BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

Drishti and HDF5:

What is actually happening in my application?

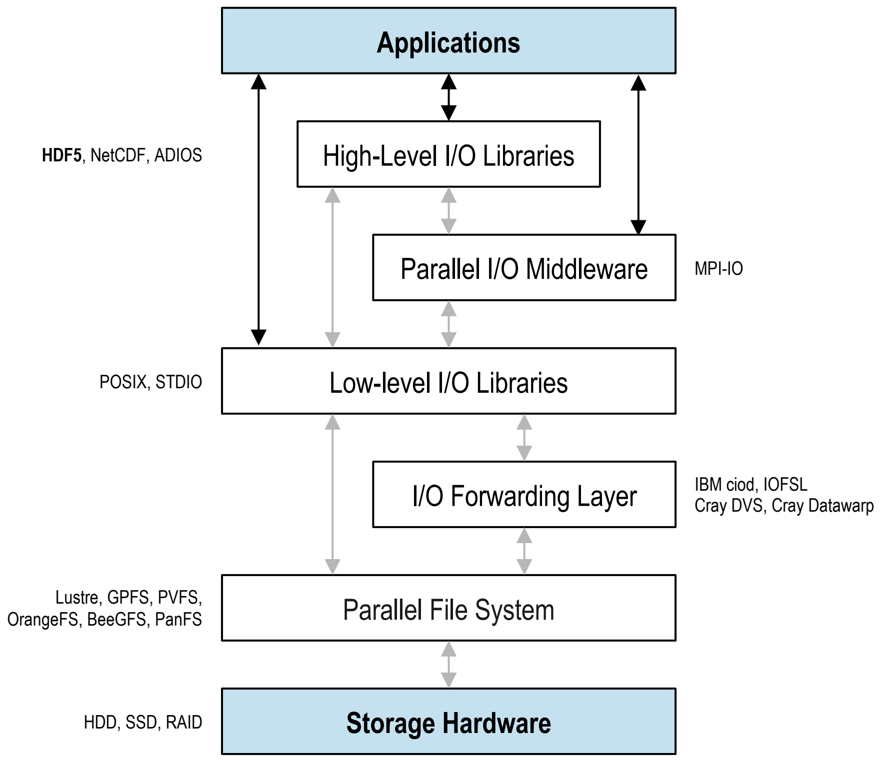
Jean Luca Bez — Scientific Data Division, LBNL

Suren Byna (OSU and LBNL) and Hammad Ather (Univ. of Oregon and LBNL)

HDF5 User Group (HUG) Meeting 2023

Complex I/O stack!

- Using the HPC I/O stack efficiently is a **tricky problem**
- **Interplay of factors** can affect I/O performance
- Various **optimizations** techniques available
- Plethora of tunable **parameters**
- Each layer brings a new set of parameters

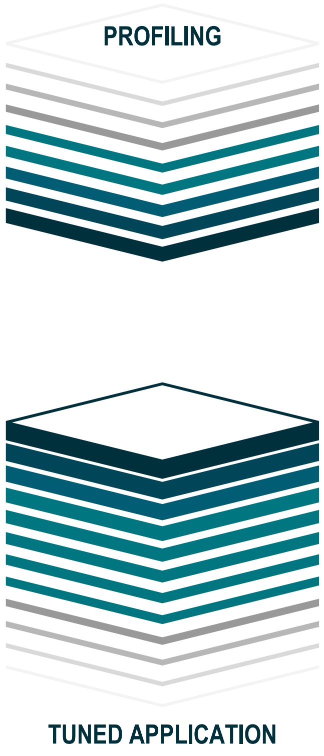


Metrics to the rescue?

- **Darshan** is a popular tool to collect **I/O profiling**
- Extended tracing mode (**DXT**) for a fine grain view
- **Recorder** and **TAU** are other I/O profiling tools
- How to optimize the I/O of my application?

