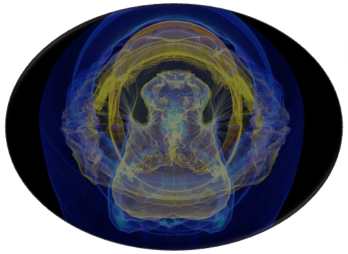
Evaluating Asynchronous Parallel IO on HPC Systems

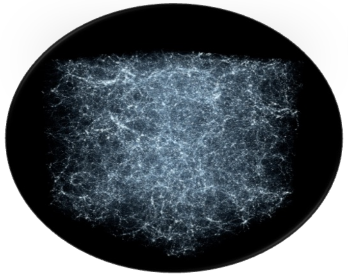
John Ravi, Suren Byna, Quincey Koziol, Houjun Tang, and Michela Becchi

North Carolina State University, The Ohio State University,
and Lawrence Berkeley National Laboratory

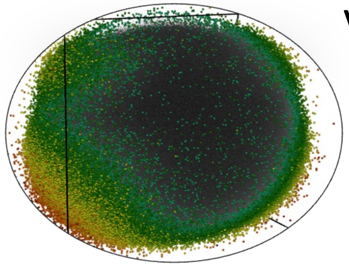
I/O – A critical tool for data storage and access



FLASH



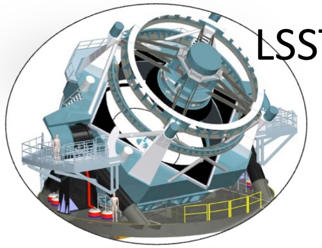
NyX



VPIC

- Simulations
  - Multi-physics (FLASH) – 10 PB
  - Cosmology (NyX) – 10 PB
  - Plasma physics (VPIC) – 1 PB
- Experimental and observational data (EOD)
  - LHC (100 PB),
  - LSST (60 PB),
  - Genomics (100 TB to 1 PB)

LHC

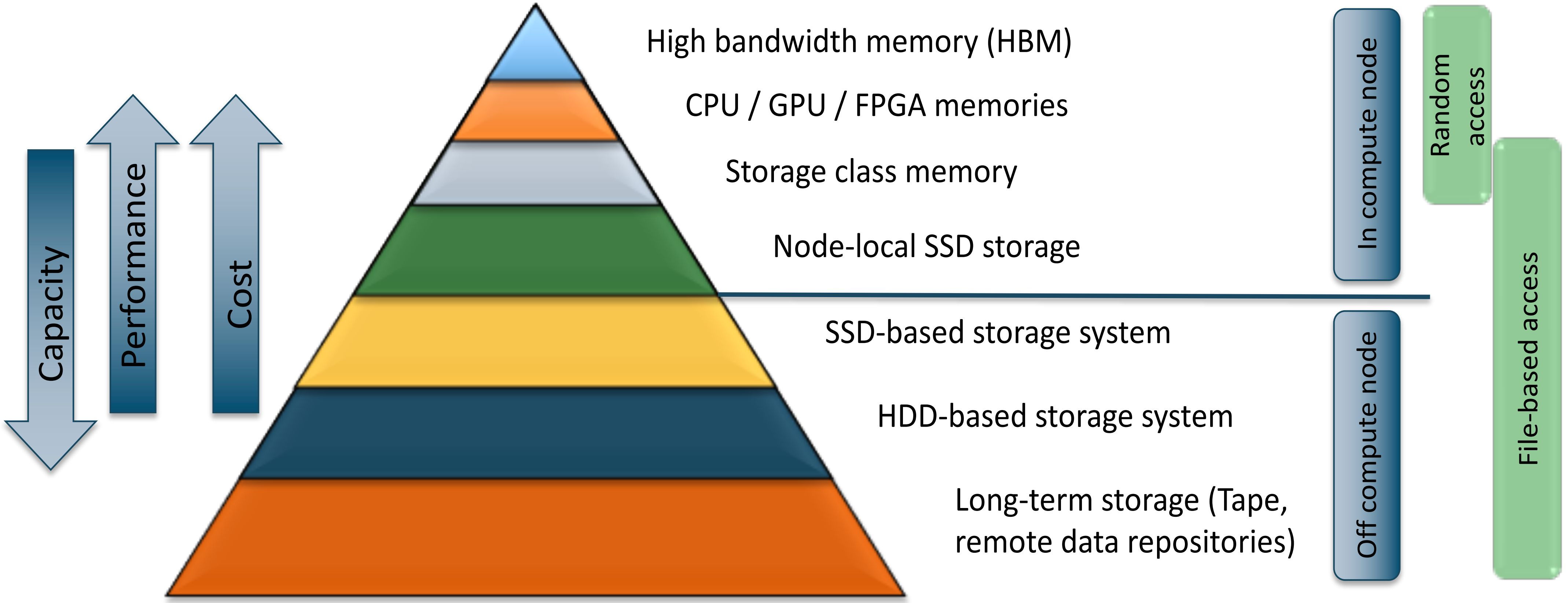


LSST

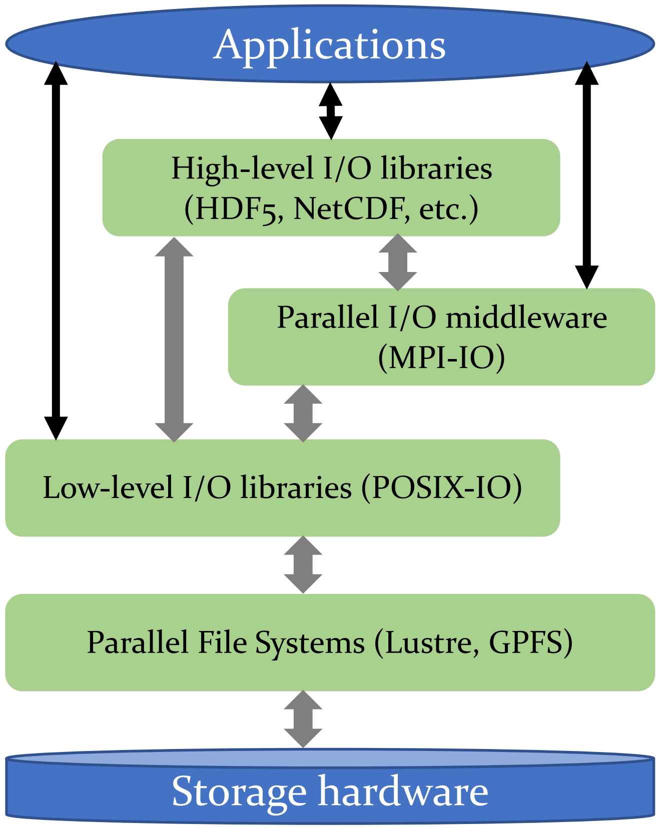
Genomics

Storage and I/O software and hardware are critical for storing and accessing these massive amounts of data.

