Coupling Chapel-Powered HPC Workflows for Python

John Byrne, Harumi Kuno, Chinmay Ghosh, Porno Shome, Amitha C, Sharad Singhal, Clarete Riana Crasta, David Emberson, Abhishek Dwaraki

June 2, 2023
The OpenFAM library for programming Fabric-Attached Memory supports shared pools of fabric-attached memory (FAM) hosted on conventional nodes.

The OpenFAM API lets programmers create and share regions of FAM, and also data items within regions.

OpenFAM uses RDMA to support operations like put, get, scatter, gather, copy, backup, and restore, as well as standard atomic operations such as fetch-and-add, compare-and-swap.

FAM bandwidth and latency are currently superior to Flash but inferior to local DRAM.
Arkouda lets Python programmers work interactively with data residing in the memory of compute nodes.

For each line of code, the Python program, acting as an Arkouda client, sends commands and metadata over the wire to the Arkouda server running on the compute nodes.
FAMArray Storage Manager

FAM Dataset Storage Manager
- famDatasetStore class
  - Operations (e.g., filter, scatter, gather, sortrun, topXperrun, toPdarray, toDataFrame) on FAM Datasets.

Arkouda Client Python Module
- famarray and famstore classes
  - Over-the-wire messages that let Arkouda clients request that the Arkouda server perform operations on arrays of data residing in FAM.
  - Movement of data between Arkouda pdarrays and named arrays of data stored in FAM.
  - Store FAM objects in Arkouda Symbol Table.

Arkouda Server (Chapel application)
- FamMsg Module
  - Multi-locale put, get, gather, scatter of data stored in FAM.
  - OpenFAM API for single-locale operations on data stored in FAM.

Chapel programming language

FAMArrayStore Module
- OpenFAM Module
- OpenFAM C Linkage Library

OpenFAM
- OpenFAM Client
- OpenFAM Server
FAMArray Storage Manager for Chapel and Arkouda

Python code (Jupyter Notebook)

```python
// Create large FamArray
famArray1 = fam_region.create (‘aName’, ak.int, 100000000)

// Create parray of random ints
parray1 = ak.randint(0,100000000,10000)

// Create parray with int sequence
parray2 = ak.arange(0,10000,1)

// Scatter contents of parray2 across famArray1
famArray1[parray1] = parray2
```

For each line of code, the Python program, acting as an Arkouda client, sends commands and metadata over the wire to the Arkouda server running on the compute nodes.

The Arkouda server, acting as a FAM client, uses the FAM Array Store to explode the `famArray1[parray1] = parray2` assignment into scatter operations executed on each locale for the subset of `parray1` and `parray2` located on that locale.

Arkouda Server
(Chapel program on compute nodes)

- `parray1` and `parray2` are distributed in memory across compute nodes.
- `famArray1` is a proxy for a FAM distributed array.

FAM Array Store / OpenFAM modules (on FAM nodes)

- `famArray1` distributed across FAM.
- FAMArrayStore invokes OpenFAM module to execute the scatter operation.
Operations (e.g., filter, scatter, gather, sortruns, topXperrun, toPdarray, toDataFrame) on FAM Datasets.

Over-the-wire messages that let Arkouda clients request that the Arkouda server perform operations on arrays of data residing in FAM. Movement of data between Arkouda pdarrays and named arrays of data stored in FAM. Store FAM objects in Arkouda Symbol Table.

Multi-locale put, get, gather, scatter of data stored in FAM.

OpenFAM API for single-locale operations on data stored in FAM.
Getting the most from FAM + Arkouda

- Enable Python programmers to solve problems using datasets too large to fit in the memory of compute nodes.
- Remote memory has lower bandwidth and higher latency than local DRAM.
- Move data between the FAM pool and the local memory of the compute nodes such that:
  - Data is placed in memory that is “close” to the processors that will operate upon that data.
  - Computations operate on distinct subsets of program data.
Store data in FAM in batches, as discrete arrays. Present data to the programmer as integrated Datasets.
Derived FAM Datasets represent indices into base data

filter(Trip, 'fare', 'gt', 10000)
Can use the index to gather columns of interest

Trip.fare_gt_10000:

Gather(Trip.fare_gt_10000, 'pickuptime')

Number of Batches: 2
Some operations produce derived columns

\[
\text{subtract(}
\text{Trip.fare}_{\geq}10000,
\text{'dropofftime'},
\text{'pickuptime'},
\text{'duration'})
\]
1. Ingested data is stored in famarrays in a FAM Array Store.

2. The FAM Dataset Storage Manager presents integrated views of related base and derived data items as a collection of FAM Datasets.

3. Multiple Arkouda processes can attach to the same FAM Dataset Store, sharing FAM data while maintaining their own symbol tables and internal working data sets.

Data is managed in terms of ordered batches, so it can be incrementally processed, and so that previous results can be leveraged to speed time-to-results for future results.

Derived datasets could be updated using an automatic update mechanism as new batches of data are ingested.
Working vs. Shared Data and Metadata

A transient FAM Dataset Storage Manager instance runs within the Jupyter Notebook process. Working metadata resides in memory here.

Data and metadata “persist” in memory on FAM nodes as FAM Arrays.

Within the Arkouda server processes, the FAM Dataset Storage Manager “pages” batches of data between fmaarrays and Arkouda parallel distributed arrays (pdarrays). Working data resides in the local memory of compute nodes.
Arkouda and Chapel programs can use FAM to share results.

Diagram showing the integration of Arkouda and Chapel programs with FAM, highlighting the communication flow between different nodes and servers.
In addition, we also provide an HDF5 interface for FAM

- Hierarchical Data Format version 5 (HDF5) is an extensible data model/specification (backed by a library of open source software) that makes it easier to organize and share large, complex, heterogeneous data.
- It works with a variety of backing stores, such as POSIX, DAOS, and AWS S3.
- The OpenFAM connector for HDF5 maps FAM storage into the HDF5 data model through the VOL layer.
- This connector enables applications, regardless of programming language, to read/write HDF5 datasets on FAM.

This figure is based on a drawing from slide 31 of:
Take-Aways and Next Steps

**Take-Aways**

- Chapel does a very nice job translating array operations into parallel computation, and Arkouda brings this to Python programmers.

- FAMArray extensions to Chapel and Arkouda help Chapel and Python programmers work with disaggregated memory when using a compute cluster.

- FAM Dataset Storage Manager leverages these extensions to help Python programmers derive datasets through workflows, to maintain derived index and column data automatically and incrementally, and to save and share results.

**Potential Next Steps**

- FAM-side computation

- Native paging across FAM