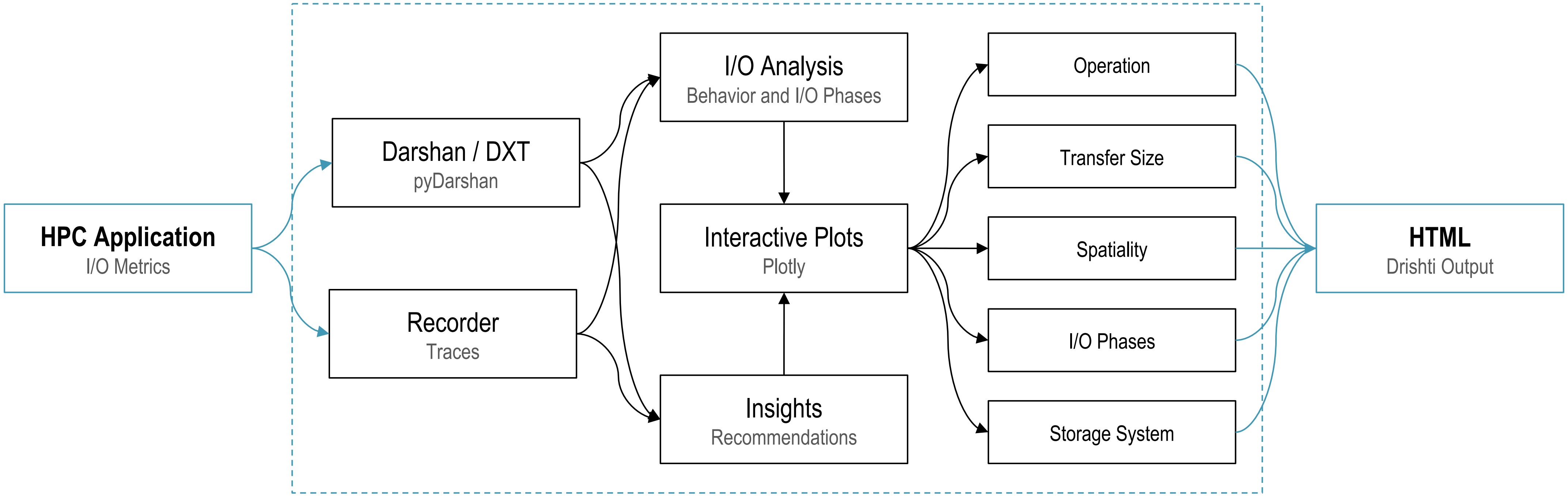
What is the problem?

- There is still a **gap** between profiling and tuning
- How to convert I/O metrics to meaningful information?
  - **Visualize** characteristics, behavior, and bottlenecks
  - **Detect** root causes of I/O bottlenecks
  - **Map** I/O bottlenecks into actionable items
  - **Guide** end-user to tune I/O performance

