Accelerating Data Analytics with Arkouda on GPUs

Chapel Implementers and Users Workshop, 2 June 2023

Josh Milthorpe,
Brett Eiffert, and Jeffrey S. Vetter

ORNL Advanced Computing Systems Research, milthorpejj@ornl.gov

ORNL is managed by UT-Battelle LLC for the US Department of Energy

This research used resources of the Experimental Computing Laboratory (ExCL) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Accelerating Arkouda with GPUs

αρκούδα
massive scale
data science

https://github.com/Bears-R-Us/arkouda

- Arkouda promises 'HPC-enabled exploratory data analytics'
- Compute on large data → memory bandwidth

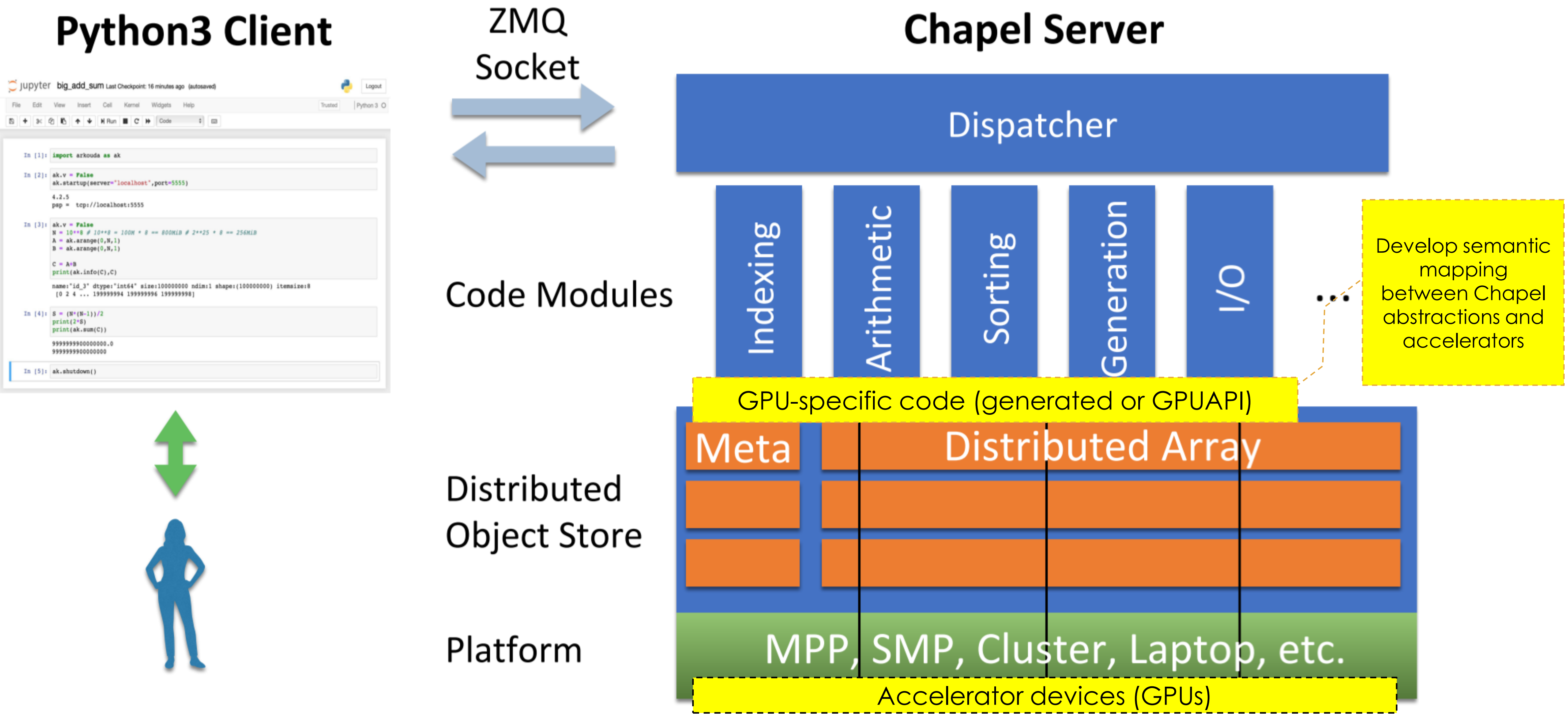
	CPU-DRAM	GPU-HBM
Summit (2018)	340 GB/s	2,700 GB/s
Frontier (2022)	205 GB/s	13,080 GB/s