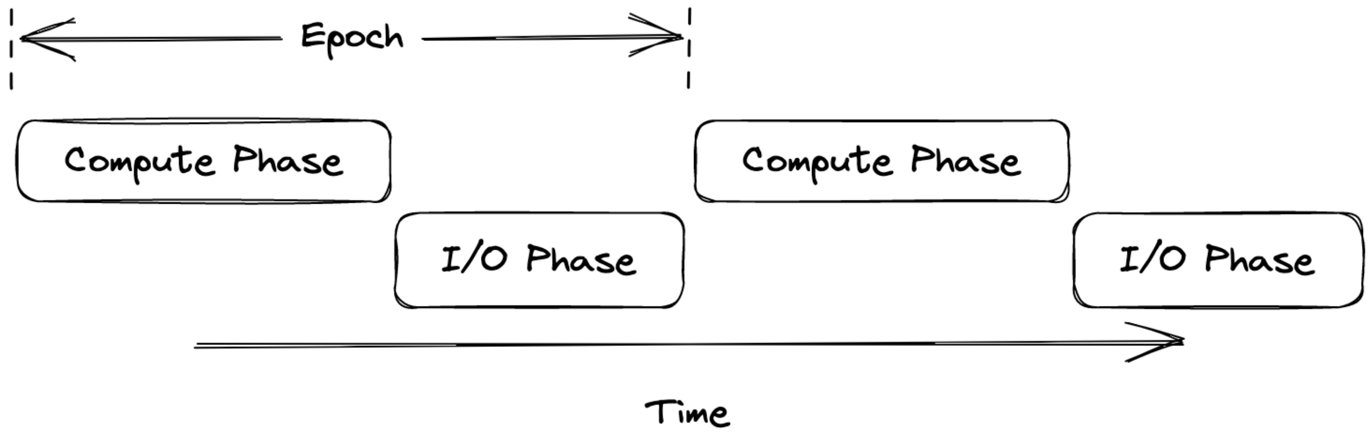
Architectural trends impacting I/O on HPC systems – deep memory and storage hierarchy



Parallel I/O – A stack of software libraries and hardware



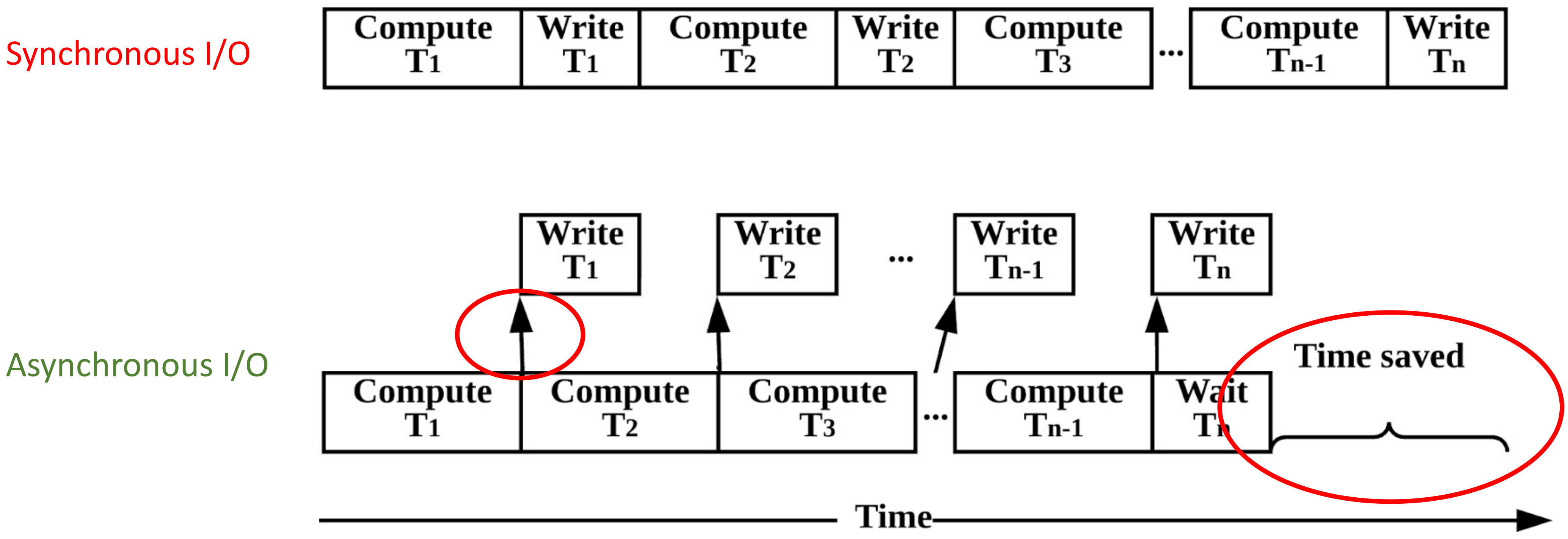
I/O phases could be slow and slowdown applications



- Many large-scale applications have distinct compute and I/O phases
- Simulations checkpoint state or save visualization data
  - *EQSIM* (earthquake simulator), *Nyx* and *Castro* (adaptive mesh refinement, cosmological hydrodynamics)
- Machine learning training iteratively reads data
  - *Cosmoflow (3D convolutional neural network)*

Asynchronous I/O to the rescue

- Hiding I/O latency by overlapping with computation → Common async I/O approach

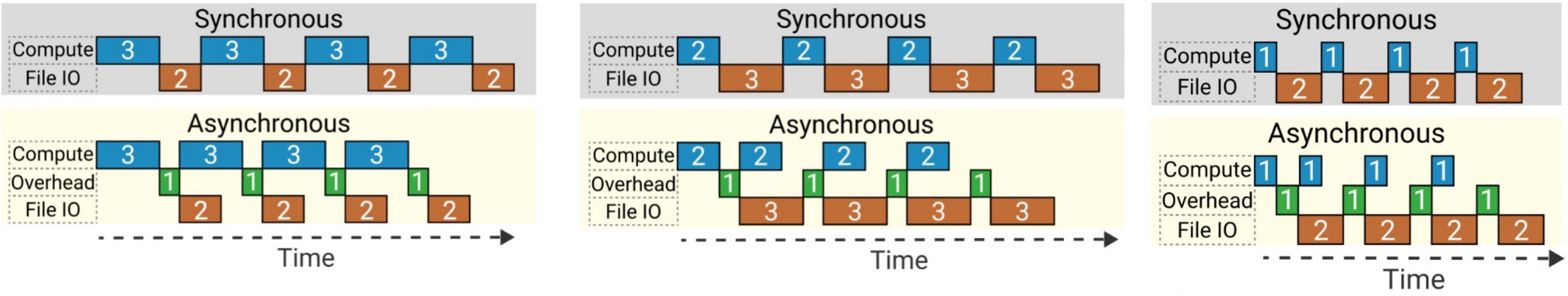


Asynchronous I/O – Implemented in several I/O libraries

- POSIX
- MPI-IO
- ADIOS
- Data Elevator and ARCHIE
- Proactive Data Containers (PDC)
- HDF5

A systematic study of benefits and limitations of asynchronous I/O is lacking

Asynchronous I/O Scenarios



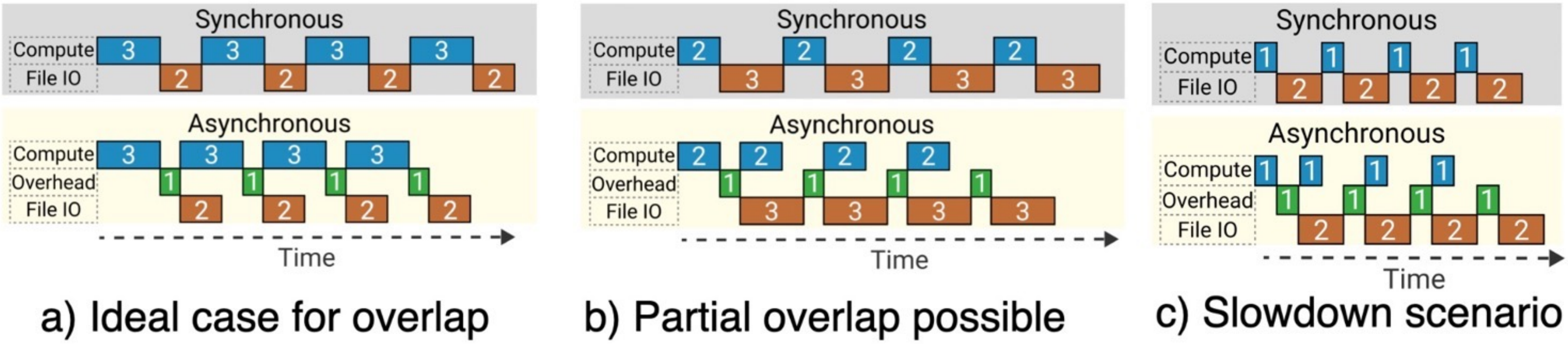
a) Compute Time > I/O Time

b) Compute Time < I/O Time

c) Compute Time <= Overhead

- Computation phase, Overhead for setting up async I/O, I/O latency
- Scenarios
  - Longer computation phases than I/O latency
  - Shorter computation phases than I/O latency

