Hewlett Packard Enterprise

CUG2023

CRAY USER GROUP | SUSTAINABLE EXASCALE

Helsinki, Finland

ARKOUDA: A HIGH-PERFORMANCE DATA ANALYTICS FRAMEWORK

Michelle Strout with Scott Bachman, Brad Chamberlain, Ben McDonald, and Elliot Ronaghan

Cray User Group (CUG)
May 10, 2023

MOTIVATION FOR ARKOUDA

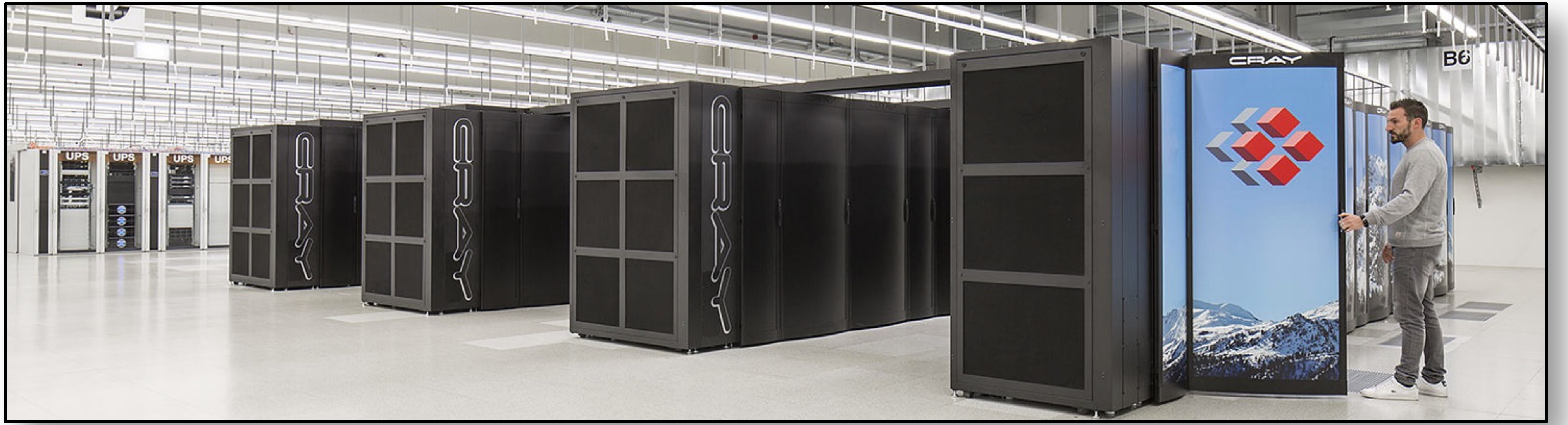
Motivation: Say you have…

…a bunch of Python programmers

…HPC-scale data science problems to solve

…access to HPC systems

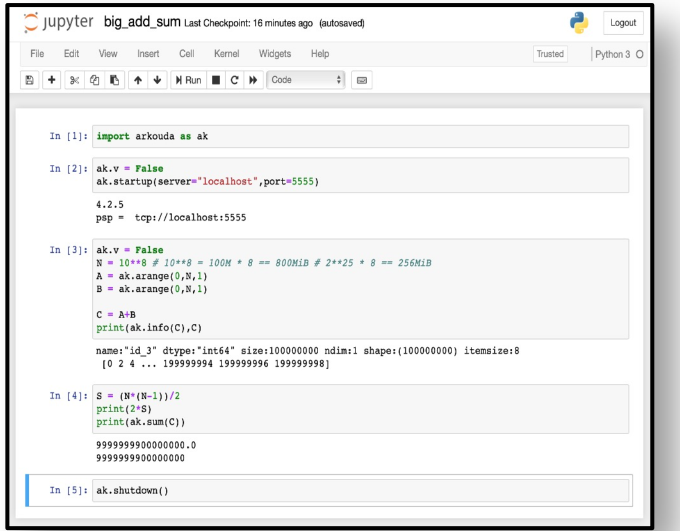
https://www.cscs.ch/computers/piz-daint/



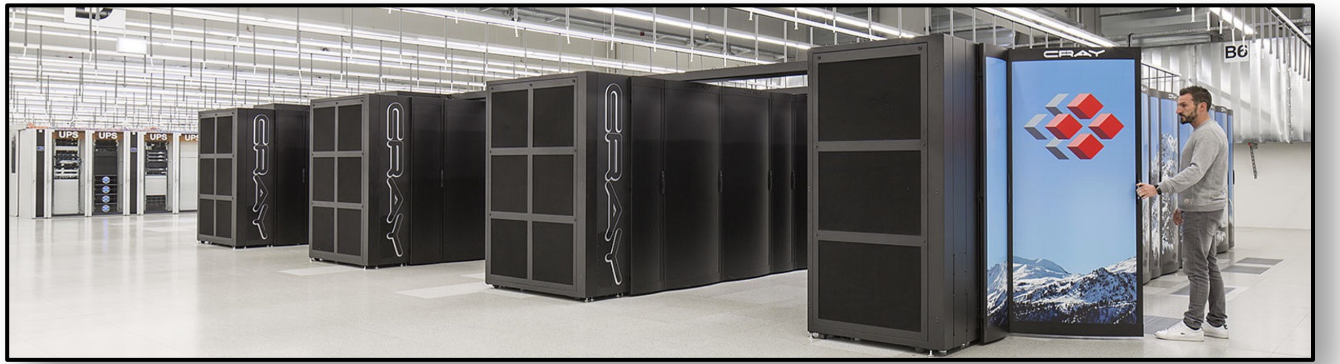
How can you enable your Python programmers to solve large problems?

ARKOUDA'S HIGH-LEVEL APPROACH

Arkouda Client
(written in Python)



Arkouda Server (written in Chapel)



Writes Python code in Jupyter
Invoking NumPy operations

EXAMPLE ARKOUDA CODE

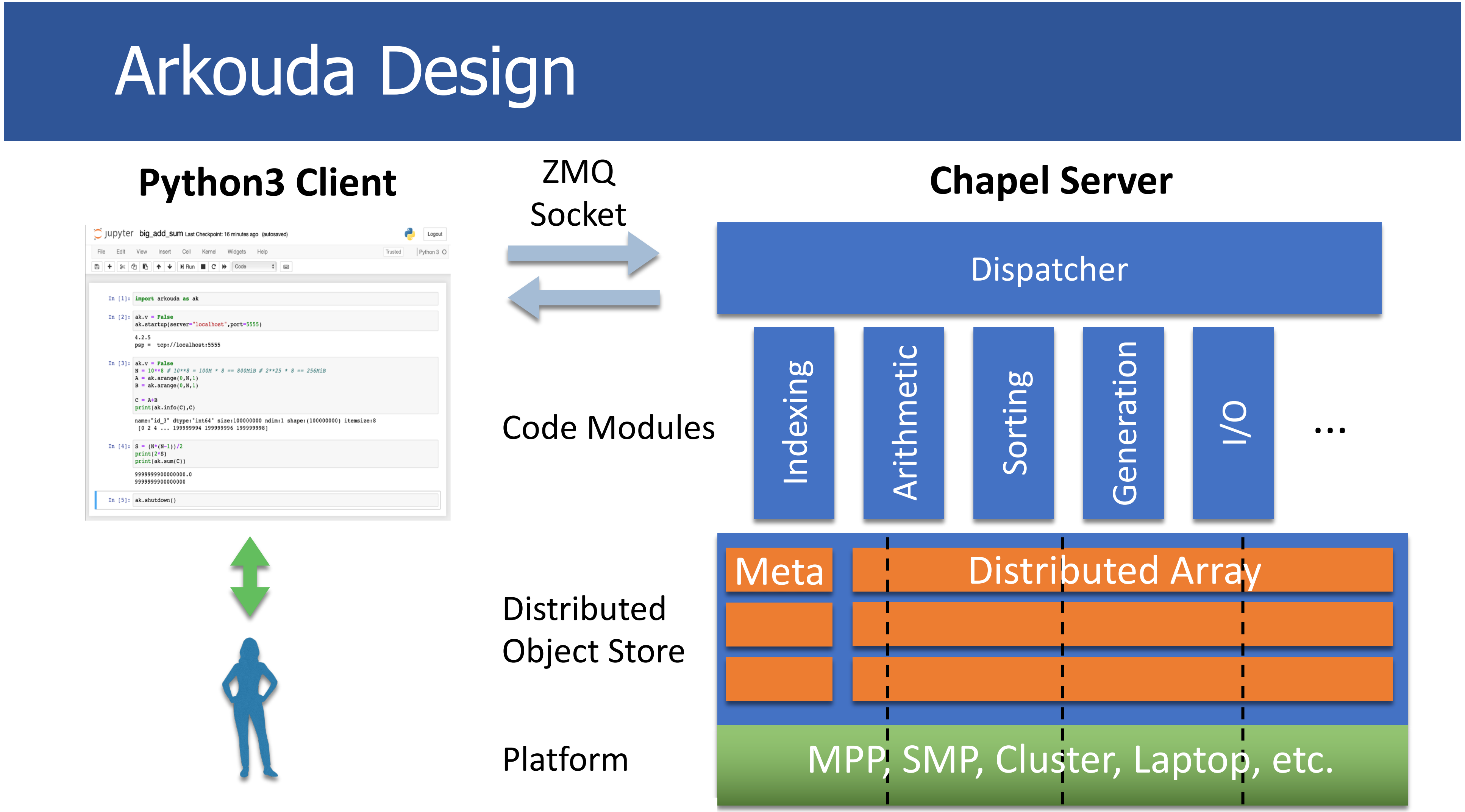
Summing numbers similar to how one would do with NumPy

```
>>> N = 10**6
>>> A = ak.arange(1, N+1, 1)    # creating a large array on server
>>> print(A.sum())              # compute sum and returning result to the Python client
```

