Squeezing Performance out of Arkouda

Elliot Ronaghan

CHIUW 2020

May 22, 2020

 elliot.ronaghan@hpe.com

 chapel-lang.org

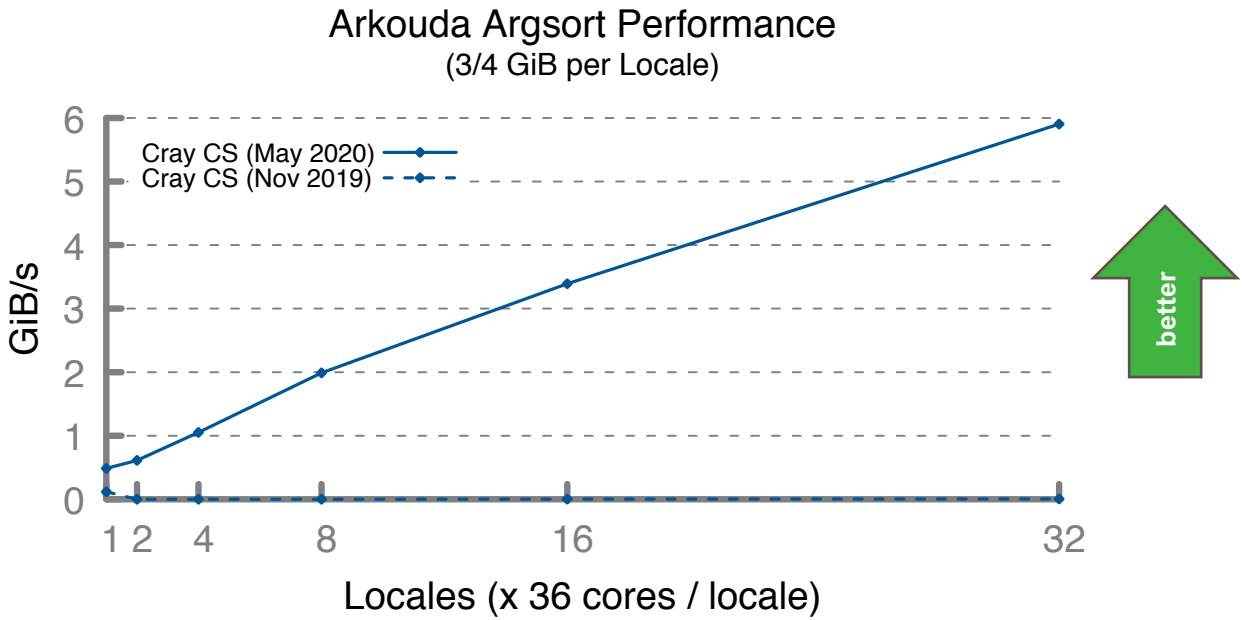
 @ChapelLanguage

CRAY®

a Hewlett Packard Enterprise company

Teaser

- Achieved more than a 1000x speedup for sorting on InfiniBand systems



Testing Hardware

- Most performance results will be shown for Cray CS and Cray XC systems
  - 32-node Cray CS with 56 Gb/s InfiniBand network (1,152 cores)
  - 512-node Cray XC with Aries network (18,432 cores)
- Per-node hardware is similar for both systems
  - 36-core (72-thread) 2.1 GHz Broadwell processors
  - 128 GiB RAM
- Results for other networks will be shown at the end
  - Cray CS with 100 and 200 Gb/s InfiniBand networks
  - AWS clusters with 25 and 100 Gb/s Ethernet networks

Outline

- NumPy vs Arkouda Performance
- Sorting Background
- Small Message Performance
- Message Aggregation
- Aggregated Sort Performance
- Future work

Arkouda Background

- See Bill Reus's CHIUW 2020 keynote on Arkouda for details
- At a high level: Arkouda provides NumPy-like arrays at HPC scale
  - A NumPy/Pandas Python interface, backed by Chapel
  - Chapel provides performance and scalability