Latency with asynchronous I/O



$$t_{sync\_epoch} = t_{io} + t_{comp}$$

$$t_{async\_epoch} = max(t_{comp}, t_{io} - t_{comp}) + t_{transact\_overhead}$$

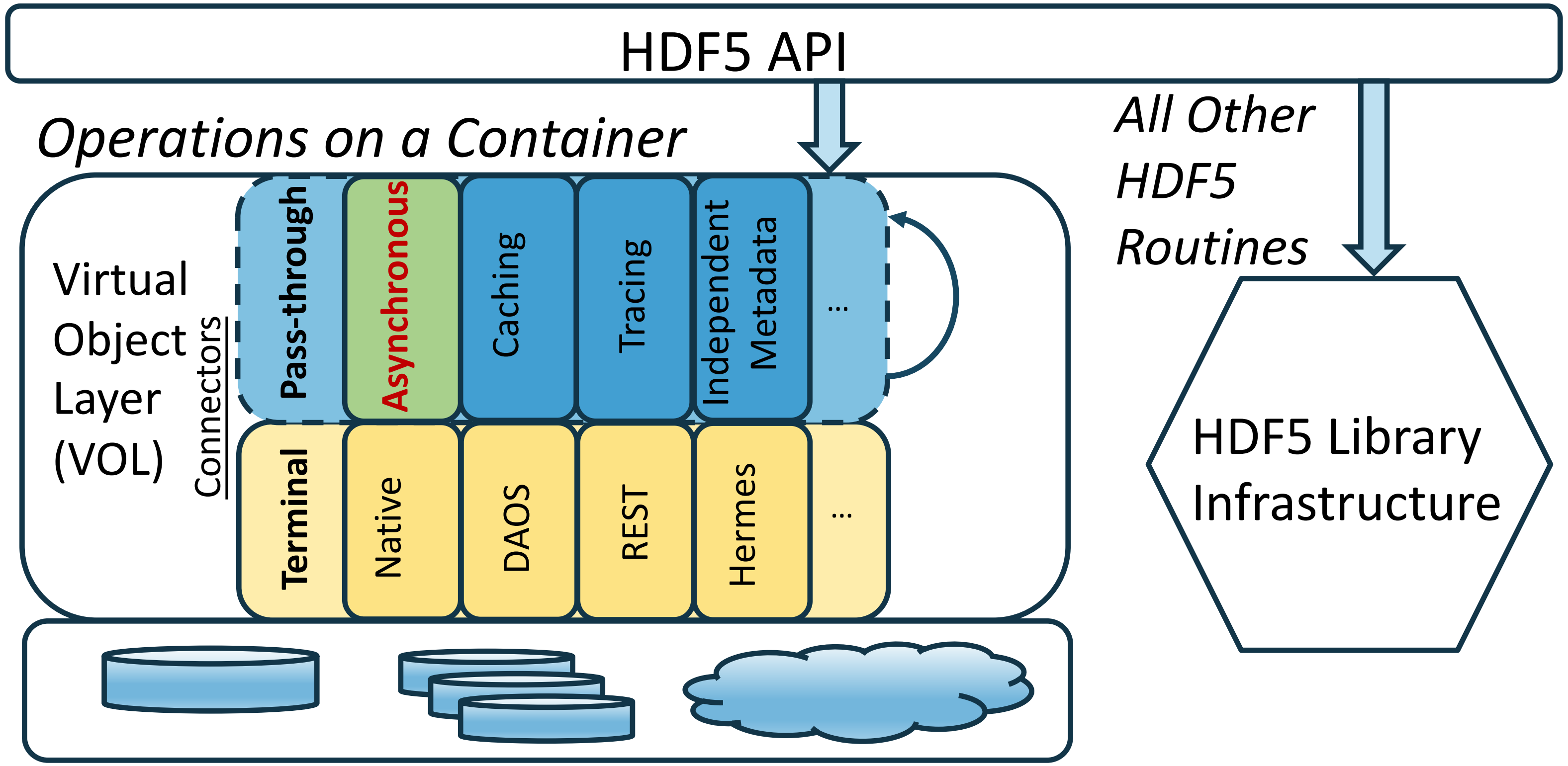
$$t_{io} = \frac{data\_size}{f_{io\_rate}}$$

Depends on an implementation of async I/O

- Background threads
- Extra buffering
- Communication for buffering
- Shared computation and communication resources
- Pressure on file system and I/O from other jobs

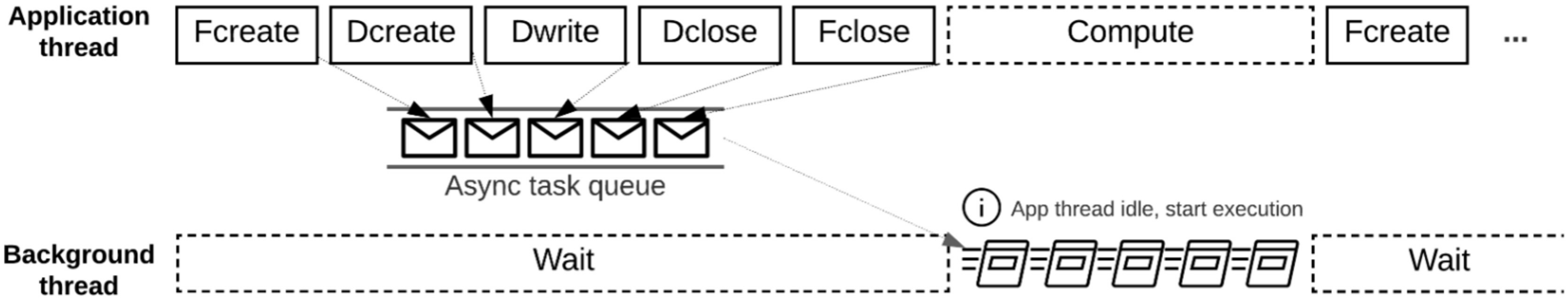
Asynchronous I/O in HDF5 – Intro to Virtual Object Layer (VOL)

- VOL allows “intercepting” HDF5 public API and implementing a different approach to storage and access



Asynchronous I/O in HDF5 – Using background threads

- A pass-through VOL connector for implementing asynchronous I/O
- Asynchronous task queue
- Transparent background thread execution



Explicit Control with Async and EventSet APIs

- Async version of HDF5 APIs
  - `H5Fcreate_async(fname, …, es_id);`
  - `H5Dwrite_async(dset, …, es_id);`
  - …

- Track and inspect multiple I/O operations with an ***EventSet ID***
  - `H5EScreate();`
  - `H5ESwait(es_id, timeout, &remaining, &op_failed);`
  - `H5ESget_err_info(es_id, ...);`
  - `H5ESclose(es_id);`

