Hewlett Packard
Enterprise

IMPLEMENTING AND OPTIMIZING PARQUET I/O IN CHAPEL

Ben McDonald, HPE

Software Engineer – Chapel team

CHIUW 2022

June 10, 2022

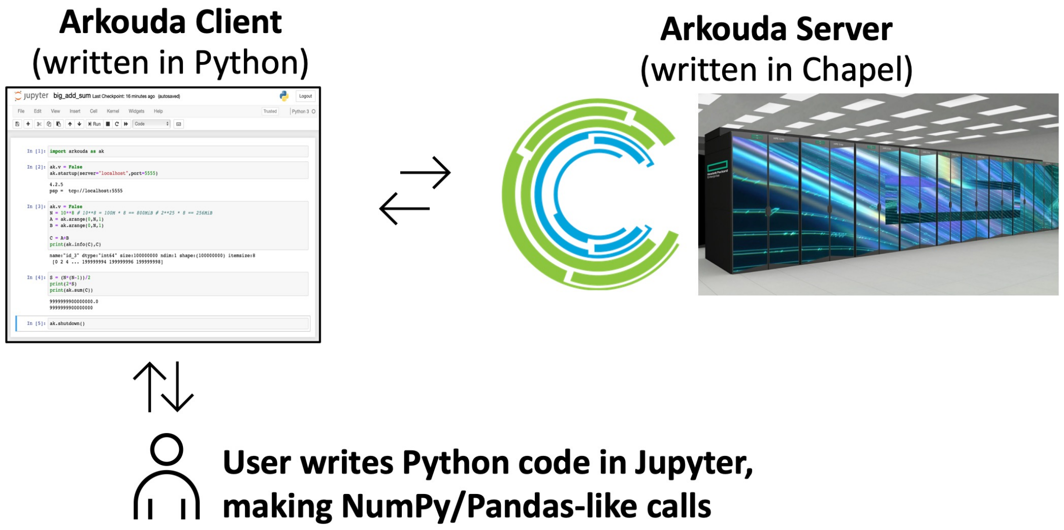
ARKOUDA SUMMARY

Arkouda

- A Python library supporting a key subset of the NumPy and Pandas interfaces for Data Science
  - Uses a Python-client/Chapel-server model for scalability and performance
  - Computes massive-scale results (multi-TB arrays) within the human thought loop (seconds to a few minutes)
- Open-source: https://github.com/Bears-R-Us/arkouda

Typical Workflow

- Read in hundreds of files containing terabytes of data
- Perform typical data science analysis on that data
  - i.e., sort, group by, etc.
- Write results to a file

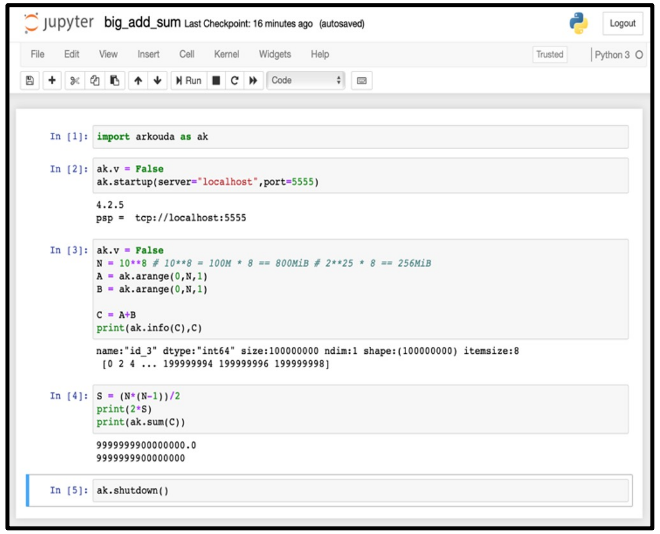


Arkouda HDF5 Support

- HDF5 previously only supported file format
- Viewed as a standard HPC file format by many
- Designed specifically for HPC use

ARKOUDA'S HIGH-LEVEL APPROACH

Arkouda Client
(written in Python)



Arkouda Server
(written in Chapel)

User writes Python code in Jupyter, making NumPy/Pandas-like calls

OBJECTIVE

Apache Parquet

- Widely-used columnar file format for data analytics
- Columnar storage allows efficient queries of single columns

File

	Column 1	Column 2	Column 3
Row 1	A0	B0	C0
Row 2	A1	B1	C1
Row 3	A2	B2	C2
Row 4	A3	B3	C3

Row storage

A0	B0	C0	A1
B1	C1	A2	B2
C2	A3	B3	C3

Columnar storage

A0	A1	A2	A3
B0	B1	B2	B3
C0	C1	C2	C3

Contiguous memory layout improves single-column read performance in Parquet

This Effort

- Goal to add support for Apache Parquet in Arkouda
  - Increasing demand for Parquet from Arkouda user community
- Goal to be within 2.5x the performance of Arkouda's HDF5 implementation
  - Set by external users requesting the functionality based on level of tolerance to HDF5 performance

