The PIO Library for Scalable HPC Performance

Ed Hartnett

10/15/20

Abstract

The PIO C and Fortran libraries enable high-performance, scalable I/O on HPC systems with many processors. Doing I/O from many processors at the same time causes system contention and inefficiencies. Instead, users may select a small number of processors to be responsible for all I/O. Code on the computational processors calls netCDF I/O functions as usual, but instead of writing directly to disk, the data are sent with MPI to the I/O processors, who execute the disk I/O. The PIO libraries are available in C and Fortran, and work with Unidata's netCDF package, the parallel-netcdf package from Argonne Labs, and the HDF5 library from the HDF Group. The PIO libraries are maintained and distributed by NCAR and NOAA, and are free and open software. Recent improvements in PIO include full netCDF integration, allowing users to use existing netCDF code bases with little modification.

PIO Summary

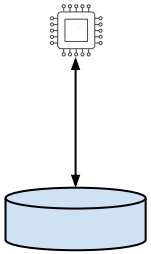
- C/Fortran libraries for HPC systems to provide scalable netCDF I/O on thousands or tens of thousands of processors.
- Uses netCDF classic (sequential), netCDF/HDF5 (parallel or sequential), pnetcdf (parallel), under the covers.
- Supports two modes:
  - Intracomm - one computational unit with shared I/O processors.
  - Async - many computational units with dedicated I/O processors.
- Supprts two APIs:
  - Classic - C/Fortran PIOc_* and PIO_ functions.
  - NetCDF Integration - C/Fortran netCDF API calls with PIO additions.
- Used in CESM, ESMF.

I/O on Small Processor Counts is Easy

- One processor can use sequential access to netCDF/HDF5 files. Easy!
- Tens of processors can use parallel access to netCDF/HDF5 files. Not as easy, but simple enough.

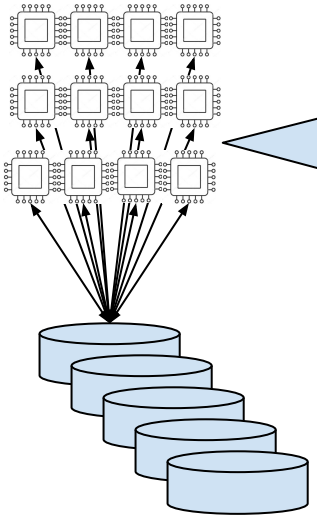
I/O on One or Few Processors

One Processor



Sequential I/O: One processor writes to disk. The good old days!

