PIO and NetCDF

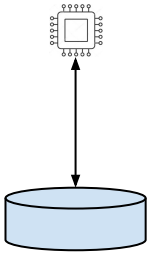
Ed
Hartnett
11/17/19

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.

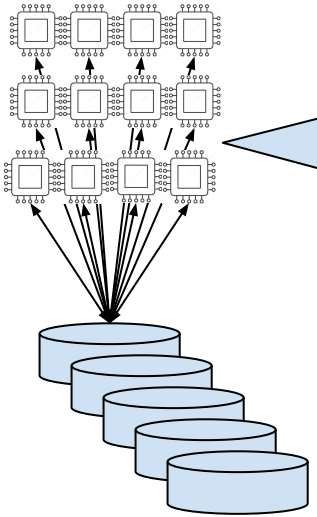
I/O on One or Few Processors

One Processor



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

Few Processors

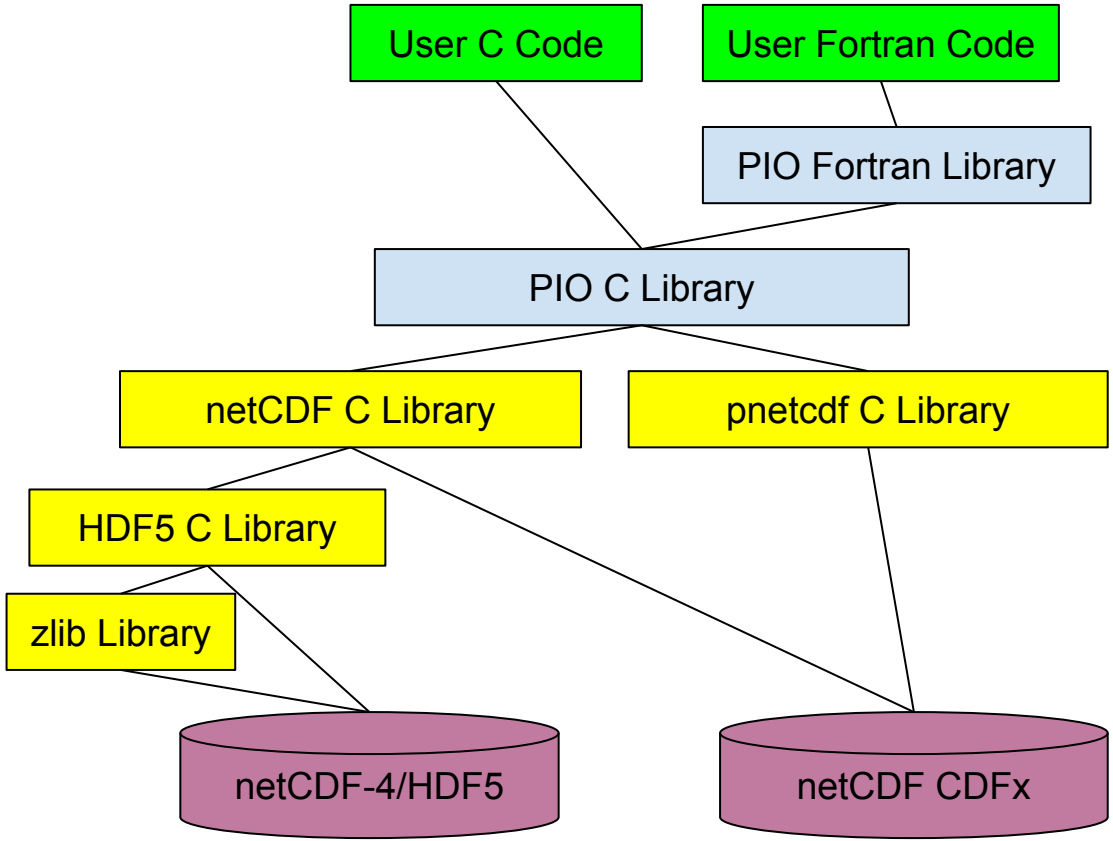


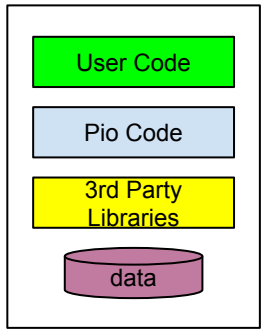
Parallel I/O: Multiple processors each read/write to parallel disk system. Higher bandwidth is available than with sequential I/O. Does not scale well past 10s or 100s of processors.

