SC21

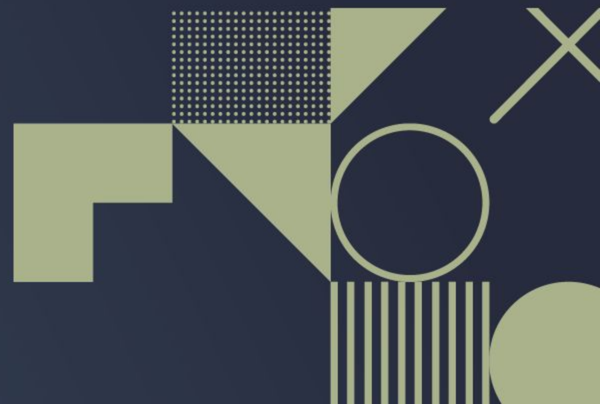
St. Louis, MO | science & beyond.

pMEMCPY: a simple, lightweight, and portable I/O library for storing data in persistent memory

Luke Logan, Jay Lofstead, Scott Levy, Patrick Widener, Xian-He Sun, Anthony Kougkas

Illinois Institute of Technology and Sandia National Labs

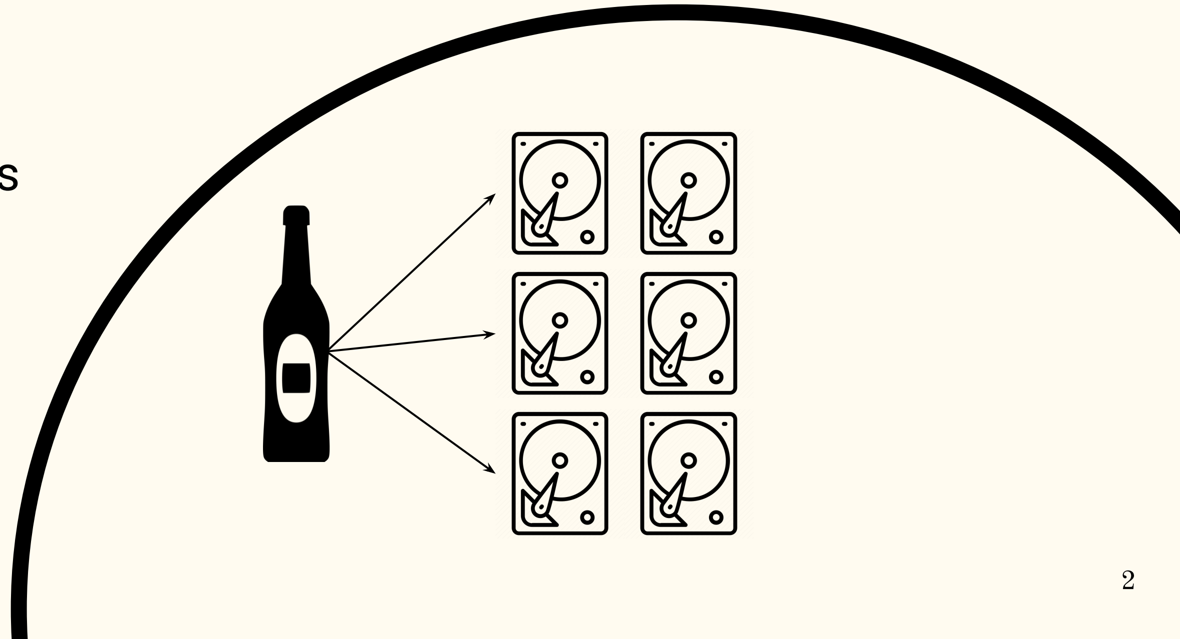
llogan@hawk.iit.edu, {glofst, sllevy, pwidene}@sandia.gov, {sun, akougkas}@iit.edu



I/O Bottleneck

- Parallel I/O (PIO) libraries are used by scientific applications to manage the complexity of loading/storing massive amounts of data

- Storing massive amounts of data leads to the I/O bottleneck problem



Persistent Memory (PMEM) as Storage

- PMEM is typically considered an extension to main memory
- PMEM can also be used as a fast storage tier to temporarily store data

However...

PIO libraries fundamentally rely on per-node POSIX filesystems to store data, which causes unnecessary data copying when using PMEM!

PIO libraries should utilize memory mapping to prevent this.

PIO Libraries can also be complicated!

