Advanced Message-Passing Programming

Challenges of Parallel IO

Reusing this material

Overview

- Lecture will cover
  - Why is IO difficult
  - Why is parallel IO even worse
  - Straightforward solutions in parallel
  - What is parallel IO trying to achieve?
  - Files as arrays
  - MPI-IO and derived data types

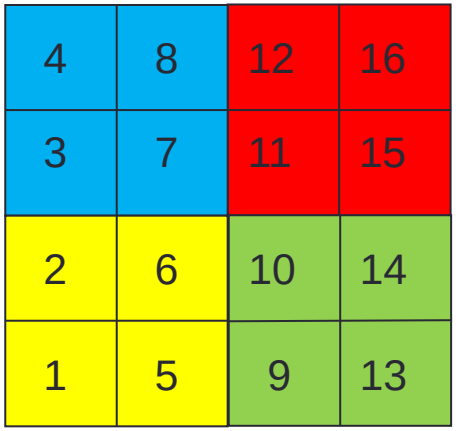
Why is IO hard?

- Breaks out of the nice process/memory model
  - data in memory has to physically appear on an external device
- Files are very restrictive compared to data arrays
  - random access (fseek) very inefficient
  - linear access to file may require remapping of program data
- Many, many system-specific options to IO calls
- Different formats
  - text, binary, big/little endian, Fortran unformatted, HDF5, NetCDF, ...
- Disk systems are very complicated
  - RAID disks, many layers of caching on disk, in memory, ...
- IO is the HPC equivalent of printing!

Serial IO of a 4x4 array

```
double x[4][4];
fwrite(x, sizeof(double), 4*4, fp);
```

