Data Science Needs Interactive Supercomputing

Dr. William Reus

US Department of Defense
Why Data Science?

• Science: advancing human understanding of systems by building models tested against observation

• In most sciences, models are coupled to understanding
  Better understanding $\rightarrow$ more accurate model $\rightarrow$ more useful technology

• Data makes accurate* models possible without human understanding

• So why not skip to Data Technology?
"Can" Does Not Imply "Should"

Data Science

- Exploratory Data Analysis
  Advancing human understanding by testing hypotheses against data

Data Technology

- Artificial Intelligence
  Delegating human decisions to machines

- Analytics
  Answering fixed questions with data

Science is critical:

- Technology is not always the right goal
- Tech. without science will fail

And yet...

- Technology is what everyone talks about
- Large-scale tools favor tech. over science
(Data) Science is Interactive

“Hypothesis Testing”

Data → I/O → Summarize → Enrich → Inspect → Model → Transform → Filter → Output
Implications for Computing

• Stay in memory
• Compute in small, reversible steps
• Enable introspection (code and state)
• Use other people’s code
• Avoid boilerplate
• Maximize \( \frac{t_{thinking}}{t_{thinking} + t_{coding} + t_{waiting}} \)

So, basically Python...

...but fast
Interactive Computational Ladder

- Goal: Move seamlessly between tiers
  - Same data formats
  - Same UI (Jupyter)
  - Same APIs (NumPy/Pandas)
- Lower two tiers are easy
Interactive Computational Ladder

• We need the upper tier
  • Cybersecurity data >> 6 TB
• But hardware is the easy part
  • Need serious data engineering
  • Need to rethink job scheduling
  • Need an HPC shell
An HPC Shell for Data Science

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 Design

Jupyter/Python3

Chapel-Based Server

MPP
SMP
Cluster
Workstation
Laptop
Arkouda Design

• Why Chapel?
  • High-level language with C-comparable performance
  • Parallelism is a first-class citizen
  • Great distributed array support
  • Portable code: from laptop up to supercomputer
Where Does Arkouda Fit In?

• Unique approach
  • Other efforts: interactivity → parallel, distributed execution
  • Arkouda: proven HPC performance → interactivity

• Arkouda uses the HPC
  • Scales positively to at least 10k cores
  • Exploits features of high-speed interconnects
  • Leverages parallel filesystems
  • All without user fine-tuning

• Current drawbacks
  • Still adding major features
  • Only one I/O format (HDF5)
  • No GPU support
Arkouda Startup

1) In terminal:
   > arkouda_server -nl 96
   server listening on hostname:port

2) In Jupyter:

   In [2]: import arkouda as ak
   ak.connect(hostname, port)

   4.2.5
   psp = tcp://nid00104:5555
   connected to tcp://nid00104:5555
Data Exploration with Arkouda and NumPy

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)
Hypothesis Testing on 50 Billion Records

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Approximate 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[A == 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>30 - 300</td>
</tr>
<tr>
<td>Group by Key</td>
<td>G = ak.GroupBy(A)</td>
<td>60</td>
</tr>
<tr>
<td>Aggregate per Key</td>
<td>G.aggregate(B, ‘sum’)</td>
<td>15</td>
</tr>
<tr>
<td>Get Item</td>
<td>print(A[42])</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>Export to NumPy</td>
<td>A[:10**6].to_ndarray()</td>
<td>2</td>
</tr>
</tbody>
</table>

- A, B are 50 billion-element arrays
- Timings measured on real data
- Hardware: Cray XC40
  - 96 nodes
  - 3072 cores
  - 24 TB
  - Lustre filesystem
What about Model?

- Vision: Expose HPC libraries to Python via Arkouda
  - FFT
  - Tensor decomposition
  - Graph algorithms
  - Solvers
  - CHGL (Chapel HyperGraph Library from PNNL)
  - Anything you could link into a Chapel application (via C or LLVM)
- Need to standardize a distributed array interface with the HPC community
Python Implementation Details

• Python pdarray class: a shim for the distributed array on the Arkouda server
  • Stores server-side name of array
  • Has a NumPy-like dtype
  • Has methods that translate operators into server commands

• Arkouda relies on Python to reduce complexity
  • Scoping
  • Reference counting
  • Garbage collection
  • Exceptions

• Arkouda integrates with and uses NumPy
  • Dtype
  • Argument validation
  • Type conversion
Chapel Implementation Details

- A restricted Chapel interpreter:
  - Symbol table holding multi-type array wrappers
  - Code to parse commands from Python and select functions, operators, and types

- Chapel does some things really well
  - Makes parallelism easy (often implicit!)
  - Abstracts away inter-node communication and data layout
  - Compiler templates some functions
  - Allows dynamic casts from generic arrays to typed arrays

- But some things are hard
  - Large “select” statements for choosing functions, operators, types (an issue for all statically-typed languages)
  - Long compile times

- Far too many details to cover here...
Future Directions

• Open source release

• Tactical functionality
  • Strings and/or categorical dtype
  • Actual DataFrame class
  • Segmented arrays for sparse linear algebra (e.g. GraphBLAS)

• Strategic goals
  • Integration of Parallel Libraries
  • Multi-user support
Conclusion

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

It’s not crazy.
Acknowledgements

• Michael Merrill – inventor and lead developer
• Cray Chapel team – enthusiastic, responsive support