Drishti

WHERE IS THE I/O BOTTLENECK?

JEAN LUCA BEZ, HAMMAD ATHER, YANKUN XIA, SUREN BYNA

U.S. DEPARTMENT OF ENERGY
Office of Science

BERKELEY LAB

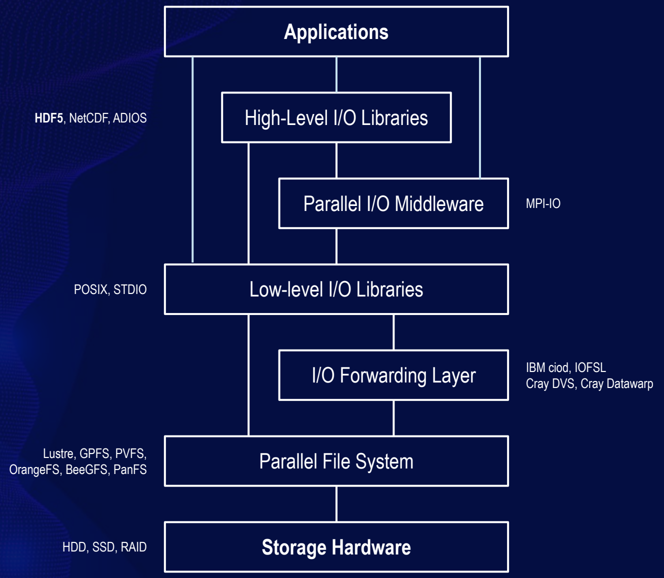
Scientific Data Division

The Ohio State University

University of Oregon

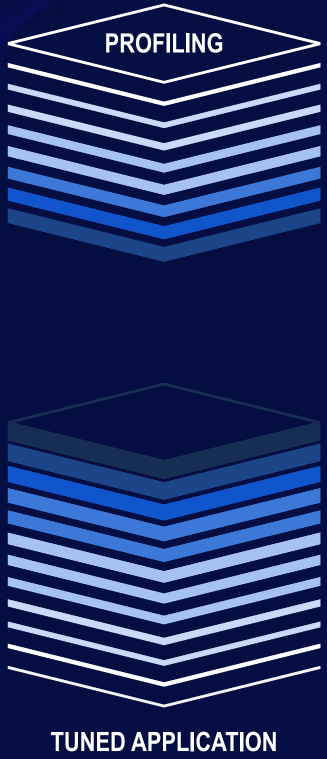
COMPLEX I/O STACK!