- Complex configuration spaces
- Many specialized APIs
- Large learning curve

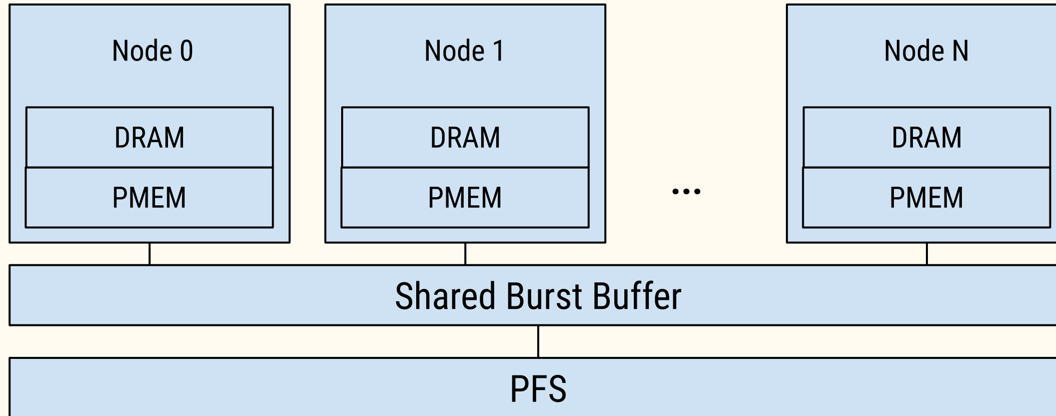
```
1.    #include <hdf5.h>
2.    int main (int argc, char **argv) {
3.     int nprocs, rank;
4.     MPI_Init(&argc, &argv);
5.     MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
6.     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
7.     hid_t file_id, dset_id;
8.     hid_t filespace, memspace;
9.     hsize_t     count = 100;
10.    hsize_t offset = rank*100;
11.    hsize_t dimsf = nprocs*100;
12.    hid_t plist_id;
13.    herr_t      status;
14.    char *path = argv[1];
15.    int data[100];
16.
17.    plist_id = H5Pcreate(H5P_FILE_ACCESS);
18.    H5Pset_fapl_mpio(plist_id,
19.      MPI_COMM_WORLD, MPI_INFO_NULL);
20.    file_id = H5Fcreate(path,
21.      H5F_ACC_TRUNC, H5P_DEFAULT, plist_id);
22.    H5Pclose(plist_id);
23.   ....
```

Our Approach: pMEMCPY

- pMEMCPY utilizes the efficient memory mapped I/O interface provided by PMEM to avoid unnecessary data copies.
- pMEMCPY offers a simplistic key-value store interface, which can significantly reduce code complexity