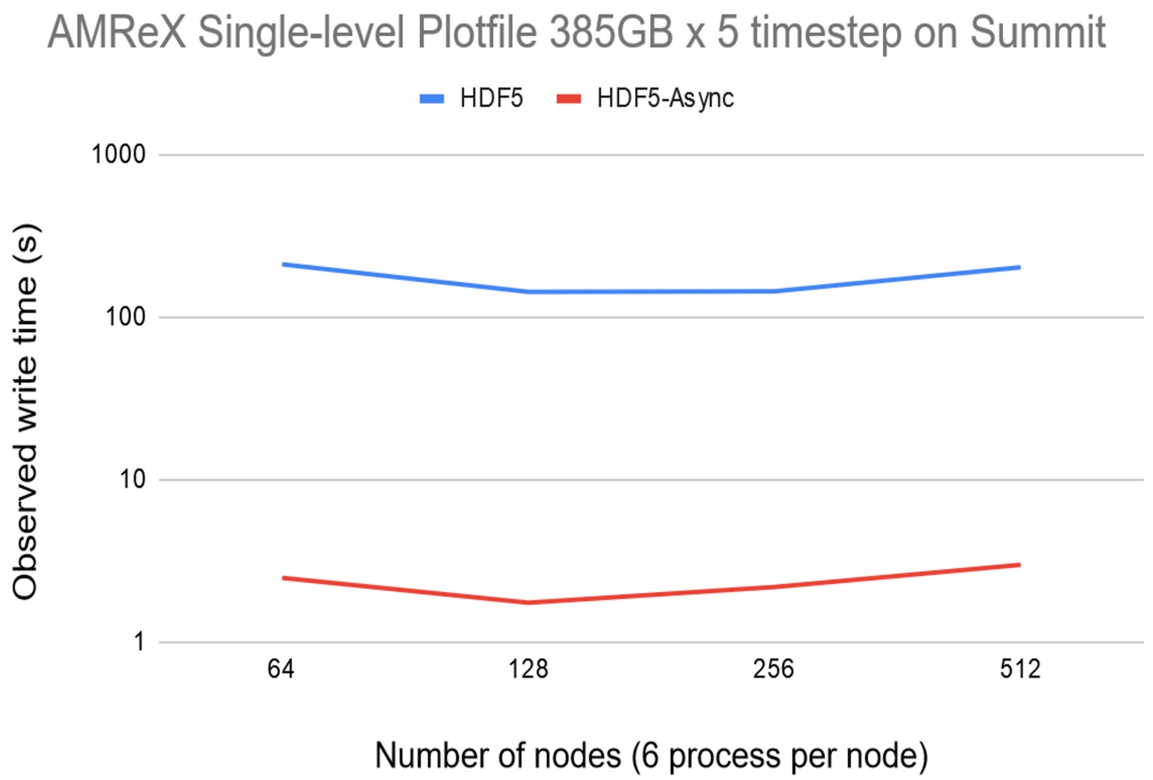
Converting Existing HDF5 Codes

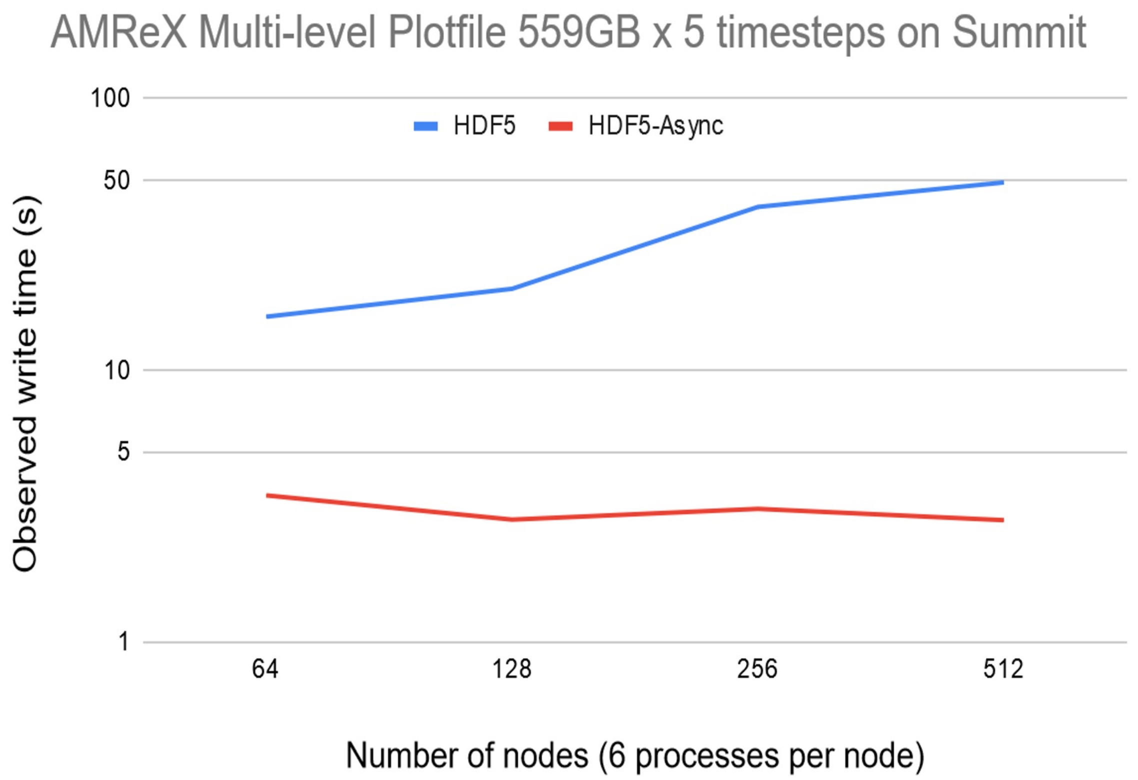
```
// Synchronous file create
fid = H5Fcreate(...);
// Synchronous group create
gid = H5Gcreate(fid, ...);
// Synchronous dataset create
did = H5Dcreate(gid, ..);
// Synchronous dataset write
status = H5Dwrite(did, ..);
// Synchronous dataset read
status = H5Dread(did, ..);
...
// Synchronous file close
H5Fclose(fid);
// Continue to computation

...

...
// Finalize
```

```
// Create an event set to track async operations
es_id = H5EScreate();
// Asynchronous file create
fid = H5Fcreate_async(.., es_id);
// Asynchronous group create
gid = H5Gcreate_async(fid, .., es_id);
// Asynchronous dataset create
did = H5Dcreate_async(gid, .., es_id);
// Asynchronous dataset write
status = H5Dwrite_async(did, .., es_id);
// Asynchronous dataset read
status = H5Dread_async(did, .., es_id);
...
// Asynchronous file close
status = H5Fclose_async(fid, .., es_id);
// Continue to computation, overlapping with asynchronous
    operations
...
// Finished computation, Wait for all previous operations in the
    event set to complete
H5ESwait(es_id, H5ES_WAIT_FOREVER, &n_running, &op_failed);
// Close the event set
H5ESclose(es_id);
...
// Finalize
```

Detailed description in https://github.com/hpc-io/vol-async

Async HDF5 VOL Connector – Benefits

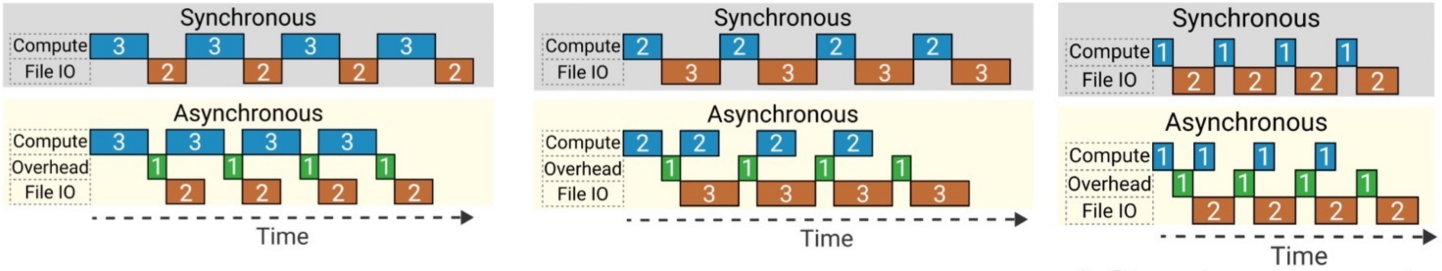




Houjun Tang, Quincey Koziol, John Ravi, and Suren Byna, "Transparent Asynchronous Parallel I/O using Background Threads", IEEE TPDS - Special Section on Innovative R&D toward the Exascale Era, 2021

Questions for a detailed evaluation

- For computation phases longer than I/O phases, async I/O is beneficial
- What about other conditions?
  - When does asynchronous I/O slow down applications?

