Lawrence Livermore National Laboratory

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Optimizing Molecular Dynamics AI model using **HDF5 and DYAD**

Presented by: **Hariharan Devarajan** devarajan1@llnl.gov

HDF5 User Group Meeting (HUG 2024) Chicago, IL

flux

LLNL-PRES-867285

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC. This material is based upon work supported by LLNL LDRD 23-ERD-045 and 24-SI-005..

Acknowledgements

Current and Past Flux team:

Thomas Scogland, Albert Chu, Tapasya Patki, Stephen Herbein, Mark Grondona, Becky Springmeyer, Christopher Moussa, Jim Garlick, Daniel Milroy, Clay England, Michela Taufer, Ryan Day, Dong H. Ahn, Barry Rountree, Zeke Morton, Jae-Seung Yeom, James Corbett, Vanessa Sochat, and William Hobbs.

LLNL's MuMMI Team:

Loïc Pottier, Konstantia Georgouli, Tim Hsu, and Timo Bremer

LLNL's IOPP Team:

Chen Wang and Kathryn Mohror

University of Tennessee, Knoxville:

Michela Taufer, Ian Lumsden, and Jack Marquez

Lawrence Livermore National Laboratory

THE UNIVERSITY OF TENNESSEE KNOXVILLE

flux

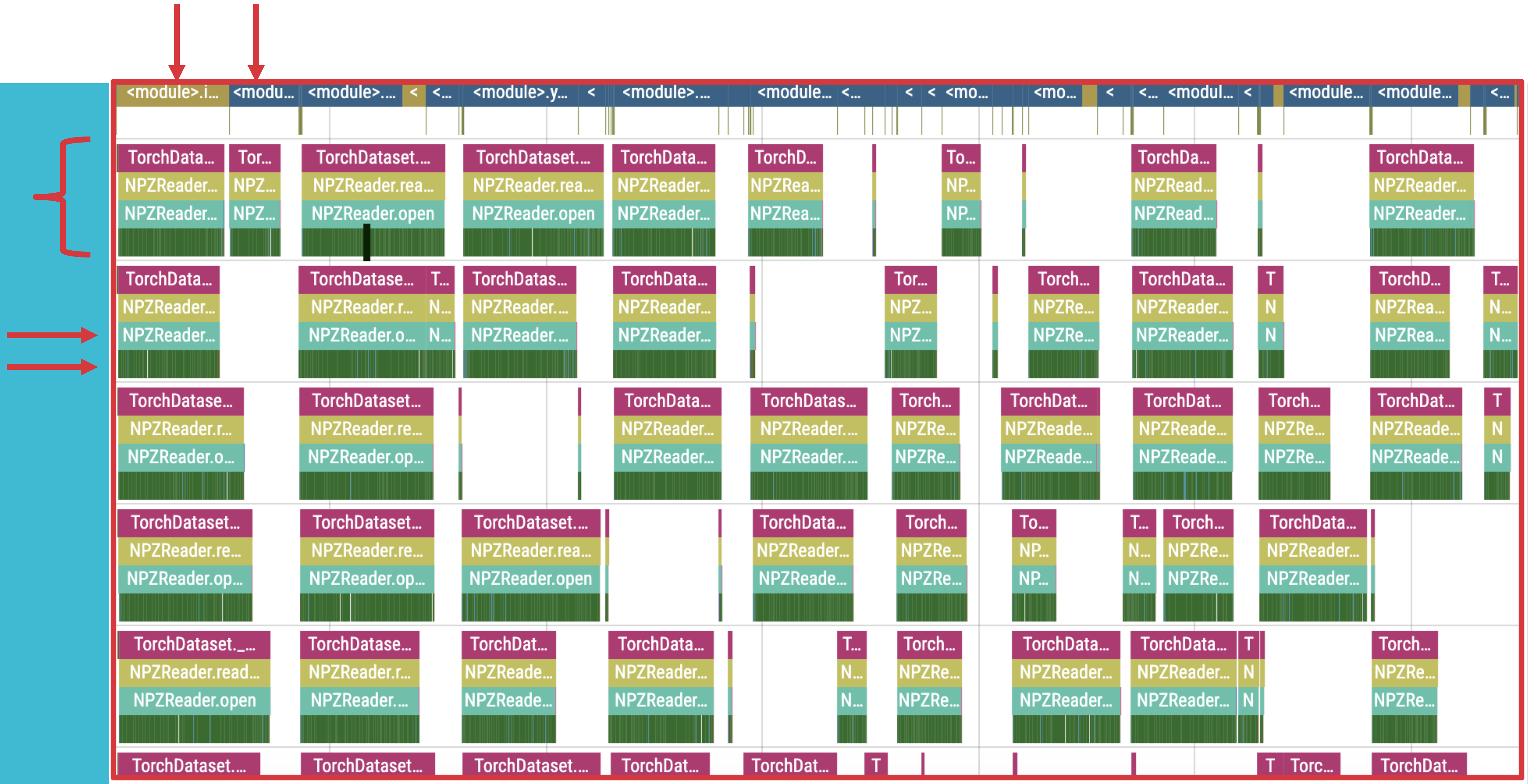
The Massively Parallel Multiscale Machine-Learned Modeling Infrastructure (MuMMI)