Few Processors

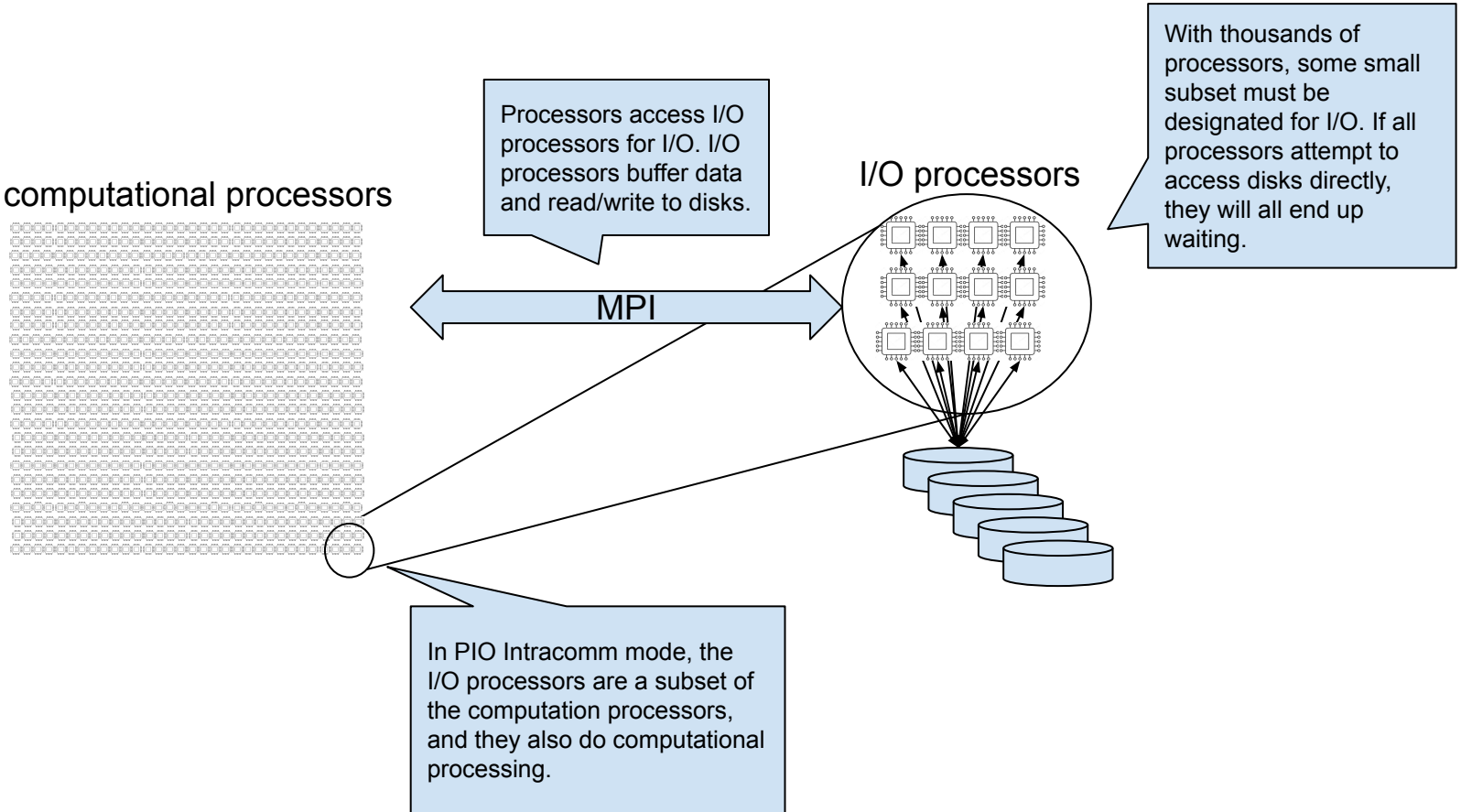


Parallel I/O: Multiple processors each read/write to parallel disk system. Higher bandwidth is available than with sequential I/O. Does not scale well past 10s or 100s of processors.

I/O on Large Processor Counts is Harder

- Now we need to run on tens of thousands of processors.
- Parallel I/O does not scale - once the (relatively few) I/O channels to disk hardware are filled, processors wait.
- A solution is to designate a subset of processors to handle all I/O, and buffer I/O operations.
- This may be done with the PIO library.