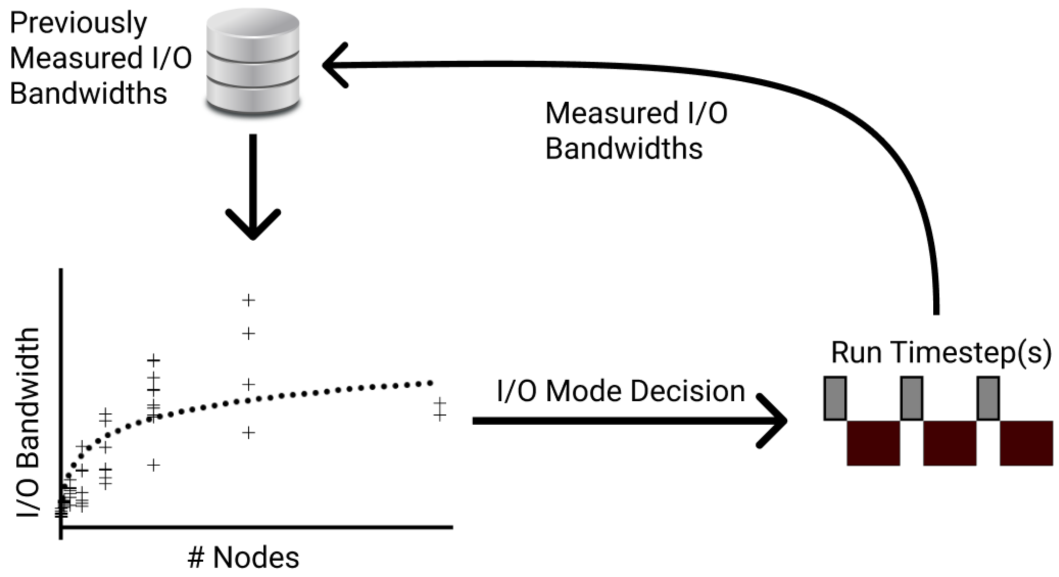


a) Ideal case for overlap b) Partial overlap possible c) Slowdown scenario

- Can we predict synchronous and asynchronous I/O time to decide on using them?

Experimental evaluation

- Systems
  - Summit at OLCF with ~4k nodes with GPFS Parallel File system
  - Cori at NERSC with ~12k nodes with Lustre Parallel File System
- Estimation of I/O cost
  - Empirical model using linear regression
  - Aggregate bandwidth scales with data size, # ranks for each I/O request



Benchmarks and Applications

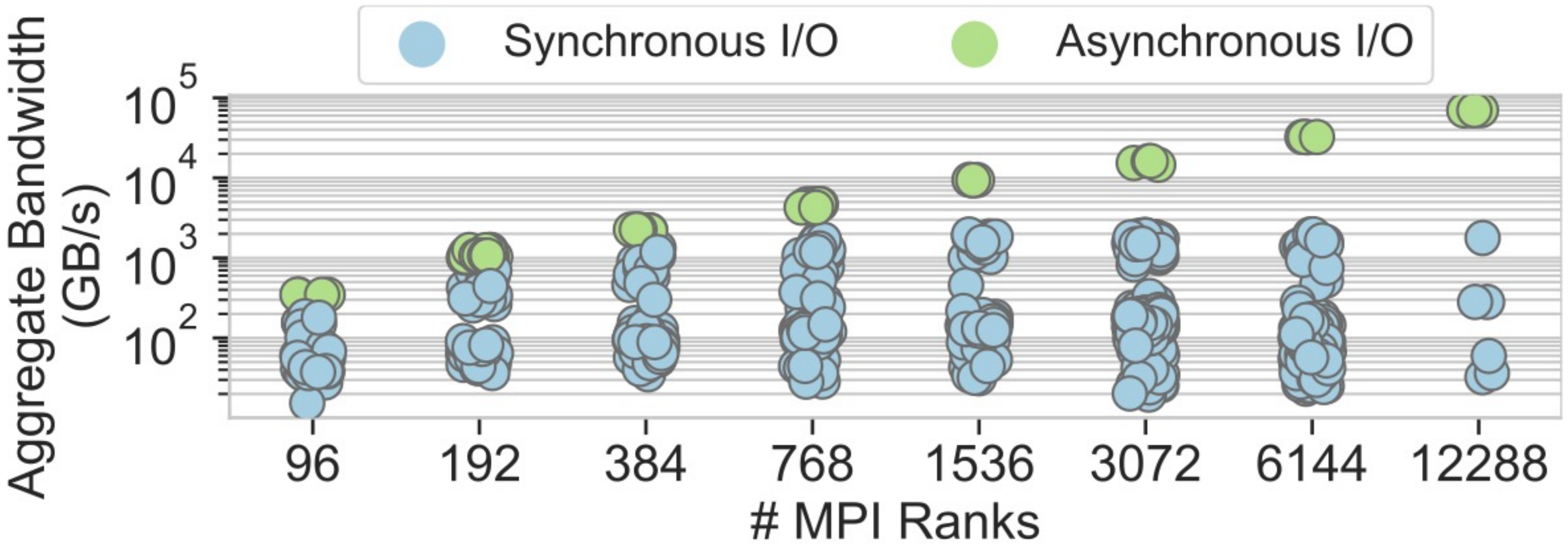
- VPIC-IO
  - A data write benchmark, extracted from a plasma physics simulation
- BD-CATS-IO
  - A read benchmark, extracted from a clustering analysis code
- Nyx
  - A massively parallel, adaptive mesh, cosmology simulation code
- Castro
  - A cosmology simulation solving compressible radiation & hydrodynamics equations
- EQSIM
  - A regional earthquake simulation code
- Cosmoflow
  - A deep learning code to process large 3D matter distributions using CNN

https://github.com/hpc-io/h5bench

Configurations

I/O kernel / App	Data dimensions	Other notes
VPIC-IO	8 variables, 2^{10} particles	1D HDF5 dataset
BD-CATS-IO	Any number of given variables, 2^{10} particles	Same as VPIC-IO, read pattern
Nyx	Small: 256x256x256, every 20 time steps	20 MB data per time step
	Large: 2048x2048x2048 dimensions, every 50 time steps	10 GB data per time step
Castro	128x128x128 dimensions with 6 components in each multi-fab and 2 particles per cell	128 MB data per time step
EQSIM	Grid size of 50 with 30000x30000x17000 dimensions; checkpoint every 100 time steps	Computation phases are often very long compared to checkpointing phases
Cosmoflow	128^3 Voxels dataset, 4 epochs and with batch size of 8	Computation on GPUs, data for I/O is transferred to main memory before CPU performs I/O.

