Arkouda
αρκούδα

NumPy-like arrays at massive scale!

Michael Merrill
CHIUW 2019
June 22, 2019
Why?

• “Python is the new bash”

• We have data analyses which need to be done at a much larger scale... because sampling to run at smaller scale alters what can be seen in the data

• We need to enable our data scientists with tools they know... so why not co-opt an interface or two

• Because we can and it’s fun!
We want some of our Data Scientists to drive an F22!

Jupyter allows Data Scientists to drive a cool plane!
Data Science Workflow

Data

Characterization
Hypothesis Testing
Model Construction for
- Latent variable inference
- Graph algorithms
- Prediction (ML)
- Other mature applications

Principles: stay in memory (interactive) and use packages
Goal: NumPy for HPC

- Distributed arrays with parallel primitives
- Familiar, interactive interface
- Smooth integration with mature HPC code
Why Python/NumPy API?

- NumPy is pervasive across Jupiter Notebooks
- Python data science packages communicate via NumPy arrays
- NumPy arrays wrap C and Fortran code for heavy lifting
- Need similar integration point for distributed HPC code!
Approach

- Other efforts approach from the interactive/interpreted side
- We decided to approach from the HPC side
- Nobody that we knew of was starting with HPC-level performance and working towards interactivity
- Preserving as close to interactive speed as we can
What?

• Present the user with a familiar interface

• Allow different execution contexts to coexist and communicate
  • A single threaded context: Python3
  • A multithreaded and distributed context: Chapel

• Work without users knowing about all the HPC stuff

• Currently a very targeted definition to allow specific analyses to be done
Structure

Chapel-Based Server

Massive Flame Front
Supercomputer
...For...
Well Funded Discriminating Zelots
...or any other computer even your laptop

Jupyter/Python3

ØMQ
Why Chapel?

• High level — makes for less code
• Close to “Pythonic” (for a statically type language)
• Great support for array operations and distributed arrays
• Direct support for sync/atomic variables
• Same code runs on single or multi-locale — laptop to supercomputer
In [1]: import arkouda as ak

In [2]: ak.v = False
   ak.connect(server="localhost",port=5555)
   4.2.5
   psp = tcp://localhost:5555

In [3]: ak.v = False
   N = 10**8 # 10**8 = 100M * 8 == 800MiB # 2**25 * 8 == 256MiB
   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:100000000 ndim:1 shape:(100000000) itemsize:8
   [0 2 4 ... 199999994 199999996 199999998]

In [4]: S = (N*(N-1))/2
   print(2*S)
   print(ak.sum(C))
   999999999999999999.0
   999999999999999999.0

In [ ]:
In [9]: `import arkouda as ak`

In [10]: `ak.v = True
ak.connect(server="localhost",port=5555)`

4.2.5

psp = tcp://localhost:5555
[Python] Sending request: startup
[Python] Received response: arkouda server started tcp://*:5555

In [11]: `N = 10**8 # 10**8 = 100M * 8 = 800MB
A = ak.arange(0,N,1)
B = ak.arange(0,N,1)
C = A+B
print(ak.info(C),C)`

[Python] Sending request: arange 0 100000000 1
[Python] Received response: created id_7 int64 100000000 1 (100000000) 8
id_7 int64 100000000 1 [100000000] 8
[Python] Sending request: delete id_4
[Python] Received response: deleted id_4
[Python] Sending request: arange 0 100000000 1
[Python] Received response: created id_8 int64 100000000 1 (100000000) 8
id_8 int64 100000000 1 [100000000] 8
[Python] Sending request: delete id_5
[Python] Received response: deleted id_5
[Python] Sending request: binopv + id_7 id_8
[Python] Received response: created id_9 int64 100000000 1 (100000000) 8
id_9 int64 100000000 1 [100000000] 8
[Python] Sending request: delete id_6
[Python] Received response: deleted id_6
[Python] Sending request: info id_9
[Python] Received response: name:"id_9" dtype:"int64" size:100000000 ndim:1 shape:(100000000) itemsize:8
```
[Python] Received response: deleted id_4
[Python] Sending request: arange 0 100000000 1
[Python] Received response: created id_8 int64 100000000 1 (100000000) 8
id_8 int64 100000000 1 [100000000] 8
[Python] Sending request: delete id_5
[Python] Received response: deleted id_5
[Python] Sending request: binopv + id_7 id_8
[Python] Received response: created id_9 int64 100000000 1 (100000000) 8
id_9 int64 100000000 1 [100000000] 8
[Python] Sending request: delete id_6
[Python] Received response: deleted id_6
[Python] Sending request: info id_9
[Python] Received response: name:"id_9" dtype:"int64" size:100000000 ndim:1 shape:(100000000) itemsize:8
name:"id_9" dtype:"int64" size:100000000 ndim:1 shape:(100000000) itemsize:8
[Python] Sending request: str id_9 100
[Python] Received response: [0 2 4 ... 199999994 199999996 199999998]
[0 2 4 ... 199999994 199999996 199999998]

In [12]: S = N*(N-1))/2
print(2*S)
print(ak.sum(C))

9999999900000000.0
[Python] Sending request: reduction sum id_9
[Python] Received response: int64 9999999900000000
9999999900000000

In [ ]:
```
Chapel Implementation Details

• I could make a whole talk about this…

• Implementing the array operations is straightforward in Chapel.

• Implementing function, operator, and type selection is where most of the code is in the implementation. This is an issue for all statically type languages.

• Flat multi-type symbol table…
  • Enum to mirror dtypes/types used, testable at runtime.
  • Chapel dynamic casts were important.

• Select constructs everywhere… moving to vtable-like approach indexed by dtype/type enum.
Chapel
Implementation Details

• Needed generic fields when using BlockDist arrays in a class

• Using PrivateDist for some optimizations

• Type-based nested procedures

• Need more meta programming facilities like macros or a way to auto generate from a template.

• Need a Chapel type primer/tutorial to show how to explicitly state a type… suggesting init() variants is sometimes not helpful.
Python
Implementation Details

• We rely on Python’s scoping, reference counting, and GC.

• GC issue — Jupyter prevents garbage collection when you put a var in a cell to get the repr... the Out[] in Jupyter creates a reference to the object.

• Importing NumPy and using types and other features to extend functionality

• Pdarray object is a shim with a handle(name) of the array object in the Arkouda server
HDF5 Array I/O

• Currently the data we operate on comes in CSV files

• We use a Python Pandas/HDF5 process to convert CSV files into HDF5 files

• pip3 install hdflow

• Arkouda only has HDF5 I/O at the moment
A point of integration for HPC libraries and Python3

• Parallel Libraries:
  • FFT
  • Tensor
  • Graphs
  • Solvers
  • CHGL — Chapel Hyper Graph Library
  • Many others

• Anything you could link into a Chapel application and interface with…
Future Possibilities

• Better Fileset/Dataset I/O

• More NumPy/SciPy/Pandas functionality

• Linking in parallel libraries to make them available

• Persisted and shared workspaces

• Multiuser

• Maybe even send and interpret ASTs from Python3

• Julia?

• Sharing… open source - approved, just waiting on OGC licensing opinion
Conclusions

• Enable your data scientists to do larger scale analyses.

• Look at more data and gain insights from the experience.

• It was not that hard and a lot of goodness from several months of work.

• You could use this pattern for other useful things.