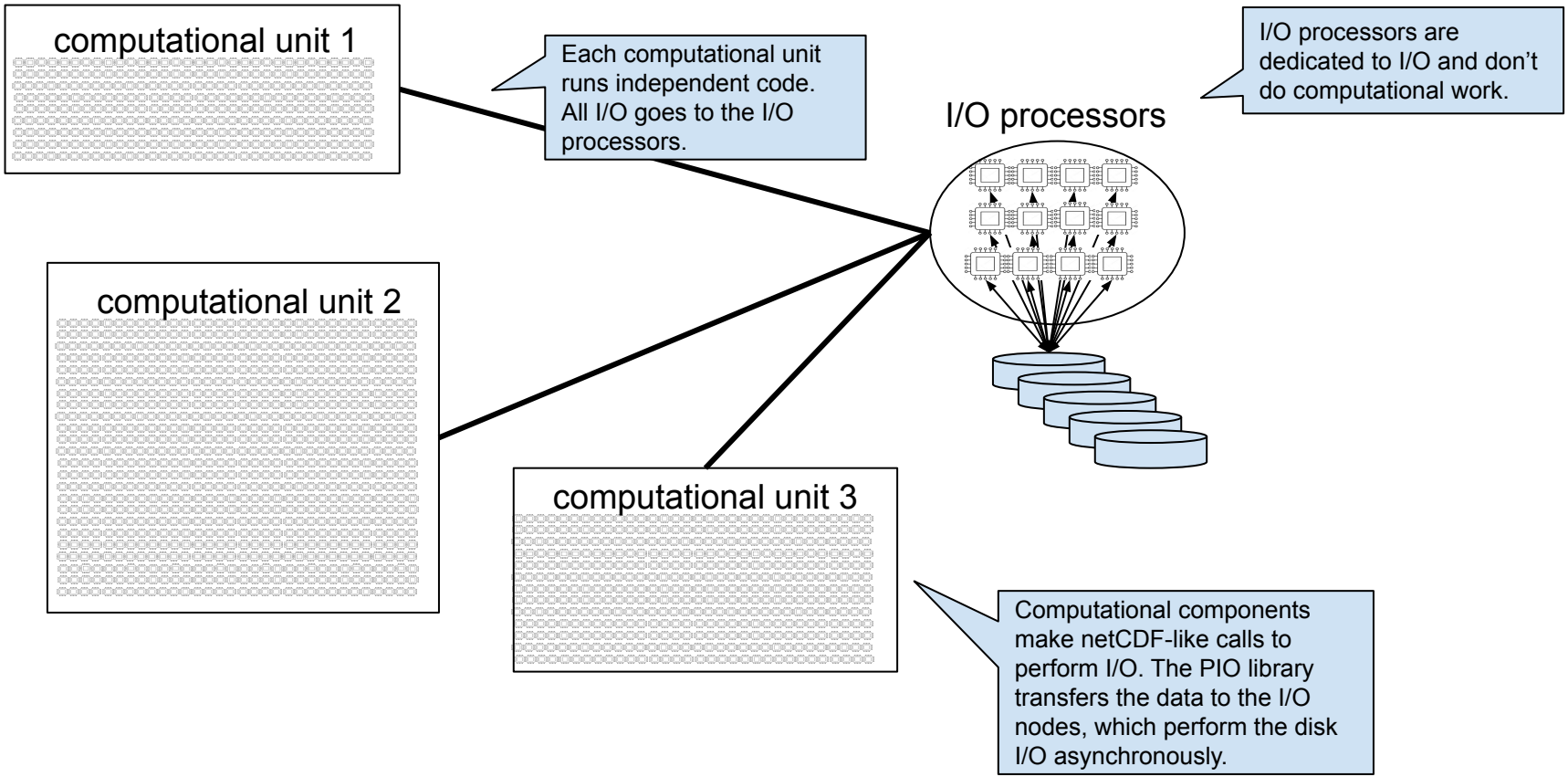
I/O on Many Processors
(PIO Intracomm Mode)



Multi-Level Parallelism

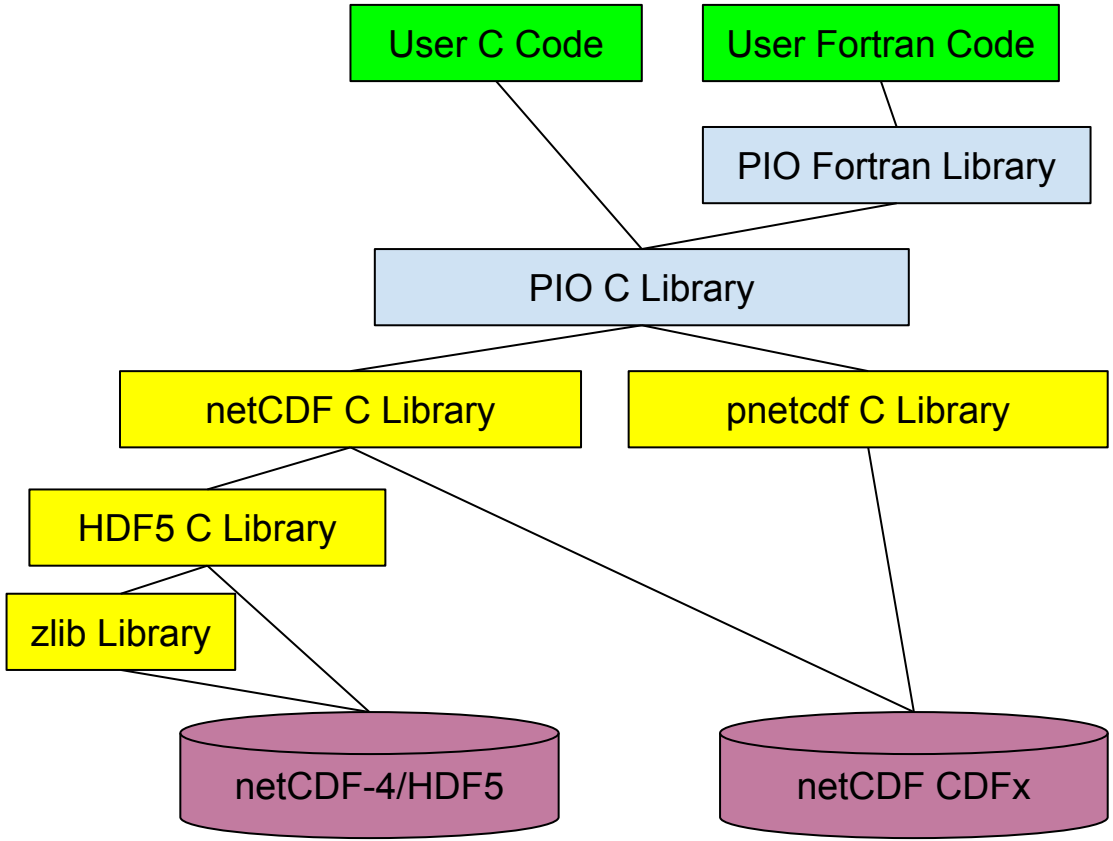
- A further refinement is to have multiple computational components, all using the same dedicated I/O component to do I/O.
- Now computation can proceed while I/O is taking place.