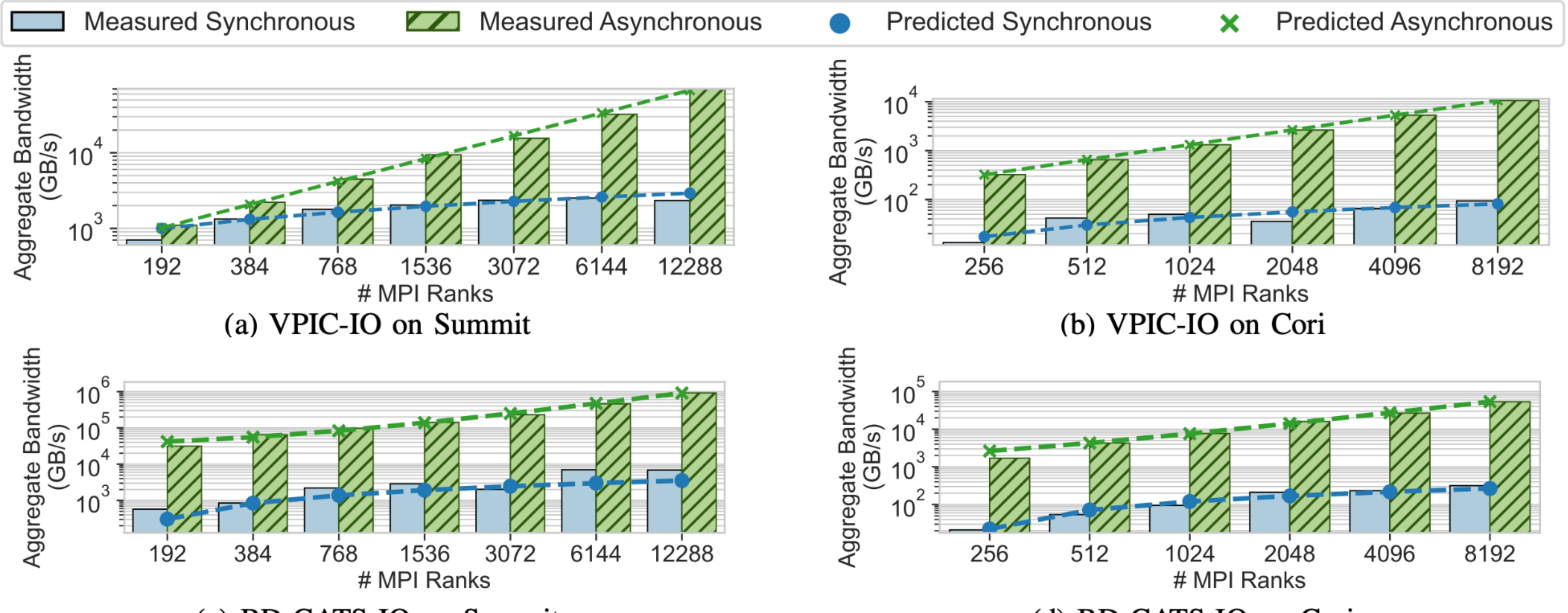
Estimation of I/O cost



VPIC-IO on Summit

- Each point represents a separate run at a different time
- Synchronous I/O varies in performance (about 2 orders of magnitude at high node count)
- History of best achieved bandwidth

Weak scaling tests on Summit and Cori

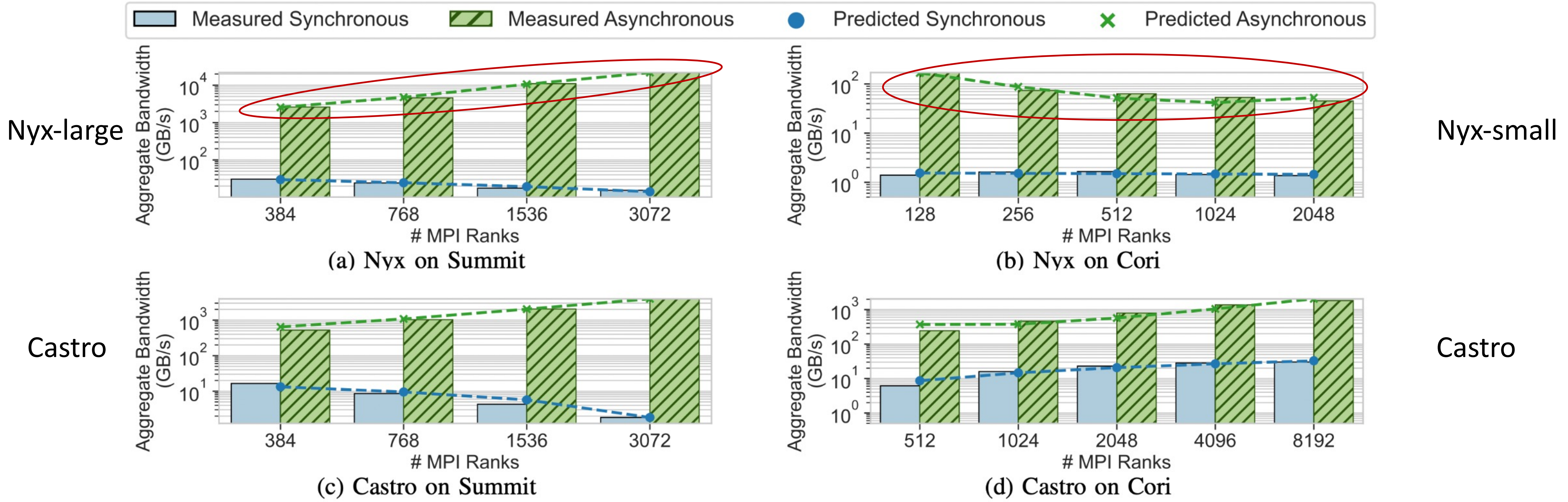


(a) VPIC-IO on Summit (b) VPIC-IO on Cori

(c) BD-CATS-IO on Summit (d) BD-CATS-IO on Cori

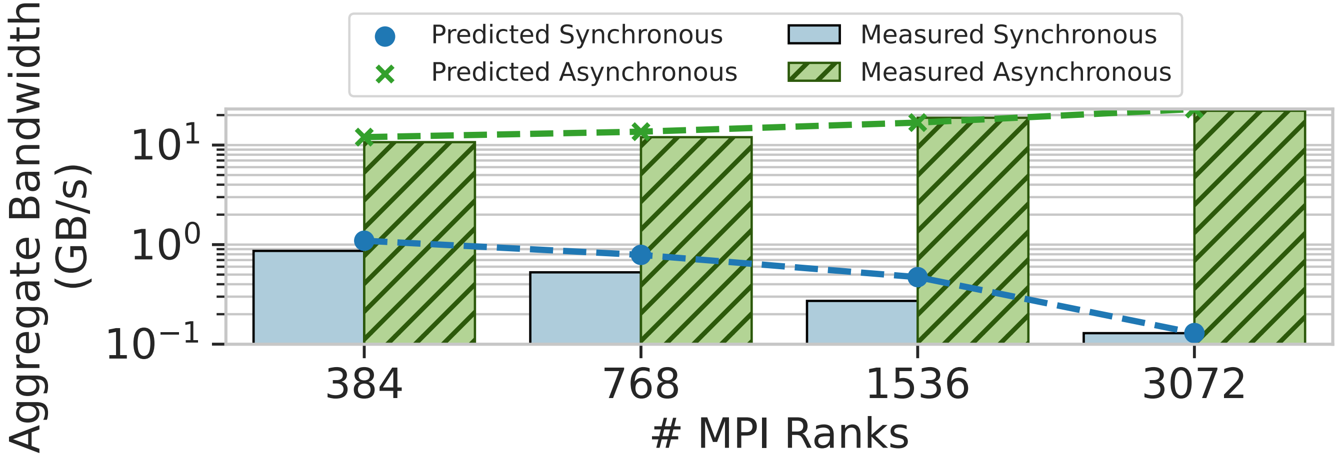
- *The aggregate bandwidth scales similarly on both systems for both synchronous and asynchronous epochs*
- *Analytical model fits well with the trend of synchronous write aggregate bandwidth which is based on a linear-log regression*