- A framework for executing multiscale modeling simulations of large molecular systems
- Studies the plasma membrane and RAS-RAF-14-3-3
- AI-assisted workflow with offline training, online inference and simulation with feedback mechanisms.
- Accuracy of AI training improves the initial conditions for molecular interaction which significantly improve resolution of the workflow.

AI training segment of MuMMI.

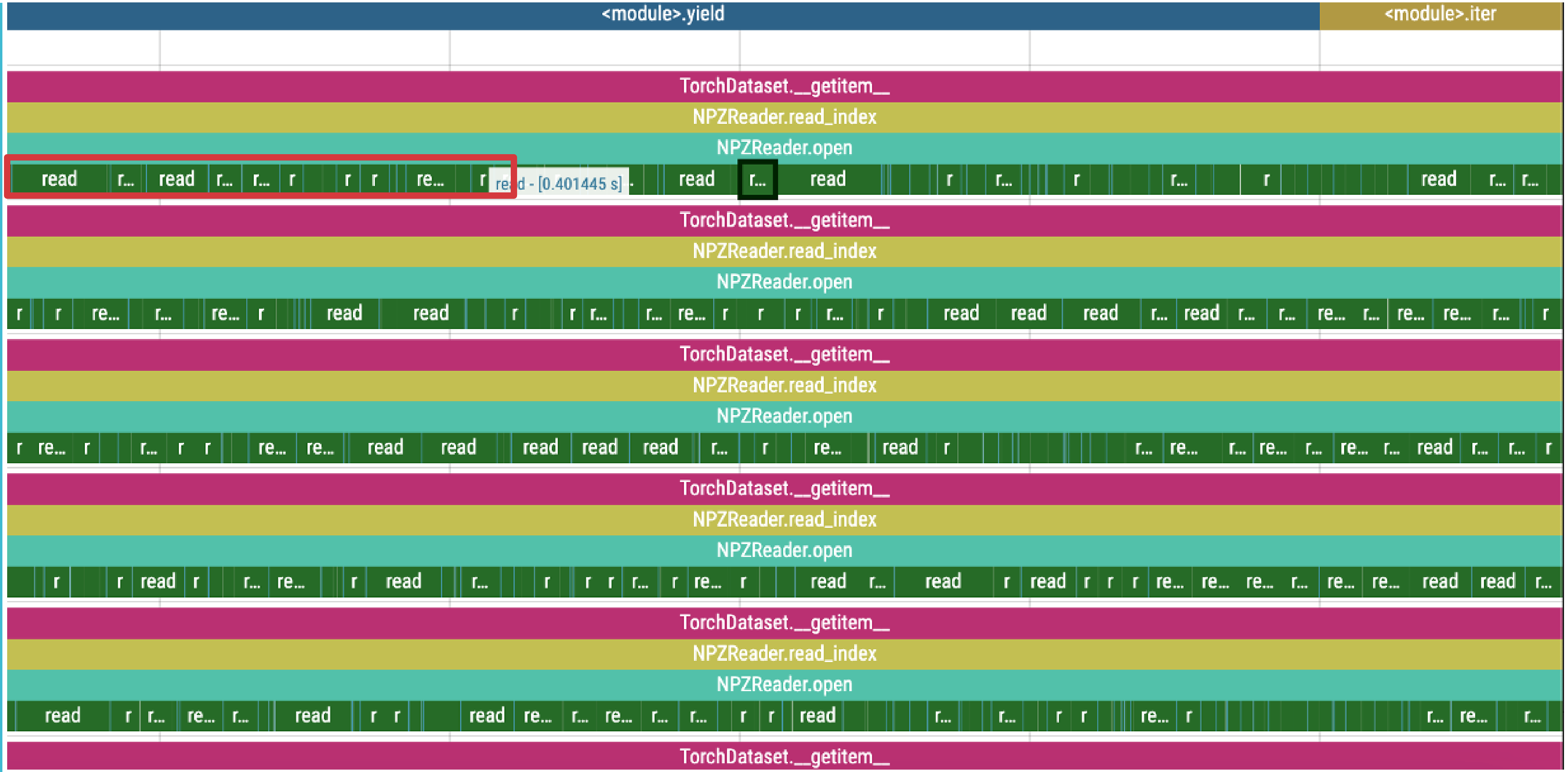
- **Training Dataset (320 GB):** 600 files with 8k samples.
  - Each sample has 8,691 elements.
  - Each element is 8 Bytes.
  - Data format NPZ files.
- **PyTorch Data Loader:** Map-style data loader
  - With 6 worker processes per GPU
  - Batch size of 256 samples
- **Computation**: step is .133 seconds
- **Hardware (used 32 nodes)**
  - Corona cluster at LLNL with 8 GPUs per node
  - 256 GB of RAM and 3.2 TB NVMe SSD per node
- **Tools** DFTracer
  - DFAnalyzer
  - Perfetto UI