Design

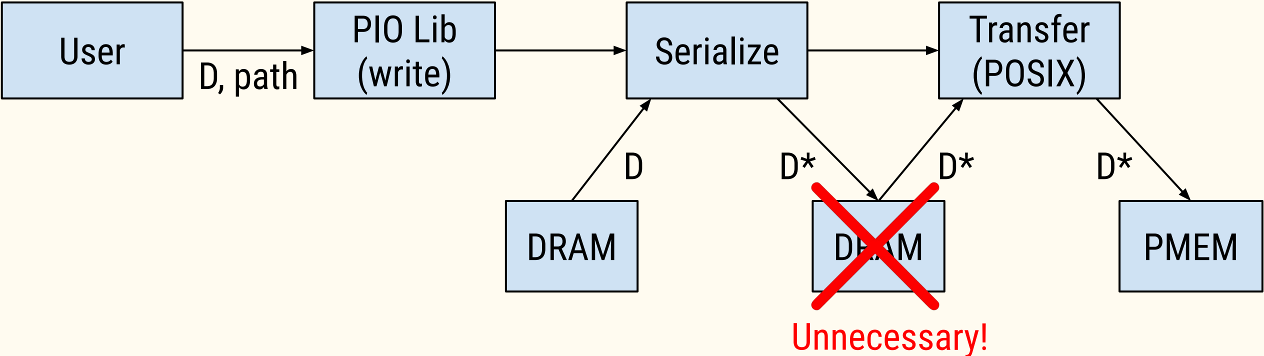
Assumed Machine Architecture

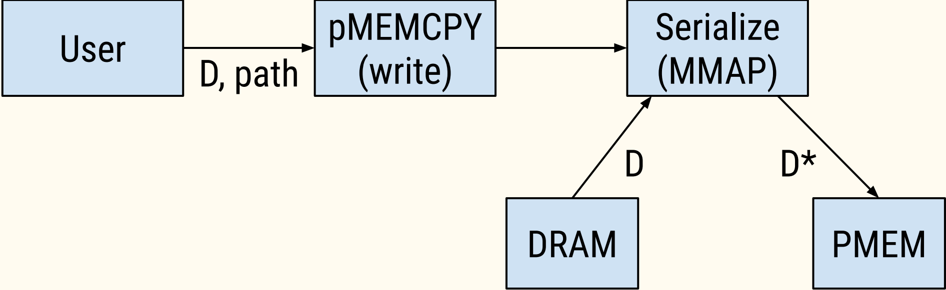


Avoiding Unnecessary Data Copies

D is a data structure

D* is the serialized form of D





API

- Simple key-value store API
- Can store primitive types and arrays of those types
- Templating allows for storing complex data types

```
1.  #include <pmemcpy/pmemcpy.hpp>
2.  pmemcpy::PMEM pmem(pmemcpy::SerializerType default_);
3.  pmem.mmap(std::string filename, int comm);
4.  pmem.munmap();
5.
6.  pmem.store<T>(std::string id, T &data,
7.   pmemcpy::SerializerType s = Default);
8.  pmem.alloc<T>(std::string id,
9.   int ndims, size_t *dims,
10.  pmemcpy::SerializerType s = Default);
11. pmem.store<T>(std::string id, T *data,
12.  int ndims, size_t *offsets, size_t *dimspp);
13.
14. pmem.load<T>(std::string id);
15. pmem.load<T>(std::string id, T &num);
16. pmem.load<T>(std::string id, T *data,
17.  int ndims, size_t *offsets, size_t *dimspp);
18. pmem.load_dims(std::string id,
19.  int *ndims, size_t *dim);
```

pMEMCPY Example

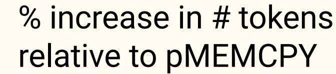
```
1.    #include <pmemcpy/pmemcpy.h>
2.    int main(int argc, char** argv) {
3.       int rank, nprocs;
4.       MPI_Init(&argc,&argv);
5.       MPI_Comm_rank(MPI_COMM_WORLD, &rank);
6.       MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
7.       pmemcpy::PMEM pmem;
8.       size_t count = 100;
9.       size_t off = 100*rank;
10.      size_t dimsf = 100*nprocs;
11.      char *path = argv[1];
12.
13.      double data[100] = {0};
14.      pmem.mmap(path, MPI_COMM_WORLD);
15.      pmem.alloc<double>("A", 1, &dimsf);
16.      pmem.store<double>("A", data, 1, &off,
      &count);
17.      MPI_Finalize();
18.   }
```

pMEMCPY for writing a 1-D array of 100 doubles per-process

HDF5 Comparison

```
1.     #include <hdf5.h>
2.     int main (int argc, char **argv) {
3.      int nprocs, rank;
4.      MPI_Init(&argc, &argv);
5.      MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
6.      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
7.      hid_t file_id, dset_id;
8.      hid_t filespace, memspace;
9.      hsize_t      count = 100;
10.     hsize_t offset = rank*100;
11.     hsize_t dimsf = nprocs*100;
12.     hid_t plist_id;
13.     herr_t       status;
14.     char *path = argv[1];
15.     int data[100];
16.
17.     plist_id = H5Pcreate(H5P_FILE_ACCESS);
18.     H5Pset_fapl_mpio(plist_id,
19.       MPI_COMM_WORLD, MPI_INFO_NULL);
20.     file_id = H5Fcreate(path,
21.       H5F_ACC_TRUNC, H5P_DEFAULT, plist_id);
22.     H5Pclose(plist_id);
23.     filespace = H5Screate_simple(1, &dimsf, NULL);
24.     dset_id = H5Dcreate(file_id, "dataset",
25.       H5T_NATIVE_INT, filespace, H5P_DEFAULT,
26.       H5P_DEFAULT, H5P_DEFAULT);
27.     H5Sclose(filespace);
28.     memspace = H5Screate_simple(1, &count, NULL);
29.     filespace = H5Dget_space(dset_id);
30.     H5Sselect_hyperslab(filespace,
31.       H5S_SELECT_SET, &offset,
32.       NULL, &count, NULL);
33.
34.     plist_id = H5Pcreate(H5P_DATASET_XFER);
35.     status = H5Dwrite(dset_id, H5T_NATIVE_INT,
36.       memspace, filespace, plist_id, data);
37.
38.     H5Dclose(dset_id);
39.     H5Sclose(filespace);
40.     H5Sclose(memspace);
41.     H5Pclose(plist_id);
42.     H5Fclose(file_id);
43.     MPI_Finalize();
44.     return 0;
45.    }
```