Serial data



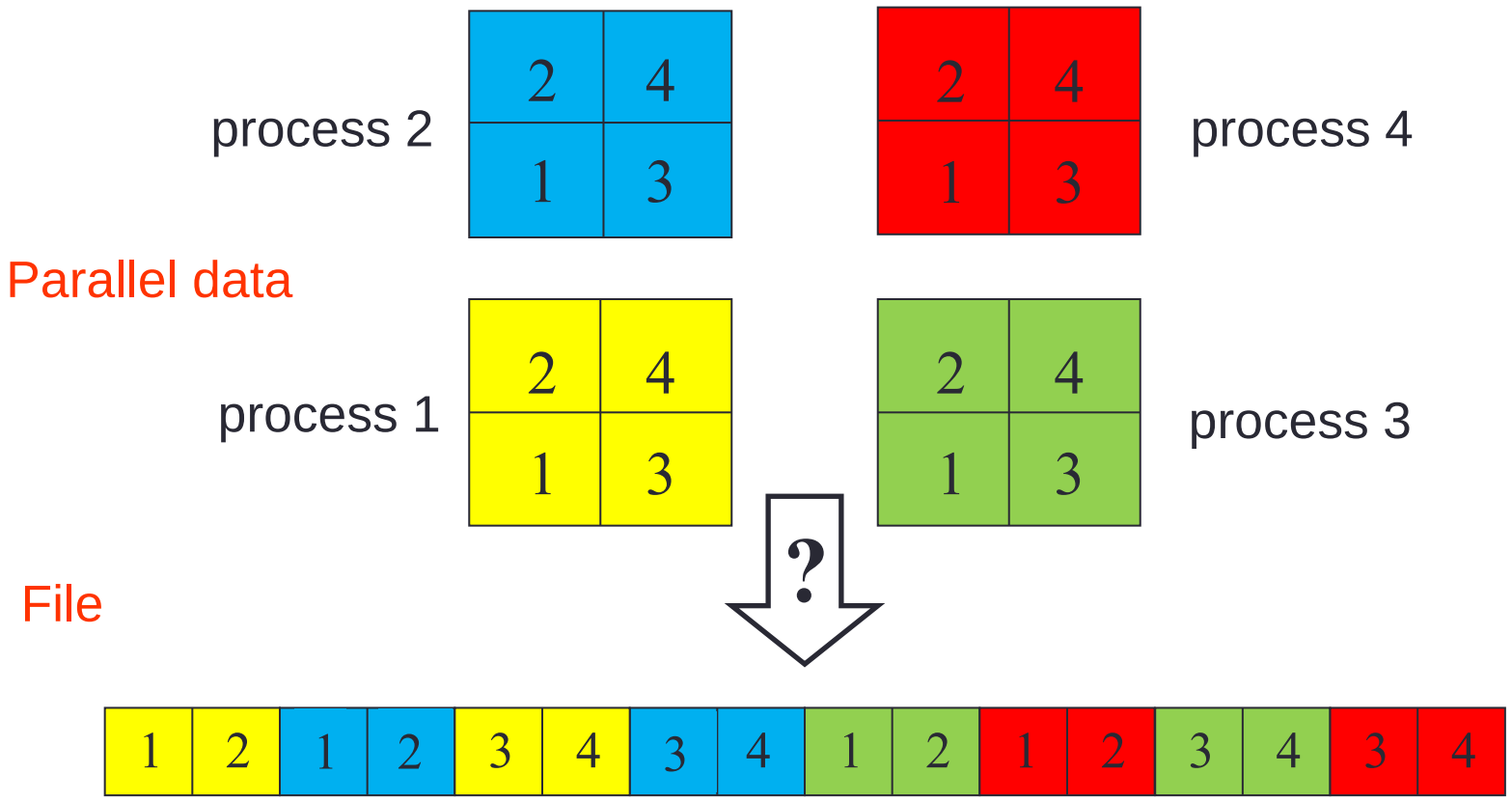
File

epcc

THE UNIVERSITY OF EDINBURGH

Parallel IO of a 4x4 distributed array

```
double x[2][2]; // On each of 4 processes
// Insert magic parallel IO call here!
```



epcc

