Utilizing Persistent Memory in Parallel I/O Libraries

Luke Logan Jay Lofstead Scott Levy Patrick Widener Xian-He Sun Anthony Kougkas
llogan@hawk.iit.edu, {glofst,sllevy,pwidene}@sandia.gov, {sun,akougkas}@iit.edu

SCALABLE COMPUTING SOFTWARE LABORATORY

ILLINOIS INSTITUTE OF TECHNOLOGY

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

7. Testbed

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

Figure 1: Chameleon Cloud

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

3. Avoiding Data Copying Costs

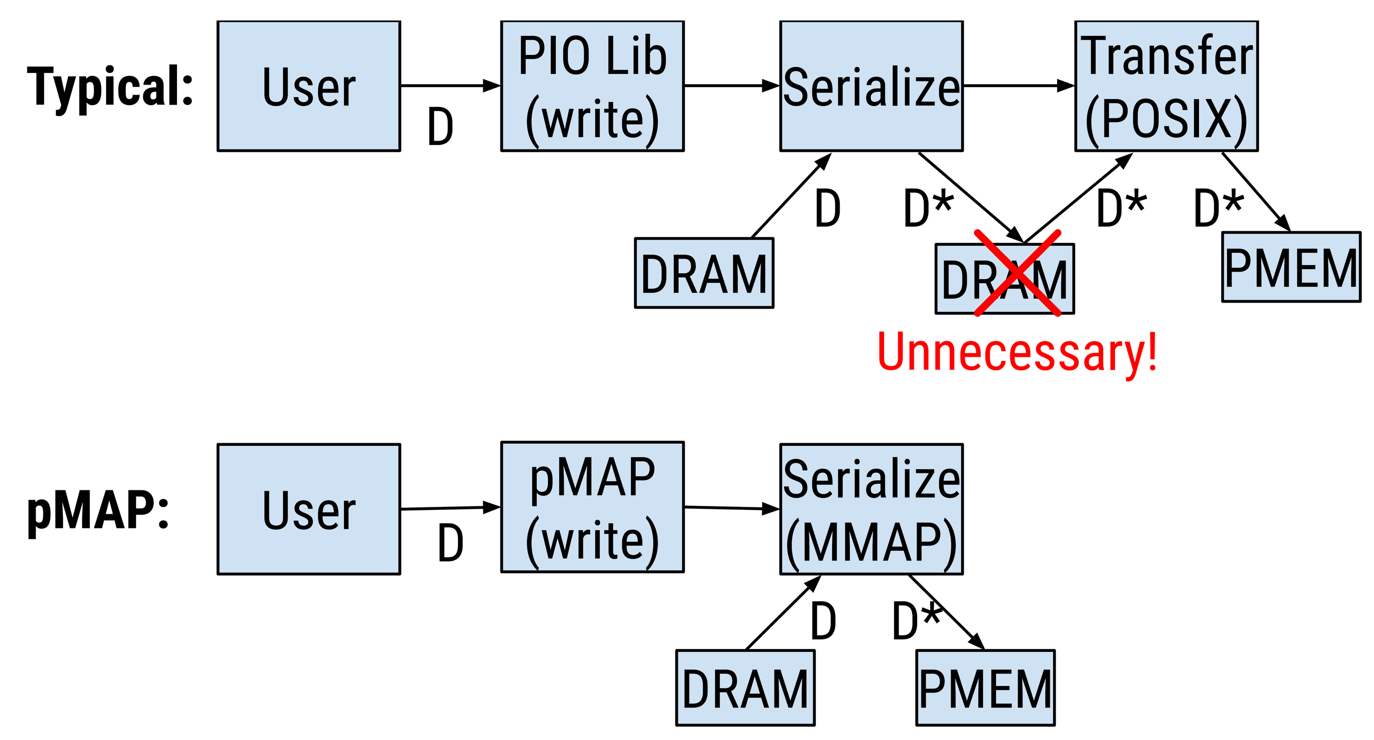

Figure 2: Storing a data structure D in PMEM using a typical PIO library vs pMAP

1. To store data structures, PIO libraries must serialize the entire data structure in-memory and then copy to PMEM
2. pMAP serializes/deserializes data directly from node-local PMEM using memory mapping and well-known serialization libraries, such as CapnProto

4. pMAP API

```
#include <pmap/pmap.hpp>
pmap::PMEM pmem();
pmem.mmap(std::string filename, int comm);
pmem.munmap();

pmem.store<T>(std::string id, T &data,
  pmap::SerializerType s = Default);
pmem.alloc<T>(std::string id,
  int ndims, size_t *dims,
  pmap::SerializerType s = Default);
pmem.store<T>(std::string id, T *data,
  int ndims, size_t *offsets, size_t *dimspp);

pmem.load<T>(std::string id);
pmem.load<T>(std::string id, T &num);
pmem.load<T>(std::string id, T *data,
  int ndims, size_t *offsets, size_t *dimspp);
pmem.load_dims(std::string id,
  int *ndims, size_t *dim);
```