Keeping arrays and results on the server

```
# Generate two (server-side) arrays of random integers 0-9
>>> B = ak.randint(0, 10, N)
>>> C = ak.randint(0, 10, N)
>>> D = B * C                   # multiply them on the server
# Print a small representation of the array
# This does NOT move the array to the client
>>> print(D)
>>> minVal = D.min()            # compute min and max and bring over to Python
>>> maxVal = D.max()
>>> print(minVal, maxVal)
```

ARKOUDA BLOCK DIAGRAM



presented by Bill Reus at CHIUW 2020 on May 22, 2020

ARKOUDA DETAILS

- A Python library supporting data science operations at massive scales and interactive rates
  - massive scales = dozens of terabytes
  - interactive rates = operations that run within the human thought loop (i.e., seconds to small numbers of minutes)
  - implemented using a Client-Server model

- Arkouda client library:
  - a normal Python library, written natively in Python
    - available to Python programmers in standard ways (e.g., Jupyter notebooks, Python interpreter)
  - supports a key subset of operations from the standard NumPy and Pandas libraries
    - e.g., numerical operations, reductions, histograms, sorting, groupby, gather/scatter, ...

- Arkouda server back-end:
  - implemented in Chapel
  - key datatype: 1D distributed arrays

Arkouda is the word for "bear" in Greek

ARKOUDA: WHAT, WHO, AND ON WHAT

Arkouda is a framework for interactive, high performance data analytics

- Users can and have created more complex computations in Python with Arkouda
- Modular configuration and build
- Server written in Chapel, thus can be extended to any parallel/distributed computations
- Open-source: https://github.com/Bears-R-Us/arkouda

Creators, maintainers, and users

- Mike Merrill, Bill Reus, et al., US DOD, created it within about 9 months of part time work in consultation with Brad Chamberlain at Cray/HPE in 2019
- Elliot Ronaghan and Ben McDonald from the Chapel team help support it
- Scott Bachman is a visiting climate scientist from NCAR who has been experimenting with it

Systems it has and is being run on

- ~360 node Cray XC (11,320 cores)
- 576 nodes of an HPE Apollo with HDR-100 IB (73,728 cores of AMD Rome)
- 896 nodes of an HPE Cray EX with Slingshot 11 (114,688 cores of AMD Milan)
- Other systems: 12TB HPE Superdome X, Cheyenne (SGI ICE XA and IB), Summit (IBM Power 9 and Nvidia Tesla)

ARKOUDA'S DESIGN

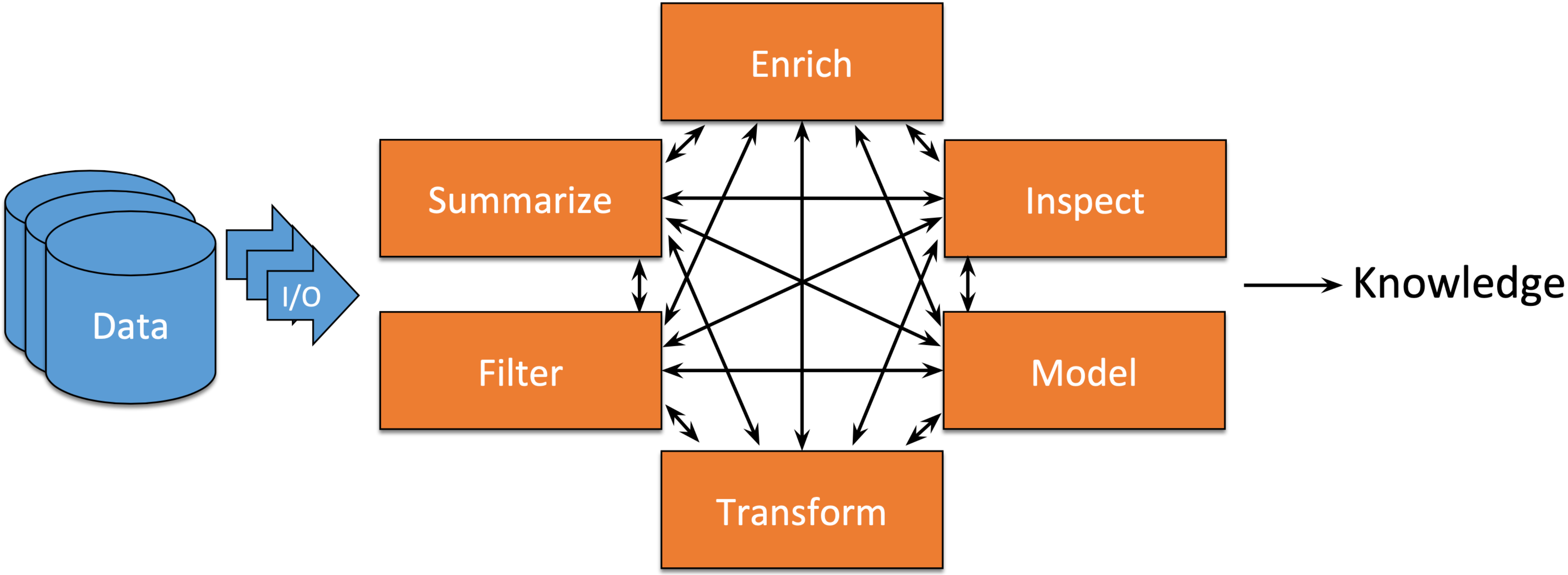
- Some of the reasons given for picking the Chapel programming language
  - High-level language with C-comparable performance
  - Parallelism is a first-class citizen
  - Great distributed array support
  - Portable code: from laptop up to supercomputer
  - Integrates with [distributed] numeric libraries
  - Close to Pythonic (for a statically typed language)
    – provides a gateway for data scientists ready to go beyond Python

```
var D = {1..1000, 1..1000} dmapped Block(...),
    A: [D] real;

forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/1000;
```