THE UNIVERSITY OF EDINBURGH

Naive solution #1

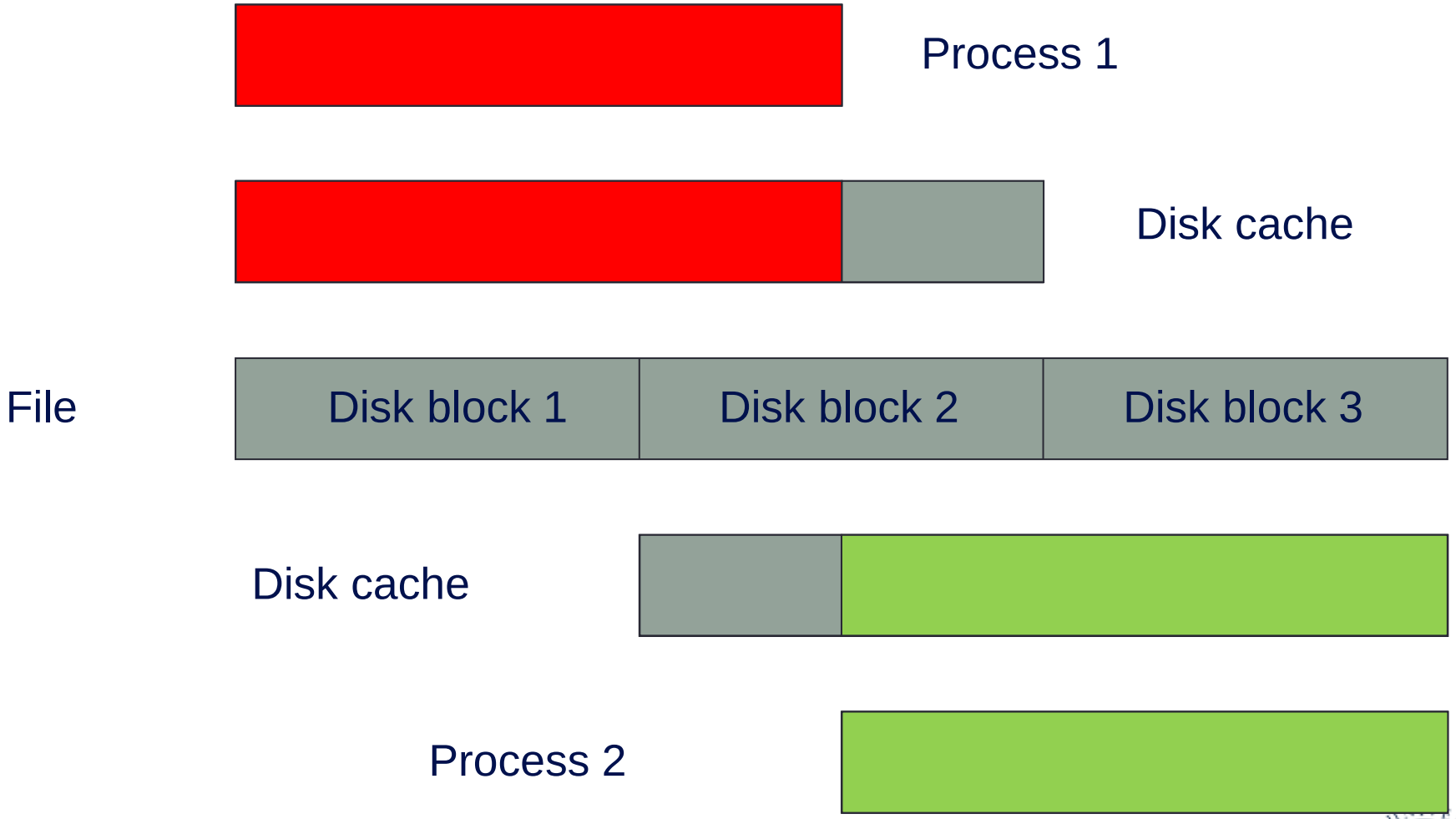
```
double x[2][2]; // On each of 4 MPI processes
FILE *fp;
fp = fopen("globalfile.dat", "w");
fseek(fp, process-specific-offset, SEEK_SET);
fwrite(...);
...
```