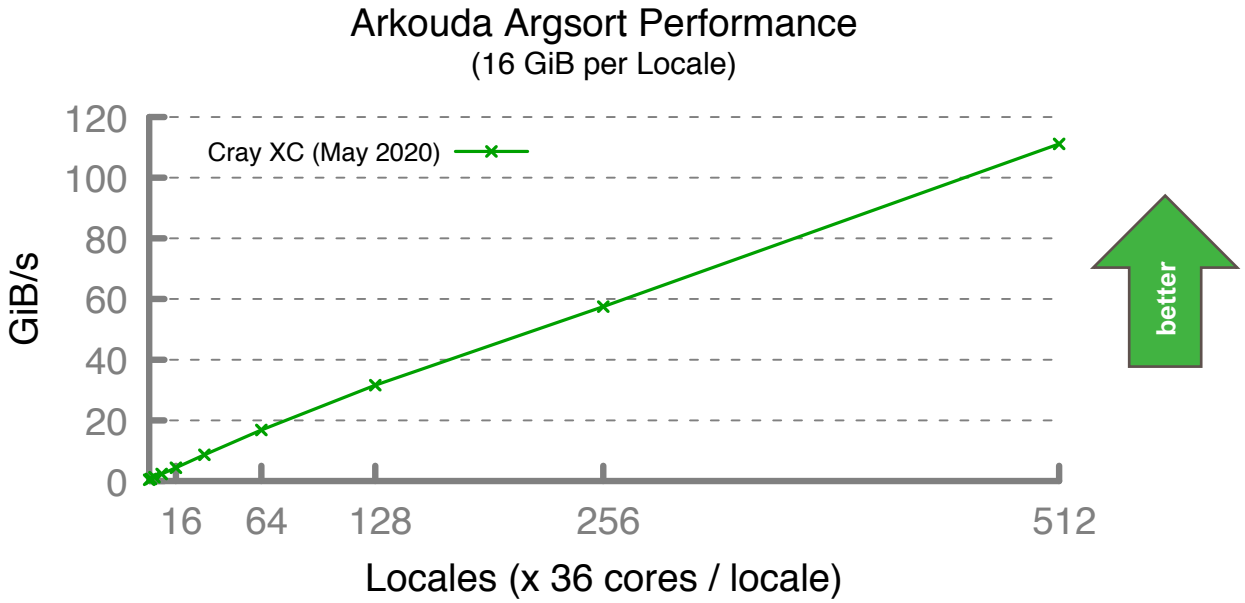
NumPy vs Arkouda Performance

- Performance normalized to NumPy on a Cray XC

benchmark	numpy	ak (serial)	ak (36-core)	ak (512-node)
argsort	1.0	2.0	16.7	1837.3
coargsort	1.0	2.3	16.7	984.7
gather	1.0	0.4	11.7	469.1
reduce	1.0	1.2	12.0	4412.4
scan	1.0	0.8	3.2	266.6
scatter	1.0	1.0	11.8	781.8
stream	1.0	0.7	6.2	1590.4