DATA SCIENCE DEMANDS INTERACTIVITY

Understanding Physics of Datasets

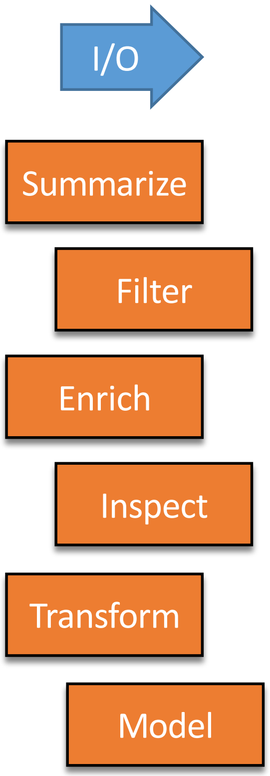


Many names: Exploratory Data Analysis, Data Wrangling, Data Modeling, etc.

presented by Bill Reus at CHIUW 2020 on May 22, 2020

DATA SCIENCE DEMANDS SCALING: REAL WORKFLOW (LATE 2019)

Data Science on 50 Billion Records



Operation	Example	Approx. Time (seconds)
Read from disk	A = ak.read_hdf()	30-60
Scalar Reduction	A.sum()	< 1
Histogram	ak.histogram(A)	< 1
Vector Ops	A + B, A == B, A & B	< 1
Logical Indexing	A[B == val]	1 - 10
Set Membership	ak.in1d(A, set)	1
Gather	B = Table[A]	4 - 120
Get Item	print(A[42])	< 1
Sort Indices by Value	I = ak.argsort(A)	15
Group by Key	G = ak.GroupBy(A)	30
Aggregate per Key	G.aggregate(B, ‘sum’)	10

- A, B are 50 billion-element arrays of 32-bit values
- Timings measured on real data
- Hardware: Cray XC40
  - 96 nodes
  - 3072 cores
  - 24 TB
  - Lustre filesystem

29

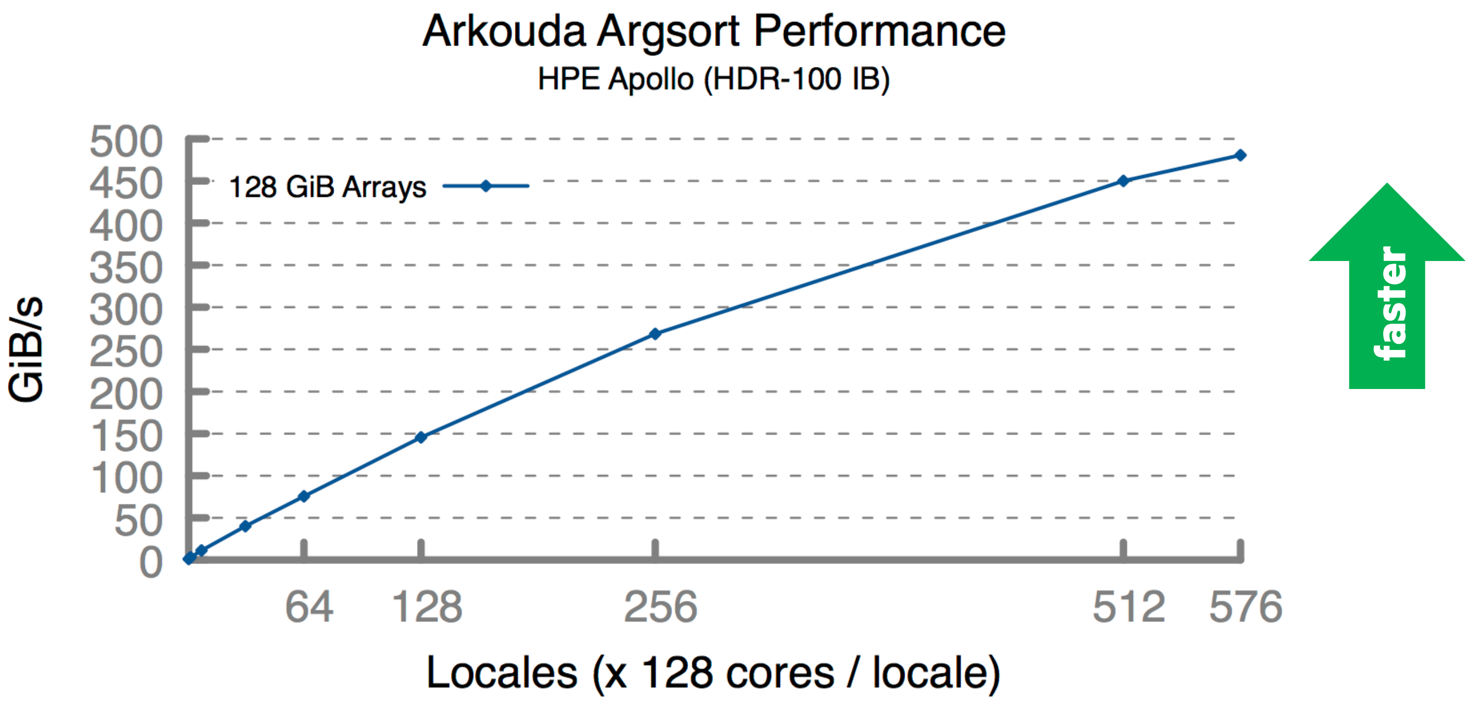
presented by Bill Reus at CHIUW 2020 on May 22, 2020

ARKOUDA PERFORMANCE COMPARED TO NUMPY ON CRAY XC (MAY 2020)

benchmark	NumPy 0.75 GB	Arkouda (serial) 0.75 GB 1 core, 1 node	Arkouda (parallel) 0.75 GB 36 cores x 1 node	Arkouda (distributed) 384 GB 36 cores x 512 nodes
argsort	0.03 GiB/s --	0.05 GiB/s **1.66x**	0.50 GiB/s **16.7x**	55.12 GiB/s **1837.3x**
coargsort	0.03 GiB/s --	0.07 GiB/s **2.3x**	0.50 GiB/s **16.7x**	29.54 GiB/s **984.7x**
gather	1.15 GiB/s --	0.45 GiB/s 0.4x	13.45 GiB/s **11.7x**	539.52 GiB/s **469.1x**
reduce	9.90 GiB/s --	11.66 GiB/s **1.2x**	118.57 GiB/s **12.0x**	43683.00 GiB/s **4412.4x**
scan	2.78 GiB/s --	2.12 GiB/s 0.8x	8.90 GiB/s **3.2x**	741.14 GiB/s **266.6x**
scatter	1.17 GiB/s --	1.12 GiB/s 1.0x	13.77 GiB/s **11.8x**	914.67 GiB/s **781.8x**
stream	3.94 GiB/s --	2.92 GiB/s 0.7x	24.58 GiB/s **6.2x**	6266.22 GiB/s **1590.4x**