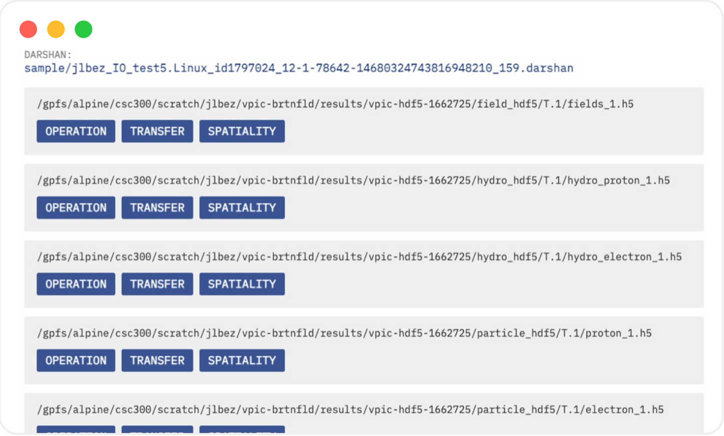


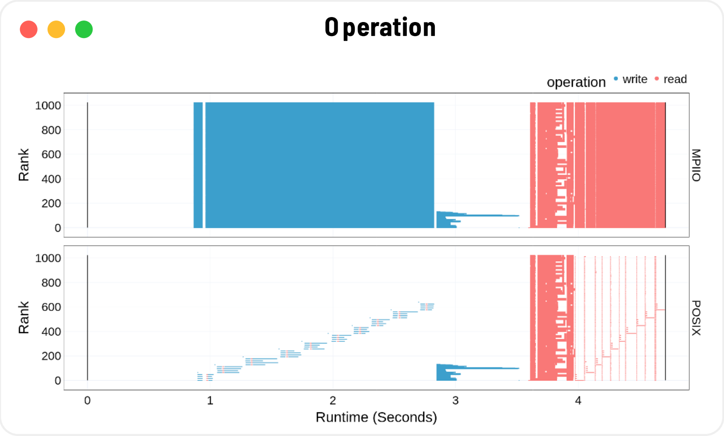
Drishti

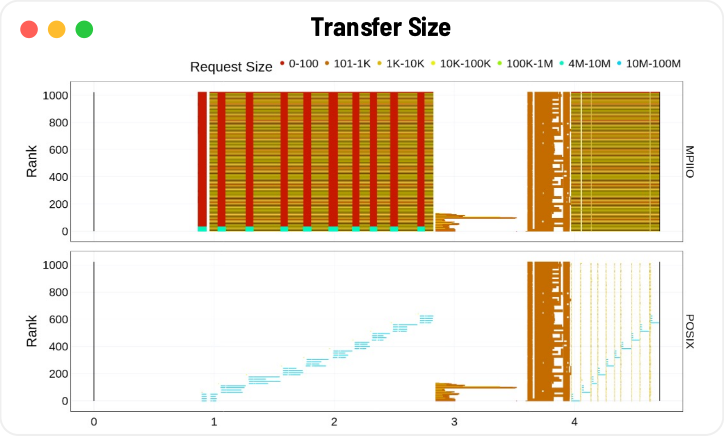
- Sanskrit word meaning "**point of focus**"
  - **Interactive** web based analysis framework
  - Pinpoint root **causes** of I/O performance problems
  - **Detects** typical I/O performance pitfalls
  - Provide a set of **actionable recommendations**
- Working to support multiple sources of I/O metrics

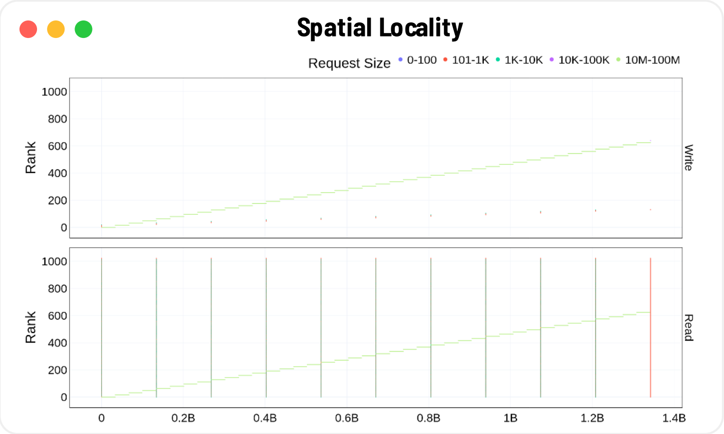

HPC Application
I/O Metrics
Darshan / DXT
pyDarshan
Recorder
Traces
I/O Analysis
Behavior and I/O Phases
Interactive Plots
Plotly
Insights
Recommendations
Operation
Transfer Size
Spatiality
I/O Phases
Storage System
HTML
Drishti Output