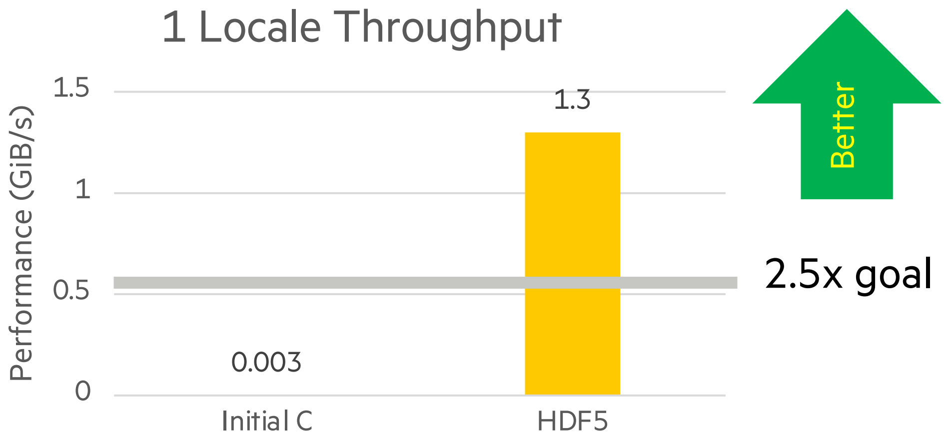
SUPPORTING PARQUET IN CHAPEL

PARQUET C LIBRARY INTEGRATION

Initial Implementation

- Clear first steps were to interoperate with the C Parquet API, given Chapel's C interoperability features
- Read performance of 400 files, 0.25 GiB integer elements each on a single node of Cray CS :
  - Higher is better on performance graph

Version	1 locale Throughput
Initial C	0.003 GiB/s
HDF5	1.30 GiB/s



- Performance well under target performance goal (~480x slower read performance)
- C implementation not fully featured, limiting room for optimizations
  - Limited metadata access, required to read into Arrow data structures, etc.
- Poor performance led to exploration using C++ implementation

PARQUET C++ LIBRARY INTEGRATION

Transition to C++ Library

- Explored the C++ library since it is the standard and most feature-complete library for Parquet
- Unfortunately, Chapel does not support interoperability with C++

Solution

- Created a thin C wrapper around C++ functions to allow Chapel to execute C++ code through C interop
  - C++/C code compiled into an object file and linked with Chapel program
- Enabled much greater control over Parquet API calls
- Enabled use of additional features not supported in C API (such as additional compression formats)

PARQUET C++ LIBRARY INTEGRATION

Chapel Code: Call C function through C interoperability

```
extern proc c_readParquetColumn(chpl_arr, …): int;
c_readParquetColumn(c_ptrTo(chpl_arr), …);
```

C Code: Call C++ to enable code to be executed from Chapel

```
int64_t c_readParquetColumn(int64_t* chpl_arr, …)
{
  return cpp_readParquetColumn(chpl_arr, …);
}
```

C++ Code: Call Parquet C++ API, do the actual work

```
int64_t cpp_readParquetColumn(int64_t* chpl_arr, …) {
  …
  parquet::ParquetFileReader::OpenFile(filename, inFile);
  parquet::ReadFile(inFile, chpl_arr);
  …
}
```

Chapel calls C code, passing pointer to Chapel array

C code calls C++ code, passing pointer to Chapel array

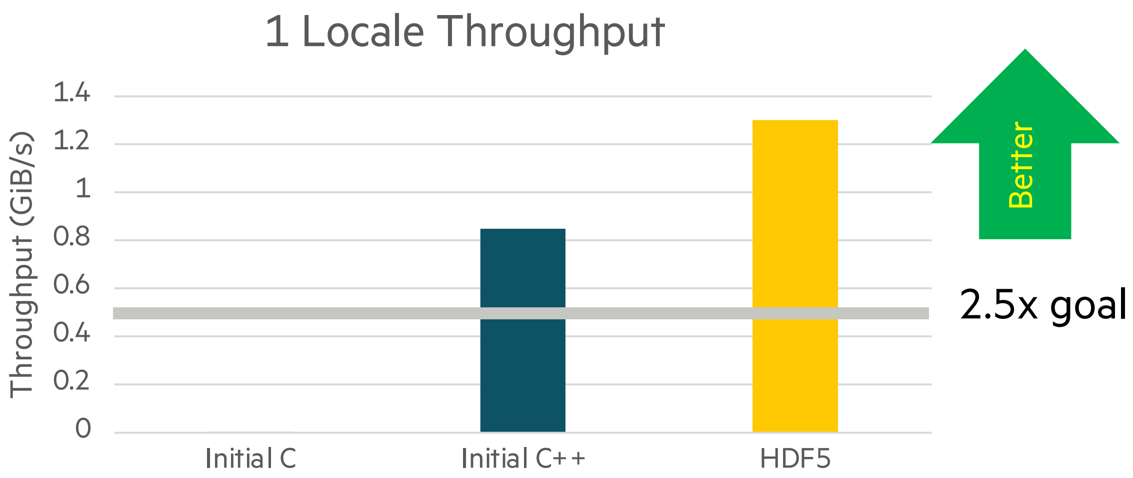
C++ code reads Parquet file into Chapel array pointer