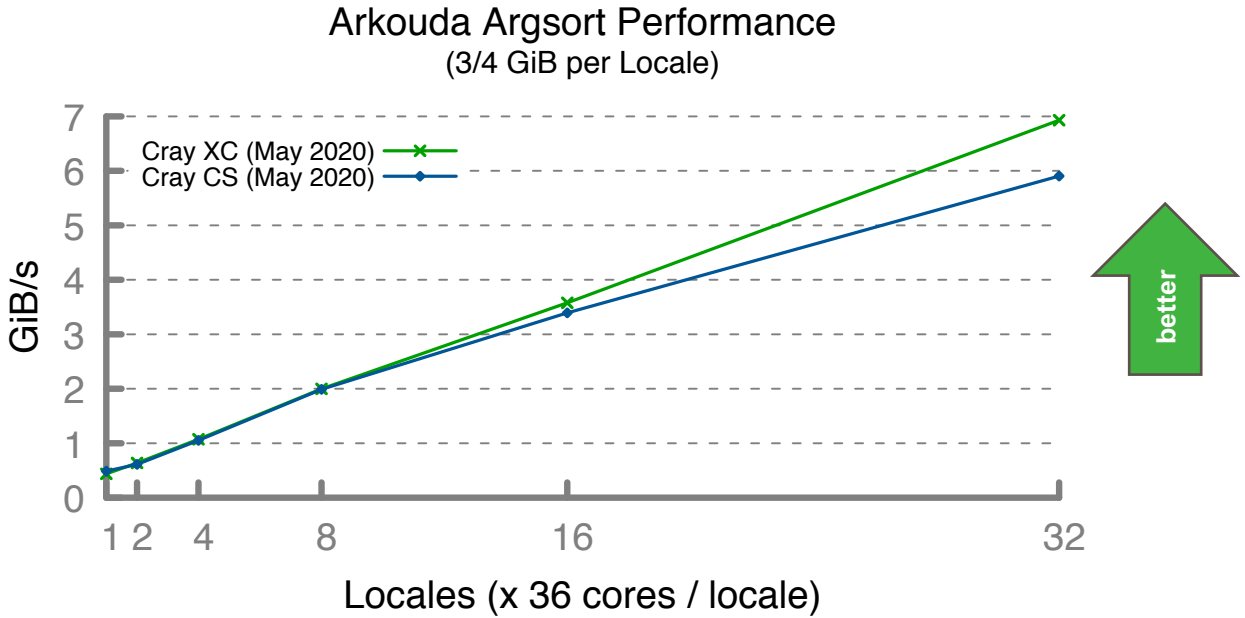
Current Argsort Performance

- Argsort has excellent performance and scalability on a Cray XC
  - 110 GiB/s at 512 nodes, sorting 8TiB of data in ~75 seconds



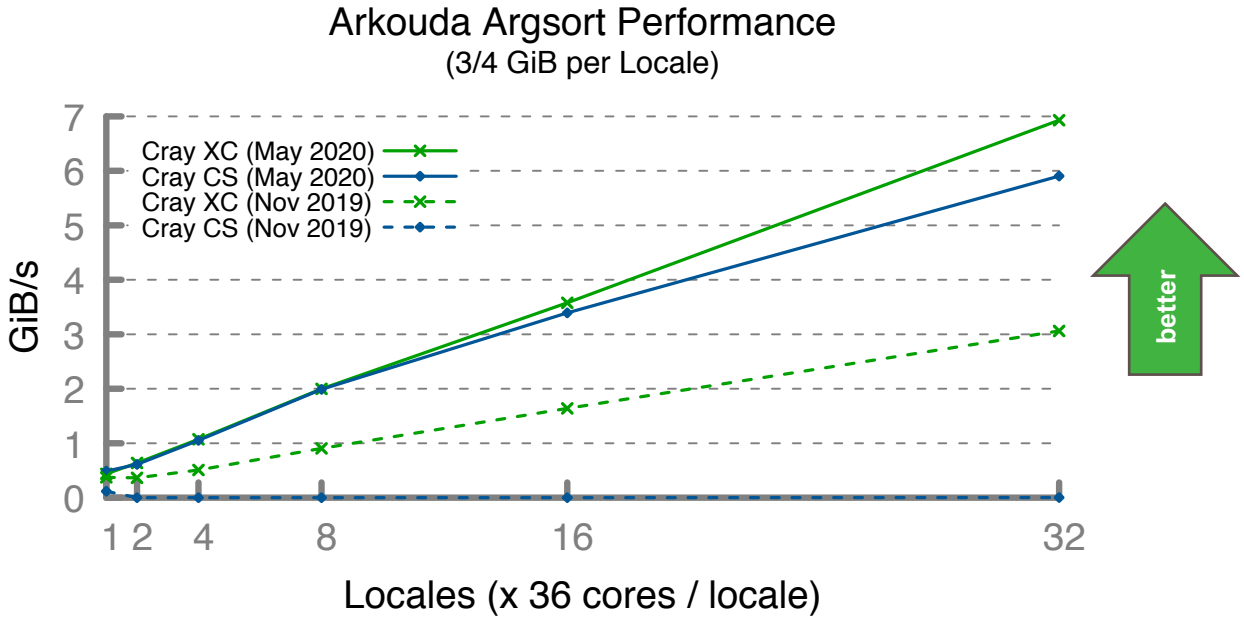
Current Argsort Performance

- Argsort performance is portable to InfiniBand networks
  - Similar performance up to 32 nodes