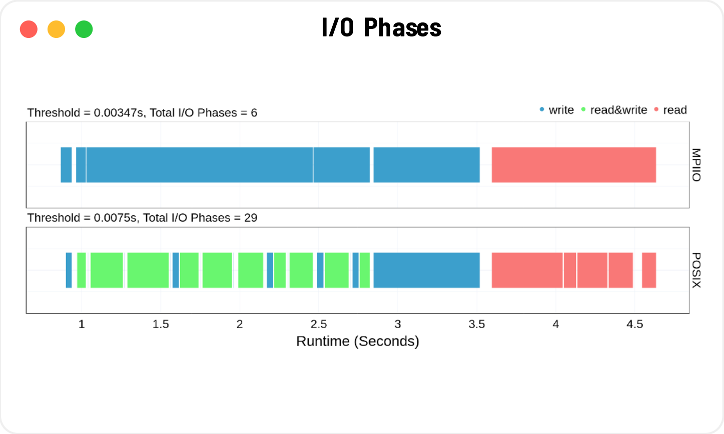
Interactive Analysis

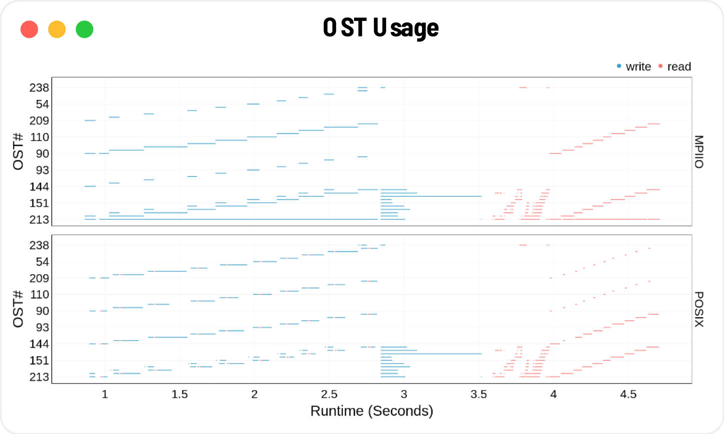










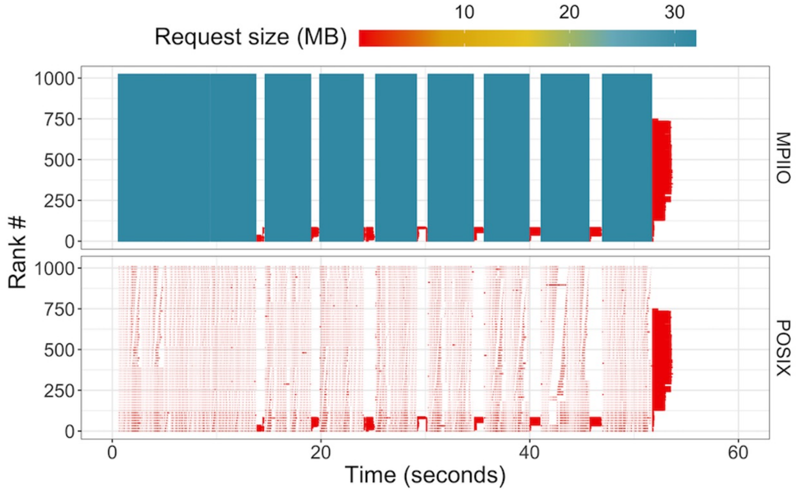


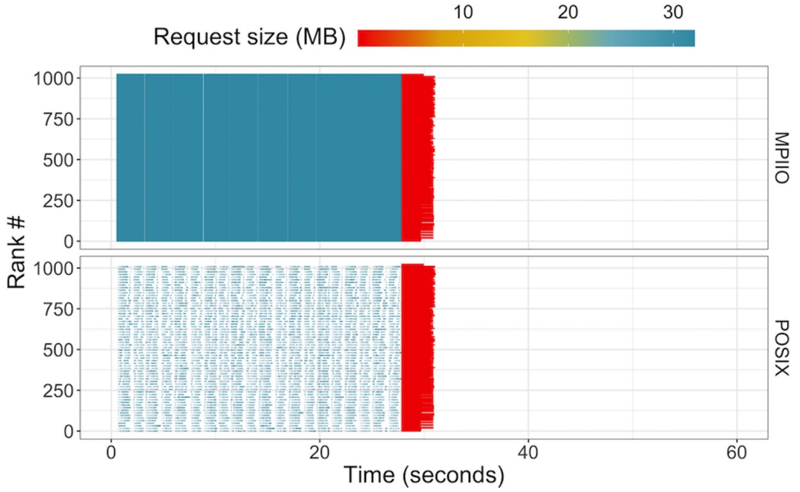
Drishti Triggers

Level	Description
HIGH	High probability of harming I/O performance.
WARN	Detected issues that could cause a significant negative impact on the I/O performance. The confidence of these recommendations is low as available metrics might not be sufficient to detect application design, configuration, or execution choices.
OK	Best practices have been followed.
INFO	Relevant information regarding application configuration.