- Challenges:
  - algorithmic portability
  - memory management
  - programmability

Carlos Jones/ORNL, CC BY 2.0 via Wikimedia Commons

OLCF at ORNL, CC BY 2.0 via Wikimedia Commons

Arkouda Architecture



Chapel GPUAPI

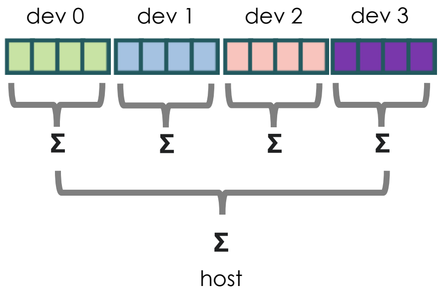
- Georgia Tech-developed framework abstracting over GPU programming models (CUDA, HIP, DPC++, SYCL)
- `GPUIterator`: exposing parallelism for kernel launch
- `GPUAPI`: device and memory management
  - low-level: C-interoperability wrappers around device functions
  - mid-level: `GPUArray` to manage memory allocation, transfer
  - there is no high-level

OAK RIDGE National Laboratory

Example: Sum on GPU (mid-level GPUAPI)

$$\sum_{i=0}^{n-1} A_i$$

`ak_arr.sum()`



Device memory allocation

Data transfer

Synchronization

```
use GPUIterator;
use GPUAPI;

extern proc launchSum(devInPtr: c_void_ptr, devOutPtr:
c_void_ptr, n: int): etype;

proc sum(A: [?aDom] ?etype) {
  var deviceSum: [0..#nGPUs] etype;
  var sumCallback = lambda(lo: int, hi: int, n: int) {
    var devA = new GPUArray(A.localSlice(lo .. hi));
    var devOut = new GPUArray(deviceSum[deviceId]);
    var deviceId: int(32);
    GetDevice(deviceId);
    devA.toDevice();
    launchSum(devA.dPtr(), devOut.dPtr(), n);
    DeviceSynchronize();
    devOut.fromDevice();
  };
  forall i in GPU(A.localSubdomain(), sumCallback) { }
  return (+ reduce deviceSum);
}
```

CUDA / OpenCL

Arkouda GPU Device Cache

- Common pattern
  - new GPUArray for local chunk
  - copy host-to-device
  - kernel launch[es]
  - [copy device-to-host]
- Where possible, leave arrays on GPU between operations

MultiTypeSymEntry.chpl

```
class SymEntry : GenSymEntry {
  proc createDeviceCache() {
  class DeviceCache {
    var isCurrent = false;
    /* Range of data for each GPU device */
    var deviceChunks: [gpuDevices] range;
    /* GPU arrays for each device */
    var deviceArrays: [gpuDevices] shared GPUArray?;
    proc toDevice(deviceId) {
      if (!isCurrent) {
        deviceArrays[deviceId]!.toDevice();
        isCurrent = true;
      }
    }
    proc fromDevice(deviceId) { … }
  }
}
```

Example: Histogram on GPU (Device Cache)