Perfetto viewer



- Timeline of the application
- Reading one sample results in Many I/O calls

Perfetto viewer (zoomed)

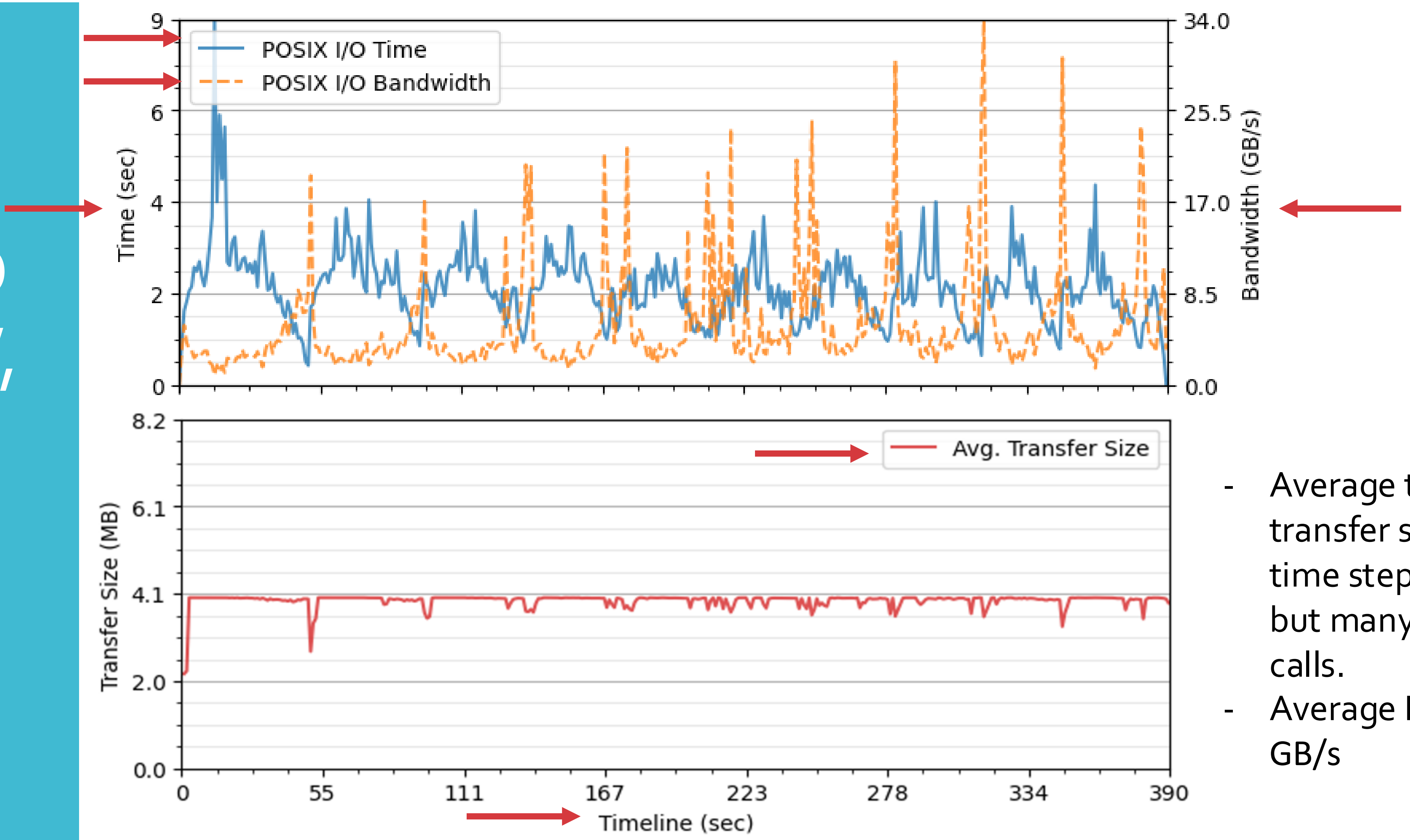


- Many small reads for one numpy open.
- Gaps in reading and numpy open call

Behavior Summary

```
                                   Summary
Allocation     Scheduler Allocation Details
               ├── Nodes: 32
               ├── Processes: 1280
               ├── Thread allocations across nodes (includes dynamically created threads)
               │   ├── Compute: 160
               │   └── I/O: 1280
               └── Events Recorded: 18M
Dataset        Description of Dataset Used
               └── Files: 604
I/O Behavior   Behavior of Application
               ├── Split of Time in application
               │   ├── Total Time: 2701 sec
               │   ├── Overall App Level I/O: 2609.967 sec
               │   ├── Unoverlapped App I/O: 2581.864 sec
               │   ├── Unoverlapped App Compute: 1.199 sec
               │   ├── Compute: 29.303 sec
               │   ├── Overall I/O: 882.836 sec
               │   ├── Unoverlapped I/O: 853.770 sec
               │   └── Unoverlapped Compute: 0.237 sec
               └── Metrics by function
                   ├── Function        |count |                 size                 |
                   ├──                 |      |min  |25   |mean |median|75   |max   |
                   ├── opendir         |208   |NA   |nan  |nan  |NA    |nan  |NA    |
                   ├── __xstat64       |312   |NA   |nan  |nan  |NA    |nan  |NA    |
                   ├── mkdir           |104   |NA   |nan  |nan  |NA    |nan  |NA    |
                   ├── open64          |8K    |NA   |nan  |nan  |NA    |nan  |NA    |
                   ├── __fxstat64      |16K   |NA   |nan  |nan  |NA    |nan  |NA    |
                   ├── lseek64         |17M   |NA   |nan  |nan  |NA    |nan  |NA    |
                   ├── read            |1M    |NA   |4MB  |4MB  |4MB   |4MB  |4MB   |
                   └── close           |8K    |NA   |nan  |nan  |NA    |nan  |NA    |
```

POSIX I/O Time, BW, and Transfer Size



- Average transfer transfer size per time step is good but many read calls.
- Average BW is 2.2 GB/s