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.

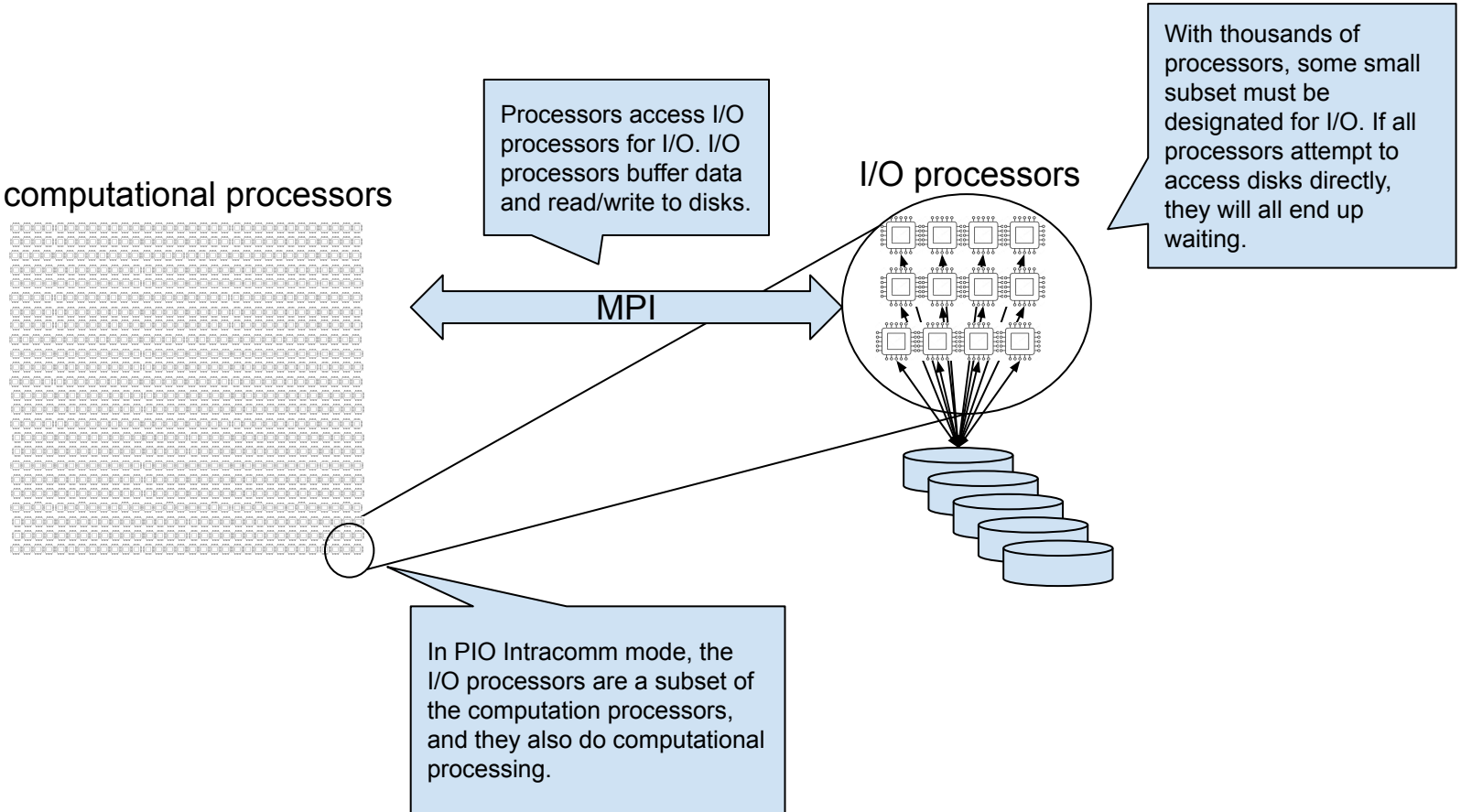
PIO Library Architecture





Ed Hartnett, 5/1/19

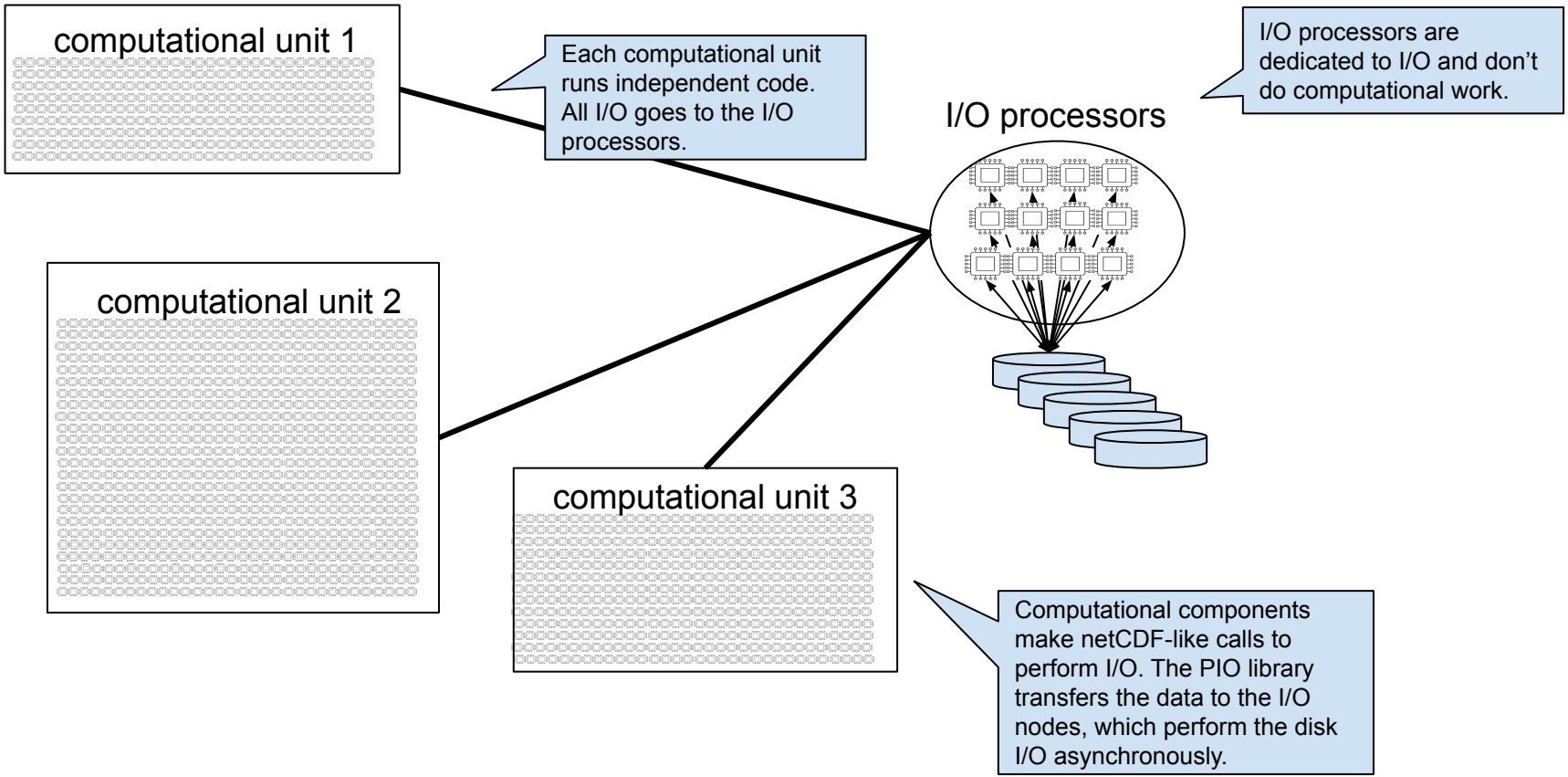
I/O on Many Processors
(PIO Intracomm Mode)



Multi-Level Parallelism

- A further refinement is to have multiple computational components, all using the same dedicated I/O component to do I/O.

I/O on Many Processors
(PIO Async Mode)



Ed Hartnett 6/1/19

Computational Components use NetCDF API

- 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.

