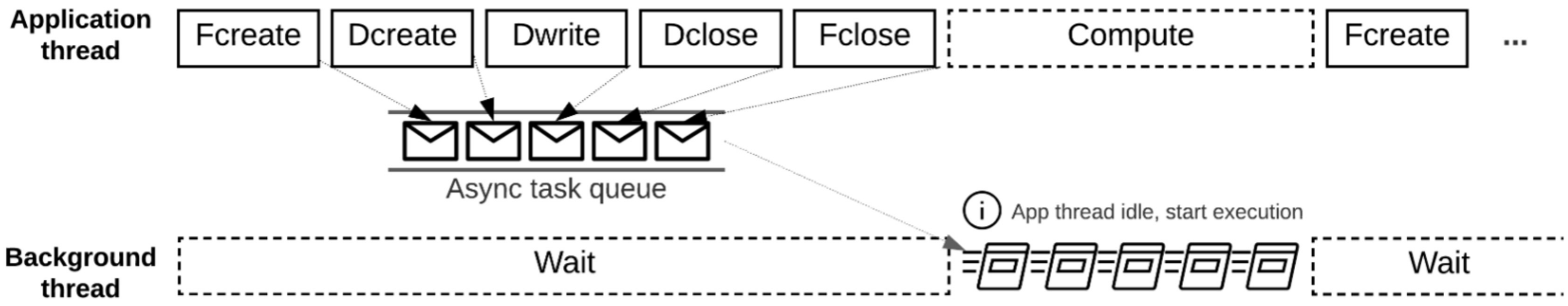
Accelerating HPC Applications with Asynchronous I/O

Houjun Tang

Lawrence Berkeley National Laboratory, USA

HDF5 Async VOL Connector

- HDF5 1.13+ with the new HDF5 asynchronous I/O APIs.
- Transparent background thread execution overlaps I/O with compute time.



Explicit Control with Async and EventSet APIs

- Async version of HDF5 APIs
  - `H5Fcreate_async(fname, …, es_id);`
  - `H5Dwrite_async(dset, …, es_id);`
  - …

- Track and inspect multiple I/O operations with an ***EventSet ID***
  - `H5EScreate();`
  - `H5ESwait(es_id, timeout, &remaining, &op_failed);`
  - `H5ESget_err_info(es_id, ...);`
  - `H5ESclose(es_id);`

Example Code from AMReX

```
#ifdef AMREX_USE_HDF5_ASYNC
        hid_t dataset = H5Dcreate_async(grp, dataname.c_str(), H5T_NATIVE_DOUBLE, dataspace, H5P_DEFAULT, dcpl_id, H5P_DEFAULT, es_id_g);
#else
        hid_t dataset = H5Dcreate(grp, dataname.c_str(), H5T_NATIVE_DOUBLE, dataspace, H5P_DEFAULT, dcpl_id, H5P_DEFAULT);
#endif
        if(dataset < 0)
            std::cout << ParallelDescriptor::MyProc() << "create data failed!  ret = " << dataset << std::endl;

#ifdef AMREX_USE_HDF5_ASYNC
        ret = H5Dwrite_async(dataset, H5T_NATIVE_DOUBLE, memdataspace, dataspace, dxpl_col, a_buffer.dataPtr(), es_id_g);
#else
        ret = H5Dwrite(dataset, H5T_NATIVE_DOUBLE, memdataspace, dataspace, dxpl_col, a_buffer.dataPtr());
#endif
        if(ret < 0) { std::cout << ParallelDescriptor::MyProc() << "Write data failed!  ret = " << ret << std::endl; break; }
```

https://github.com/AMReX-Codes/amrex/blob/development/Src/Extern/HDF5/AMReX_PlotFileUtilHDF5.cpp#L721