- 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



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

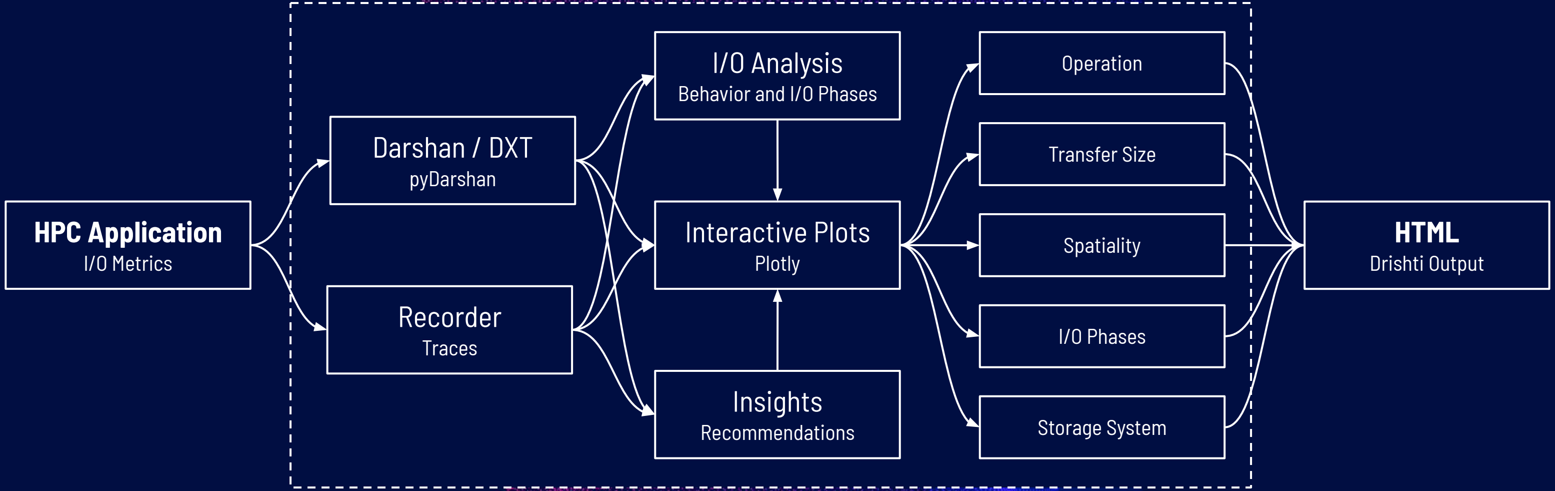


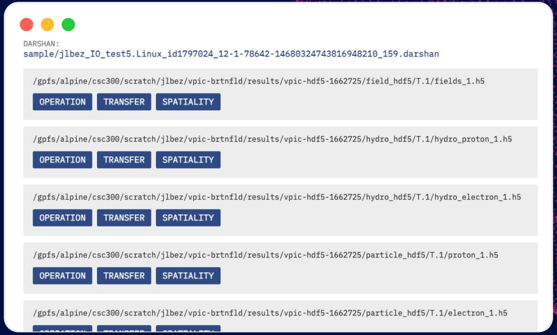
WHAT IS THE PROBLEM?

- 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

PROFILING
Drishti
TUNED APPLICATION

Drishti


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



DARSHAN:
sample/jlbez_IO_test5.Linux_id1797024_12-1-78642-14680324743816948210_159.darshan

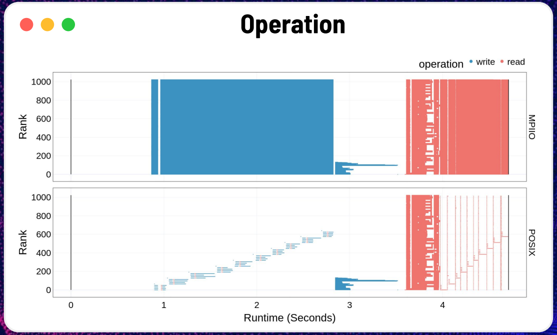
```
/gpfs/alpine/csc300/scratch/jlbez/vpic-brtnfld/results/vpic-hdf5-1662725/field_hdf5/T.1/fields_1.h5
```

```
/gpfs/alpine/csc300/scratch/jlbez/vpic-brtnfld/results/vpic-hdf5-1662725/hydro_hdf5/T.1/hydro_proton_1.h5
```

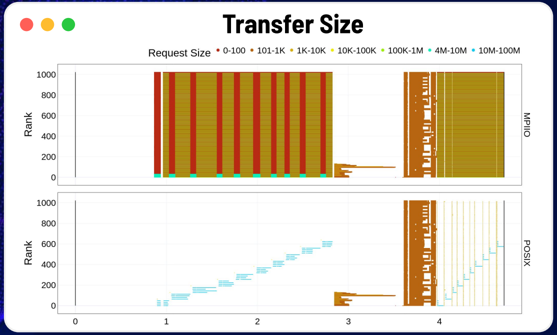
```
/gpfs/alpine/csc300/scratch/jlbez/vpic-brtnfld/results/vpic-hdf5-1662725/hydro_hdf5/T.1/hydro_electron_1.h5
```

```
/gpfs/alpine/csc300/scratch/jlbez/vpic-brtnfld/results/vpic-hdf5-1662725/particle_hdf5/T.1/proton_1.h5
```