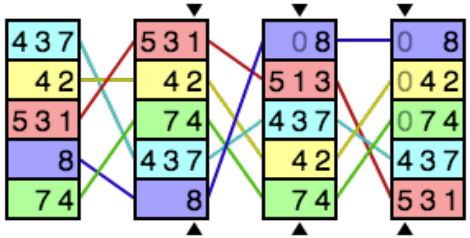
Original Argsort Performance

- Performance 6 months ago was a very different story
  - Respectable XC performance, unusable CS performance



Sorting Implementation

- Arkouda uses a simple 100 line least-significant-digit radix sort
  - Easy to parallelize, but lots of communication $O(wn)$
    - w is the number of digits (16-bit digits), n is the number of elements
    - 50B element array of 32-bit values has 100B 8-byte network transactions
  - Want to optimize without sacrificing maintainability and simplicity



https://www.growingwiththeweb.com/sorting/radix-sort-lsd/

Small Message Performance

- Previously, Arkouda used 'unorderedCopy()' for these 8-byte messages
  - Optimized copy for when no overlap will occur
- Sample performance for copying a 32 MiB local array to a remote array

```
// bulk
rArr = lArr;


// unordered
forall (r, l) in zip(rArr, lArr) do
  unorderedCopy(r, l);
```

Performance (MiB/s)		
config	**bulk**	**unordered**
Cray XC	8000.0	510.0
Cray CS	6000.0	2.0

Message Aggregation

- Added copy aggregators to Arkouda
  - Created for each task, must specify whether source or destination is remote

```
// bulk
rArr = lArr;


// unordered
forall (r, l) in zip(rArr, lArr) do
  unorderedCopy(r, l);


// aggregated
forall (r, l) in zip(rArr, lArr) with (var agg = new DstAggregator(int)) do
  agg.copy(r, l);
```

Performance (MiB/s)			
config	**bulk**	**unordered**	**aggregated**
Cray XC	8000.0	510.0	2275.0
Cray CS	6000.0	2.0	1850.0

Message Aggregation Implementation

```
config const bufferSize = 4096;


/*
 * Aggregates copy(ref dst, src). Optimized for when src is local.
 * Not parallel safe and is expected to be created on a per-task basis
 * High memory usage since there are per-destination buffers
 */
record DstAggregator {

  type elemType;
  var buffer: [LocaleSpace][0..#bufferSize] (addr, elemType);
  var bufferIdxs: [LocaleSpace] int;
```


Message Aggregation Implementation

```
inline proc copy(ref dst: elemType, srcVal: elemType) {
  // Get the locale of dst and the local address on that locale
  const loc = dst.locale.id;
  const dstAddr = getLocalAddr(dst);


  // Get our current index into the buffer for dst's locale
  ref bufferIdx = bufferIdxs[loc];


  // Buffer the address and desired value
  buffer[loc][bufferIdx] = (dstAddr, srcVal);
  bufferIdx += 1;


  // If full, flush
  if bufferIdx == bufferSize then
    flushBuffer(loc, bufferIdx);
}
```


