11.7	469.1
reduce	1.0	1.2	12.0	4412.4
scan	1.0	0.8	3.2	266.6
scatter	1.0	1.0	11.8	781.8
stream	1.0	0.7	6.2	1590.4

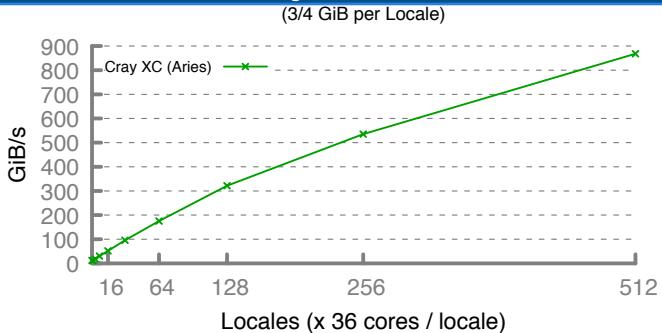
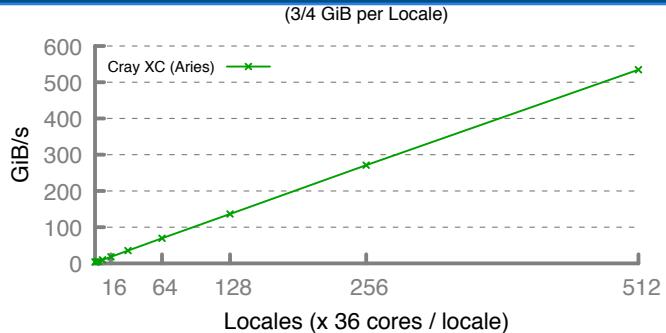


Arkouda Scaling: Aries at scale (512 locales, 18k cores)

CRAY
a Hewlett Packard Enterprise company

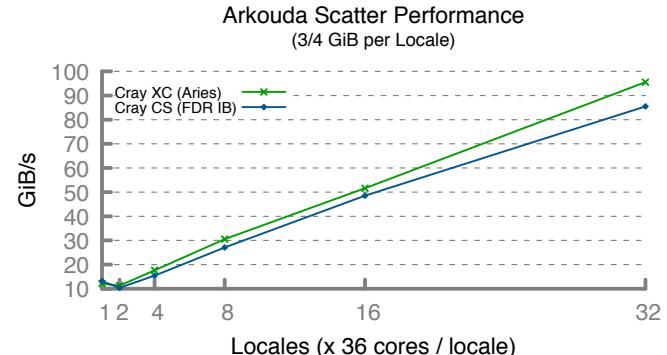
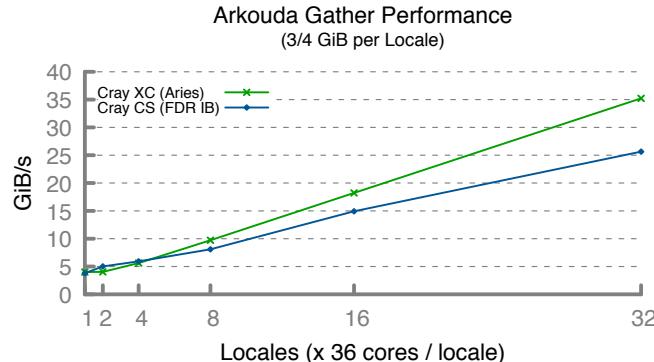
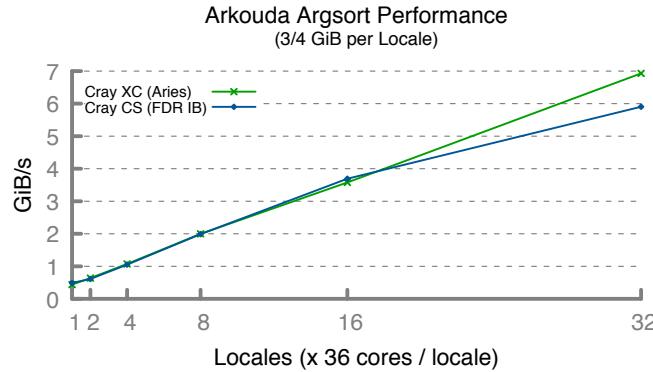
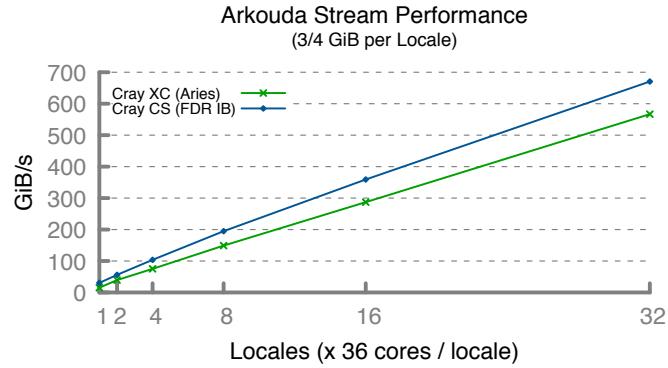


Sample result: Sorted 8TB of IPV4 addresses using 18k cores in just over a minute



Arkouda Scaling: Aries vs. IBV (32 locales, 1152 cores)

CRAY
a Hewlett Packard Enterprise company



What about Dask?