ARKOUDA ARGSORT: HERO RUN ON HPE APOLLO SYSTEM WITH IB

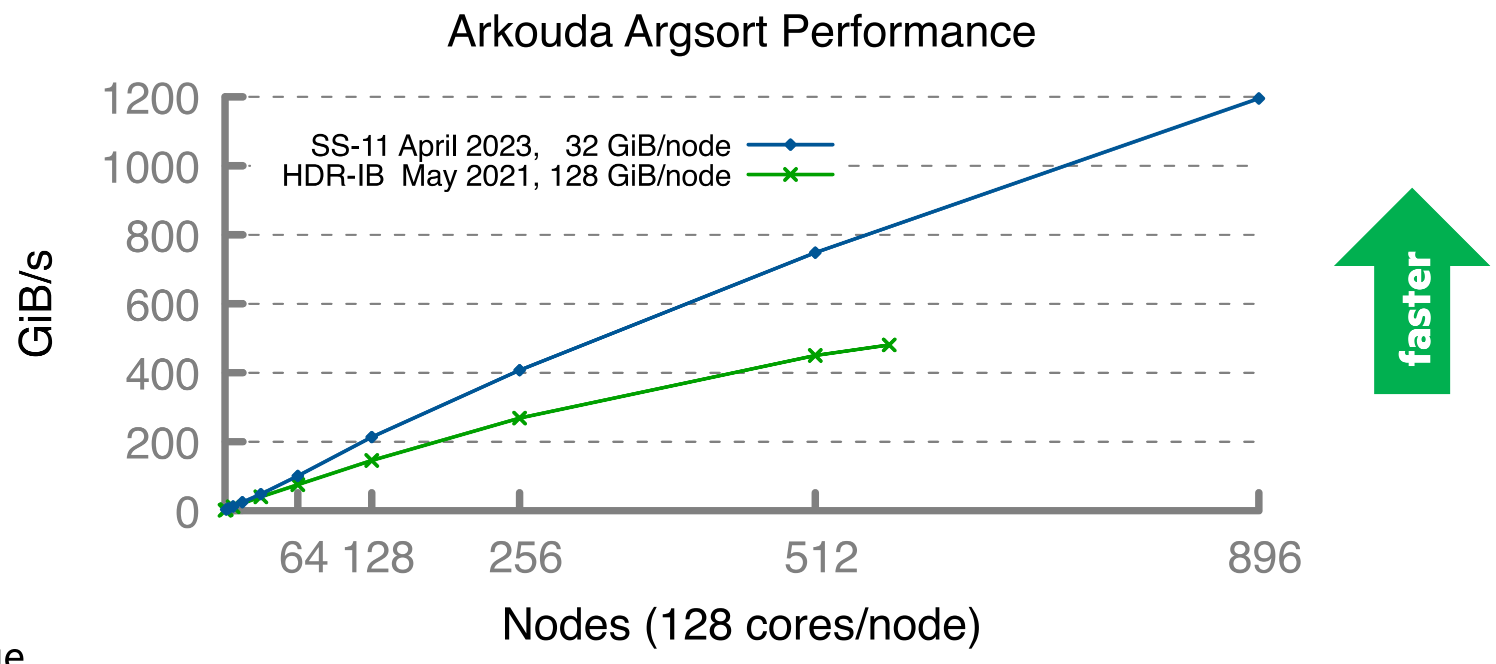
- May 2021 hero run performed on large Apollo system
  - 72 TiB of 8-byte values
  - 480 GiB/s (2.5 minutes elapsed time)
  - used 73,728 cores of AMD Rome
  - ~100 lines of Chapel code



ARKOUDA ARGSORT: HERO RUN ON HPE EX SYSTEM WITH SS-11

- In April 2023, a large HPE Cray EX system with Slingshot-11 set a new record for Arkouda argsort
  - 28 TiB of 8-byte values
  - 1200 GiB/s (24 seconds elapsed time)
  - used 114,688 cores of AMD Milan
  - similar ~100 lines of Chapel code

- Not an apples-to-apples comparison
  - Different network rates
    - Older one was 100 Gbps IB
    - Newer one was 200 Gbps SS-11
  - Different software versions
    - Aggregator optimizations
    - Improvements to the sort: bucket exchange



VISITING SCHOLAR BENCHMARKING VS DASK/NUMPY (FALL 2022)

- Many of Arkouda's capabilities also exist in NumPy and Dask
  - Dask implements many NumPy functions to run in distributed memory
  - The "go-to" library for HPC calculations in Python
  - Not necessarily straightforward to program
    - Manual control of tasks / workers

Hardware: SGI ICE XA (Cheyenne)

- 4,032 nodes
- 145,152 cores
- 64 GB memory/node
- Infiniband

- Small problems done fast – Numpy; Big problems (usually) done fast – Dask

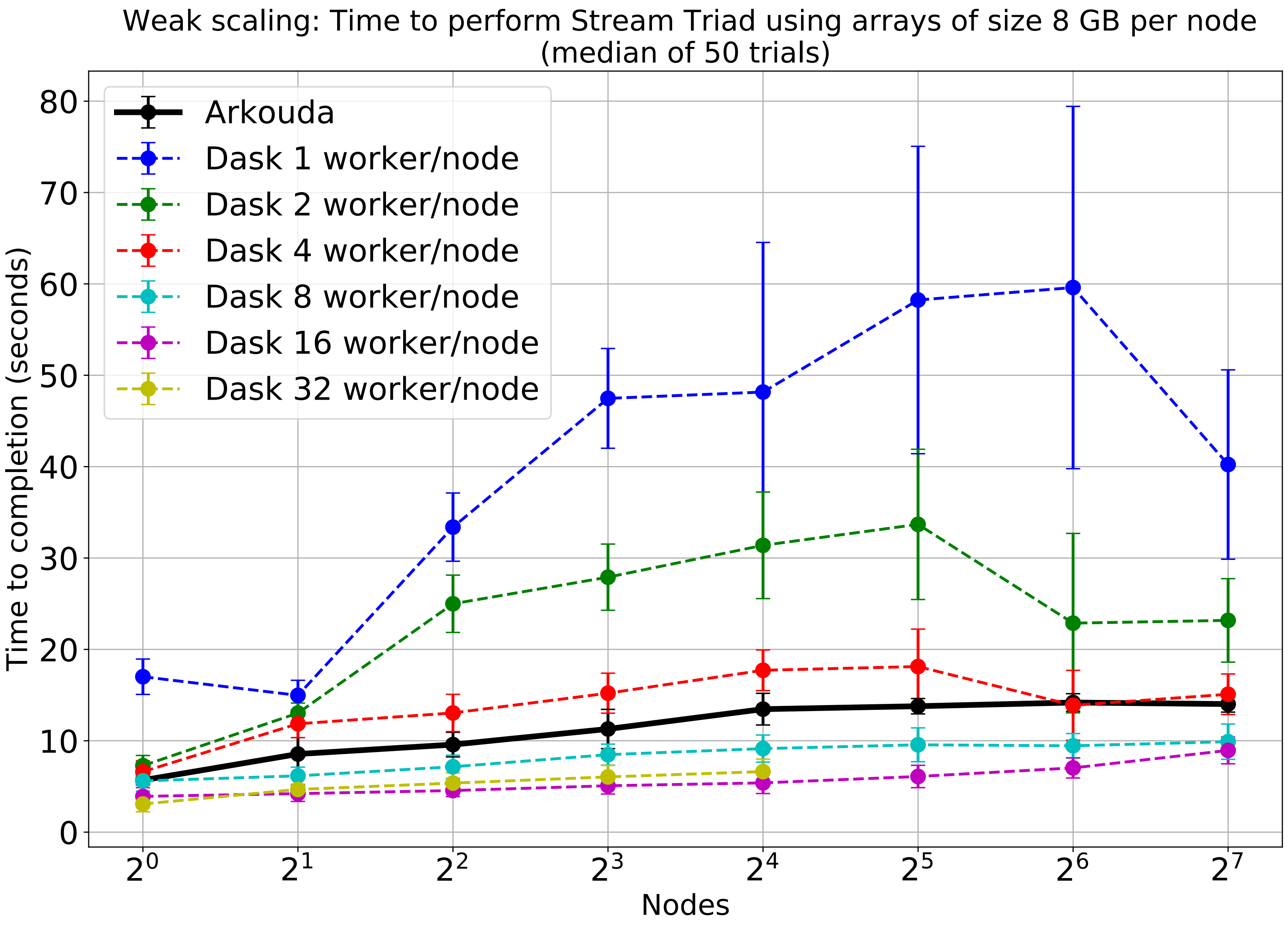
- Problems *at any scale* done fast – Arkouda

- Some of Arkouda's most powerful algorithms do not have analogues in Dask (e.g., parallel argsort)