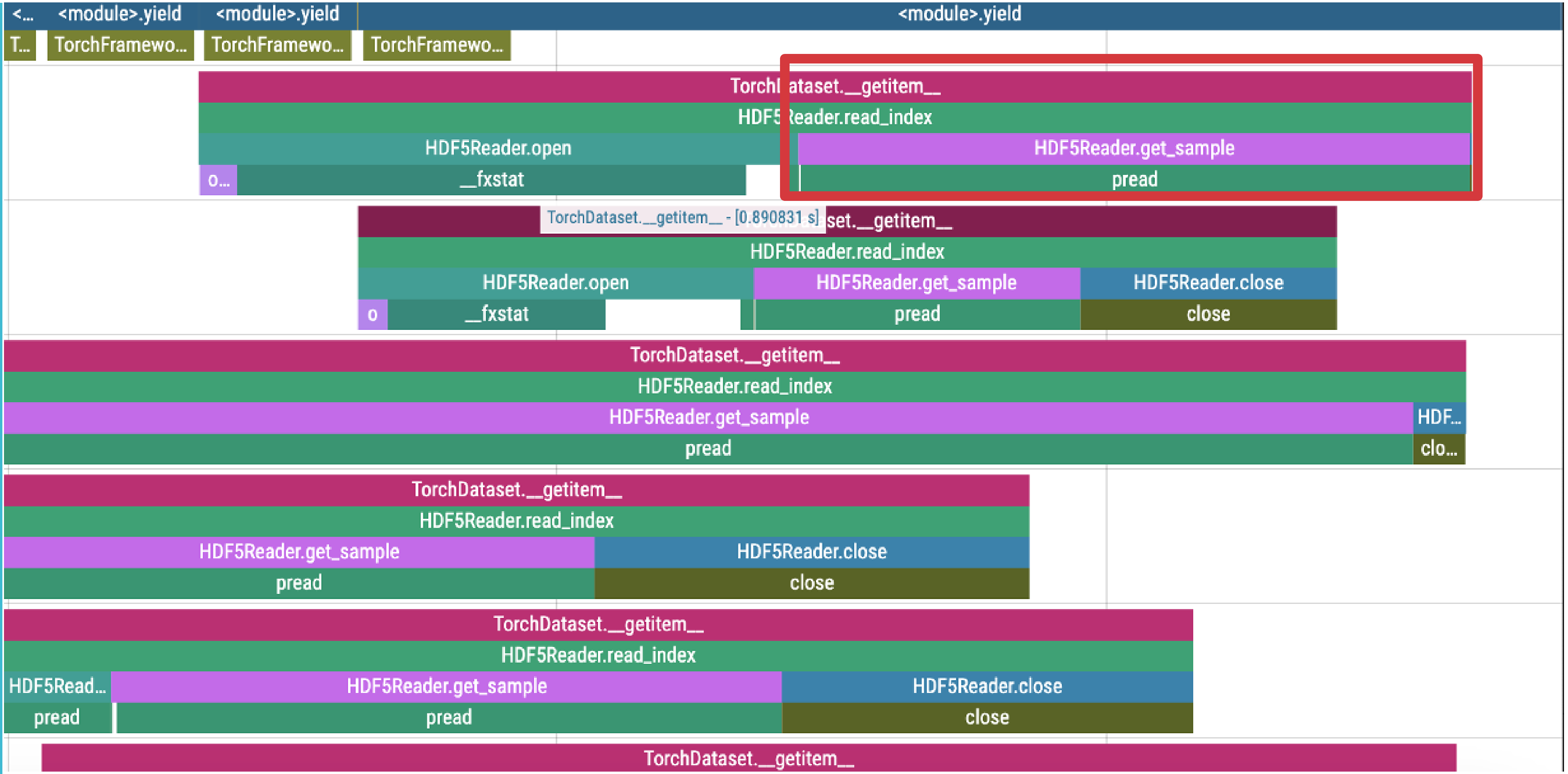
Optimization 1

Switch NPZ to HDF5

Perfetto viewer

- Faster execution time
- The reading a batch is much more efficient.

Perfetto viewer (zoomed)