Message Aggregation Implementation

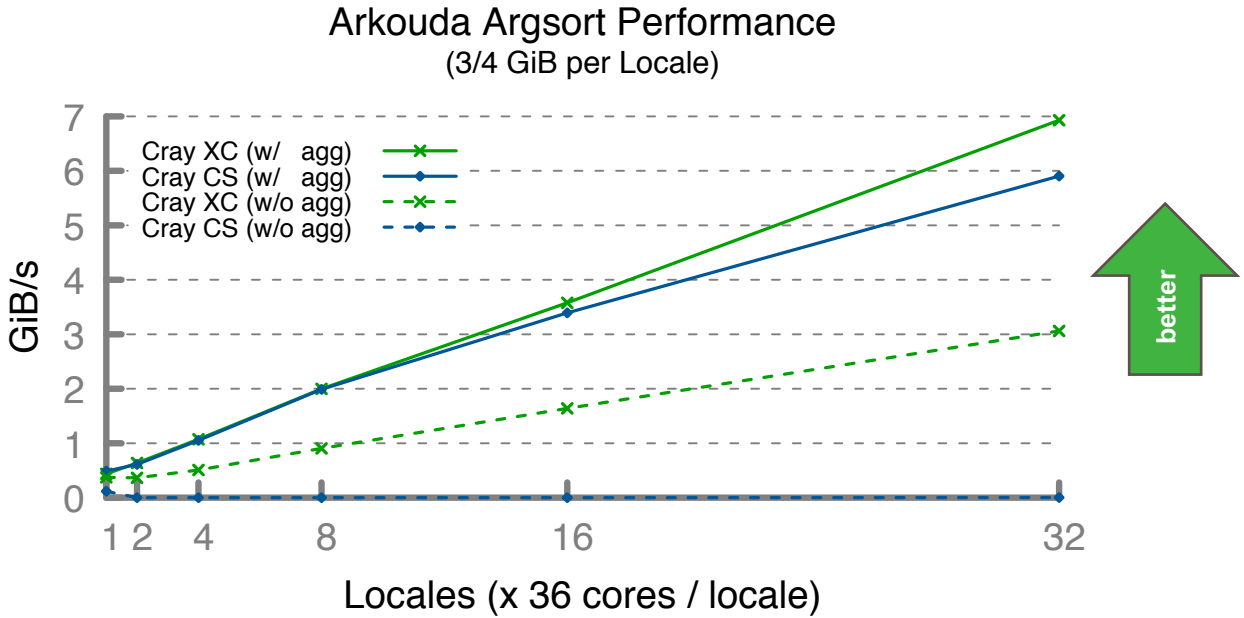
```
proc flushBuffer(loc: int, ref bufferIdx) {

  // Migrate execution to the remote node
  on Locales[loc] {
    // GET the buffered dst addrs and src values, and assign
    var localBuffer = buffer[loc][0..#bufferIdx];
    for (dstAddr, srcVal) in localBuffer do
      dstAddr.deref() = srcVal;
  }
  bufferIdx = 0;

}
```