WarpX / OpenPMD

USE CASE






```
METADATA

▶ Application is write operation intensive (60.83% writes vs. 39.17% reads)
▶ Application is write size intensive (64 15% write vs  35 85% read)
▶ Application issues a high number (100.00%) of misaligned file requests
```

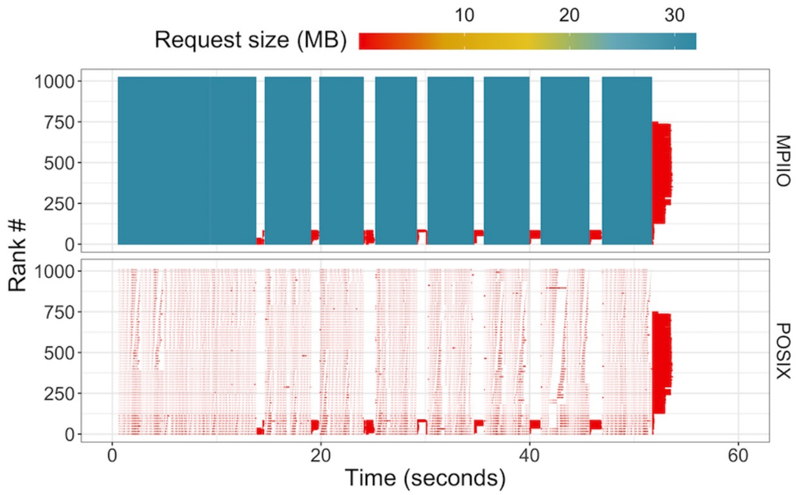
```
OPERATIONS

▶ Application issues a high number (275840) of small read requests (i.e., < 1MB) which
represents 100.00% of all read/write requests
   ↪ 275840 (100.00%) small read requests are to "8a_parallel_3Db_0000001.h5"
▶ Application issues a high number (427386) of small write requests (i.e., < 1MB) which
represents 99.75% of all read/write requests
   ↪ 275840 (64.38%) small write requests are to "8a_parallel_3Db_0000001.h5"
▶ Application mostly uses consecutive (97.67%) and sequential (2.16%) read requests
▶ Application mostly uses consecutive (97.85%) and sequential (1.17%) write requests
▶ Application uses MPI-IO and write data using 7680 (92.50%) collective operations
▶ Application could benefit from non-blocking (asynchronous) reads
▶ Application could benefit from non blocking (asynchronous) writes
```

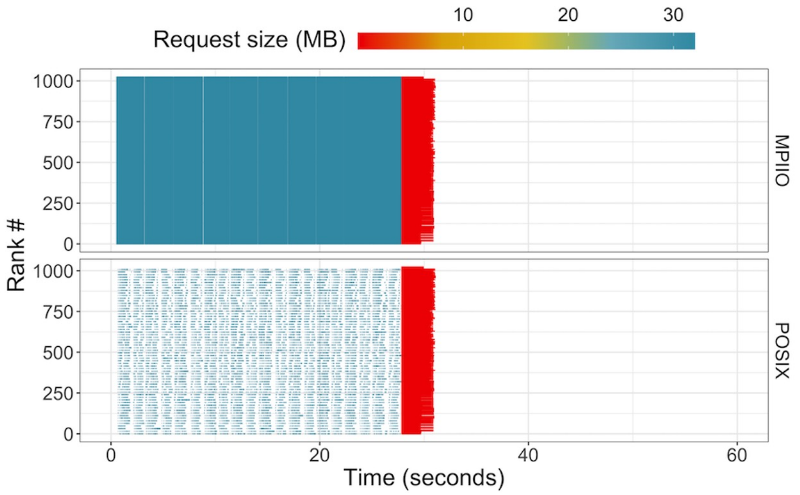
```
METADATA

▶ Application is write operation intensive (90.85% writes vs. 9.15% reads)
▶ Application is write size intensive (91 14% write vs  8 86% read)
▶ Application might have redundant read traffic (more data read than the highest offset)
```

```
OPERATIONS

▶ Application is issuing a high number (565) of random read operations (35.25%)
▶ Application mostly uses consecutive (88.56%) and sequential (7.02%) write requests
▶ Application uses MPI-IO and write data using 8448 (100.00%) collective operations
▶ Application could benefit from non-blocking (asynchronous) reads
▶ Application could benefit from non-blocking (asynchronous) writes
```