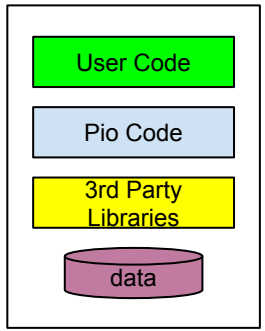
I/O on Many Processors
(PIO Async Mode)



Ed Hartnett 6/1/19

PIO Library Architecture





Ed Hartnett, 5/1/19

PIO Classic (without NetCDF Integration)

```
ret = PIO_openfile(pio_tf_iosystem_, pio_file, tgv_iotype, tgv_fname, PIO_write)

ret = PIO_redef(pio_file)

ret = PIO_def_dim(pio_file, 'dummy_dim_def_var', 100, pio_dim)

ret = PIO_def_var(pio_file, 'dummy_var_def_var', PIO_int, (/pio_dim/), pio_var)

 ret = PIO_enddef(pio_file)

...
```

PIO with NetCDF Integration

```
 if ((ret = nc_create(filename, NC_CLOBBER|NC_PIO, &ncid)))
    return ret;
if ((ret = nc_def_dim(ncid, DIM_NAME_S1, DIM_LEN_S1, &dimid)))
    return ret;
if ((ret = nc_def_var(ncid, VAR_NAME_S1, NC_INT, NDIM_S1, &dimid, &varid)))
    return ret;
if ((ret = nc_enddef(ncid)))
    return ret;
```

Computational Code uses NetCDF API for PIO

- IO System must be initialized with a function call nc_init_intracomm()/nc_init_async().
- Files are opened/created with NC_PIO flag.
- The computational components make netCDF calls.
- The PIO library handles the transferring of data to/from the I/O processors, which do the actual disk I/O.
- Distributed data read/writes are handled with new functions nc_get_vard_*()/nc_put_vard_*().