- This will not work in general
  - multiple processes writing to the same file is problematic
- Performance will be poor
  - would require many cycles of seek, write, seek, write, ...

Why is Parallel IO Harder?

- Cannot have multiple processes writing to a single file
  - Unix generally cannot cope with this
  - data cached in units of disk blocks (eg 4K) and is *not coherent*
  - not even sufficient to have processes writing to distinct parts of file
    - a parallel FS may get correct results by locking but this serialises the IO
- Even reading can be difficult
  - 1024 processes opening a file can overload the filesystem (fs)
- Data is distributed across different processes
  - processes do not in general own contiguous chunks of the file
  - cannot easily do linear writes
  - local data may have halos to be stripped off

Simultaneous Access to Files

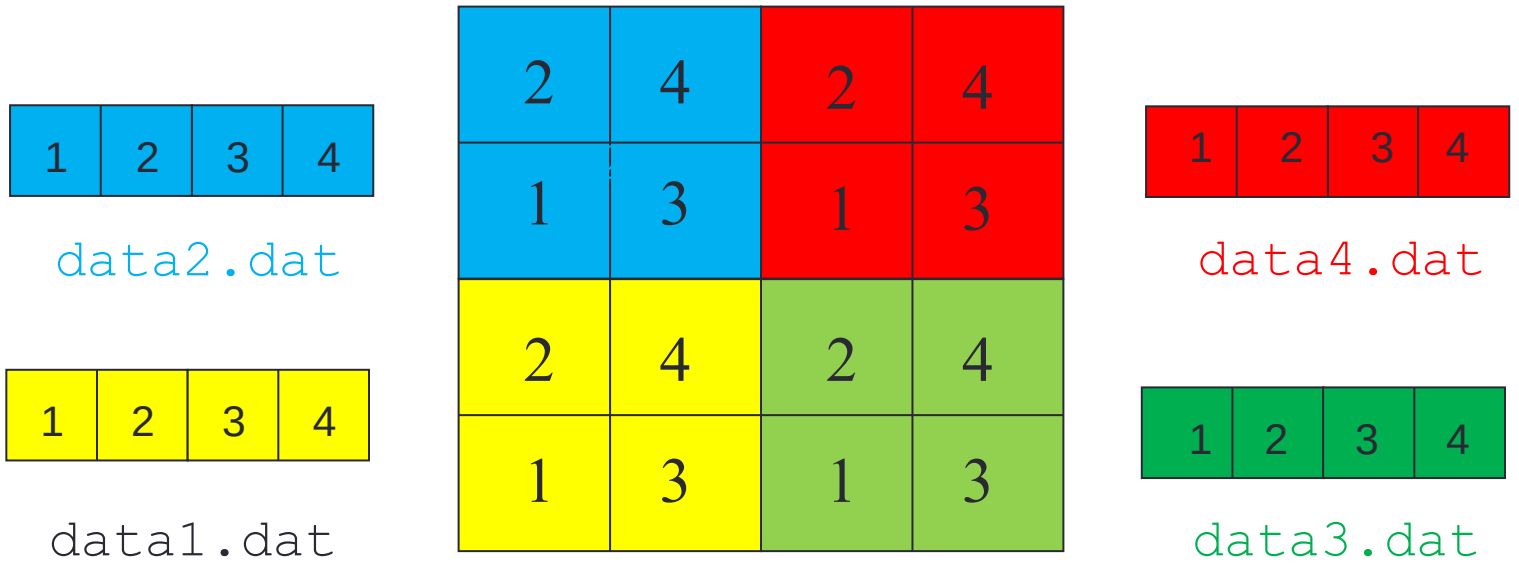


Naive solution #2: Independent I/O

- Also called File-Per-Process
- Each process opens its own data file
  - writes local data
- Simple to implement but ...
  - a nightmare to cope with in practice (thousands of files)
  - often requires a lot of pre- and post-processing
- Data format depends on number of processes
  - completely different output from serial code
- I find this philosophically wrong!
  - parallel computing should get the **same result** but **faster**

Parallel IO of a 4x4 distributed array

```
double x[2][2]; // On each of 4 processes
fp = fopen(locafile, "w");
fwrite(x, sizeof(double), 2*2, fp);
```

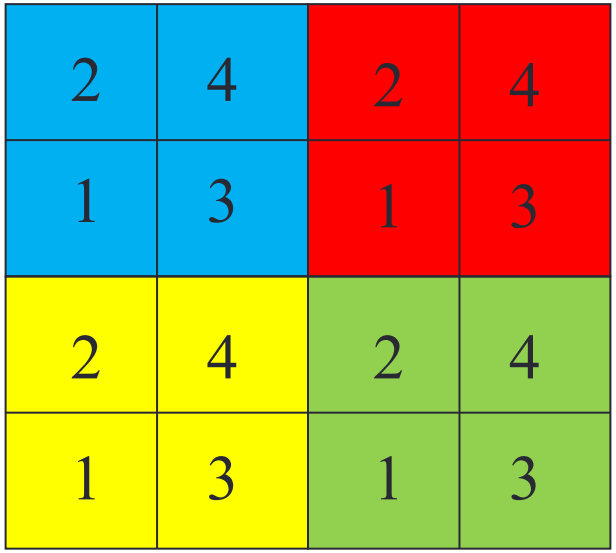


epcc

THE UNIVERSITY OF EDINBURGH

This is what we want to achieve

Parallel Data



File

epcc

THE UNIVERSITY OF EDINBURGH

Naive Solution #3: Controller IO

- Controller IO
  - send all data to/from single controller process
  - e.g. scatter/gather from/to other process from rank 0
  - write/read a single file
- Main advantage is that output is the same as serial
- Issues
  - quickly run out of memory on the controller
    - or have to write in many small chunks
- Performance is poor
  - only a single process is writing
  - a serial solution – no benefit from parallel filesystem (see later)

What do we Need?

- A way to do parallel IO properly
  - where the IO system deals with all the system specifics
- Want a single file format
  - We already have one: the serial format

