Utilizing Persistent Memory in Parallel I/O Libraries

llogan@hawk.iit.edu, {glofst,sllevy,pwidene}@sandia.gov, {sun,akougkas}@iit.edu

1. Motivation

1. Scientific applications use parallel I/O (PIO) libraries to read/write data
2. PIO libraries have not adequately adapted to the emergence of PMEM as a new tier of storage
3. PIO libraries currently depend on MPI-IO, POSIX, and filesystems for I/O, which cause significant performance loss due to data copying and network communications
4. PIO libraries have complicated APIs that cause significant burden on programmers to store basic data structures

2. Proposed Solution

We design and implement a lightweight I/O library, pMAP:
1. Memory mapping is used to interact with PMEM as opposed to POSIX and MPI-IO in order to avoid data copying
2. A simple key-value store API to store data structures is employed reduce programming burden
3. We evaluate our solution using realistic workloads and compare against various PIO libraries

3. Avoiding Data Copying Costs

We evaluate our solution using realistic workloads and compare against various PIO libraries, and found pMAP has:
- 25% fewer tokens than ADIOS
- 36% fewer tokens than NetCDF
- 90% fewer tokens than HDF5

4. pMAP API

1. #include <pmap/pmap.h>
2. int main(int argc, char** argv) {
3.   int rank, nprocs;
4.   MPI_Init(&argc,&argv);
5.   ... 1, &dimsf);
6.   pmem.store<double>(
7.     "A", data, 1, &off, &count);
8.   MPI_Finalize();
9. } # Lines of Code # Tokens
pMAP 16 132
ADIOS 24 164
NetCDF 26 180
HDF5 42 253

5. pMAP API Example

1. #include <pmap/pmap.h>
2. int main(int argc, char** argv) {
3.   int rank, nprocs;
4.   MPI_Init(&argc,&argv);
5.   ... 1, &dimsf);
6.   pmem.store<double>(
7.     "A", data, 1, &off, &count);
8.   MPI_Finalize();

6. API Comparison

We rebuilt the examples shown in the above API example using other PIO libraries, and found pMAP has:
- 90% fewer tokens than HDF5
- 36% fewer tokens than NetCDF
- 25% fewer tokens than ADIOS

pMAP is more compact and user-friendly than other interfaces.

7. Testbed

Figure 1: Chameleon Cloud

- Skylake
- DRAM 192GB
- Cores 24
- Threads 48
- Ubuntu 20.04
- Kernel 5.4.0-36

8. Workload

- 40GB 3-D domain decomposition problem
- Processes read/write same amount of data
- pMAP A has MAP_SYNC flag enabled
- pMAP B has MAP_SYNC flag disabled

9. PIO Library Write Comparison

Figure 3: Writes

- pMAP-A outperforms ADIOS by 15% after 24 procs
- pMAP-A outperforms NetCDF and pNetCDF by 2.5x
- pMAP-B experiences latency penalty from MAP_SYNC, causing it to be no better than ADIOS

10. PIO Library Read Comparison

Figure 5: Reads

- pMAP-A outperforms ADIOS by 2x regardless of scale
- pMAP-A outperforms NetCDF and pNetCDF by 5x
- pMAP-B experiences latency penalty from MAP_SYNC, causing it to be no better than ADIOS

11. Conclusion

1. Memory mapping can improve PIO read/write performance to PMEM by between 15% - 2x
2. A simple KVS interface for storing data structures can reduce code size by up to 90% compared to other PIO libraries
3. pMAP is more compact and user-friendly than other interfaces.