WarpX / OpenPMD

USE CASE






```
METADATA

▶ Application is write operation intensive (60.83% writes vs. 39.17% reads)
▶ Application is write size intensive (64 15% write vs  35 85% read)
▶ Application issues a high number (100.00%) of misaligned file requests

OPERATIONS

▶ Application issues a high number (275840) of small read requests (i.e., < 1MB) which
represents 100.00% of all read/write requests
   ↪ 275840 (100.00%) small read requests are to "8a_parallel_3Db_0000001.h5"
▶ Application issues a high number (427386) of small write requests (i.e., < 1MB) which
represents 99.75% of all read/write requests
   ↪ 275840 (64.38%) small write requests are to "8a_parallel_3Db_0000001.h5"
▶ Application mostly uses consecutive (97.67%) and sequential (2.16%) read requests
▶ Application mostly uses consecutive (97.85%) and sequential (1.17%) write requests
▶ Application uses MPI-IO and write data using 7680 (92.50%) collective operations
▶ Application could benefit from non-blocking (asynchronous) reads
▶ Application could benefit from non blocking (asynchronous) writes
```

```
METADATA

▶ Application is write operation intensive (90.85% writes vs. 9.15% reads)
▶ Application is write size intensive (91 14% write vs  8 86% read)
▶ Application might have redundant read traffic (more data read than the highest offset)

OPERATIONS

▶ Application is issuing a high number (565) of random read operations (35.25%)
▶ Application mostly uses consecutive (88.56%) and sequential (7.02%) write requests
▶ Application uses MPI-IO and write data using 8448 (100.00%) collective operations
▶ Application could benefit from non-blocking (asynchronous) reads
▶ Application could benefit from non-blocking (asynchronous) writes
```

AMReX

USE CASE

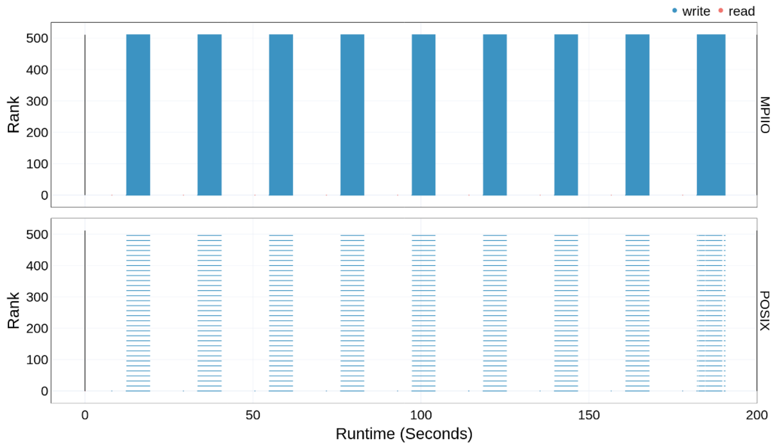
- Uses highly parallel Adaptive Mesh Refinement (AMR) algorithms
  - Solve partial differential equations
  - Block-structured meshes
- Astrophysics, atmospheric modeling, combustion, cosmology, and particle accelerators
- Experimental setup:
  - 512 ranks (32 nodes) in Cori
  - 1024 domain size
  - 1 level, 6 components, 2 particles per cell
  - 10 output plot files

AMReX

USE CASE

2.1× speedup

from 211 to 100 seconds

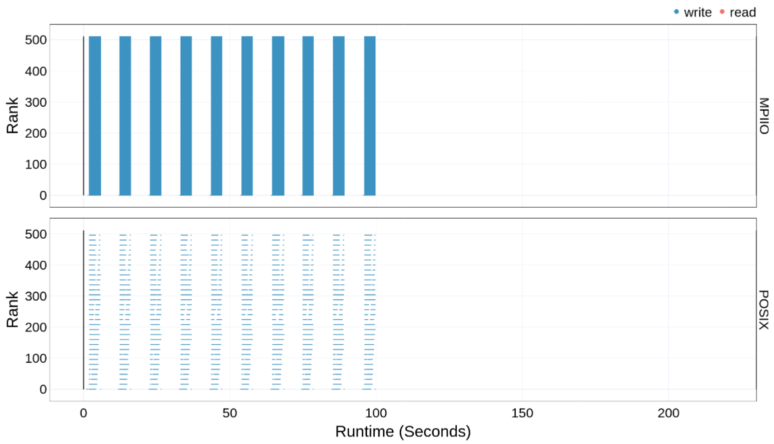


METADATA

- Application is write operation intensive (99.98% writes vs. 0.02% reads)
- Application is write size intensive (100.00% write vs. 0.00% read)