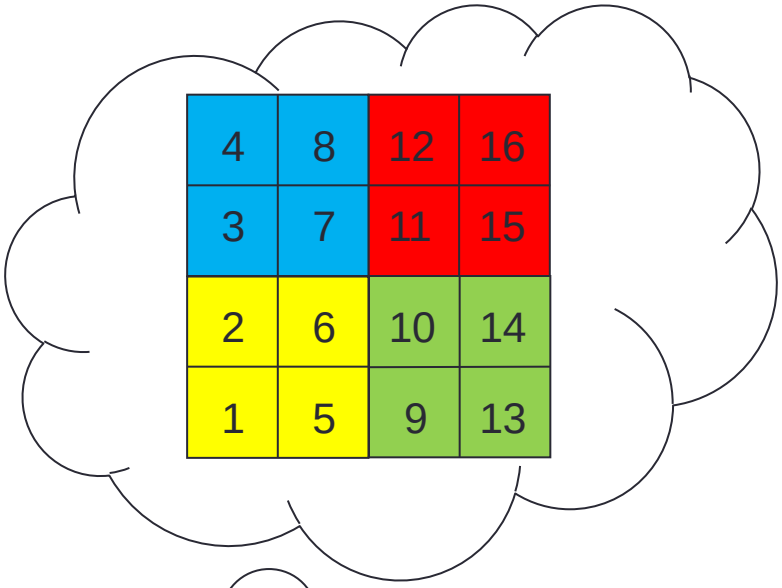
- All files should have same format as a serial file
  - entries stored according to position in global array
    - not dependent on which process owns them
  - order should always be 1, 2, 3, 4, ..., 15, 16

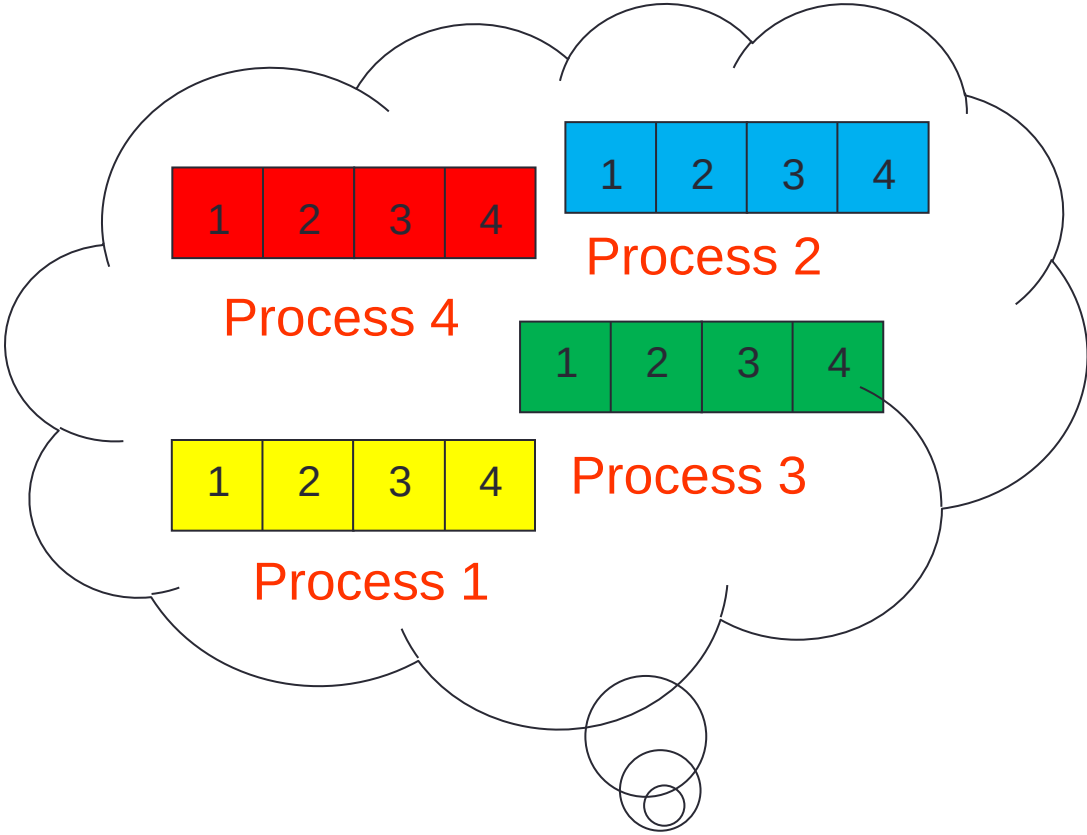
- A number of libraries exist
  - we will consider MPI-IO first