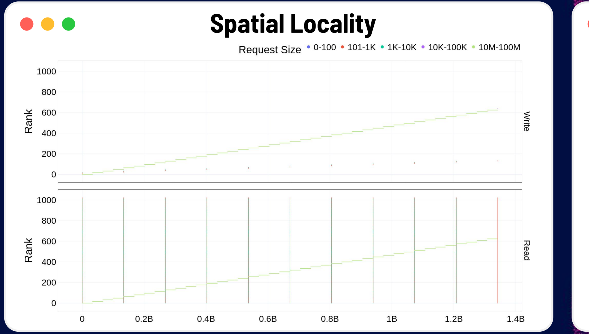
```
/gpfs/alpine/csc300/scratch/jlbez/vpic-brtnfld/results/vpic-hdf5-1662725/particle_hdf5/T.1/electron_1.h5
```



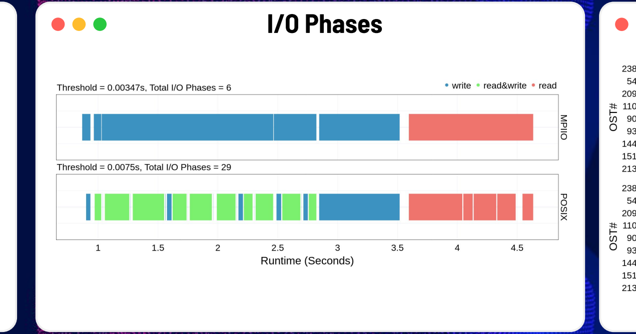
Operation



Transfer Size



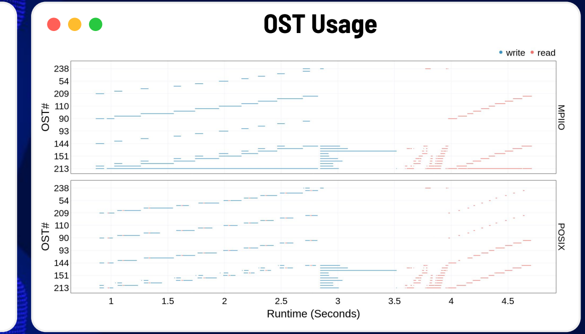
Spatial Locality



I/O Phases

Threshold = 0.00347s, Total I/O Phases = 6

Threshold = 0.0075s, Total I/O Phases = 29



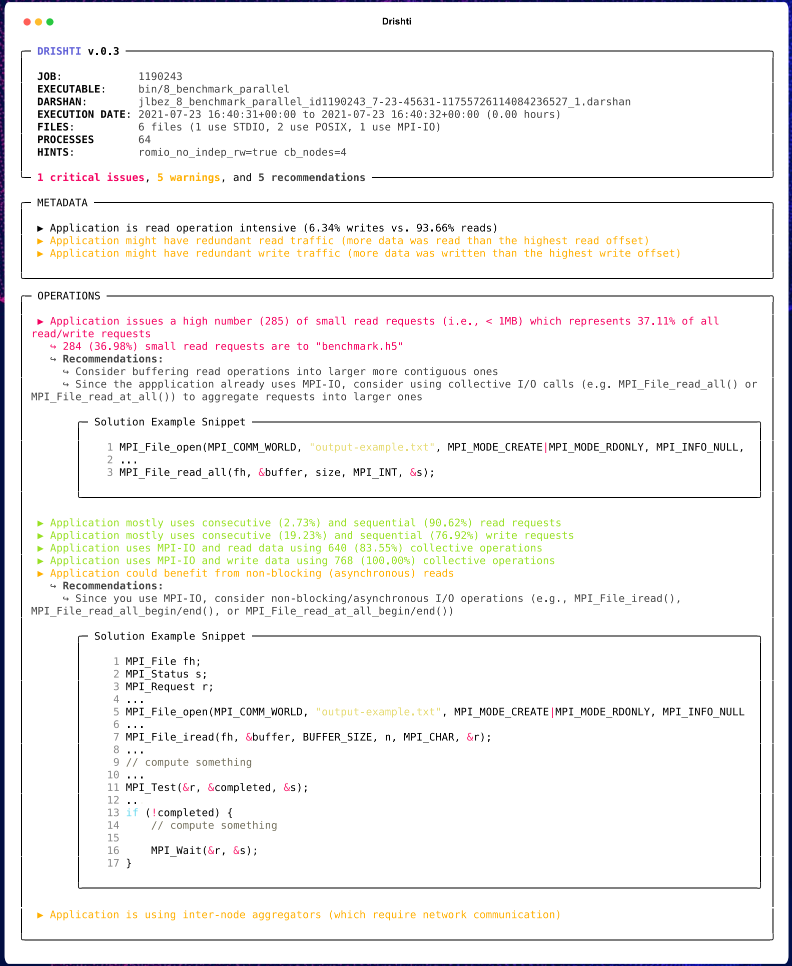
OST Usage

Number of **critical issues**, warning, and **recommendations**

Drishti checks metrics for **over 30 triggers**

Highlight the **file** that triggered the issue

Sample **code solutions** are provided



```
DRISHTI v.0.3

JOB:            1190243
EXECUTABLE:     bin/8_benchmark_parallel
DARSHAN:        jlbez_8_benchmark_parallel_id1190243_7-23-45631-11755726114084236527_1.darshan
EXECUTION DATE: 2021-07-23 16:40:31+00:00 to 2021-07-23 16:40:32+00:00 (0.00 hours)
FILES:          6 files (1 use STDIO, 2 use POSIX, 1 use MPI-IO)
PROCESSES       64
HINTS:          romio_no_indep_rw=true cb_nodes=4

1 critical issues, 5 warnings, and 5 recommendations

METADATA

▶ Application is read operation intensive (6.34% writes vs. 93.66% reads)
▶ Application might have redundant read traffic (more data was read than the highest read offset)
▶ Application might have redundant write traffic (more data was written than the highest write offset)

OPERATIONS

▶ Application issues a high number (285) of small read requests (i.e., < 1MB) which represents 37.11% of all
read/write requests
  ↪ 284 (36.98%) small read requests are to "benchmark.h5"
  ↪ Recommendations:
    ↪ Consider buffering read operations into larger more contiguous ones
