## Overview

<table>
<thead>
<tr>
<th>Set Operations</th>
<th>NumPy and Arkouda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arkouda design</td>
<td>Optimization of set operations</td>
</tr>
<tr>
<td>Performance results</td>
<td></td>
</tr>
</tbody>
</table>
What are set operations?

- Like you learned about in math: union, intersect, difference, xor
- Commonly used in Arkouda and data science in general

```python
>>> import arkouda as ak
>>> ak.connect()
>>> a = ak.array([1,2,3])
>>> b = ak.array([2,3,4])
>>> ak.intersect1d(a,b)
array([2, 3])
>>> ak.union1d(a,b)
array([1, 2, 3, 4])
>>> ak.setxor1d(a,b)
array([1, 4])
>>> ak.setdiff1d(a,b)
array([1])
```
NumPy
• Widely used Python package for array operations
• Uses pre-compiled C code

Data science
• Analyzing large sets of data
• Provides statistics, trends, information
Background: Arkouda

• Open-source Python package backed by Chapel
• Similar functionality to NumPy, but scalable
• Enables interactive supercomputing for data science
• Interoperates with NumPy
Design of Arkouda

Python3 Client

Chapel Server

Client to server communication

Arkouda User
Optimization of Set Operations

• **Previously**: written purely on the client/Python side

• **Problem**: server communication taking up bulk of execution time

• **Goal**: move operations to purely server-side operations to minimize communication

• **Result**: saw performance improvements ranging from ~10%-90%
Chapel vs. Python

def intersect1d(pda1: pdarray, pda2: pdarray, assume_unique: bool=False) -> pdarray:
    if not assume_unique:
        pda1 = unique(pda1)
        pda2 = unique(pda2)
    aux = concatenate((pda1, pda2), ordered=False)
    aux_sort_indices = argsort(aux)
    aux = aux[aux_sort_indices]
    mask = aux[1:] == aux[1:-1]
    int1d = aux[1:-1][mask]
    return int1d

proc intersect1d(a: [] int, b: [] int, assume_unique: bool) {
    if (!assume_unique) {
        a = uniqueSort(a, false);
        b = uniqueSort(b, false);
    }
    var aux = radixSortLSD_keys(concatset(a,b));
    const ref head = aux[..aux.domain.high-1];
    const ref tail = aux[aux.domain.low+1..];
    const mask = head == tail;
    return boolIndexer(head, mask);
}
Performance Results

Arkouda Set Operations Performance using 16-locales of a Cray-XC

<table>
<thead>
<tr>
<th>Operation</th>
<th>Before Changes (GiB/s)</th>
<th>After Changes (GiB/s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersect</td>
<td>1.07</td>
<td>2.01</td>
<td>88%</td>
</tr>
<tr>
<td>Union</td>
<td>1.12</td>
<td>1.95</td>
<td>74%</td>
</tr>
<tr>
<td>Exclusive or</td>
<td>1.05</td>
<td>1.90</td>
<td>81%</td>
</tr>
<tr>
<td>Set difference</td>
<td>0.45</td>
<td>0.49</td>
<td>8.9%</td>
</tr>
</tbody>
</table>
Conclusion

• Optimizations like this should be done selectively

• Goal of Arkouda is to enable data scientists to interactively utilize supercomputers
Acknowledgements

• Thank you, Chapel team
• Thank you, Arkouda team
Thank you!

Questions?