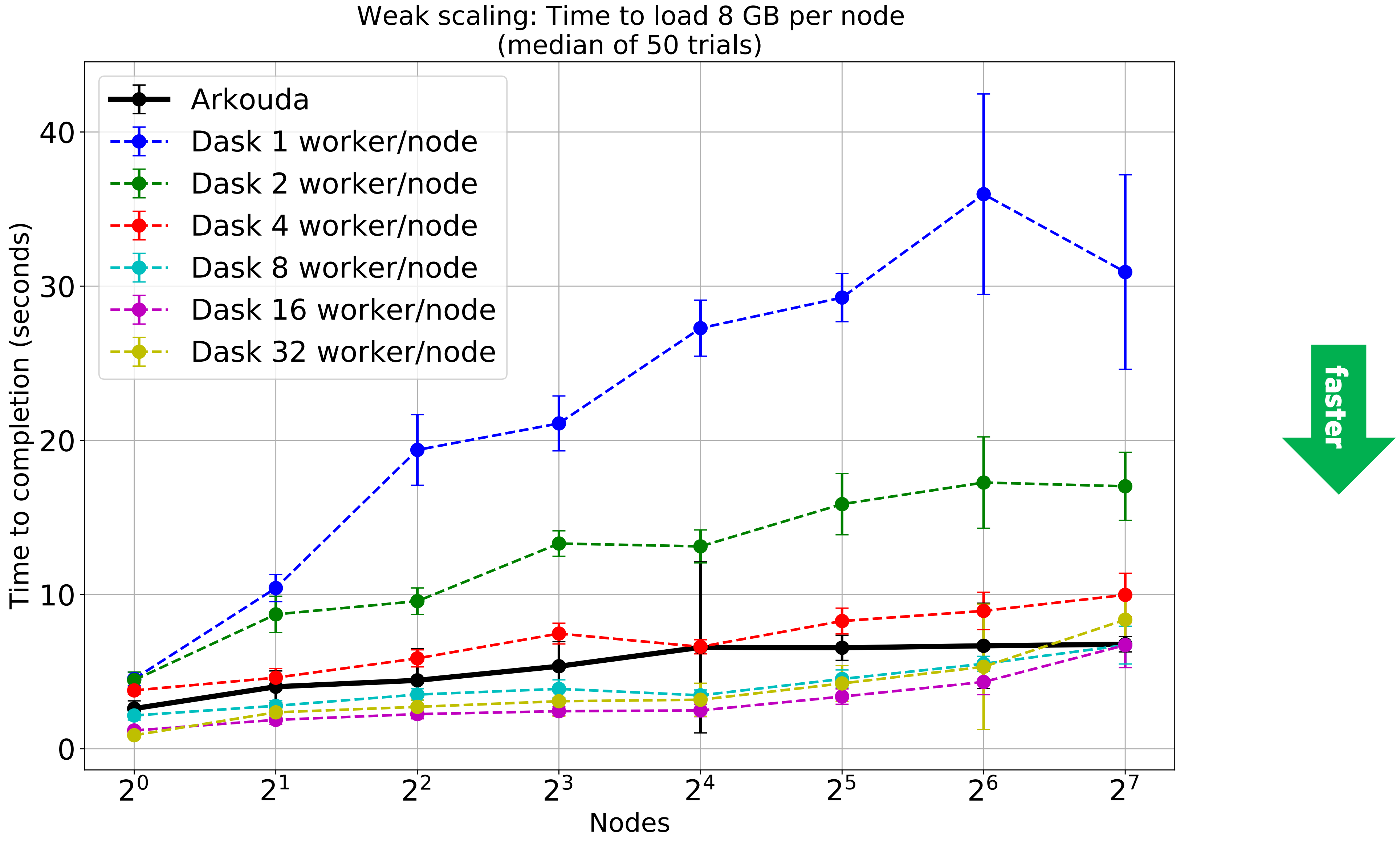
- The following slides show timing comparisons for several key functions
  - Weak scaling (variable node count, variable input size)
  - Chapel 1.27; Dask 2.30.0