Strong scaling tests on Summit and Cori



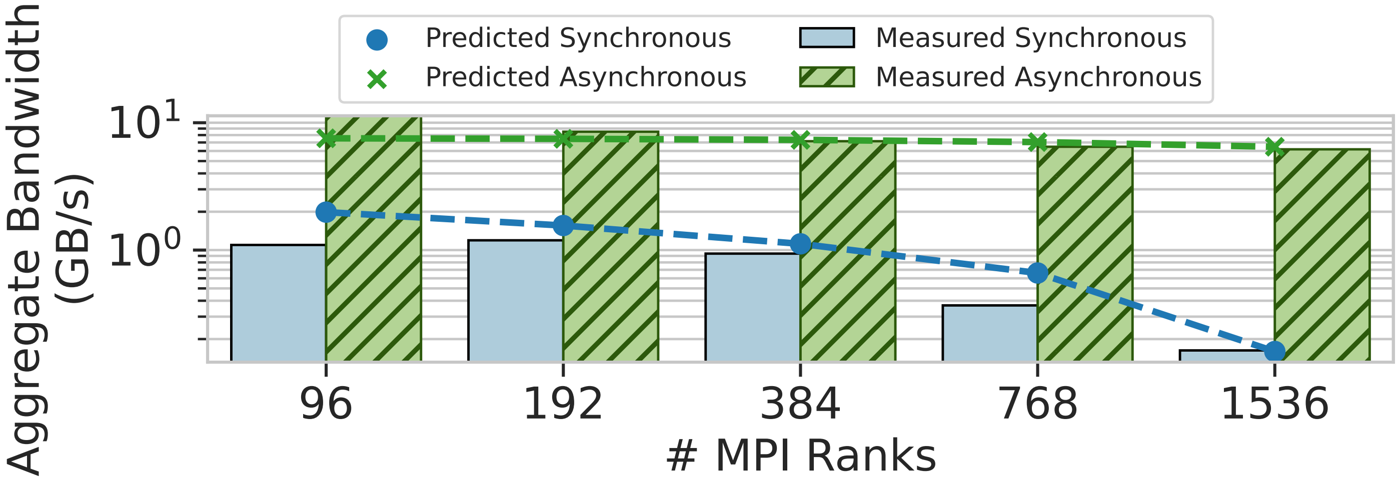
- *The async I/O overhead is low with smaller amount of data (with increasing number of ranks), increases async I/O rate on Summit*
- *On Cori, for smaller data size (Nyx-small configuration), increasing scale doesn't increase I/O rate much. Async I/O still much better than sync I/O*

EQSIM and Cosmoflow – Async I/O wins significantly

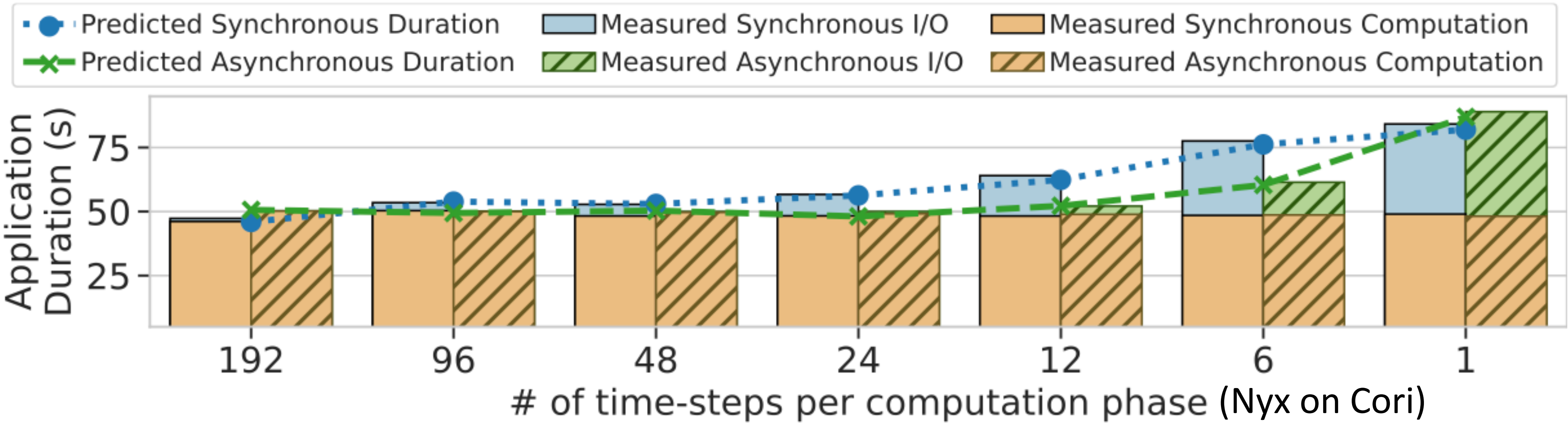


EQSIM on Summit – Synchronous I/O slows down with scale. Async I/O is effective.

Cosmoflow on Summit with GPUs – Synchronous I/O slows down after 128 nodes. Async I/O is effective (includes GPU to CPU memory copy overhead)



Frequent I/O phases with async I/O slows down applications



(Nyx on Cori)

- Checkpointing every timestep with asynchronous I/O enabled resulted in an overall slowdown
- Extra overhead introduced with asynchronous I/O could not be hidden
- Requires a dynamic decision at runtime to enable Asynchronous I/O

Conclusions

- Asynchronous I/O can hide I/O latency in cases where computation > async overhead + I/O time
- Analytical models for estimating I/O latency using linear regression to evaluate efficacy of async I/O
- Model-based automatic selection of async I/O → in progress
- Other Async I/O optimizations
  - Combine multiple small I/O requests → ESSA 2023 paper
  - Multi-dataset I/O in HDF5 to reduce the number of I/O requests

Contact: Suren Byna – https://sbyna.github.io

Async I/O with HDF5: https://github.com/hpc-io/vol-async

Back up slides

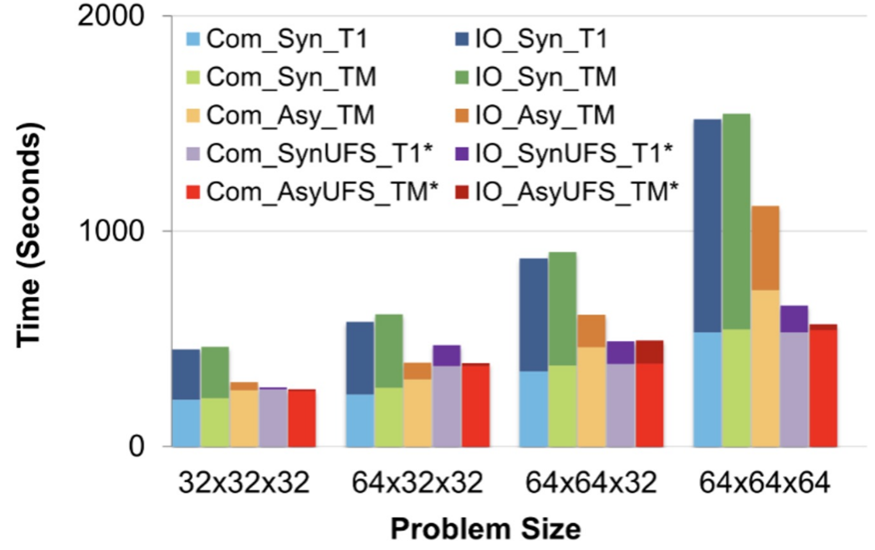
Results: Sod

Sod is a compressible flow explosion problem widely used for verification of shock-capturing simulation codes.

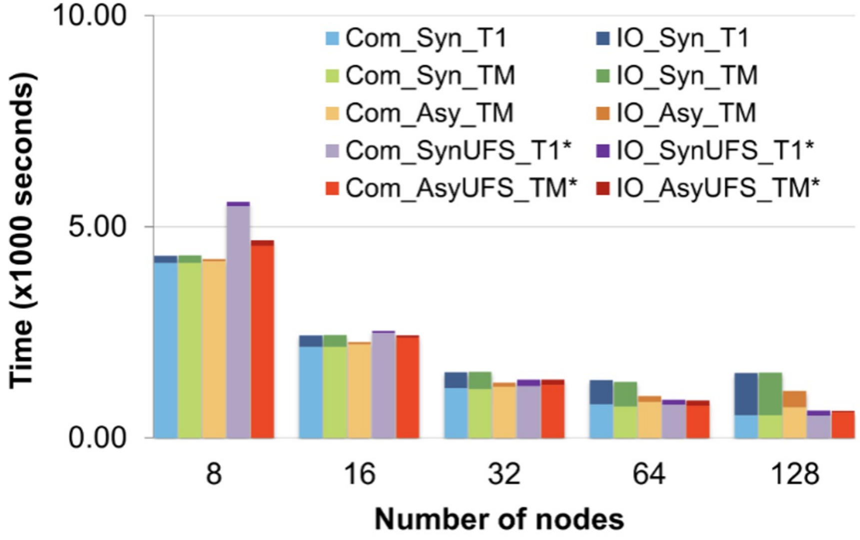
3D Sod problem with tracer particles.