    ↪ Since the appplication already uses MPI-IO, consider using collective I/O calls (e.g. MPI_File_read_all() or
MPI_File_read_at_all()) to aggregate requests into larger ones

        Solution Example Snippet

          MPI_File_open(MPI_COMM_WORLD, "output-example.txt", MPI_MODE_CREATE|MPI_MODE_RDONLY, MPI_INFO_NULL,
          ...
          MPI_File_read_all(fh, &buffer, size, MPI_INT, &s);

▶ Application mostly uses consecutive (2.73%) and sequential (90.62%) read requests
▶ Application mostly uses consecutive (19.23%) and sequential (76.92%) write requests
▶ Application uses MPI-IO and read data using 640 (83.55%) collective operations
▶ Application uses MPI-IO and write data using 768 (100.00%) collective operations
▶ Application could benefit from non-blocking (asynchronous) reads
  ↪ Recommendations:
    ↪ Since you use MPI-IO, consider non-blocking/asynchronous I/O operations (e.g., MPI_File_iread(),
MPI_File_read_all_begin/end(), or MPI_File_read_at_all_begin/end())

        Solution Example Snippet

          MPI_File fh;
          MPI_Status s;
          MPI_Request r;
          ...
          MPI_File_open(MPI_COMM_WORLD, "output-example.txt", MPI_MODE_CREATE|MPI_MODE_RDONLY, MPI_INFO_NULL
          ...
          MPI_File_iread(fh, &buffer, BUFFER_SIZE, n, MPI_CHAR, &r);
          ...
          // compute something
          ...
          MPI_Test(&r, &completed, &s);
          ..
          if (!completed) {
              // compute something

              MPI_Wait(&r, &s);
          }

▶ Application is using inter-node aggregators (which require network communication)
```