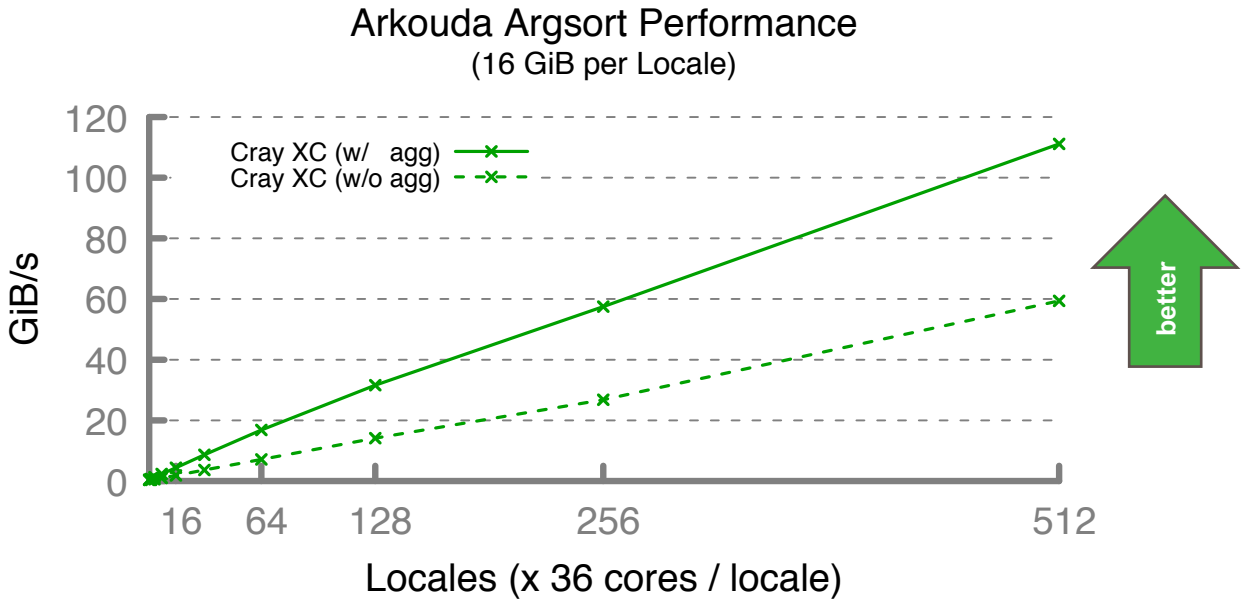

Aggregated Argsort Performance

- Aggregation improved performance by more than 2x on XC and 1000x on CS



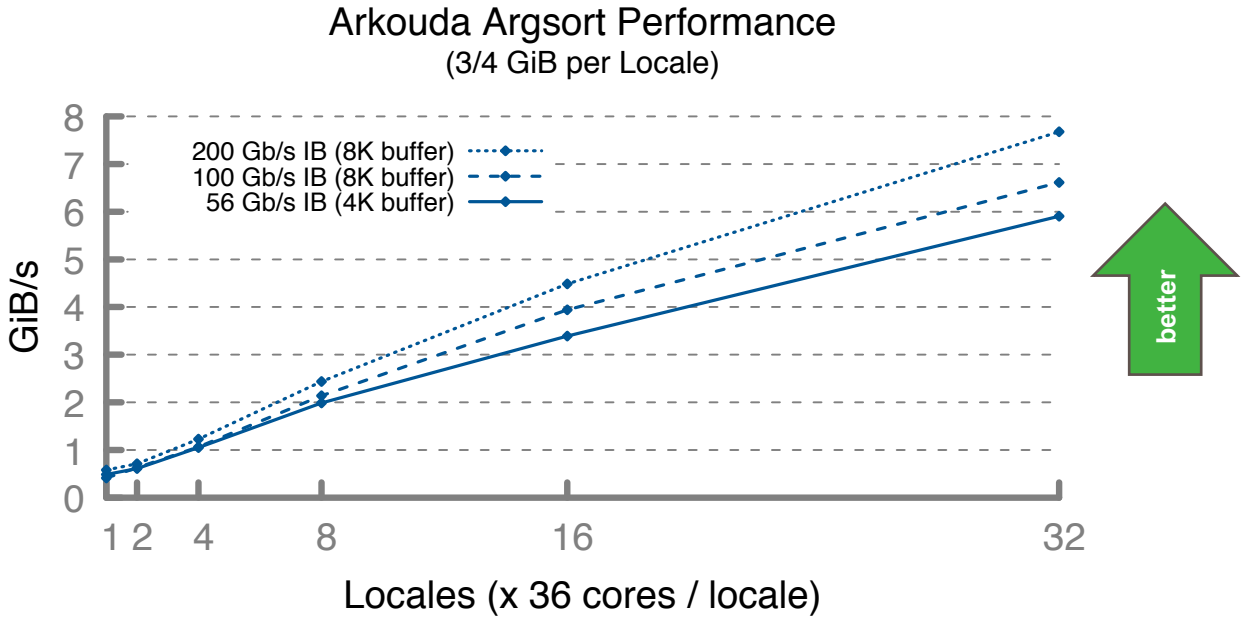
Aggregated Argsort Performance

- Aggregation has good scalability despite simple design



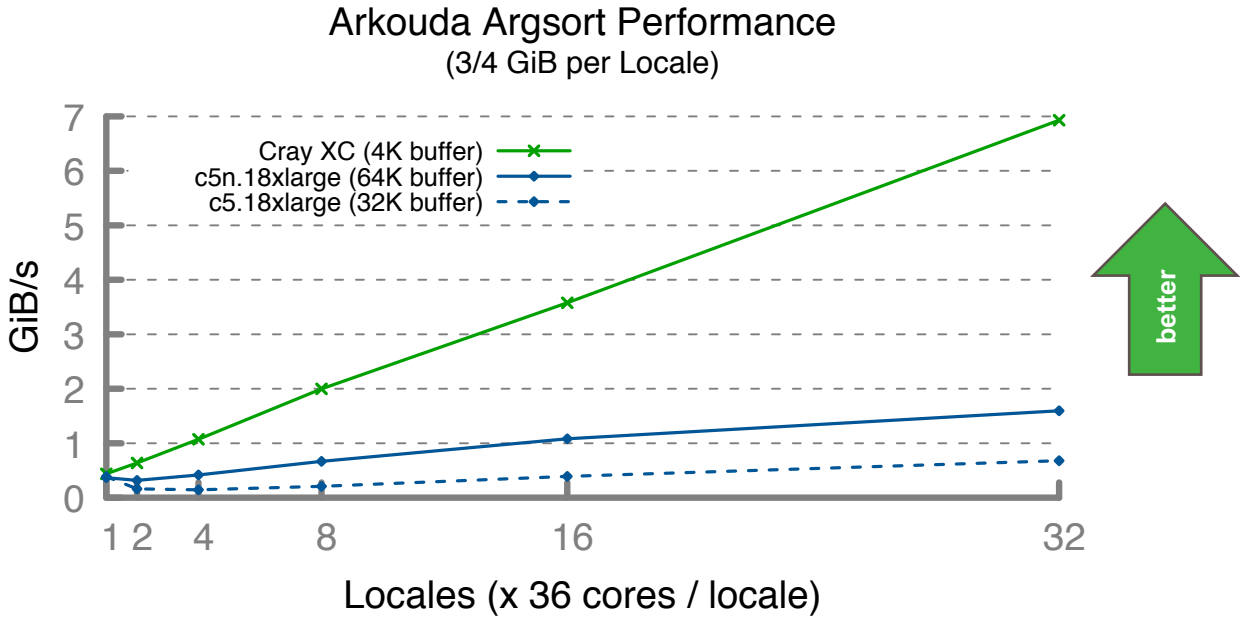
InfiniBand Performance

- Cray CS performance with 56, 100, and 200 Gb/s InfiniBand networks
  - Collected using 36-cores (200 Gb/s has 48 cores, but only 36 used)



Ethernet Performance

- AWS performance with 25 and 100 Gb/s Ethernet networks