$$\sum_{i=0}^{n-1} A_i$$

`ak_arr.sum()`

Device memory allocation

Data transfer

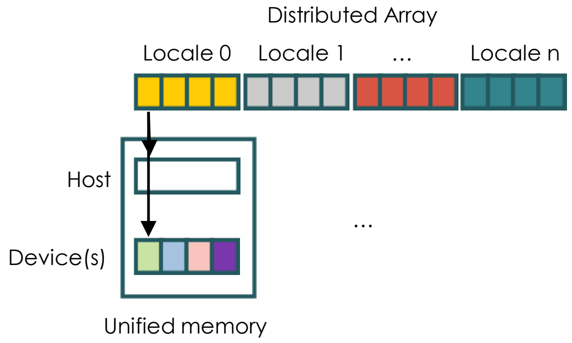
Synchronization

```
use GPUIterator;
use GPUAPI;

extern proc launchSum(devInPtr: c_void_ptr, devOutPtr:
c_void_ptr, n: int): etype;

proc sum(e: SymEntry) {
  e.createDeviceCache(); // idempotent
  var deviceSum: [0..#nGPUs] e.etype;
  var sumCallback = lambda(lo: int, hi: int, n: int) {
    var devOut = new GPUArray(deviceSum[deviceId]);
    var deviceId: int(32);
    GetDevice(deviceId);
    e.toDevice(deviceId); // idempotent
    launchSum(e.getDeviceArray(deviceId).dPtr(),
devOut.dPtr(), n);
    DeviceSynchronize();
    devOut.fromDevice();
  };
  forall i in GPU(e.a.localSubdomain(), sumCallback) { }
  return (+ reduce deviceSum);
}
```

GPUUnifiedDist: Arkouda Arrays in Shared Virtual Memory

- Host and device(s) share pointers to a single unified memory space
- Any access to memory that is currently in a different physical memory will result in a page fault, handled transparently with hardware support
- User-defined Chapel distribution `GPUUnifiedDist`
  - based on `BlockDist`
  - allocates memory for `LocGPUUnifiedArr` using `makeArrayFromPtr(umemPtr, …)`

```
module SymArrayDmap …
proc makeDistDom(size:int, param GPU:bool = false) where GPU == true {
  select MyDmap {
    when Dmap.blockDist {
      return {0..#size} dmapped GPUUnified(…);
    }
    …
  }
}
```



https://github.com/milthorpe/chapel-gpu
https://github.com/milthorpe/arkouda

Example: Histogram on GPU (unified memory)

$$\sum_{i=0}^{n-1} A_i$$

`ak_arr.sum()`

```
use GPUIterator;
use GPUAPI;

extern proc launchSum(devInPtr: c_void_ptr, devOutPtr:
c_void_ptr, n: int): etype;

proc cubSum(ref e: SymEntry) where e.GPU == true {
  var deviceSum: [0..#nGPUs] e.etype;
  var sumCallback = lambda(lo: int, hi: int, n: int) {
    var devOut = new GPUArray(deviceSum[deviceId]);
    var deviceId: int(32);
    GetDevice(deviceId);
    e.prefetchLocalDataToDevice(lo, hi, deviceId);
    launchSum(e.c_ptrToLocalData(lo), devOut.dPtr(), n);
    DeviceSynchronize();
    devOut.fromDevice();
  };
  forall i in GPU(e.a.localSubdomain(), sumCallback) { }
  return (+ reduce deviceSum);
}
```