Global vs. Local Arrays

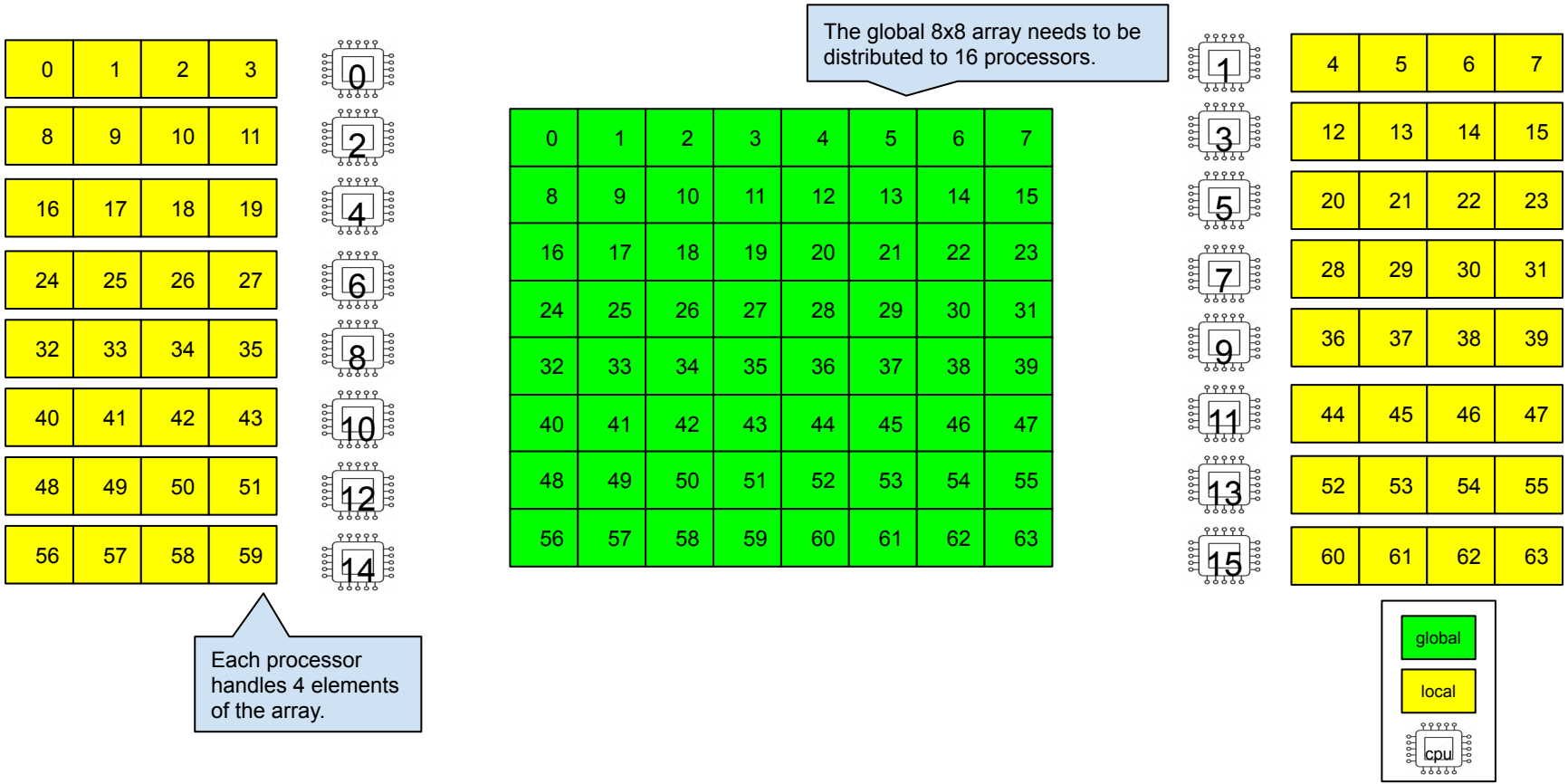
- The shape of a netCDF record defines the global data space.
- Once divided on to many processors, each processor has a subset of the global data space - the local array.
- Together, all local arrays add up to the global array.
- There may be halos - data that are needed for computation but are outside the area that the processor should be writing.

PIO Distributed Arrays

- Each processor within a computational unit has its region of responsibility within the global variable data space.
- PIO allows users to specify this decomposition.
- Different read and write decompositions may be used to support halos.

PIO Decomposition

Decomposing an 8x8 Array over 16 Processors



Reading/Writing Distributed Data

```
/* Calculate a decomposition for distributed arrays. */
elements_per_pe = DIM_LEN_X * DIM_LEN_Y / (ntasks - num_io_procs);
for (i = 0; i < elements_per_pe; i++)
    compdof[i] = (my_rank - num_io_procs) * elements_per_pe + i;

/* Create the PIO decomposition for this test. */
if (nc_def_decomp(iosysid, PIO_INT, NDIM2, &dimlen[1], elements_per_pe,
                  compdof, &ioid, 1, NULL, NULL)) PERR;

/* Create some data on this processor. */
if (!(my_data = malloc(elements_per_pe * sizeof(int)))) PERR;
for (i = 0; i < elements_per_pe; i++)
    my_data[i] = my_rank * 10 + i;

/* Write some data with distributed arrays. */
if (nc_put_vard_int(ncid, varid, ioid, 0, my_data)) PERR;
```

Decompositions Stored in Files

- Once a decomposition has been created, it can be written to file, and read in again to initialize a decomposition object.
- Decomposition files can be text (legacy) or netCDF (new).