PERFORMANCE OPTIMIZATIONS & RESULTS

PERFORMANCE RESULTS

- Read performance of 400 files, 0.25 GiB integer elements each on a Cray CS with InfiniBand HDR:
  – Higher is better on performance graph

Version	1 Locale Throughput	16 Locale Throughput
Initial C	0.003 GiB/s **480x**	--
Initial C++	0.85 GiB/s **1.5x**	10.75 GiB/s
HDF5	1.30 GiB/s **--**	12.04 GiB/s



- C++ performance was within goal-range out of the box

PERFORMANCE OPTIMIZATIONS

Two key optimizations

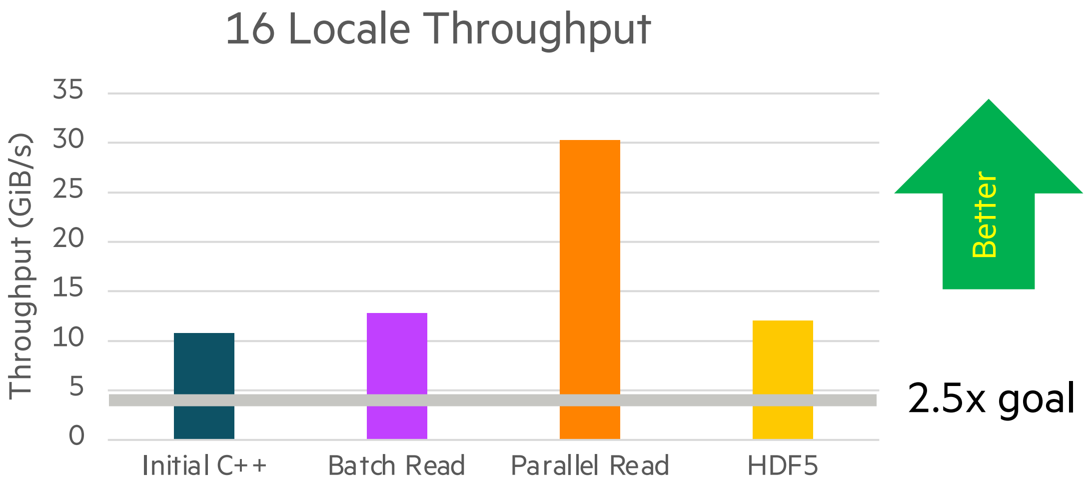
1. Switch to low-level batching API from standard API
   - Removed need to store temporary copy in an intermediate data structure
   - Allowed reading of large chunks of data directly into Chapel array
   - Resulted in modest performance improvements

2. Parallelize file reads to read multiple files concurrently
   - Separate Parquet file reader objects can read files in parallel
   - As simple as changing a ‘for’ loop to a ‘forall’ loop in Chapel
   - Resulted in significant performance improvements when reading multiple files

PERFORMANCE RESULTS

- Read performance of 400 files, 0.25 GiB integer elements each on a Cray CS with InfiniBand HDR:
  - Higher is better on performance graph

Version	1 Locale Throughput	16 Locale Throughput
Initial C++	0.85 GiB/s	10.75 GiB/s
Batch read	0.96 GiB/s	12.82 GiB/s
Parallel read	12.05 GiB/s	30.28 GiB/s
HDF5	1.30 GiB/s	12.04 GiB/s



- HDF5 is not a thread-safe library, so can only read 1 file at a time on each locale
  - Single-file HDF5 reads still outperform single-file Parquet reads

- Parquet is designed specifically for Data Science, natural fit with Arkouda

CONCLUSION

- We were able to interoperate with C++ in Chapel by using a thin C-wrapper around C++ functions
- Single node read performance improved from 0.003 GiB/s to 30.28 GiB/s
  - Over 1000x improvement
- Surprisingly, Parquet outperformed HDF5 read performance for our 16-locale benchmark by 2.5x
  - 30.28 GiB/s for Parquet; 12.04 GiB/s for HDF5
- Largest performance gain came from parallel features in Chapel
  - Simple change of a ‘for’ loop to a ‘forall’ loop

THANK YOU

https://chapel-lang.org
@ChapelLanguage