Computational Components Use NetCDF Code

```
 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;
```

Global vs. Local Arrays

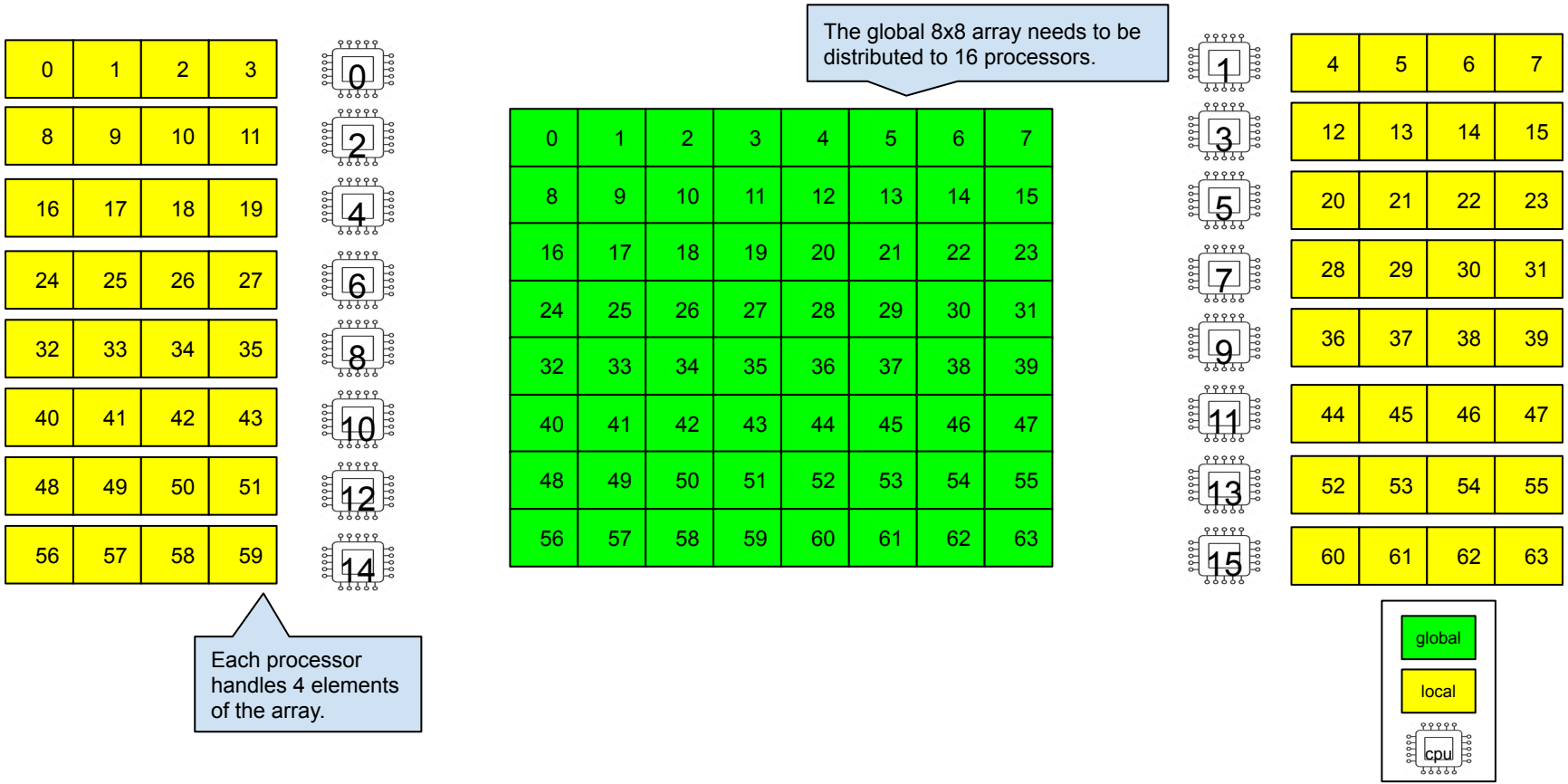
- The shape of a netCDF record defines the global data space.
- Once divided on to many processors, each processor has a subset of the global data space - the local array.
- Together, all local arrays add up to the global array.
- There may be halos - data that are needed for computation but are outside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



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.