Parallel I/O

CPS343

Parallel and High Performance Computing

Spring 2013
Outline

1. Overview of parallel I/O
   - I/O stratagies

2. MPI I/O

3. Parallel HDF5
   - Layers
   - Hyperslabs
   - Example
Some material used in creating these slides comes from

- ref 1
- ref 2
Parallel cluster structure

compute nodes

network / interconnect

storage nodes
1. Overview of parallel I/O
   - I/O strategies

2. MPI I/O

3. Parallel HDF5
   - Layers
   - Hyperslabs
   - Example
One file per process: all write

- Each process writes to its own file.
- Relatively simple I/O strategy.
- Independent writes can perform well because multiple storage servers can support parallel I/O to many separate files.
- Does not scale well if many nodes will subsequently need to access many different files.

Image source: https://fs.hlrs.de/projects/craydoc/docs/books/S-2490-40/html-S-2490-40/chapter-ht5w8mp0-oswald-iostrategies.html
Single shared file: one writes

- All processes involved in a write operation send data to one process, which then writes the data to a single shared file.
- Relatively simple; consolidates data which may be an advantage.
- Uses data aggregation.
- Writes are sequential and performance is limited by bandwidth of single process.

Image source: https://fs.hlrs.de/projects/craydoc/docs/books/S-2490-40/html-S-2490-40/chapter-ht5w8mp0-oswald-iostrategies.html
Single shared file: all write

- All processes involved in a write operation send data to a single shared file.
- Requires extra work by the application to maintain the separate file offsets for each process.
- Parallel I/O can be achieved.

Image source: https://fs.hlrs.de/projects/craydoc/docs/books/S-2490-40/html-S-2490-40/chapter-ht5w8mp0-oswald-iostrategies.html
A subset of all processes involved in a write operation sends data to a single shared file.

- Uses data aggregation.
- Sometimes used to strike a balance between many processes writing at the same time (with resulting frequent extent lock contention) and only one process writing (with the resulting loss of parallel I/O).

Image source: https://fs.hlrs.de/projects/craydoc/docs/books/S-2490-40/html-S-2490-40/chapter-ht5w8mp0-oswald-iostrategies.html
Objective

- Developed by IBM in 1994 and subsequently appeared in MPI-2 standard
- Uses underlying MPI send/receive routines to move data
- Supports MPI derived datatypes
- Both blocking and non-blocking send/receive modes are supported; allows I/O in parallel with computation
- read/write operations can be independent or collective
Independent I/O
- The “one process per file” mode we’ve already seen is an example of independent I/O
- I/O operations can occur in parallel
- Only one process is reading to and/or writing from the file

Collective I/O
- Like other MPI collective operations, all processes sharing a communicator must participate
- I/O operations can occur in parallel
- Typically each process reads or writes a portion of the file corresponding to its own memory space
Outline

1. Overview of parallel I/O
   - I/O strategies

2. MPI I/O

3. Parallel HDF5
   - Layers
   - Hyperslabs
   - Example
Parallel I/O stack

The parallel I/O stack on a Cray XT system is diagrammed as

[Diagram showing the parallel I/O stack with layers including MPI Application, POSIX I/O, Lustre, HDF5, and NetCDF]
Outline

1. Overview of parallel I/O
   - I/O strategies

2. MPI I/O

3. Parallel HDF5
   - Layers
   - Hyperslabs
   - Example
In HDF5 a *hyperslab* is a section of data. It is specified by four arrays, each having the same dimension as the dataspace the hyperslab belongs to. For dimension $i$:

- **offset** (also called *start*) The offset to the start of the hyperslab in dimension $i$
- **stride** The increment between one element in the hyperslab to the next element in dimension $i$
- **count** The number of blocks to read/write from dimension $i$
- **block** The number of elements in a block along dimension $i$

Outline

1. Overview of parallel I/O
   - I/O strategies

2. MPI I/O

3. Parallel HDF5
   - Layers
   - hyperslabs
   - Example
Cartesian grid

See


Compare with

http://www.cs.gordon.edu/courses/cps343/code/cart.c which is essentially the same program. This program collects the data from each process on the rank 0 process and displays it rather than writing a file.