DASK VS. ARKOUDA: STREAM TRIAD BENCHMARK

$C = A + \alpha B$

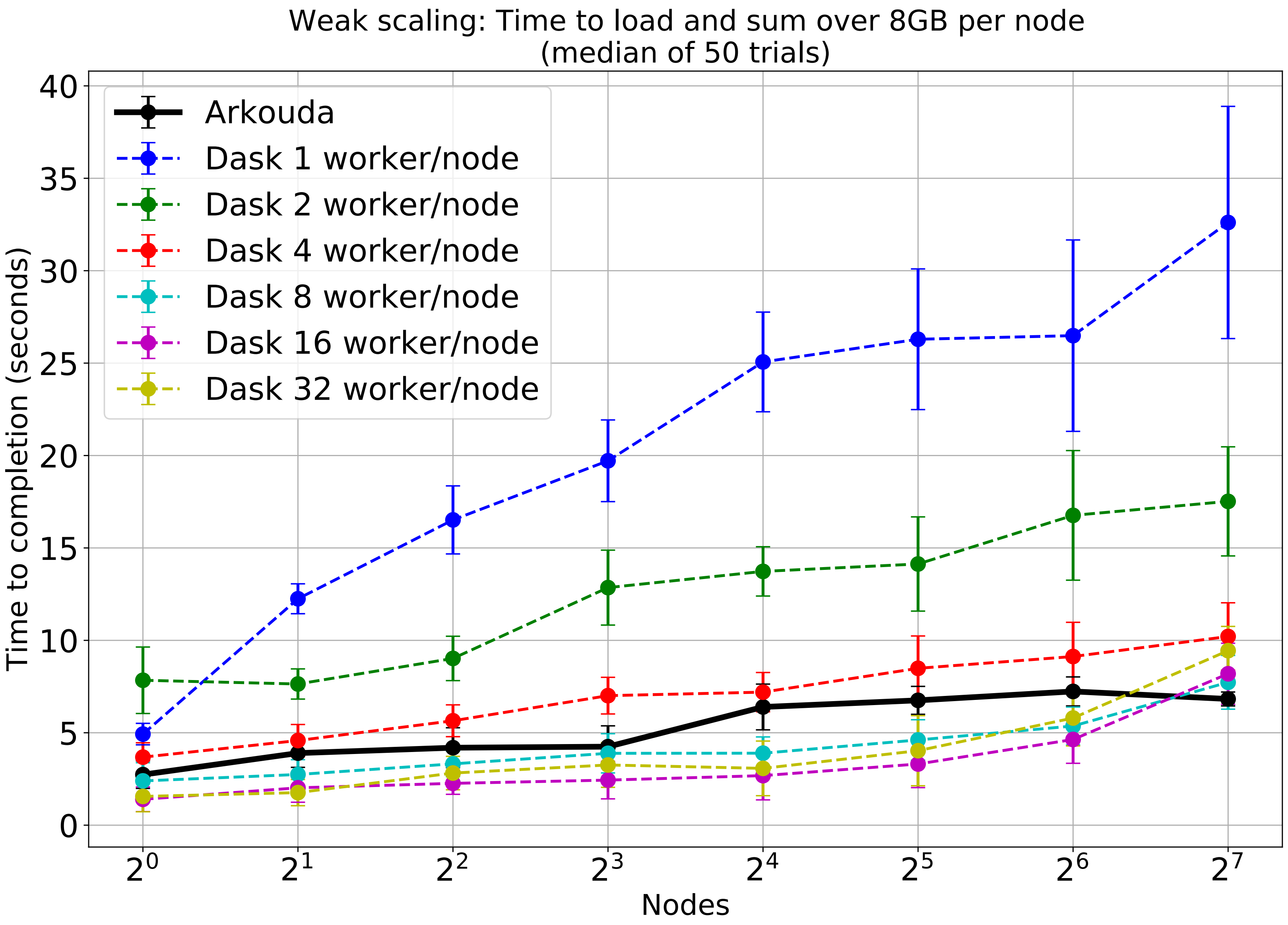


DASK VS. ARKOUDA: LOAD HDF5 BENCHMARK



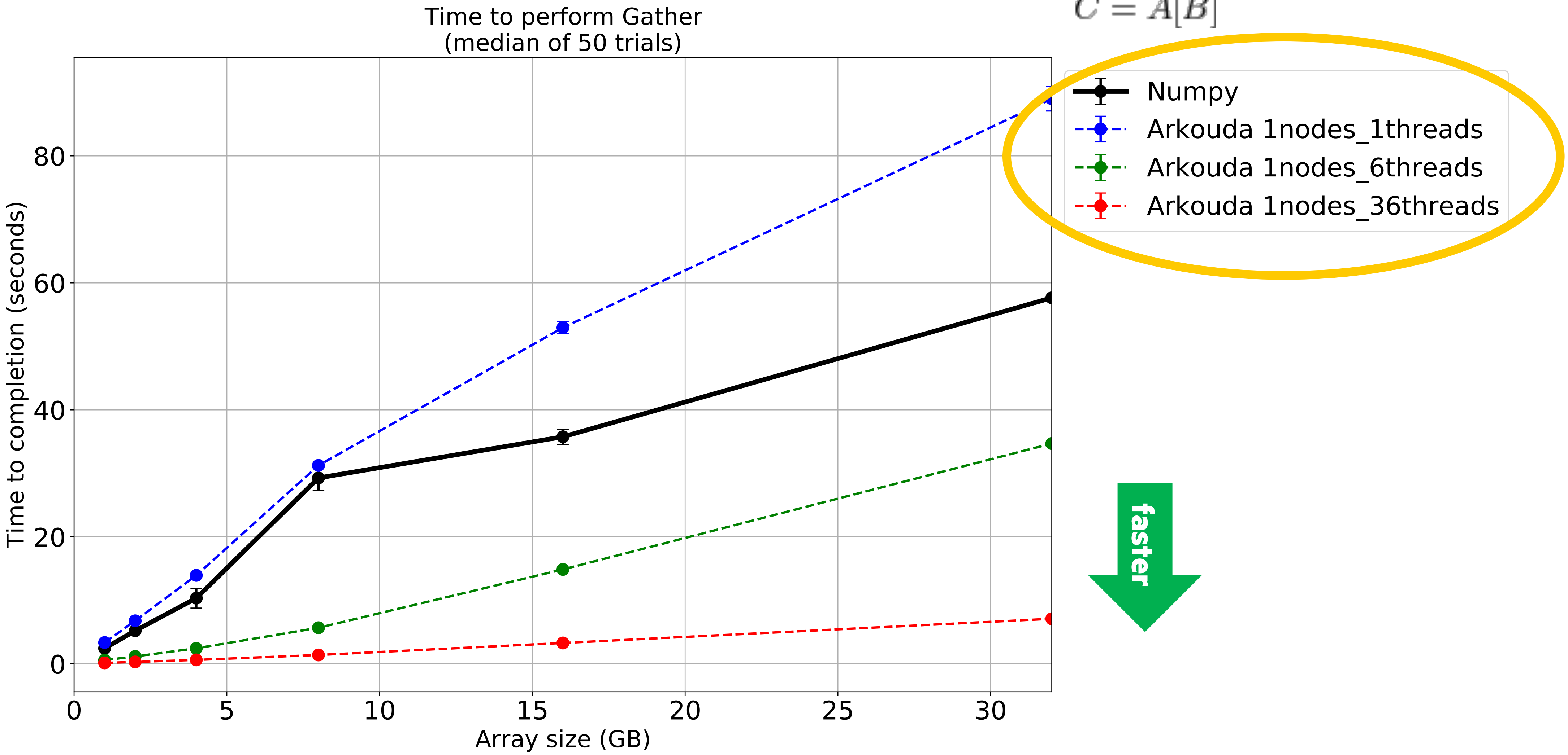
DASK VS. ARKOUDA: REDUCE BENCHMARK

$$c = \sum_i A_i$$



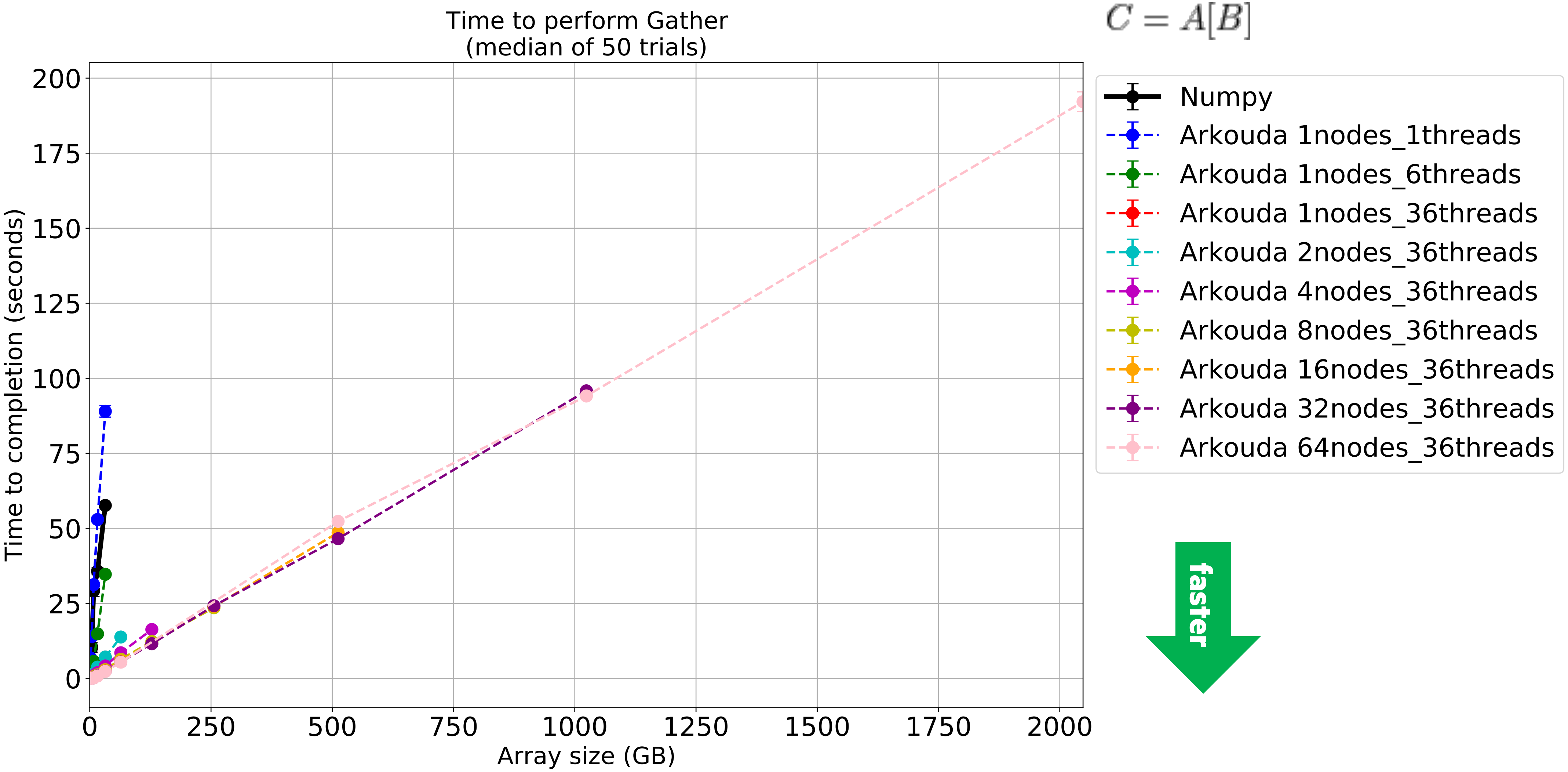
NUMPY VS. ARKOUDA: GATHER BENCHMARK ON UP TO 30 GB DATASETS

$C = A[B]$



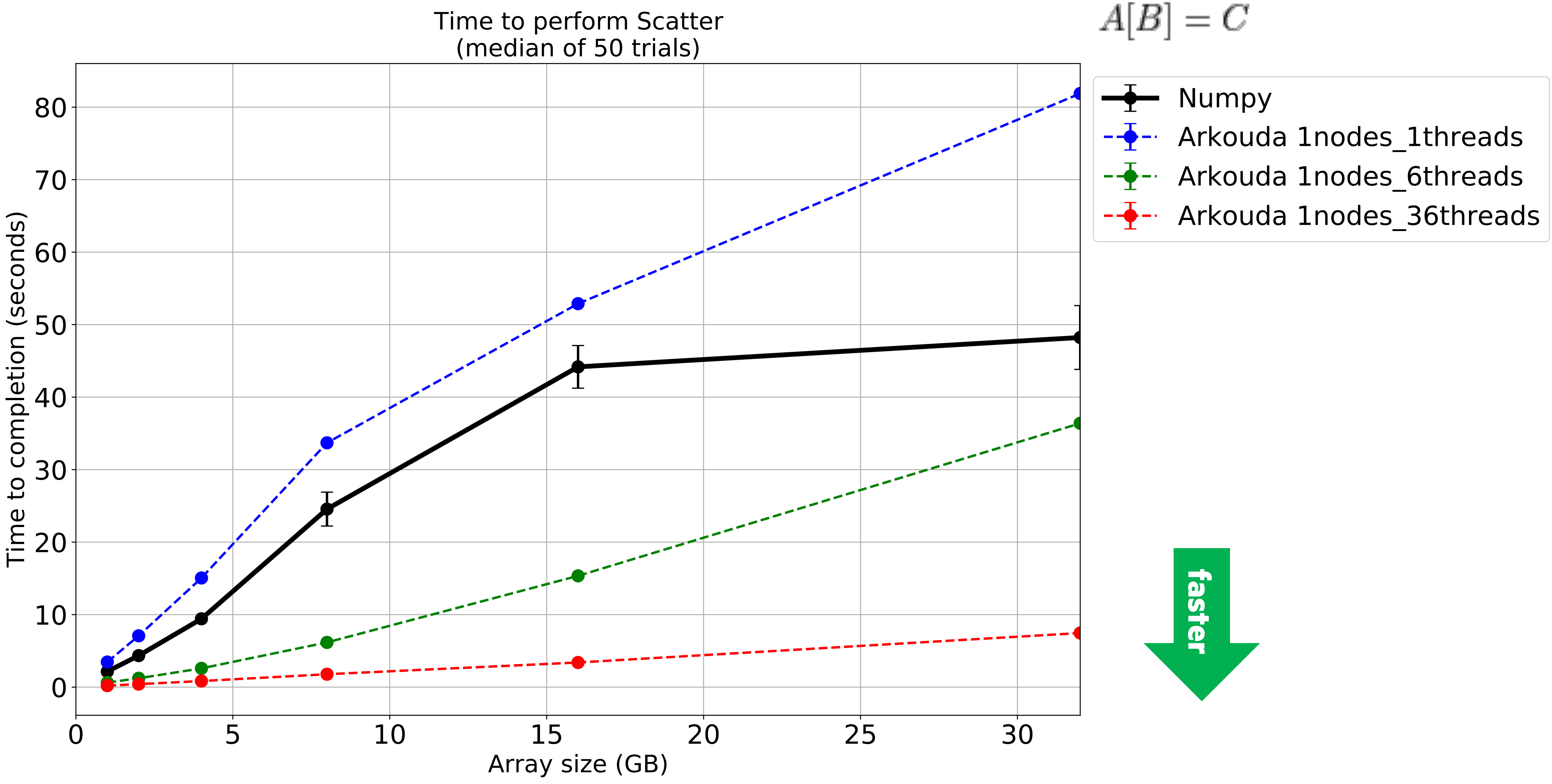
NUMPY VS. ARKOUDA: GATHER BENCHMARK ON UP TO 2000 GB DATASETS

$C = A[B]$

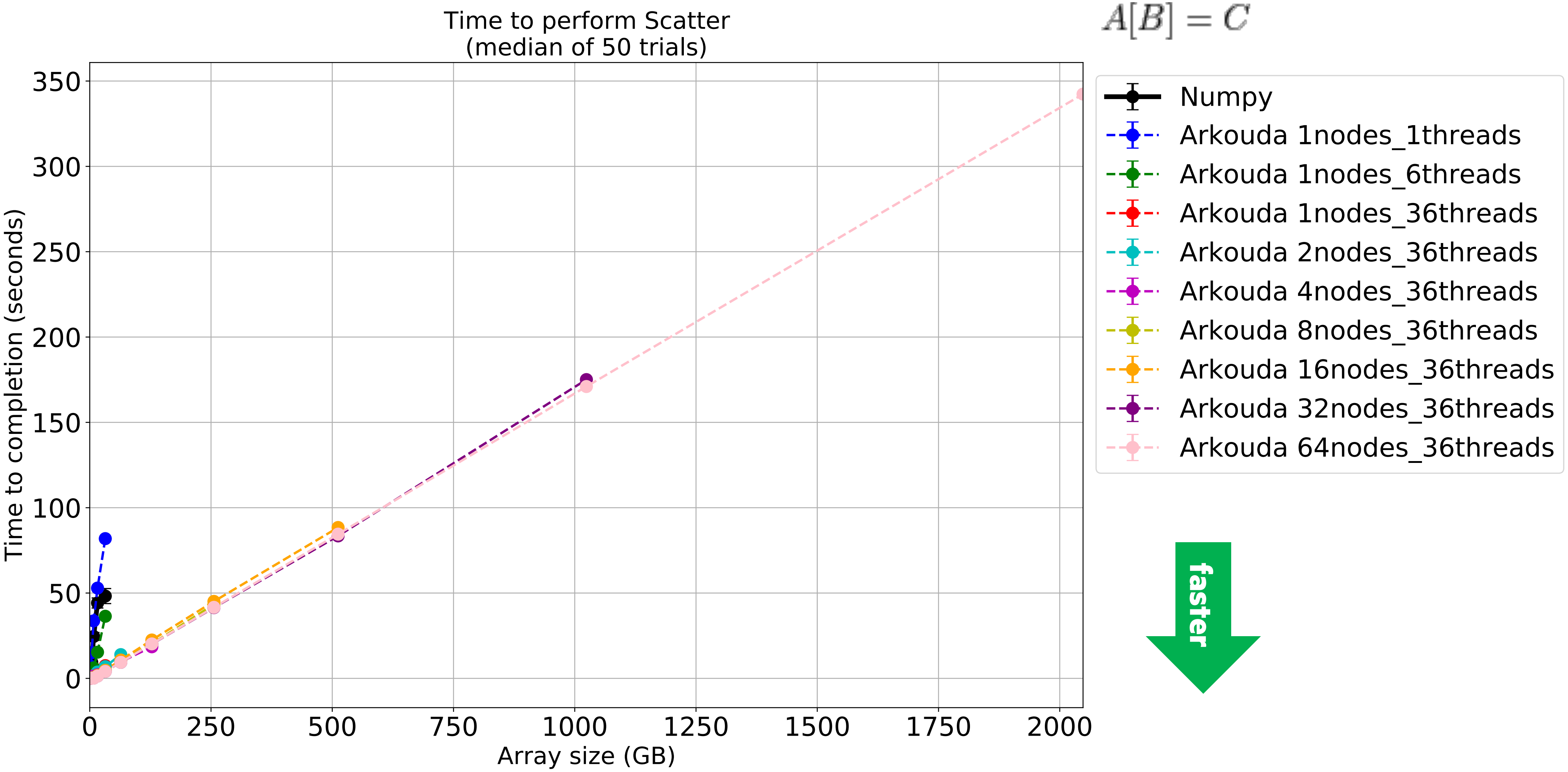


NUMPY VS. ARKOUDA: SCATTER BENCHMARK ON UP TO 30 GB DATASETS