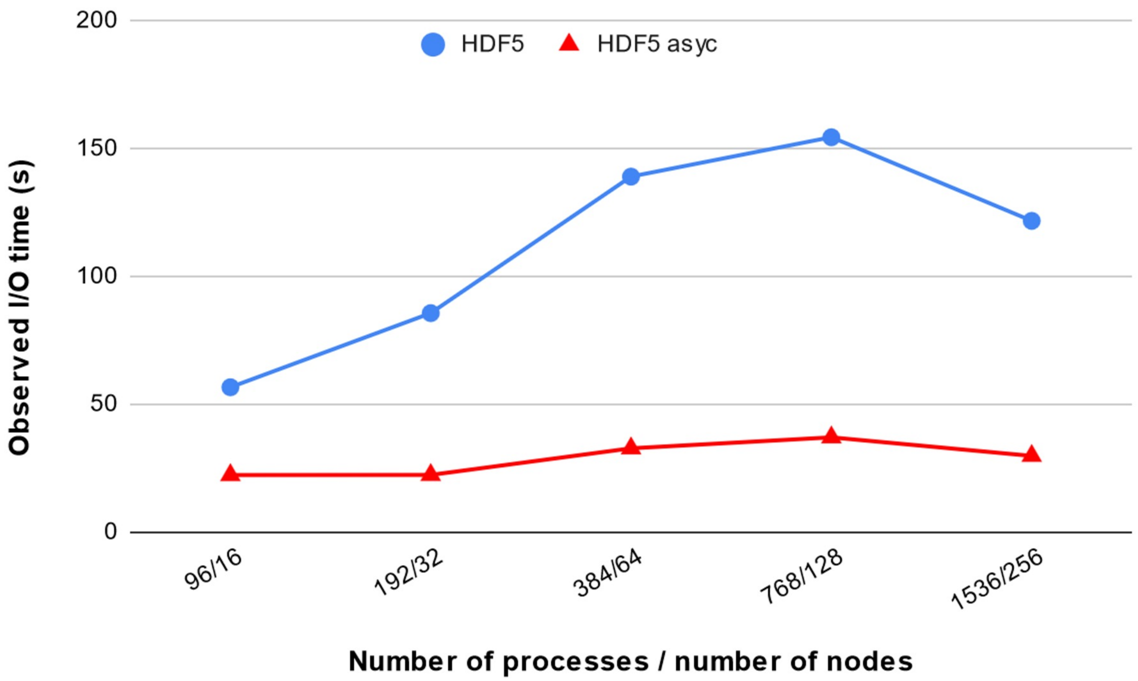
How to use Async VOL

Detailed description in https://hdf5-vol-async.readthedocs.io

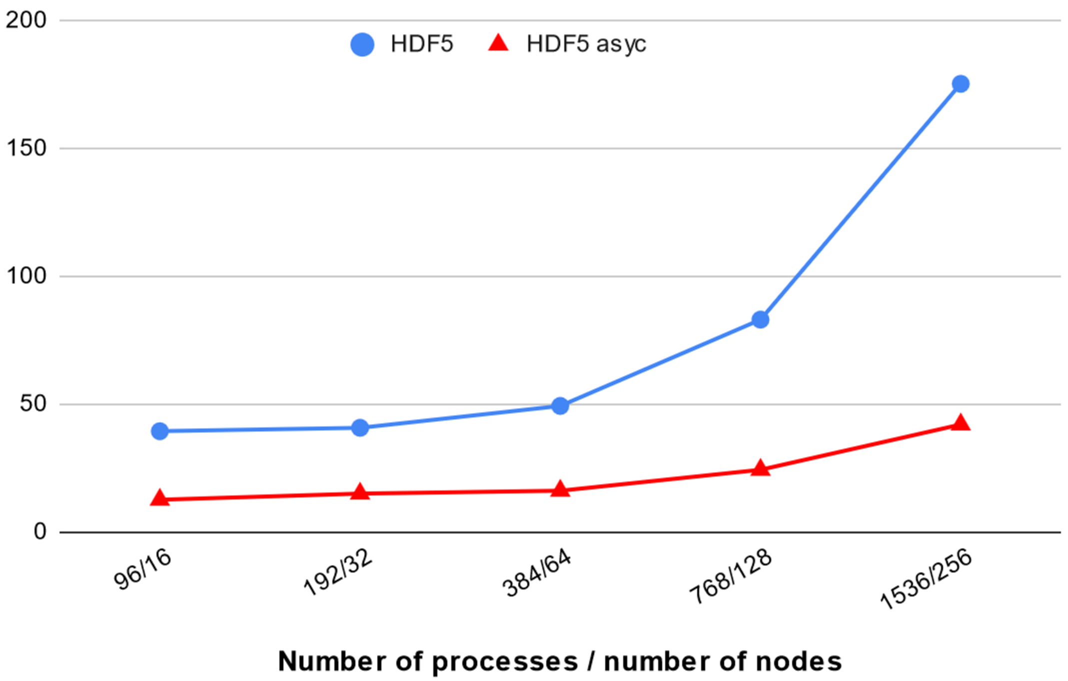
spack install hdf5-vol-async

- **Installation**
  - Compile HDF5 (github develop branch or released version 1.13+), with **thread-safety** support
  - Compile Argobots threading library
  - Compile Async VOL connector
    - "-DENABLE_WRITE_MEMCPY" flag to have async vol copy write buffer
- **Set environment variables**
  - `export `**`LD_LIBRARY_PATH`**`=$VOL_DIR/lib:$H5_DIR/lib:$ABT_DIR/lib:$LD_LIBRARY_PATH`
  - `export `**`HDF5_PLUGIN_PATH`**`="$VOL_DIR/lib"`
  - `export `**`HDF5_VOL_CONNECTOR`**`="async under_vol=0;under_info={}"`
  - `(optional) export HDF5_ASYNC_EXE_FCLOSE=1`
  - `(optional) export HDF5_ASYNC_MAX_MEM_MB=67108864`
- **Run the application (using the async and EventSet APIs)**
  - MPI must be initialized with **`MPI_THREAD_MULTIPLE`**

Speedup with AMReX Applications on Summit



NyX workload, single refinement level,
writes 385GB x 5 steps, emulated compute time.



Castro workload, 3 refinement levels,
writes 559GB x 5 steps, emulated compute time.

BERKELEY LAB
The HDF Group
Argonne NATIONAL LABORATORY

EXASCALE COMPUTING PROJECT

Best Practice & Lessons Learned

- Async is effective when I/O time is a significant portion of the total application execution time, and there is enough compute time to overlap with.
- Some operations cannot be done asynchronously, avoid if possible.
    - E.g. `H5Dget_space` need to perform sync I/O, use async debug log for identification.
- `MPI_THREAD_MULTIPLE` has overhead.
- Background thread interference.
    - Minimal interference for GPU-accelerated applications.
    - OpenMP applications should leave 1 core/thread for the async background thread.
- Memory allocation needs to be handled properly.
    - Peak memory usage could be higher than sync mode, due to double buffering.
    - Will switch to sync mode when not enough system memory is available.

Thank you!

Questions?

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.