  - Collected on 36-core c5.18xlarge and c5n.18xlarge w/ Elastic Fabric Adapter



More Information

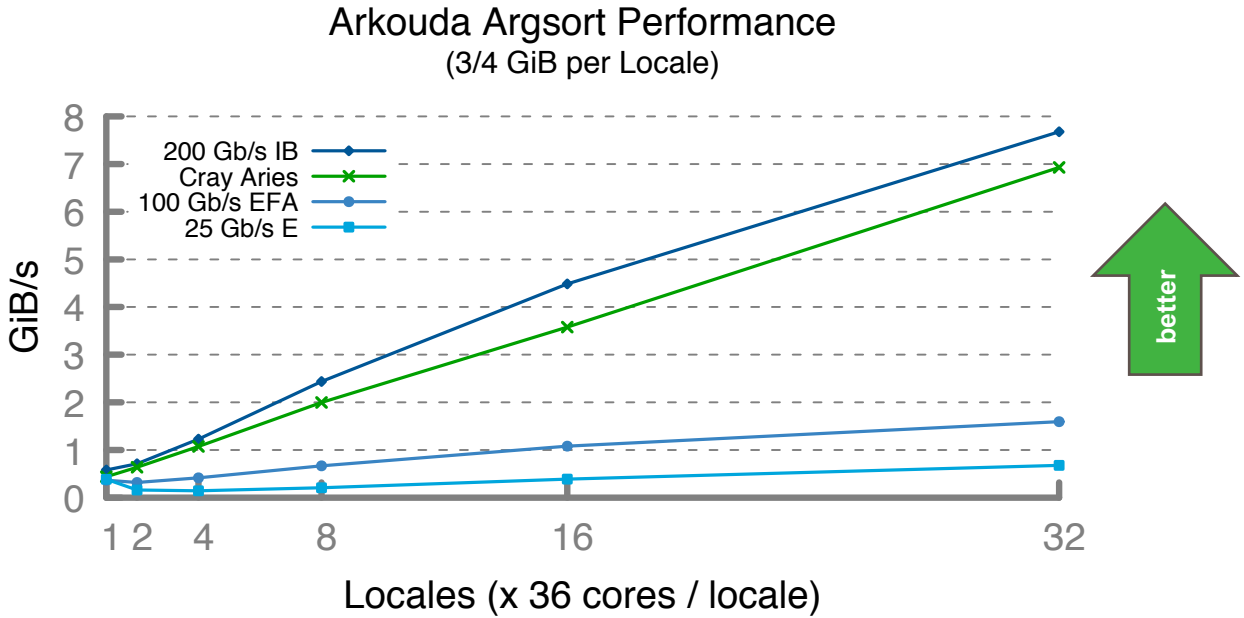
- For more information about other Arkouda optimizations see:
  - https://chapel-lang.org/releaseNotes/1.21/05-user-opt.pdf
- For more information about recent Chapel performance optimizations see:
  - https://chapel-lang.org/releaseNotes/1.21/04-perf-opt.pdf
- For general information about Chapel's performance and tuning tips see:
  - https://chapel-lang.org/performance.html