Current version only checks **profiling** metrics

Severity based on certainty and impact: **high**, **medium**, **low**, info

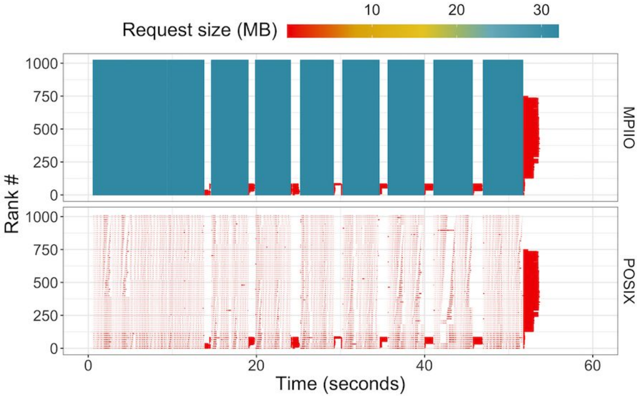
Provides **actionable feedback** for users

Drishti can check for HDF5 usage to fine **tune the recommendations**

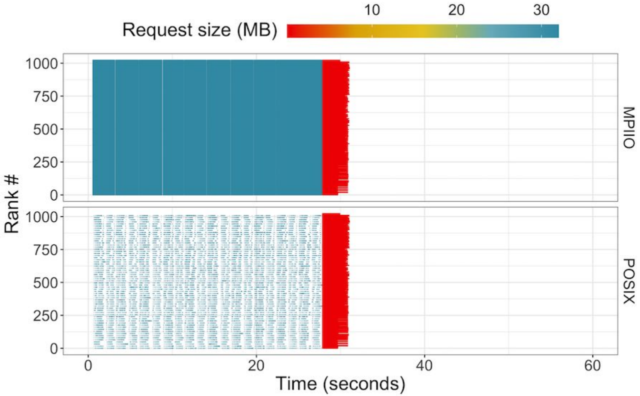
Multiple output **formats**: textual, SVG, HTML

WARPX / OPENPMD

USE CASE



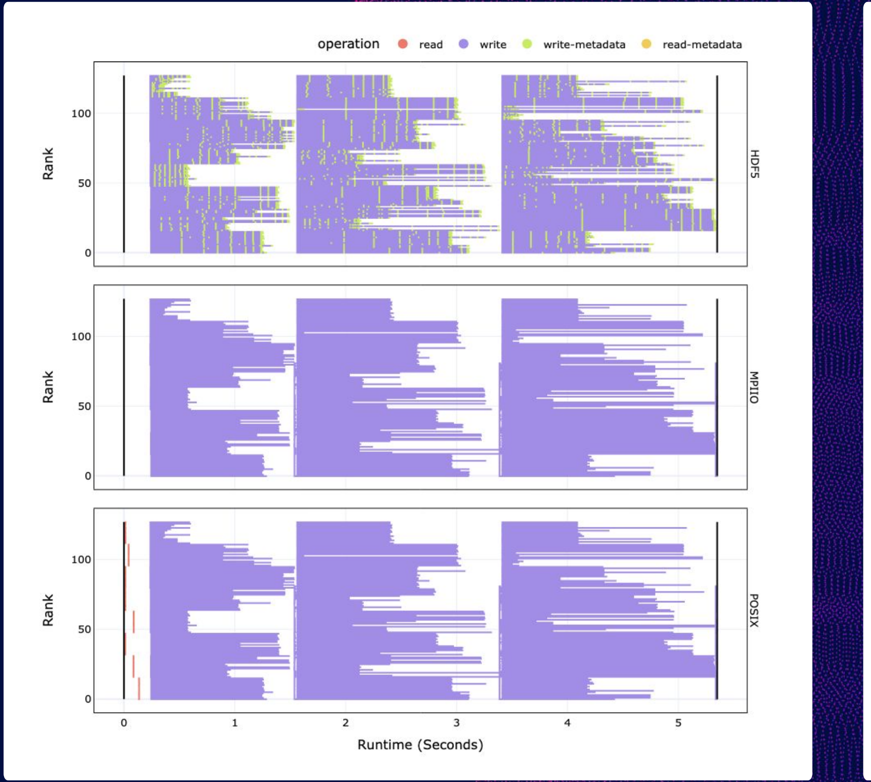
```
╭─ METADATA ──────────────────────────────────────────────────────────────╮

 ▶ 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

╰─────────────────────────────────────────────────────────────────────────╯
```