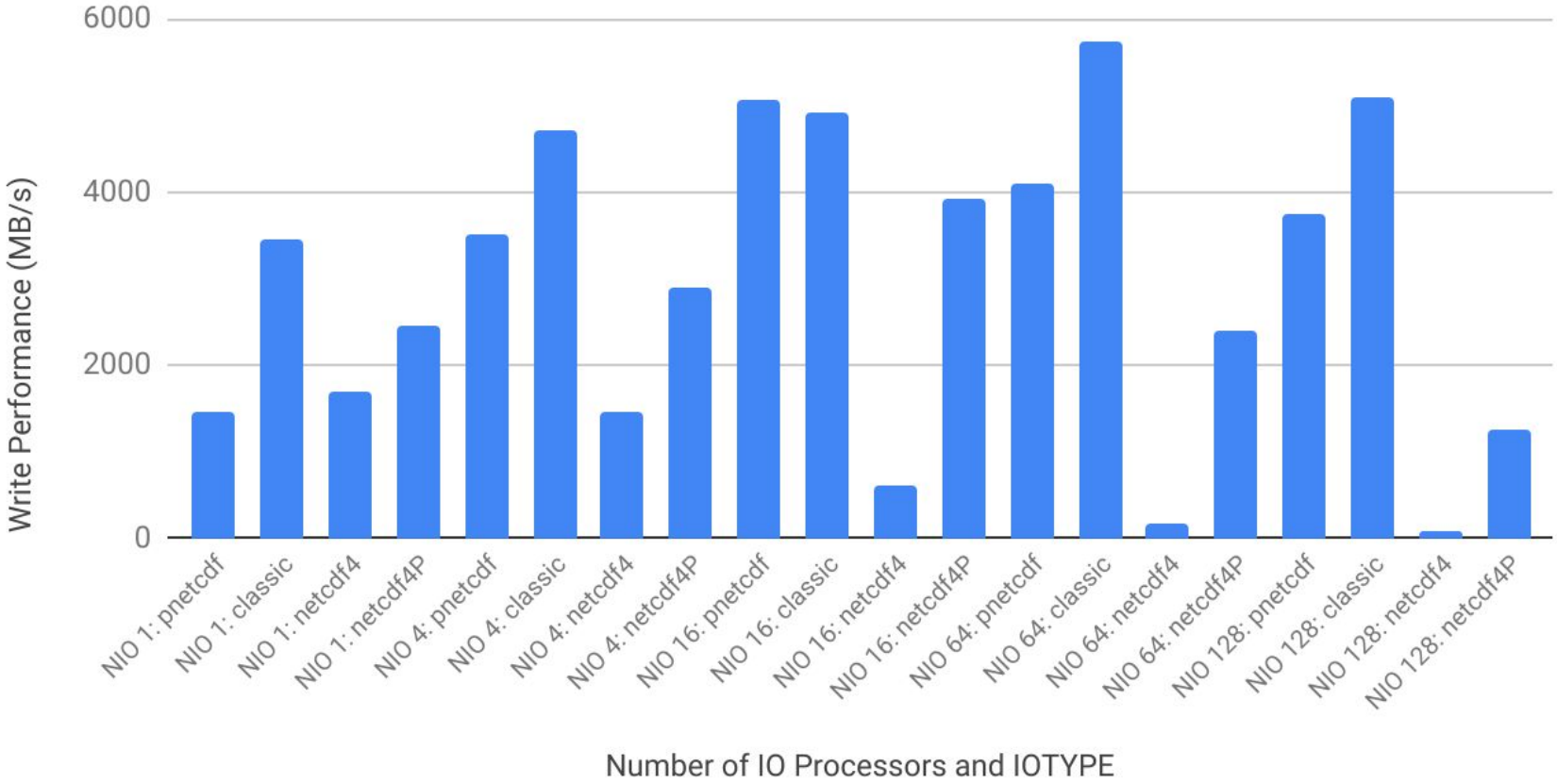
```
netcdf darray_no_async_decomp {
dimensions:
      dims = 2 ;
      task = 16 ;
      map_element = 4 ;
variables:
      int global_size(dims) ;
      int maplen(task) ;
      int map(task, map_element) ;

// global attributes:
            :PIO_library_version = "2.4.2" ;
            :max_maplen = 4 ;
            :title = "Example Decomposition from
darray_no_async.c" ;
            :history = "This file is created by the
program darray_no_async in the PIO C library" ;
            :source = "Decomposition file produced by
PIO library." ;
            :array_order = "C" ;
            :backtrace = “...”
```

Where Is PIO Used?