Future Work

- Collect Dask performance results
  - We have only anecdotal evidence that Arkouda is much faster
- Optimize aggregation performance and reduce memory footprint
  - Autotune buffer size on startup
  - Aggregate within a node before sending over the network
- Tune performance more for commodity networks and newer HPC networks
  - Ethernet and modern InfiniBand performance seem low
- Add aggregation to Chapel's standard library
  - Enable arbitrary message aggregation, not just copy aggregation

Summary

- Message aggregation enables a portable high performance sort in Arkouda
  - Significant performance optimization opportunities remain



FORWARD LOOKING STATEMENTS

This presentation may contain forward-looking statements that involve risks, uncertainties and assumptions. If the risks or uncertainties ever materialize or the assumptions prove incorrect, the results of Hewlett Packard Enterprise Company and its consolidated subsidiaries ("Hewlett Packard Enterprise") may differ materially from those expressed or implied by such forward-looking statements and assumptions. All statements other than statements of historical fact are statements that could be deemed forward-looking statements, including but not limited to any statements regarding the expected benefits and costs of the transaction contemplated by this presentation; the expected timing of the completion of the transaction; the ability of HPE, its subsidiaries and Cray to complete the transaction considering the various conditions to the transaction, some of which are outside the parties' control, including those conditions related to regulatory approvals; projections of revenue, margins, expenses, net earnings, net earnings per share, cash flows, or other financial items; any statements concerning the expected development, performance, market share or competitive performance relating to products or services; any statements regarding current or future macroeconomic trends or events and the impact of those trends and events on Hewlett Packard Enterprise and its financial performance; any statements of expectation or belief; and any statements of assumptions underlying any of the foregoing. Risks, uncertainties and assumptions include the possibility that expected benefits of the transaction described in this presentation may not materialize as expected; that the transaction may not be timely completed, if at all; that, prior to the completion of the transaction, Cray's business may not perform as expected due to transaction-related uncertainty or other factors; that the parties are unable to successfully implement integration strategies; the need to address the many challenges facing Hewlett Packard Enterprise's businesses; the competitive pressures faced by Hewlett Packard Enterprise's businesses; risks associated with executing Hewlett Packard Enterprise's strategy; the impact of macroeconomic and geopolitical trends and events; the development and transition of new products and services and the enhancement of existing products and services to meet customer needs and respond to emerging technological trends; and other risks that are described in our Fiscal Year 2018 Annual Report on Form 10-K, and that are otherwise described or updated from time to time in Hewlett Packard Enterprise's other filings with the Securities and Exchange Commission, including but not limited to our subsequent Quarterly Reports on Form 10-Q. Hewlett Packard Enterprise assumes no obligation and does not intend to update these forward-looking statements.