Device memory allocation

Data transfer

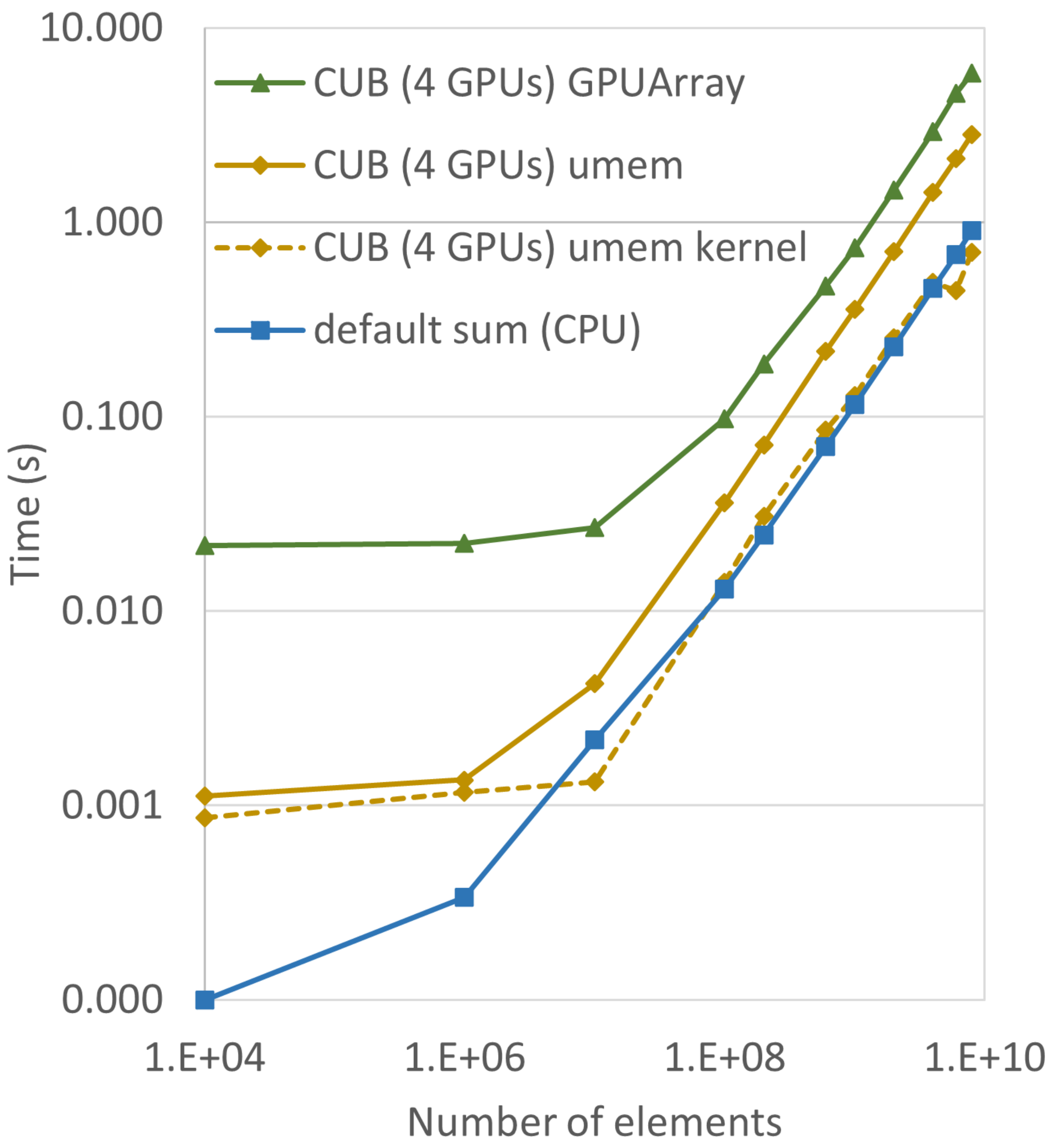
Synchronization

Experimental Evaluation

- Evaluation platform: NVIDIA DGX workstation
  - 2 × 20-core Intel Xeon E5-2698s @ 2.2GHz
  - 256GiB of DRAM
  - 4 × Tesla V100 GPUs with 32 GiB HBM
  - Chapel 1.30
  - NVHPC toolkit v22.11 (CUDA v11.8)
  - CUDA driver version 530.30.02
- Timing server-side Arkouda Chapel code directly (not from Python client)
  - Doesn't allow batching of communications