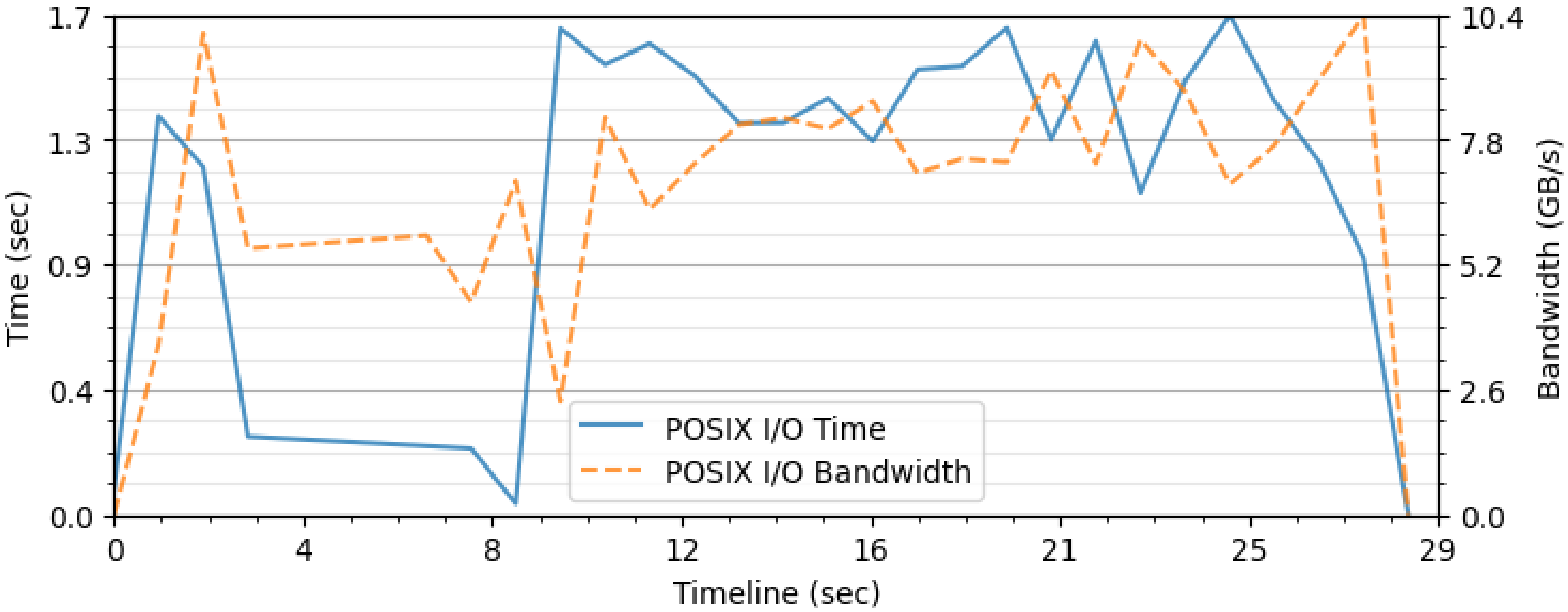


- Large single read calls

Behavior Summary

```
                                   Summary
Allocation      Scheduler Allocation Details
                ├── Nodes: 32
                ├── Processes: 1280
                ├── Thread allocations across nodes (includes dynamically created threads)
                │   ├── Compute:128
                │   └── I/O: 1280
                └── Events Recorded: 437K
Dataset         Description of Dataset Used
                └── Files: 604
I/O Behavior    Behavior of Application
                ├── Split of Time in application
                │   ├── Total Time: 32 sec
                │   ├── Overall App Level I/O: 31.818 sec
                │   ├── Unoverlapped App I/O: 20.849 sec
                │   ├── Unoverlapped App Compute: 0.620 sec
                │   ├── Compute: 11.588 sec
                │   ├── Overall I/O: 30.688 sec
                │   ├── Unoverlapped I/O: 19.723 sec
                │   └── Unoverlapped Compute: 0.624 sec
                └── Metrics by function
                    ├── Function        |count |                  size                       |
                    ├──                 |      |min   |25    |mean  |median|75    |max   |
                    ├── opendir         |368   |NA    |nan   |nan   |NA    |nan   |NA    |
                    ├── __xstat64       |552   |NA    |nan   |nan   |NA    |nan   |NA    |
                    ├── mkdir           |184   |NA    |nan   |nan   |NA    |nan   |NA    |
                    ├── open            |14K   |NA    |nan   |nan   |NA    |nan   |NA    |
                    ├── __fxstat        |14K   |NA    |nan   |nan   |NA    |nan   |NA    |
                    ├── pread           |124K  |8     |80    |512MB |512MB |512MB |512MB |
                    ├── __lxstat        |14K   |NA    |nan   |nan   |NA    |nan   |NA    |
                    └── close           |14K   |NA    |nan   |nan   |NA    |nan   |NA    |
```