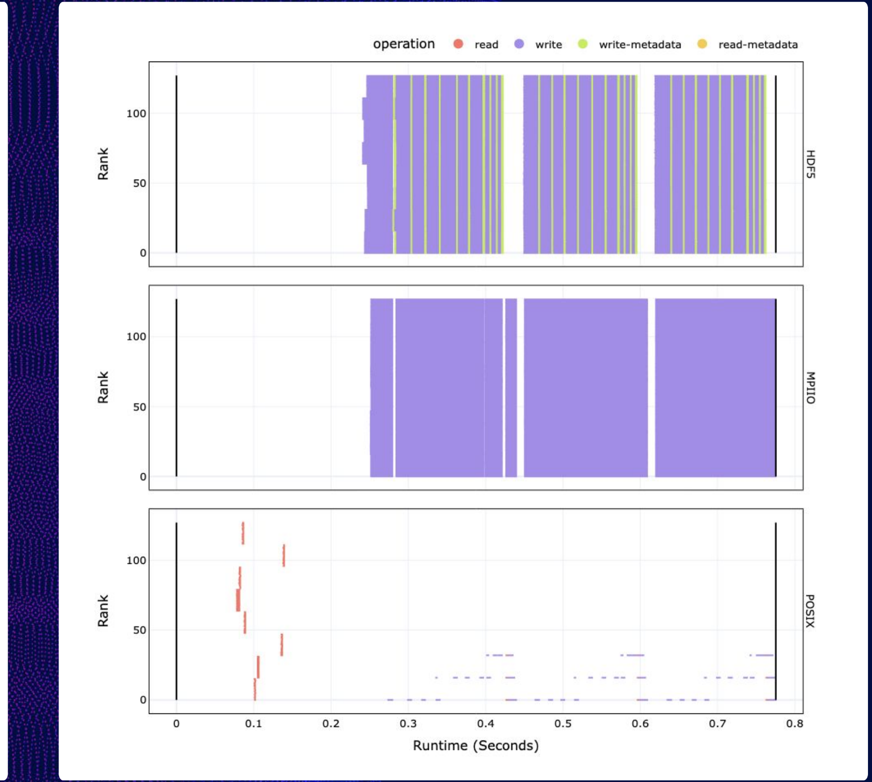


```
╭─ METADATA ──────────────────────────────────────────────────────────────╮

 ▶ 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

╰─────────────────────────────────────────────────────────────────────────╯
```