$A[B] = C$

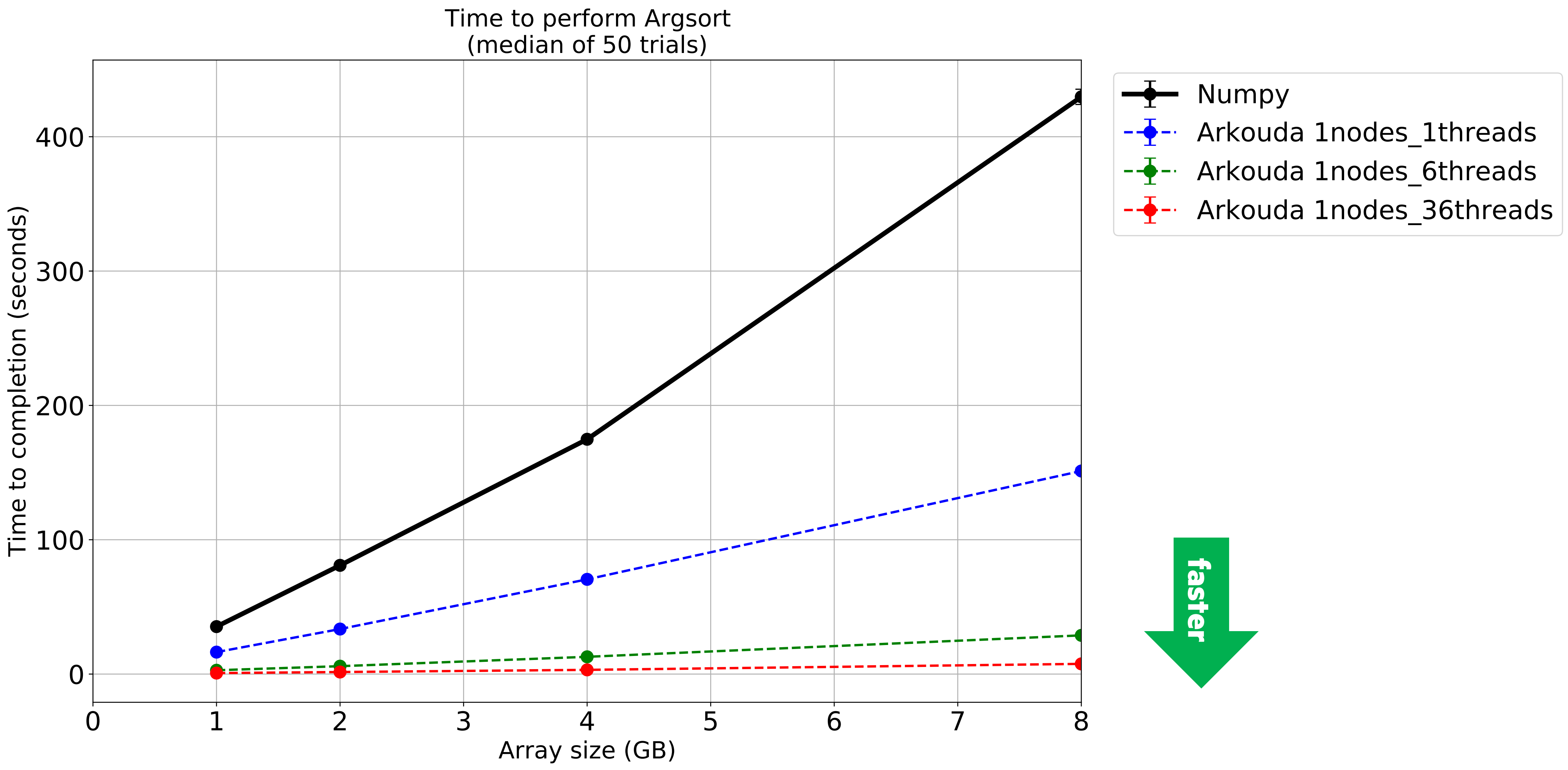


NUMPY VS. ARKOUDA: SCATTER BENCHMARK ON UP TO 2000 GB DATASETS

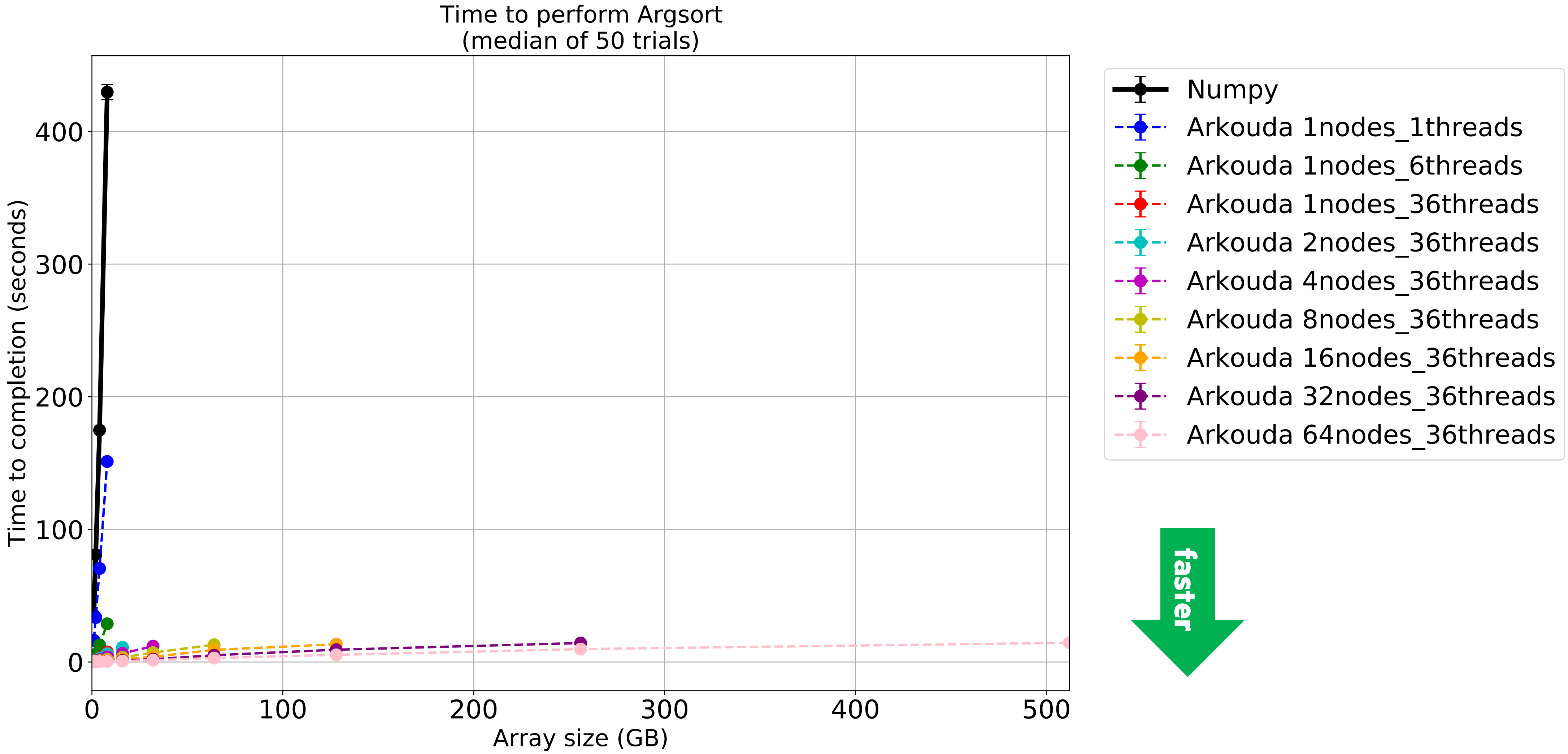


$A[B] = C$

NUMPY VS. ARKOUDA: ARGSORT ON UP TO 8 GB DATASETS



NUMPY VS. ARKOUDA: ARGSORT ON UP TO 500 GB DATASETS



SUMMARY FOR ARKOUDA

A Python data analytics framework

- massive scales = dozens of terabytes
- interactive rates = operations that run within the human thought loop (i.e., seconds to small numbers of minutes)
- crucial operations: argsort, gather, scatter, reading from HDF5 and Parquet files
- started with performance and built towards interactivity using a client-server model

High-Performance Highlights

- Great performance and scalability on HPE Apollo and HPE Cray EX
- Faster than Dask at scale
- Outperforms NumPy on a single node

Thank you!

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

https://chapel-lang.org

Next Steps

- Enable use in Climate Science by implementing the Python Array API
- Accelerate with GPUs, Josh Milthorpe and others working on at ORNL
- Persistence of data store across and between server sessions