Reduction

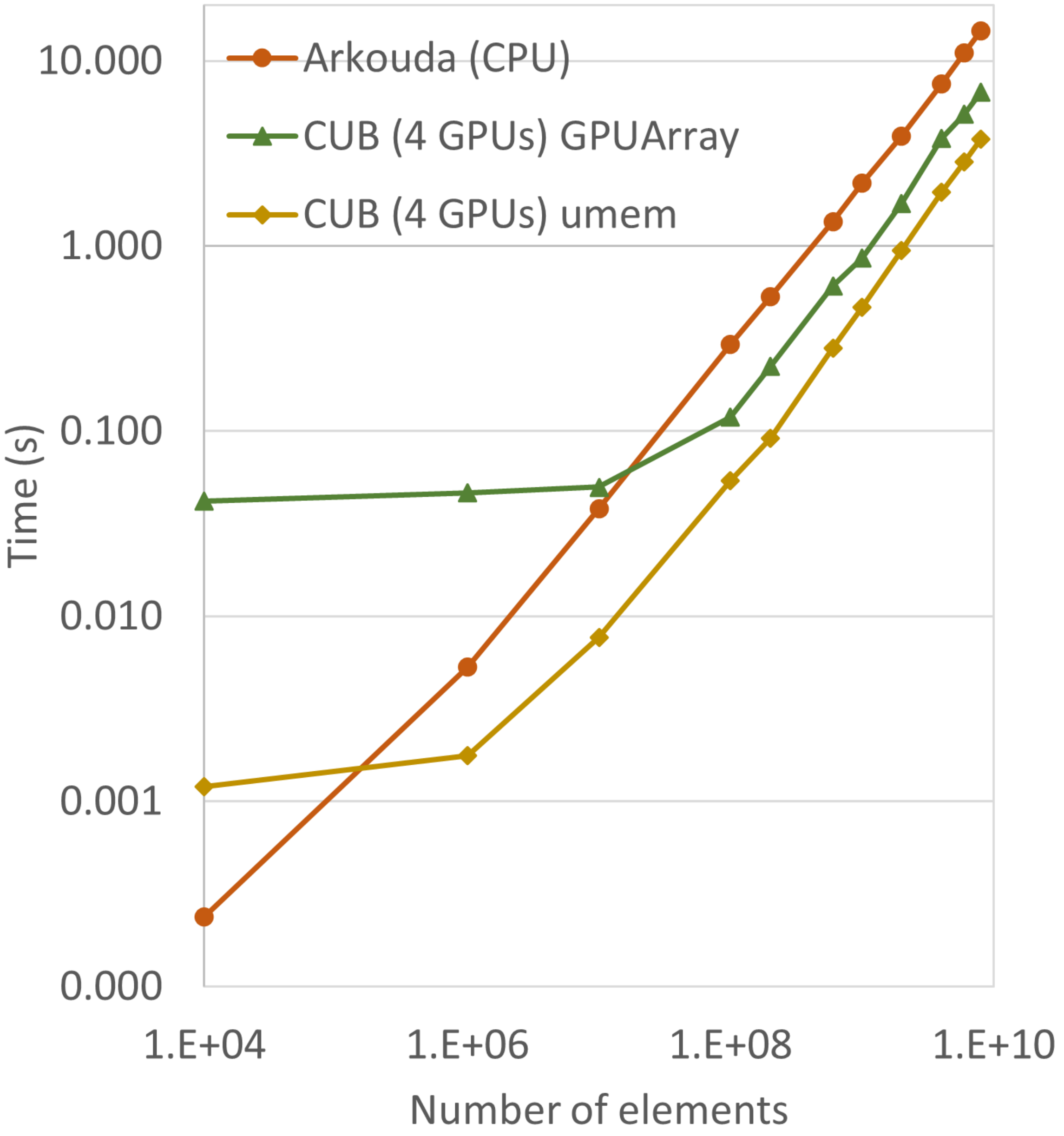
- GPUArray
  `DeviceCache`
- umem
  `GPUUnifiedDist`
- Kernel:
  - CUB library
    `DeviceReduce::Sum`
  - NCCL
    `ncclReduce`




```
ak_arr.sum()
```

Histogram

- Arkouda (CPU)
  `histogramGlobalAtomic`
- Kernel:
  - CUB library
    `DeviceHistogram::HistogramEven`
  - NCCL
    `ncclAllReduce`