POSIX I/O Time and BW

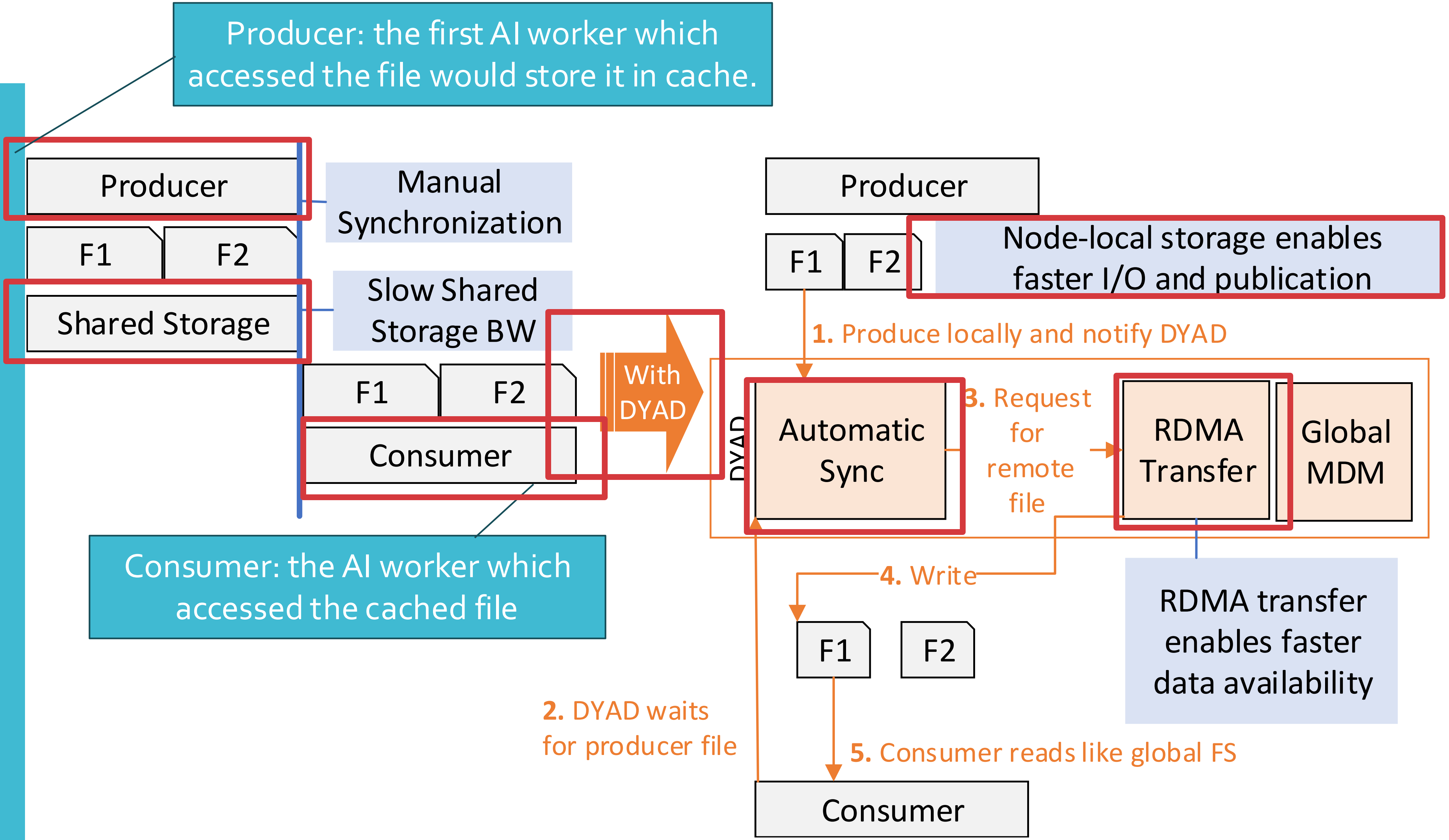


- Larger size improves bandwidth from Lustre to 8GB /s.
- The I/O time is optimized by almost **30x**.
- Overall the timeline is reduced from 390 to 29 seconds **13.4x** faster runtime.