Figure 4: pMAP API: storing data structure and arrays of data structures

5. pMAP API Example

```
#include <pmap/pmap.h>
int main(int argc, char** argv) {
  int rank, nprocs;
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  pmap::PMEM pmem;
  size_t count = 500;
  size_t off = count*rank;
  size_t dimsf = count*nprocs;
  char *path = argv[1];

  double data[100] = {0};
  pmem.mmap(path, MPI_COMM_WORLD);
  pmem.alloc<double>("A", 1, &dimsf);
  pmem.store<double>(
    "A", data, 1, &off, &count);
  MPI_Finalize();
}
```

Figure 6: Writing a 1-D array to PMEM using pMAP; each process writes 500 doubles to the array

9. PIO Library Write Comparison

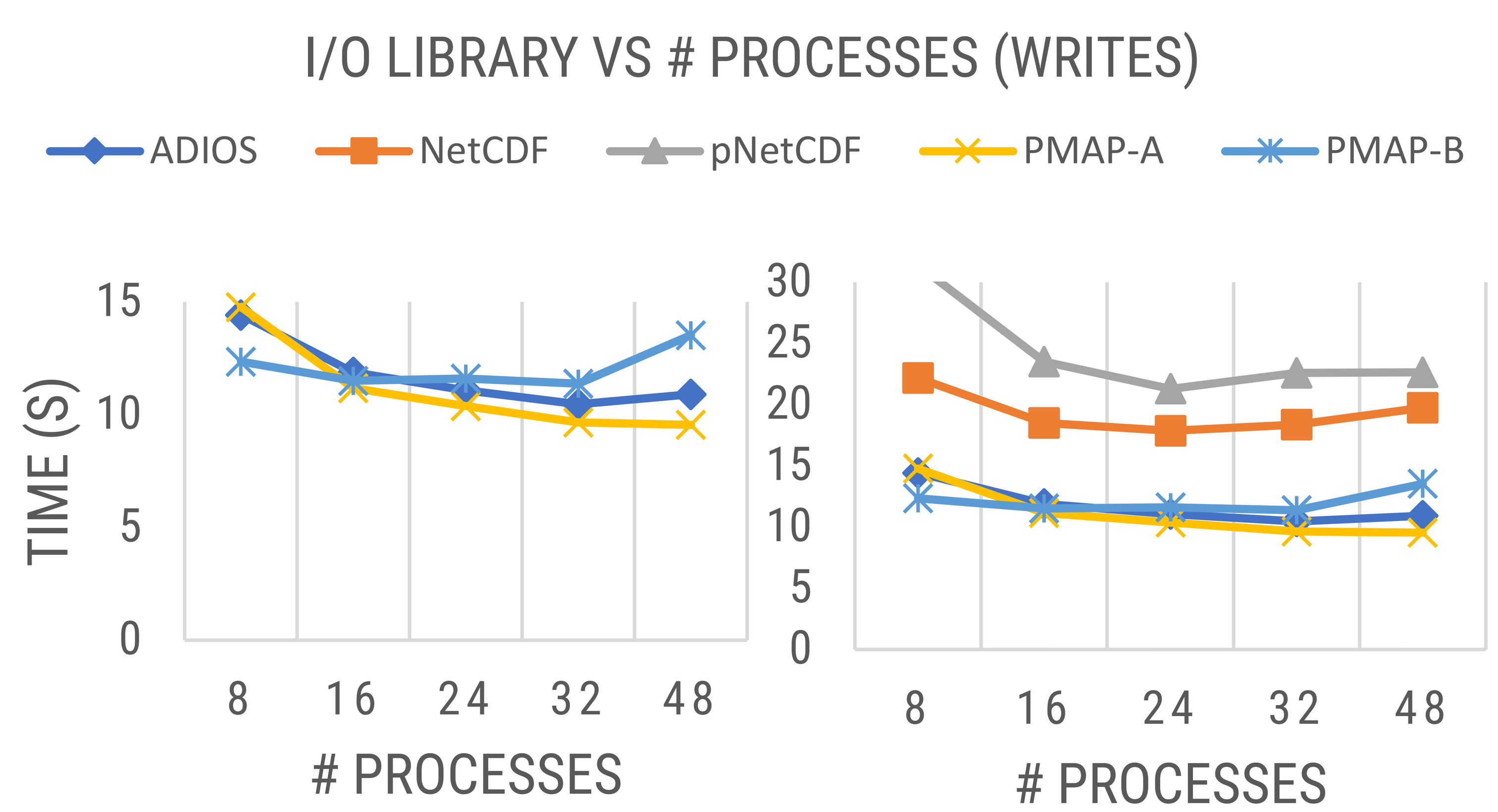

Figure 3: Writes

- 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

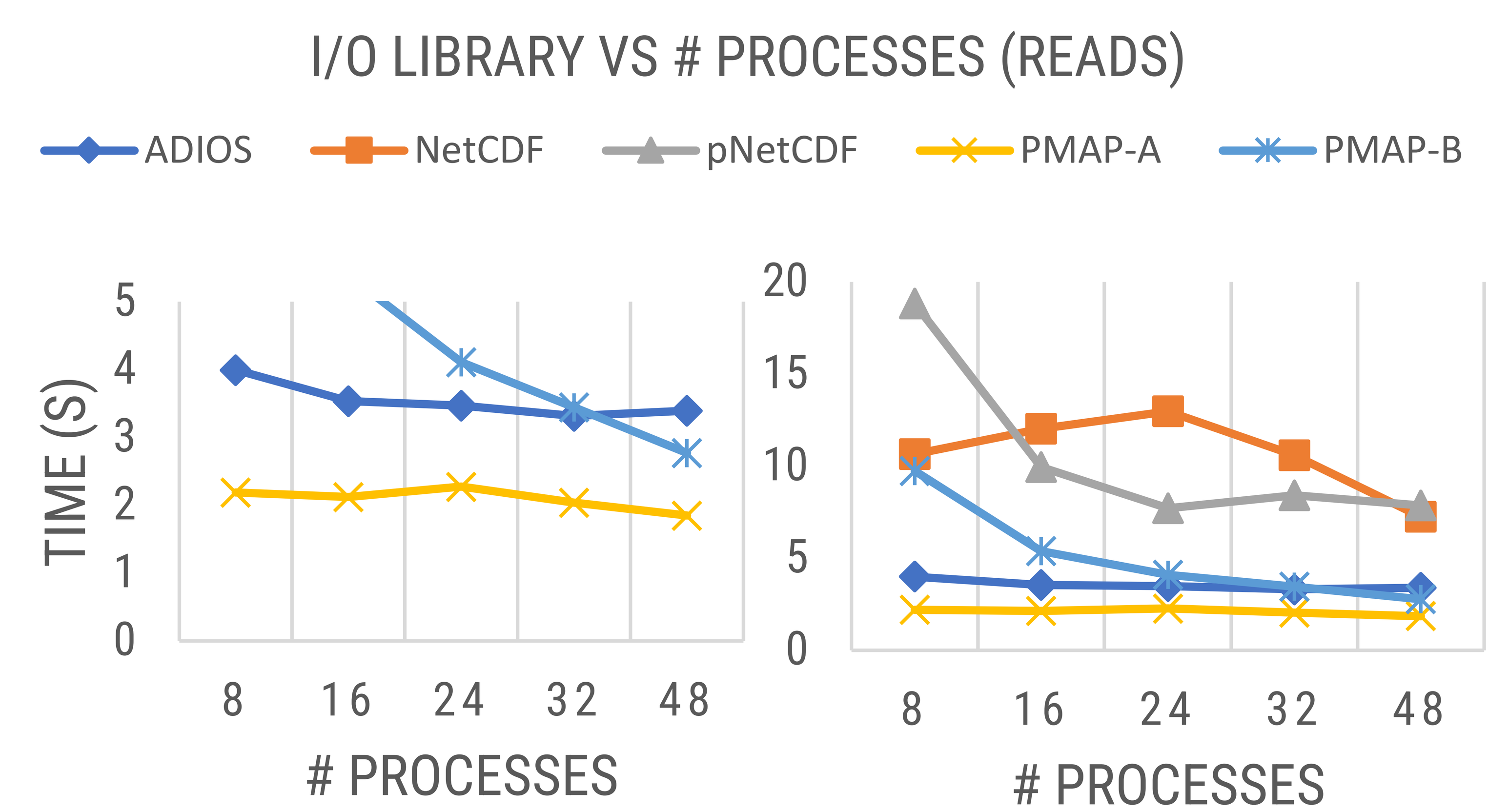

Figure 5: Reads

- 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

6. API Comparison

	Lines of Code	# Tokens
pMAP	16	132
ADIOS	24	164
NetCDF	26	180
HDF5	42	253

Figure 7: API comparison

We rebuild 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.

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