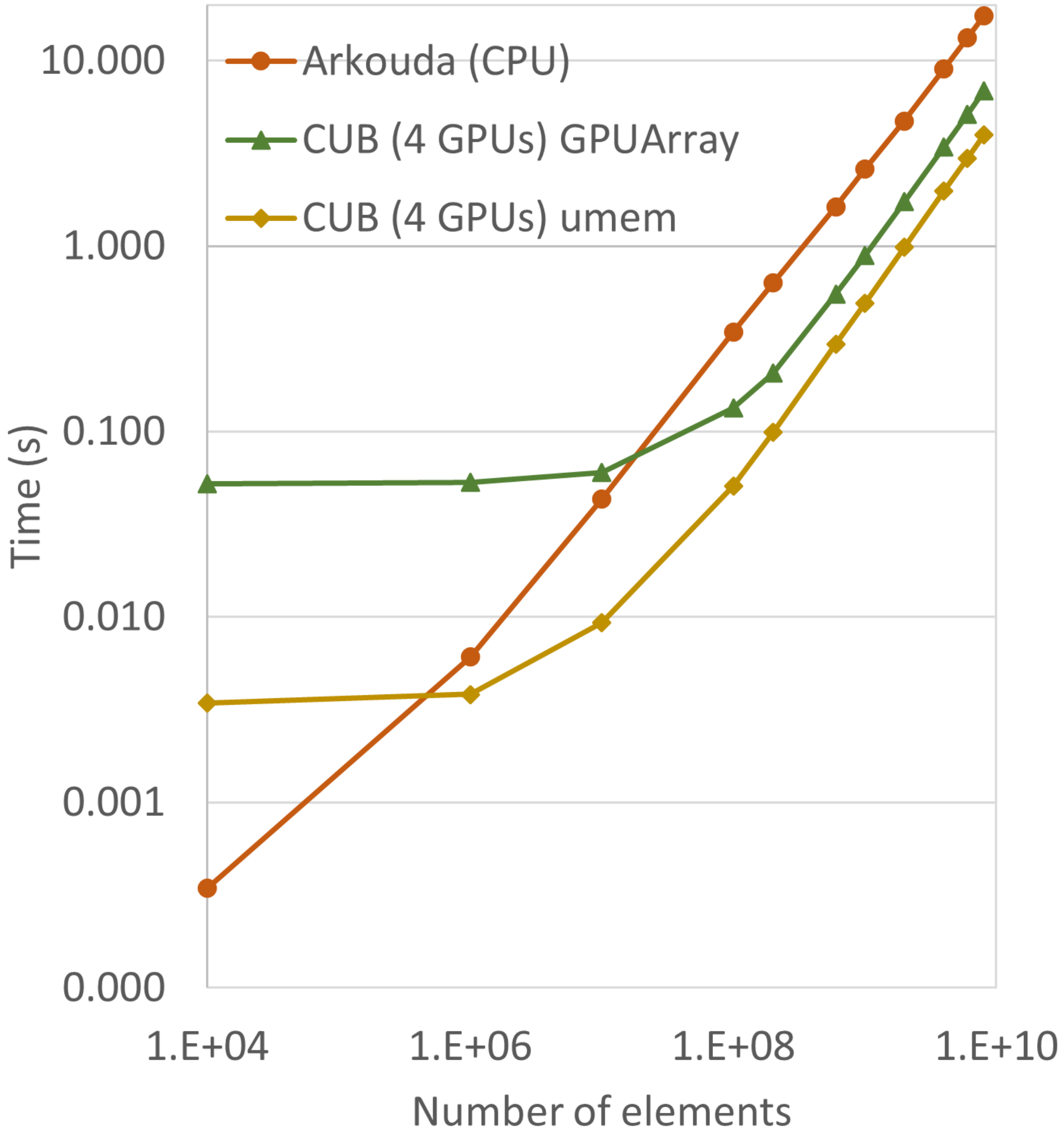



```
ak.histogram(A, sqrt(A.size))
```

Chained Operations

- DeviceCache / Unified Memory avoids multiple host-device transfers

```
A.sum()
A.min()
A.max()
ak.histogram(A, sqrt(A.size))
```