Optimization 2

Utilizing DYAD for node-local caching

Perfetto viewer

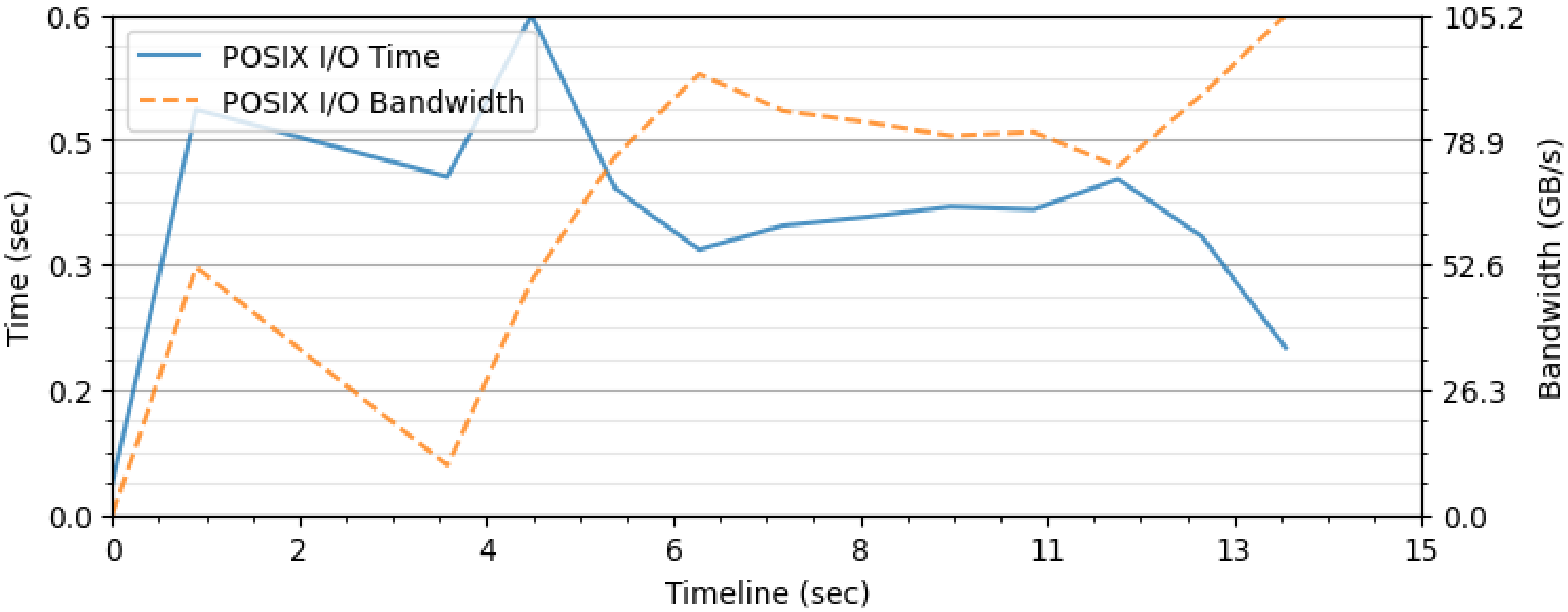


I. Lumsden, H. Devarajan, J. Marquez, S. Brink, D. Boehme, O. Pearce, J.S. Yeom, and M. Taufer. 2024. Empirical Study of Molecular Dynamics Workflow Data Movement: DYAD vs Traditional I/O Systems. In Proceedings of the 2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW).

Behavior Summary

```
Summary
Allocation    Scheduler Allocation Details
              ├── Nodes: 32
              ├── Processes: 1280
              ├── Thread allocations across nodes (includes dynamically created threads)
              │   ├── Compute:128
              │   └── I/O: 1280
              └── Events Recorded: 437K
Dataset       Description of Dataset Used
              └── Files: 604
I/O Behavior  Behavior of Application
              ├── Split of Time in application
              │   ├── Total Time: 12 sec
              │   ├── Overall App Level I/O: 6.782 sec
              │   ├── Unoverlapped App I/O: 0.573 sec
              │   ├── Unoverlapped App Compute: 313.376 sec
              │   ├── Compute: 11.2 sec
              │   ├── Overall I/O: 4.643 sec
              │   ├── Unoverlapped I/O: 0.527 sec
              │   └── Unoverlapped Compute: 0.5 sec
              └── Metrics by function
                  ├── Function        |count |                  size                        |
                  ├──                 |      |min   |25    |mean   |median|75    |max   |
                  ├── opendir         |368   |NA    |nan   |nan    |NA    |nan   |NA    |
                  ├── __xstat64       |552   |NA    |nan   |nan    |NA    |nan   |NA    |
                  ├── mkdir           |184   |NA    |nan   |nan    |NA    |nan   |NA    |