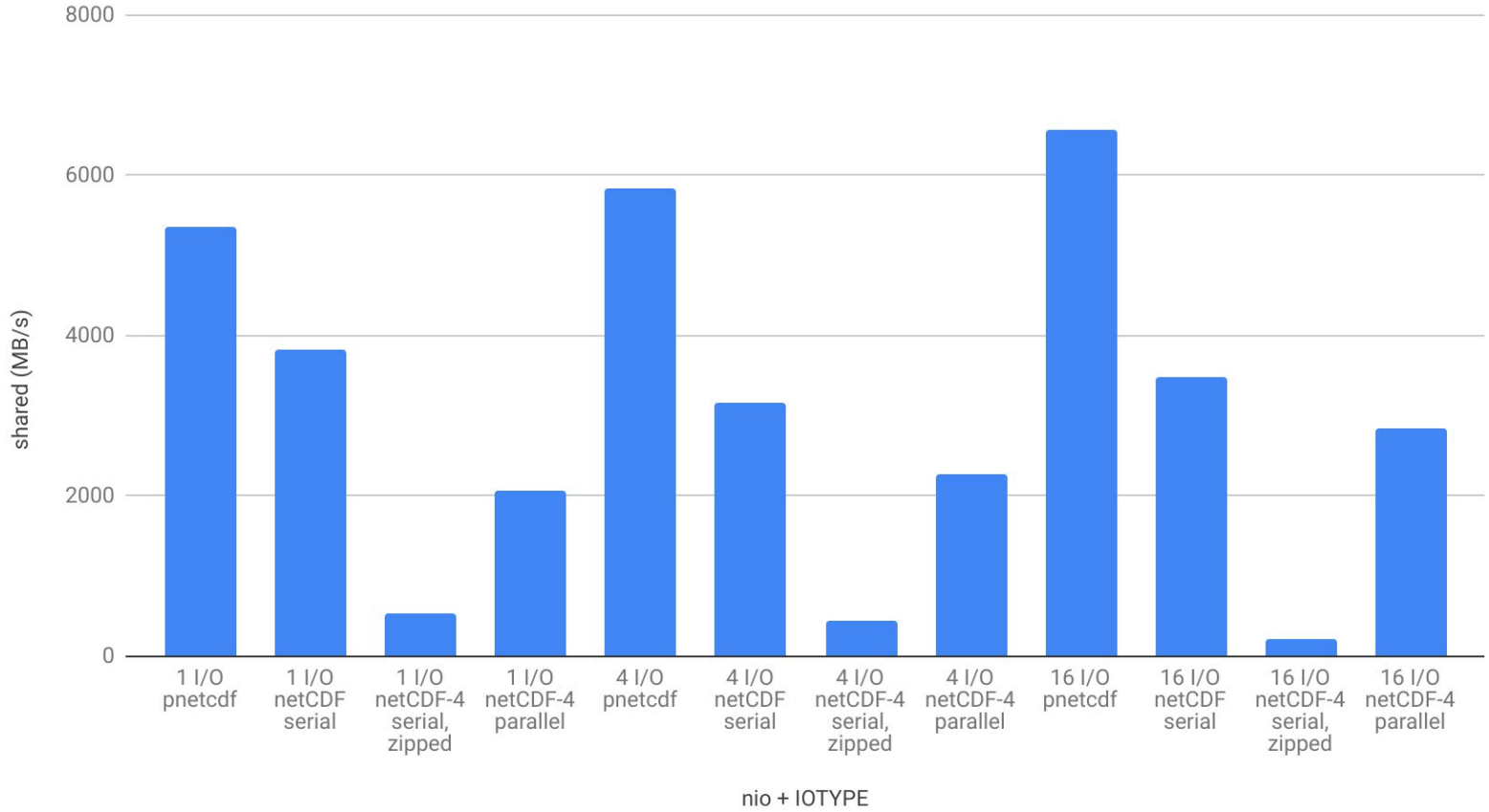
- PIO has been in use in CESM (Community Earth System Model) since around 2008.
  - Standard spatial resolution is 1 deg atmosphere and 1 degree ocn. As a climate model we don't normally write per timestep, very high temporal resolution would be hourly. High is daily and typical is monthly.
  - High spatial resolution is 1/4 degree atmosphere and 1/10 degree ocn.
- The cmip6 experiments which are currently underway have produced some 2 PB of data so far - all written using the pio library.