Each runs for 109 steps, writes a checkpoint file every 33 steps, a plot file every 10 steps, and compared the total execution time with 5 different configurations that uses Synchronous and Asynchronous I/O, with and without MPI_THREAD_MULTIPLE, and using GPFS and UnifyFS.

- For cases with async, the majority of the write operations are overlapping with Flash-X's computation. Exceptions include the initial data writes and the last step as there is no computation to overlap with.



(a) Sod - weak scaling, 16 to 128 nodes



(b) Sod - strong scaling, problem size 64x64x64

Rajeev Jain, Houjun Tang, Akash Dhruv, Austin Harris, Suren Byna, Accelerating Flash-X Simulations with Asynchronous I/O, PDSW 2022

Results: Streaming Sine Wave

- The streaming sine wave test problem is a test problem for verifying the correctness of the streaming advection operator in thornado as well as the Flash-X interface to thornado.

- This problem uses GPU and CPU (threading).

- One GPU per MPI rank, and the data is copied from GPU to CPU memory automatically by FLASH-X before being written out

- At a higher number of nodes the interference between COM_ time and IO_ is higher as the I/O time as a whole increases: it is 27.1% for the 256-node synchronous case.

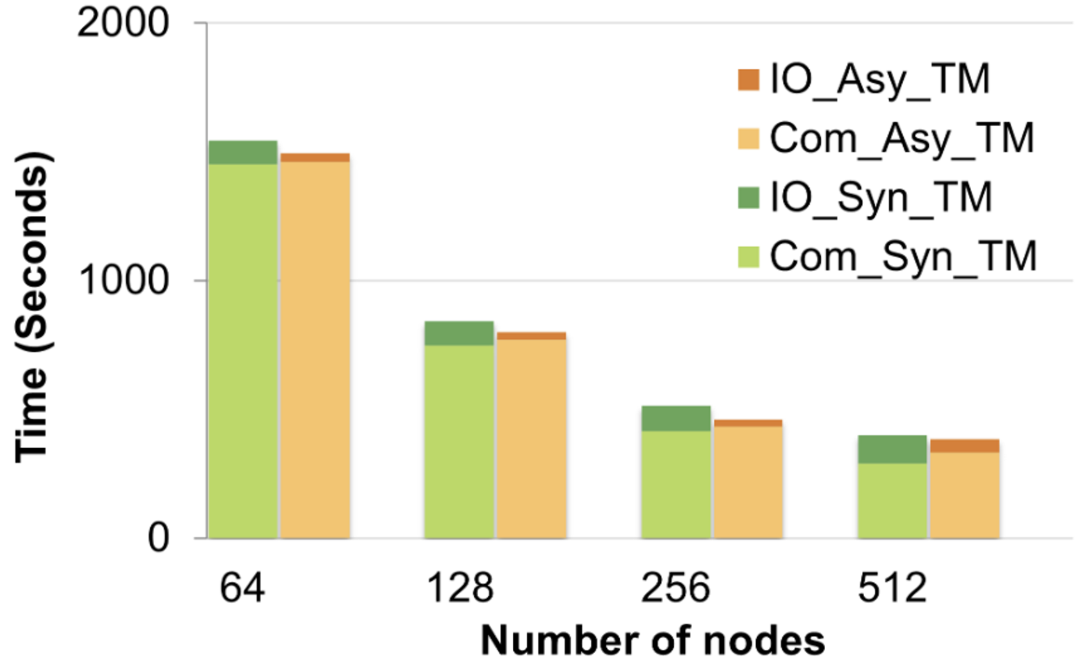


Fig. 7: Streaming sine wave - strong scaling

Results: Deforming Bubble Problem

- This is a benchmark problem for multiphase CFD applications in Flash-X. The deformation is computed by level-set advection and redistancing algorithm.
- For results shown in Fig. 6, the number of bubbles per MPI process is varied. Fig. 1 shows bubble undergo deformation under a velocity field.
- For the 64-node case the I/O time as a percentage of the total simulation time goes down from 22.3% to 4.7%.
- For the 256-node case, the I/O time is significantly higher for the synchronous case; this is due to the fact that a lot of communication is required to write the file to disk from 256 nodes (or 5,376 MPI ranks) and the GPFS file system on Summit does not scale well.
- The asynchronous I/O time for 256 nodes remains the same as for other cases, but the Com_ time has increased because a greater percentage of Com_ time overlaps with IO_ time.

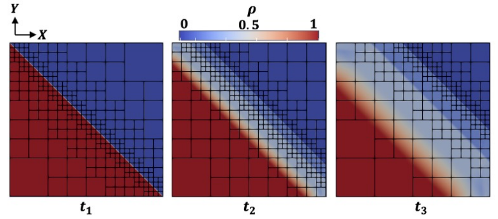


Fig. 1: Contours of energy (E) for time $t_3 > t_2 > t_1$, and an example of block structured AMR grids.

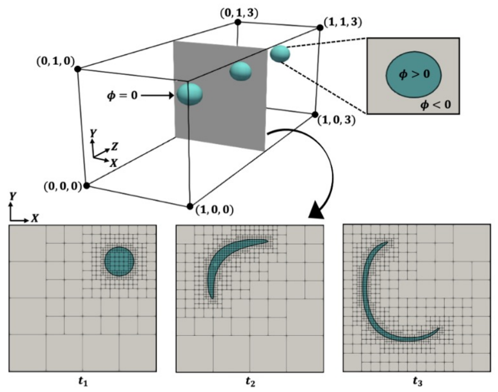


Fig. 2: Schematic of the deforming bubble problem: The bubbles are defined by using a signed distance function, ϕ, that undergoes deformation under a prescribed velocity field.

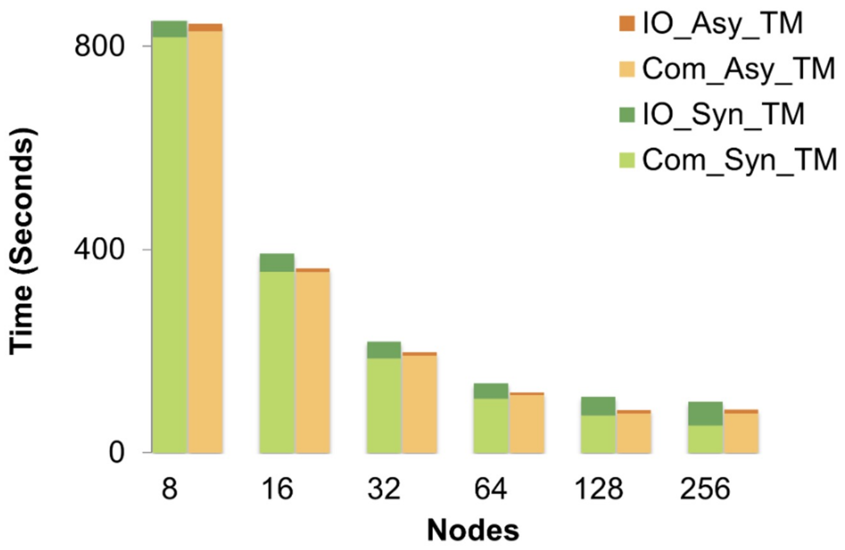


Fig. 6: Deforming bubble - strong scaling