                  ├── open            |14K   |NA    |nan   |nan    |NA    |nan   |NA    |
                  ├── __fxstat        |14K   |NA    |nan   |nan    |NA    |nan   |NA    |
                  ├── pread           |124K  |8     |80    |512MB  |512MB |512MB |512MB |
                  ├── __lxstat        |14K   |NA    |nan   |nan    |NA    |nan   |NA    |
                  └── close           |14K   |NA    |nan   |nan    |NA    |nan   |NA    |
```

POSIX I/O Time and BW



- Faster storage to share data further improves bandwidth to 62GB/s.
- The I/O time is further optimized by **7.5x**.
- Overall the timeline is reduced from 29 to 15 seconds **1.9x** faster runtime.

Key Learnings

- Format has a significant effect on I/O Performance.
- Large Data accesses do not require explicit chunking.
  - Adding explicit chunking hurt performance by 1.8x
- Prefetching policy of PyTorch is not aggressive.
  - Typically it would wait for a cache miss to do next rounds of prefetching.
- Utilizing node-local storage with RDMA for inter-node data movement can speed up AI training.
- Interacting with domain scientists to explain data format intricates is fun.

Conclusions

1. Using Numpy Array APIs lead to many small accesses and large number of metadata calls.
   - Most due to the buffering and decompression schemes within the format.
2. HDF5 format for large simulation samples are efficient to share data.
   - Some apps use shared and other use one sample per file.
   - This leads to **30x** faster performance than Numpy Array
3. Utilizing node-local storage with DYAD can optimize AI training by almost **7.5x** as compared to HDF5 with PFS.

DYAD

SCAN ME

DFTracer

SCAN ME

Analyzer

SCAN ME

Lawrence Livermore National Laboratory