Arkouda

Interactive Supercomputing for Data Science

Dr. William Reus
US Department of Defense

https://github.com/mhmerrill/arkouda
Data Science?

Data science proper is:
• Fundamental
• Difficult
• Computationally intensive
• Underemphasized
Understanding Physics of Datasets

Many names: Exploratory Data Analysis, Data Wrangling, Data Modeling, etc.
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

BLAS
C/Fortran code

and many more Python packages

NumPy

SciPy

pandas

NetworkX

GPU code

Numba

C/Fortran code
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
Dilemma: Interactivity vs. Scaling

What Data Science Needs

Single-node Python

Commodity Space

Conventional HPC
Can We Fly an HPC?

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

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

- Feature extraction
- Scalable DataFrames
- Large (hyper)graphs
- Parallel, Distributed Runtime
- GPU code?
- HPC libraries?
Arkouda Design

Python3 Client

Chapel Server

ZMQ Socket

Dispatcher

Indexing
 Arithmetic
 Sorting
 Generation
 I/O

Code Modules

Distributed Array

Distributed Object Store

Platform

Meta

MPP, SMP, Cluster, Laptop, etc.
A Chapel Interpreter

1. Client message
   “binopvv + id_1 id_2”

2. Module dispatch
   binopvv + id_1 id_2

3. Argument lookup and result allocation

4. Parallel execution

5. Return message
   “created id_3”

C = A+B

Client message

Module dispatch

Argument lookup and result allocation

Parallel execution

Return message

“created id_3”
Client Handles Bookkeeping

- A, B, C are instances of `pdarray` class
  - attributes store metadata
    - size
    - data type (subset of NumPy dtypes)
    - server-side name
  - methods issue server commands
    - e.g. operator overloads
    - object deletion issues server command to free array data

- Client language (python) handles
  - scoping
  - garbage collection
  - reference counting
  - exceptions
Chapel Is Unique

• Productivity
  • Parallelism and locality are first-class citizens
  • Arkouda server = 12k lines of code
• Performance
  • Single-threaded comparable to NumPy (C/Fortran)
  • Parallel, distributed comparable to C/OpenMP/MPI
• Portability
  • Develop on laptop, run on supercomputer
Where Does Arkouda Fit In?

• Unique approach: start with performance, build towards interactivity
• Arkouda uses the HPC
  • Scales well to at least 512 nodes / 18k cores
  • Exploits features of high-speed interconnects
  • Leverages parallel filesystems
  • All without user fine-tuning
• Current drawbacks
  • Still adding major features (e.g. authentication)
  • Only one I/O format (HDF5)
  • GPU support only for client
Arkouda Startup

1) In terminal:

   > arkouda_server -nl 96

   server listening on hostname:port

2) In Jupyter:

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

   4.2.5
   psp = tcp://nid00104:5555
   connected to tcp://nid00104:5555
Toy 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()

In [10]:
   bins = np.linspace(Cmin, Cmax, 20)
   _ = plt.bar(bins, hist.to_ndarray(), width=(Cmax-Cmin)/20)
```

MPP (Arkouda)

Login Node (Python/NumPy)
## Data Science on 50 Billion Records

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

- 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

---

### I/O

- Read from disk
- Scalar Reduction
- Histogram
- Vector Ops
- Logical Indexing
- Set Membership
- Gather
- Get Item
- Sort Indices by Value
- Group by Key
- Aggregate per Key
Sorting is Critical

Sorting (argsort and coargsort) is the rate-limiting step in most arkouda workflows:

- Grouping tabular data by one or multiple columns
- Creating sparse matrices (graphs)
- Finding unique values and reindexing
- Extracting features for statistical testing
- Computing set operations
Arkouda uses a least-significant-digit radix sort

- Requires a fast interconnect
  - communication is $O(wn)$
- But runtime is independent of data distribution
  - best case = worst case = avg. case = $O(wn)$

$$w = \lceil \log_{radix} (\text{max} - \text{min}) \rceil$$

https://www.growingwiththeweb.com/sorting/radix-sort-lsd/
Sorting Scales

Arkouda Argsort Performance
(16 GB per Locale)

Image credit: Elliot Ronaghan, HPE
Performance Is Portable

Arkouda Argsort Performance
(3/4 GB per Locale)

Image credit: Elliot Ronaghan, HPE
This example (from the arkouda source code) is very similar to `numpy.intersect1d`.
Future Directions

• Leaves
  • Implement DataFrames
  • Add sparse linear algebra (GraphBLAS)
  • ???

• Trunk
  • Authentication
  • Data sharing and access control
  • Multi-user resource management?

• Roots
  • Link in FFT, tensor decomp., solvers, etc.
  • Need to standardize a distributed array interface with the HPC community

[Your package here]
Arkouda

[Your library here]
A New (Old) Perspective on HPC

Not Just This

But Also This
Acknowledgements

• Michael Merrill – inventor and lead developer
• Elliot Ronaghan – significant performance enhancements, scaling studies
• Chapel team – instrumental in helping arkouda use Chapel to the fullest
• All our contributors!

https://github.com/mhmerrill/arkouda