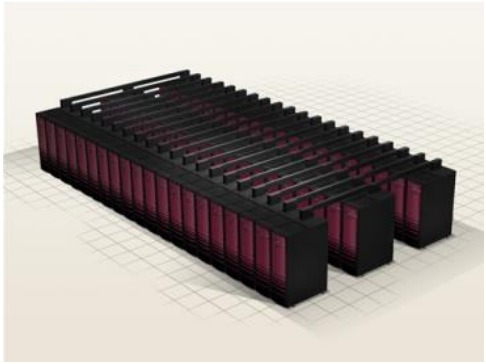
Information on Data Layout

- What does the IO system need to know about the data?
  - how the local arrays should be stitched together to form the file
- But ...
  - mapping from local data to the global file is only in the mind of the programmer!
  - the program does not know that we imagine the processes to be arranged in a 2D grid
- How do we describe data layout to the IO system?
- Crucial extra step in all parallel IO:
  - tell the IO library what portion(s) of the global file each process owns

Programmer View vs Machine View







epcc

THE UNIVERSITY OF EDINBURGH

Files vs Arrays

- Think of the file as a large array
  - forget that IO actually goes to disk
  - imagine we are recreating a single large array on a controller process

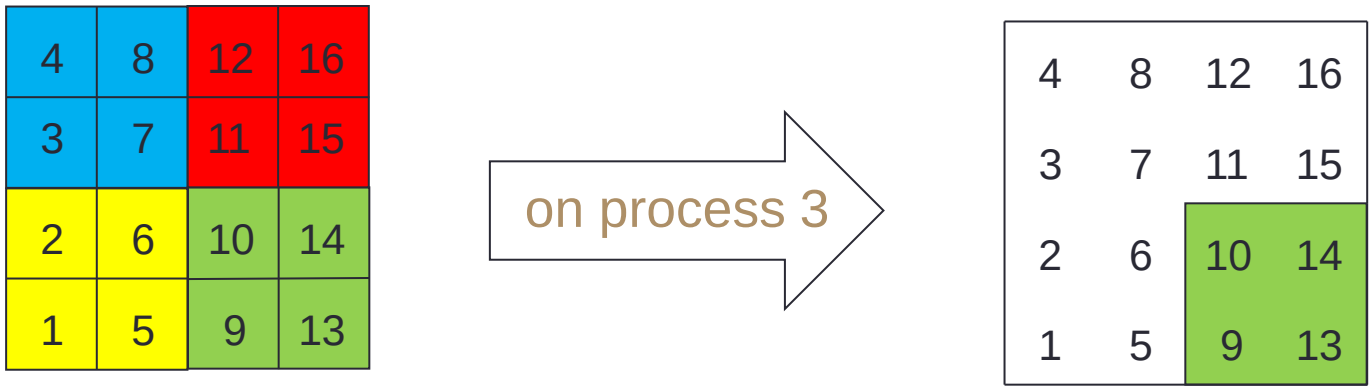
- The IO system must create this array and save to disk
  - without running out of memory
    - never actually creating the entire array
    - i.e. without doing naive controller IO
  - and by doing a small number of large IO operations
    - merge data to write large contiguous sections at a time
  - utilising any parallel features
    - doing multiple simultaneous writes if there are multiple IO nodes
    - managing any coherency issues re file blocks

MPI-IO Approach

- MPI-IO is part of the MPI standard
  - http://www.mpi-forum.org/docs/docs.html

- Each process needs to describe what subsection of the global array it holds
  - it is entirely up to the programmer to ensure that these do not overlap for write operations!

Data Sections



- Describe 2x2 subsection of 4x4 array
- Using standard MPI derived datatypes
- A number of different ways to do this

Other Parallel IO Libraries

- MPI-IO is usually the lowest level
  - you may never call it directly
- Higher level libraries exit
  - HDF5
  - NetCDF
  - ADIOS2
- Approach is the same
  - some way of describing what portion(s) of file each process owns
  - these libraries usually use MPI-IO to do their reads and writes

Summary

- Parallel IO is difficult
  - in theory and in practice
- MPI-IO provides a higher-level abstraction
  - user describes global data layout using derived datatypes
  - MPI-IO hides all the system specific filesystem details …
  - … but (hopefully) takes advantage of them for performance
- More flexible formats exist, e.g. NetCDF, HDF5 and ADIOS2
  - they gain performance by layering on top of MPI-IO
- User requires a good understanding of derived datatypes