CROSS LAYER EXPLORATION

HDF5 VOL CONNECTOR

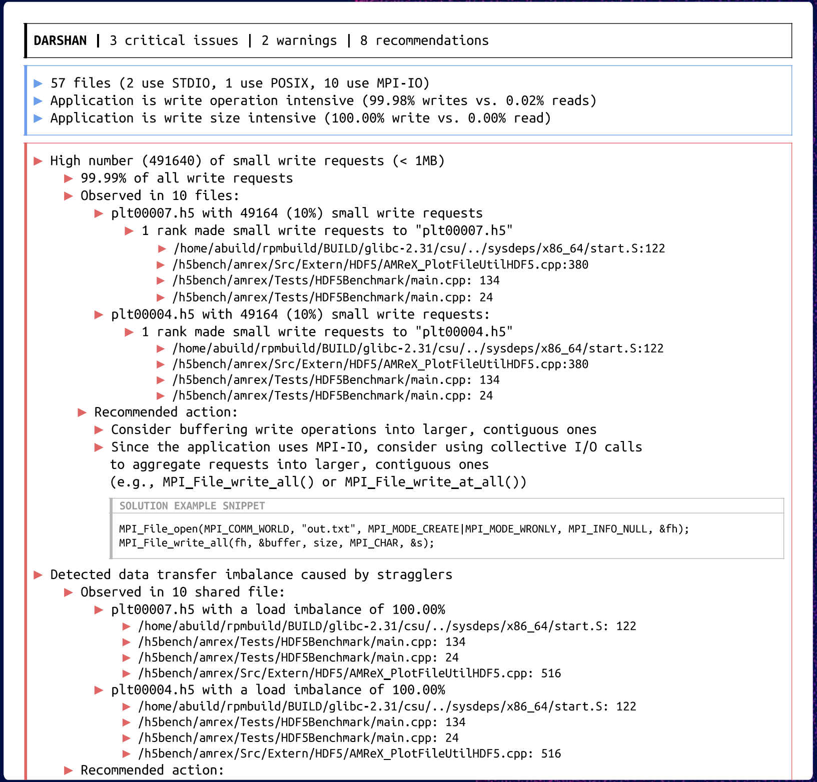




CROSS LAYER EXPLORATION

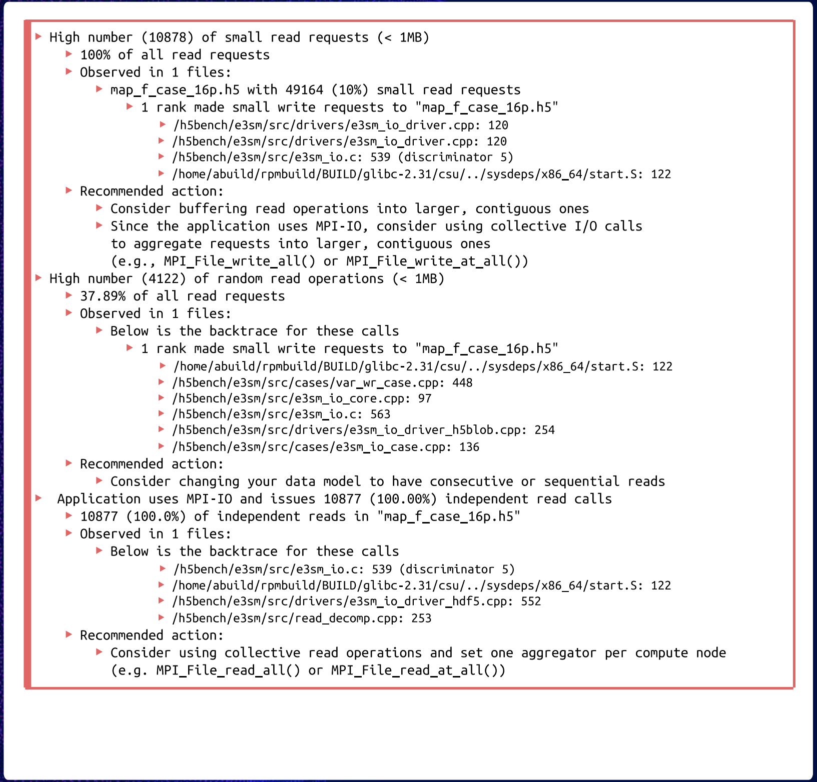
SOURCE CODE

AMREX



```
DARSHAN | 3 critical issues | 2 warnings | 8 recommendations

▶ 57 files (2 use STDIO, 1 use POSIX, 10 use MPI-IO)
▶ Application is write operation intensive (99.98% writes vs. 0.02% reads)
▶ Application is write size intensive (100.00% write vs. 0.00% read)

▶ High number (491640) of small write requests (< 1MB)
    ▶ 99.99% of all write requests
    ▶ Observed in 10 files:
        ▶ plt00007.h5 with 49164 (10%) small write requests
            ▶ 1 rank made small write requests to "plt00007.h5"
                ▶ /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S:122
                ▶ /h5bench/amrex/Src/Extern/HDF5/AMReX_PlotFileUtilHDF5.cpp:380
                ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 134
                ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 24
        ▶ plt00004.h5 with 49164 (10%) small write requests:
            ▶ 1 rank made small write requests to "plt00004.h5"
                ▶ /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S:122
                ▶ /h5bench/amrex/Src/Extern/HDF5/AMReX_PlotFileUtilHDF5.cpp:380
                ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 134
                ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 24
      ▶ Recommended action:
        ▶ Consider buffering write operations into larger, contiguous ones
        ▶ Since the application uses MPI-IO, consider using collective I/O calls
          to aggregate requests into larger, contiguous ones
          (e.g., MPI_File_write_all() or MPI_File_write_at_all())

          SOLUTION EXAMPLE SNIPPET

          MPI_File_open(MPI_COMM_WORLD, "out.txt", MPI_MODE_CREATE|MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
          MPI_File_write_all(fh, &buffer, size, MPI_CHAR, &s);

▶ Detected data transfer imbalance caused by stragglers
    ▶ Observed in 10 shared file:
        ▶ plt00007.h5 with a load imbalance of 100.00%
            ▶ /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S: 122
            ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 134
            ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 24
            ▶ /h5bench/amrex/Src/Extern/HDF5/AMReX_PlotFileUtilHDF5.cpp: 516
        ▶ plt00004.h5 with a load imbalance of 100.00%
            ▶ /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S: 122
            ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 134
            ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 24
            ▶ /h5bench/amrex/Src/Extern/HDF5/AMReX_PlotFileUtilHDF5.cpp: 516
    ▶ Recommended action:
```

E3SM



```
▶ High number (10878) of small read requests (< 1MB)
    ▶ 100% of all read requests
    ▶ Observed in 1 files:
        ▶ map_f_case_16p.h5 with 49164 (10%) small read requests
            ▶ 1 rank made small write requests to "map_f_case_16p.h5"
                ▶ /h5bench/e3sm/src/drivers/e3sm_io_driver.cpp: 120
                ▶ /h5bench/e3sm/src/drivers/e3sm_io_driver.cpp: 120
                ▶ /h5bench/e3sm/src/e3sm_io.c: 539 (discriminator 5)
                ▶ /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S: 122
    ▶ Recommended action:
        ▶ Consider buffering read operations into larger, contiguous ones
        ▶ Since the application uses MPI-IO, consider using collective I/O calls
          to aggregate requests into larger, contiguous ones
          (e.g., MPI_File_write_all() or MPI_File_write_at_all())
▶ High number (4122) of random read operations (< 1MB)
    ▶ 37.89% of all read requests
    ▶ Observed in 1 files:
        ▶ Below is the backtrace for these calls
            ▶ 1 rank made small write requests to "map_f_case_16p.h5"
                ▶ /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S: 122
                ▶ /h5bench/e3sm/src/cases/var_wr_case.cpp: 448
                ▶ /h5bench/e3sm/src/e3sm_io_core.cpp: 97
                ▶ /h5bench/e3sm/src/e3sm_io.c: 563
                ▶ /h5bench/e3sm/src/drivers/e3sm_io_driver_h5blob.cpp: 254
                ▶ /h5bench/e3sm/src/cases/e3sm_io_case.cpp: 136
    ▶ Recommended action:
        ▶ Consider changing your data model to have consecutive or sequential reads
▶  Application uses MPI-IO and issues 10877 (100.00%) independent read calls
    ▶ 10877 (100.0%) of independent reads in "map_f_case_16p.h5"
    ▶ Observed in 1 files:
        ▶ Below is the backtrace for these calls
                ▶ /h5bench/e3sm/src/e3sm_io.c: 539 (discriminator 5)
                ▶ /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S: 122
                ▶ /h5bench/e3sm/src/drivers/e3sm_io_driver_hdf5.cpp: 552
                ▶ /h5bench/e3sm/src/read_decomp.cpp: 253
    ▶ Recommended action:
        ▶ Consider using collective read operations and set one aggregator per compute node
          (e.g. MPI_File_read_all() or MPI_File_read_at_all())
```

Drishti

github.com/hpc-io/drishti