Write Performance Async Mode

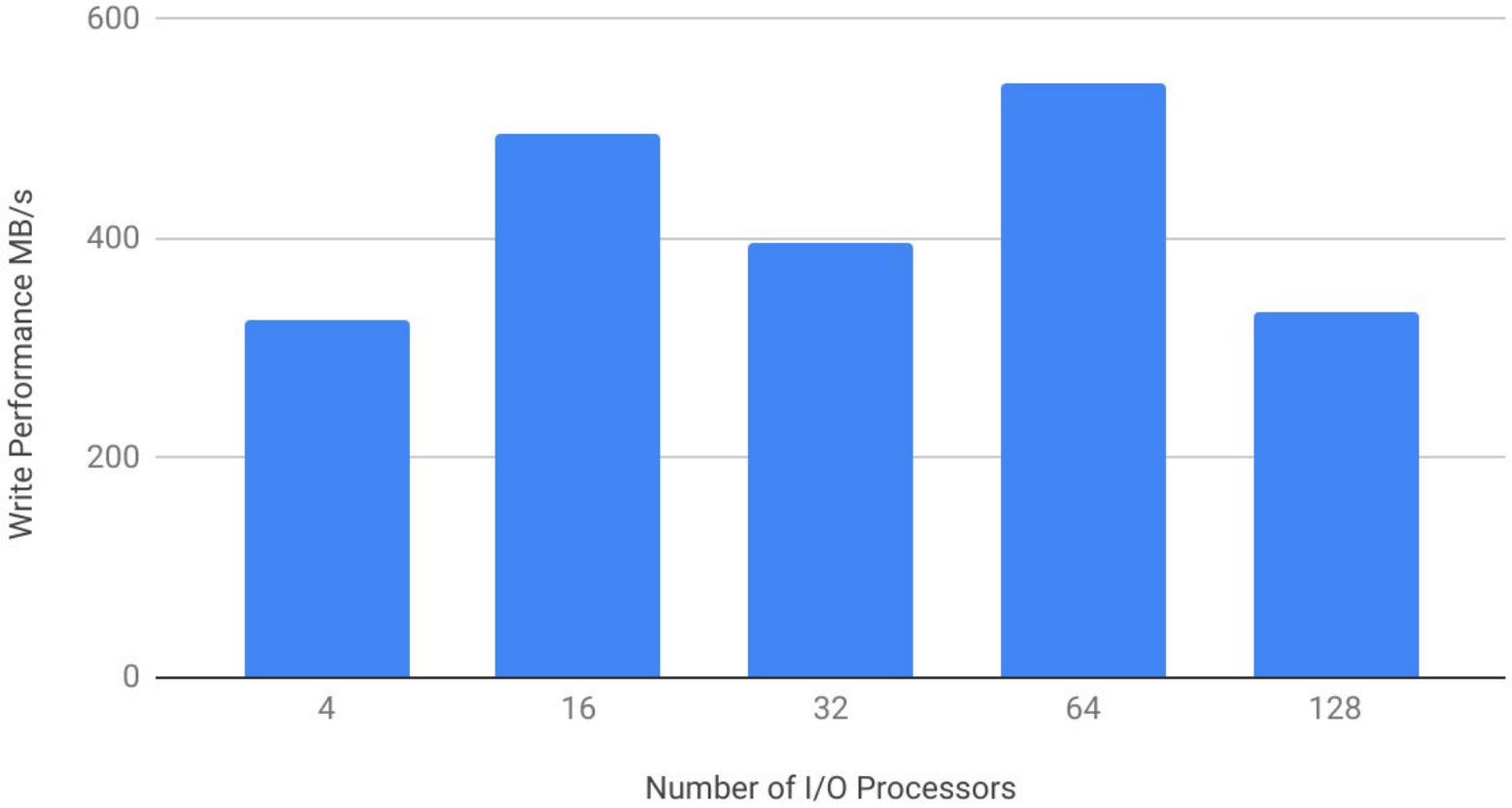
4096 processors on Jet



PIO Performance Jet



PIO Write Performance, Jet, 1024 Processors, using pnetcdf



Future Plans

- Enable new compression options for netCDF-4 files.
- More performance testing with new versions of I/O libraries.
- Further testing for NOAA FV3 I/O.

PIO Conclusion

- Get release 2.5.2 at https://github.com/NCAR/ParallelIO. (Autotools and CMake builds available.)
- Documentation: https://ncar.github.io/ParallelIO/
- Look for PIO poster at the next AMS meeting.