Arkouda Set Operation Optimizations

Ben McDonald
Gonzaga University
USA
bmcdonald3@zagmail.gonzaga.edu

Elliot Ronaghan
Hewlett Packard Enterprise
USA
elliot.ronaghan@hpe.com

ABSTRACT

This talk discusses experiences of a summer intern in getting up and running on distributed memory systems using Chapel. During the course of the summer, the presenter learned how to use supercomputers thanks to the abstraction of complicated distributed computing concepts and high-level syntax of Chapel. Furthermore, this talk highlights significant contributions made to the performance of Arkouda, which is a NumPy-like Python package with a Chapel backend server that allows data scientists to interactively utilize supercomputers. The design of Arkouda will be outlined to better explain work accomplished and performance graphs of the improvements will be shown.

1 Introduction

1.1 Background

Through the presenter's experience, we have observed that Chapel's ease-of-use can enable a programmer with no HPC experience to get ramped up on supercomputers fairly quickly. The Pythonic syntax of Chapel and it's handling of difficult distributed, parallel aspects of high performance computing for the programmer not only make programming performant code more productive, but also allow programmers to learn these concepts through programming themselves.

1.2 Design of Arkouda

Arkouda is a NumPy-like Python package designed for data scientists to interactively utilize supercomputers. Arkouda uses a client-server communication model, meaning that the server can be run on a supercomputer and then be connected to from a laptop. Arkouda's client is written in Python and implements portions of interfaces from popular data science packages, such as NumPy and Pandas, which means it has a very familiar feel to many data scientists.

Arkouda is designed with a Chapel backend server that communicates with the Python client over ZMQ sockets and also stores the data as well as performs the operations. This client-to-server communication can end up taking more time than the actual operation in some cases, and for common functions, this can result in an unnecessary, significant reduction in performance.

Arkouda is centered around "pdarrays" (parallel, distributed array). The data is stored on the server, implemented in Chapel, while the client accesses the data by calling into the Chapel server. This means that a Python user can greatly improve performance by using pdarrays, but that each array operation, such as array slicing, requires a call to the server.

2 Optimizing Set Operations in Chapel

Set operations, such as union, intersect, etc., are used frequently in data science and, in Arkouda, were originally written purely on the client-side in Python. These operations require many slices, among other array operations. Due to the design of Arkouda and the implementation of the operations, this resulted in 10-15 server calls per operation with the design of Arkouda. While this demonstrates part of Arkouda's power, being able to quickly compose complex set operation on the client side, the large amount of use the set operations were seeing warranted additional optimizations.

By writing the set operations natively in Chapel the set operations were able to be optimized, reducing the number of server calls to only a single call, as opposed to the aforementioned 10-15 calls that were happening previously. Also, with the operations written in Chapel, we were able to further optimize the operations by eliminating unnecessary work that could not be avoided, given the limitations of Python.

3 Performance Results

Figure 1 shows set operation performance from Arkouda's nightly benchmarking collected using 16-nodes of a Cray-XC.

Table 1 shows the set operation performance before and after our optimizations.

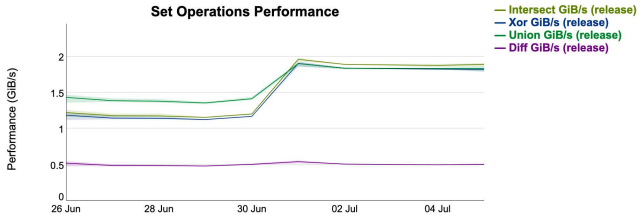

Figure 1: Set operations performance improvements

Set Operation	Before (GiB/s)	After (GiB/s)	Speedup
intersect	1.07	2.01	88%
union	1.12	1.95	74%
exclusive or	1.05	1.90	81%
set difference	0.45	0.49	8.9%

Table 1: Set Operations on 16-nodes of a Cray-XC

4 Conclusion

The high-level, Pythonic syntax and abstraction of complicated concepts provided by Chapel enable programmers to quickly become productive using supercomputers and understand parallel and distributed concepts.

The performance improvements from the translation of set operations in Arkouda to server side operations show the value of limiting server calls in Arkouda when possible and help pave the way for future work on Arkouda.