API Comparison



	LOC	# Tokens	% Larger
pMEMCPY	16	132	0%
ADIOS	24	164	24%
NetCDF	26	180	36%
HDF5	42	253	92%

pMEMCPY requires the least user effort to store basic data structures.

Evaluation

Evaluations

S3D Combustion Emulation

Writes a 40GB 3-D rectangle to PMEM

The 40GB is divided equally among each process

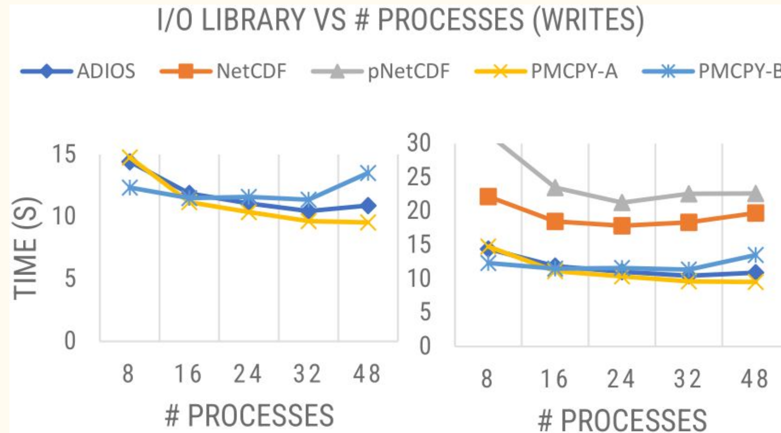
S3D Combustion Analysis Emulation

Reads the 40GB 3-D rectangle from PMEM

Each process reads the same region that was originally written

- We ran experiments on emulated PMEM in Chameleon Cloud

S3D Combustion Emulation



* pMCPY-A has MAP_SYNC disabled

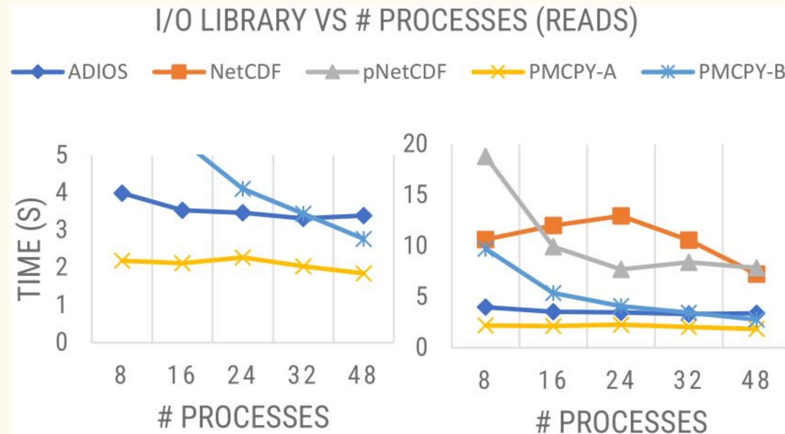
* pMCPY-B has MAP_SYNC enabled

pMCPY-A is **15%** faster than ADIOS

pMCPY-A is **2.5x** faster than NetCDF and pNetCDF

pMCPY-B is no faster than ADIOS

S3D Combustion Analysis Emulation



* pMCPY-A has MAP_SYNC disabled

* pMCPY-B has MAP_SYNC enabled

pMCPY-A is **2x** faster than ADIOS

pMCPY-A is **5x** faster than NetCDF and pNetCDF

pMCPY-B is no faster than ADIOS

Conclusion

Conclusion

- We found that pMEMCPY can reduce code size by as much as **92%** over other PIO libraries by providing a simple key-value store interface
- We found that pMEMCPY can improve writes by **15%** and reads by **2x** over other PIO libraries through memory mapping

Questions?