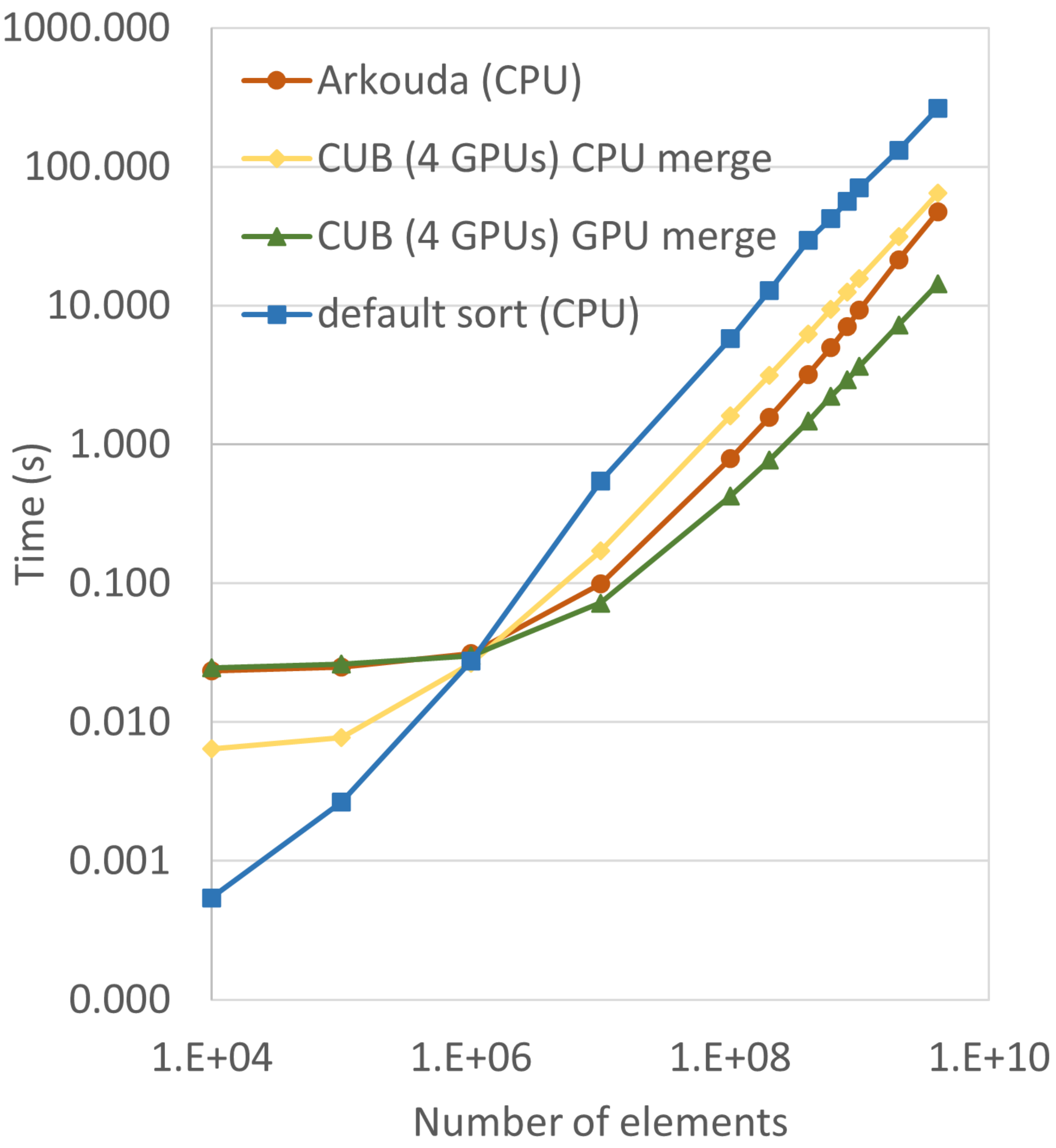
Oak Ridge National Laboratory

Milthorpe: Accelerating Data Analytics with Arkouda on GPUs (CHIUW 2023)

Sort

- Arkouda (CPU)
  `radixSortLSD_keys`
- Kernel:
  - CUB library
    `DeviceRadixSort::SortKeys`
  - merge on CPU:
    K-way merge
  - GPU merge:
    peer-to-peer swap and merge

Tobias Maltenberger, Ivan Ilic, Ilin Tolovski, and Tilmann Rabl. (2022) *Evaluating multi-GPU sorting with modern interconnects*. Intl. Conf. Management of Data. https://doi.org/10.1145/3514221.3517842



`ak_df.sort_values()`

Summary and Future Work

- Chapel GPUAPI combined with unified memory can support productive, high-performance development of GPU-accelerated data analytics
  - algorithmic portability still a challenge
- Future:
  - Application Workflows: real data analytics pipelines
    - e.g. astronomical image/spectroscopic post-processing and analysis
  - Port to AMD GPUs (HIP/ROCm)
  - Chapel GPU code generation

This research used resources of the Experimental Computing Laboratory (ExCL) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.