OPERATIONS

- Application issues a high number (491640) of small write requests (i.e., < 1MB) which represents 99.99% of all read/write requests
  - ↪ 98328 (20.00%) small write requests are to "plt00001.h5"
  - ↪ 98328 (20.00%) small write requests are to "plt00002.h5"
  - ↪ 98328 (20.00%) small write requests are to "plt00005.h5"
  - ↪ 98328 (20.00%) small write requests are to "plt00009.h5"
  - ↪ 98328 (20.00%) small write requests are to "plt00000.h5"
  - ↪ 98328 (20.00%) small write requests are to "plt00004.h5"
  - ↪ 98328 (20.00%) small write requests are to "plt00003.h5"
  - ↪ 98328 (20.00%) small write requests are to "plt00006.h5"
  - ↪ 98328 (20.00%) small write requests are to "plt00007.h5"
  - ↪ 98328 (20.00%) small write requests are to "plt00008.h5"
  - ↪ **Recommendations:**
    - ↪ Consider buffering write operations into larger more contiguous ones
    - ↪ Since the application already uses MPI-IO, consider using collective I/O calls (e.g. MPI_File_write_all() or MPI_File_write_at_all()) to aggregate requests into larger ones
- Application mostly uses consecutive (25.41%) and sequential (32.79%) read requests
- Application mostly uses consecutive (0.01%) and sequential (99.98%) write requests
- Application issues a high number (491640) of small write requests to a shared file (i.e., < 1MB) which represents 99.99% of all shared file write requests
  - ↪ 49164 (10.00%) small writes requests are to "plt00001.h5"
  - ↪ 49164 (10.00%) small writes requests are to "plt00002.h5"
  - ↪ 49164 (10.00%) small writes requests are to "plt00005.h5"
  - ↪ 49164 (10.00%) small writes requests are to "plt00009.h5"
  - ↪ 49164 (10.00%) small writes requests are to "plt00000.h5"
  - ↪ 49164 (10.00%) small writes requests are to "plt00004.h5"
  - ↪ 49164 (10.00%) small writes requests are to "plt00003.h5"
  - ↪ 49164 (10.00%) small writes requests are to "plt00006.h5"
  - ↪ 49164 (10.00%) small writes requests are to "plt00007.h5"
  - ↪ 49164 (10.00%) small writes requests are to "plt00008.h5"
  - ↪ **Recommendations:**
    - ↪ Consider coalescing write requests into larger more contiguous ones using MPI-IO collective operations
- Application uses MPI-IO and write data using 15360 (99.81%) collective operations
- Application could benefit from non-blocking (asynchronous) reads
  - ↪ **Recommendations:**
    - ↪ Since you use HDF5, consider using the ASYNC I/O VOL connector (https://github.com/hpc-io/vol-async)
    - ↪ Since you use MPI-IO, consider non-blocking/asynchronous I/O operations
- Application could benefit from non-blocking (asynchronous) writes
  - ↪ **Recommendations:**
    - ↪ Since you use HDF5, consider using the ASYNC I/O VOL connector (https://github.com/hpc-io/vol-async)
    - ↪ Since you use MPI-IO, consider non-blocking/asynchronous I/O operations



METADATA

- Application is write operation intensive (99.61% writes vs. 0.39% reads)
- Application is write size intensive (100.00% write vs. 0.00% read)

OPERATIONS

- Application mostly uses consecutive (24.79%) and sequential (33.06%) read requests
- Application mostly uses consecutive (0.16%) and sequential (99.64%) write requests
- Application uses MPI-IO and write data using 15360 (99.81%) collective operations

Coming Soon!

- Improved recommendations
  - Issues and suggestions pointing to the source-code location
  - Enhanced code snippets for tuning
- API to handle other sources of metrics
  - Full support for Recorder 2.0 (github.com/uiuc-hpc/Recorder)
  - File system metrics
- Support for interdependent and complex triggers

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

Drishti and HDF5:
What is actually happening in my application?

- Pinpointing root causes of I/O inefficiencies **requires**:
  - Detailed metric analysis
  - Understanding of the HPC I/O stack
- **Drishti** is an interactive I/O analysis framework
  - Seeks to close the gap between **trace collection, analysis, and tuning**
  - Detects common root causes of I/O performance inefficiencies
  - Provides actionable recommendations to the users

EXASCALE COMPUTING PROJECT

github.com/hpc-io/drishti

jlbez@lbl.gov and **sbyna@lbl.gov**