- Arkouda performs significantly faster in our experience*
 - Comparison involves reading from HDF5 file and doing a scalar reduction
 - *Looking to work with Dask experts to ensure a fair comparison
- Dask configuration on HPC systems is not trivial
 - Getting Arkouda performance out the box is easy
- Dask is not designed for shuffle-based workflows (sorting, grouping, etc.)
 - Arkouda targets these workflows and scales without lazy evaluation
- Dask is much more established and featureful
 - Better integration with Pandas and more supported data formats
 - Excellent cluster management tooling

Arkouda Status

CRAY
a Hewlett Packard Enterprise company

- Now 12,000+ lines of Chapel code, developed in one year
 - “without Chapel, we could not have gotten this far this fast”
- Open source:
 - Being developed on GitHub
 - Available as a PyPI package via ‘`pip install arkouda`’
- Being used on a daily / weekly basis on real data and problems
 - Features being added as requested by users



Arkouda Summary & Next Steps

CRAY
a Hewlett Packard Enterprise company

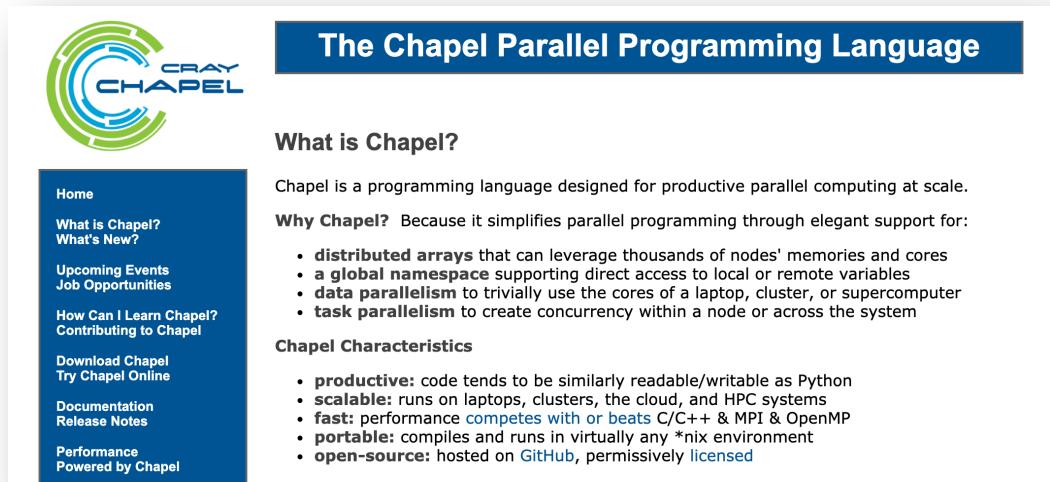
- Arkouda is a powerful tool and vision
 - NumPy/Pandas on TB-scale arrays in seconds to minutes
 - A workbench for interactive HPC-scale data science
- Arkouda takes a unique approach to interactivity in HPC
 - Starts with performance and builds towards interactivity
- Next steps involve expanding API and supporting multi-user sessions
 - Actual dataframes (currently informal collections of arrays)
 - Sparse linear algebra (GraphBLAS)
 - Wrapping existing HPC libraries
 - Data sharing and access control



For More Information

- Arkouda GitHub: <https://github.com/mhmerrill/arkouda>
- Arkouda PyPi page: <https://pypi.org/project/arkouda/>
- Arkouda Gitter Channel: <https://gitter.im/ArkoudaProject/community>
- Bill Reus's CHIUW talk: <https://chapel-lang.org/CHIUW2020.html#keynote>

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



The screenshot shows the Chapel Parallel Programming Language website. At the top right, there is a dark blue header bar with the text "The Chapel Parallel Programming Language". To the left of this header is the CRAY CHAPEL logo, which features a stylized green and blue "C" shape followed by the word "CHAPEL" in white. Below the header, there is a main content area with a white background. On the left side of this area, there is a sidebar with a dark blue background containing navigation links: "Home", "What is Chapel? What's New?", "Upcoming Events Job Opportunities", "How Can I Learn Chapel? Contributing to Chapel", "Download Chapel Try Chapel Online", "Documentation Release Notes", and "Performance Powered by Chapel". The main content area contains two sections: "What is Chapel?" and "Why Chapel? Because it simplifies parallel programming through elegant support for:". The "What is Chapel?" section has a small paragraph of text. The "Why Chapel?" section lists several bullet points about the language's features: distributed arrays, global namespace, data parallelism, and task parallelism. Below these sections is another heading, "Chapel Characteristics", followed by a list of bullet points describing the language's productivity, scalability, performance, portability, and open-source nature.

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](#)



SAFE HARBOR STATEMENT

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts.

These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

