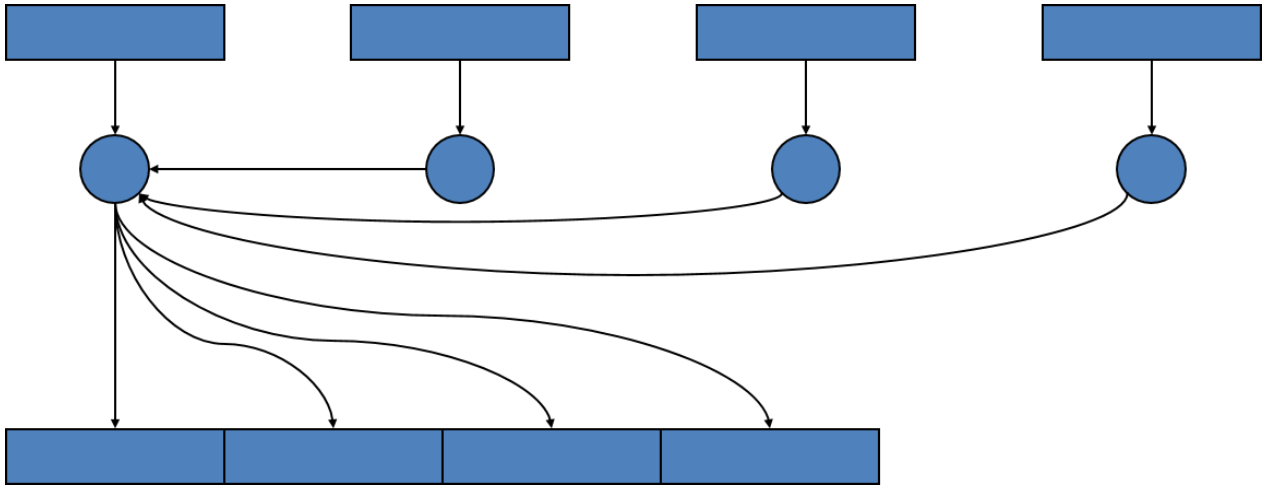
CPSC 4770/6770

Distributed and Cluster Computing

Lecture 10: Parallel IO – MPIIO

Common Ways of Doing I/O in Parallel Programs

- All processes send data to rank 0, and 0 writes it to the file



Pros:

parallel machine may support I/O from only one process (e.g., no common file system)

some I/O libraries (e.g. HDF-4, NetCDF) not parallel

resulting single file is handy for ftp, mv

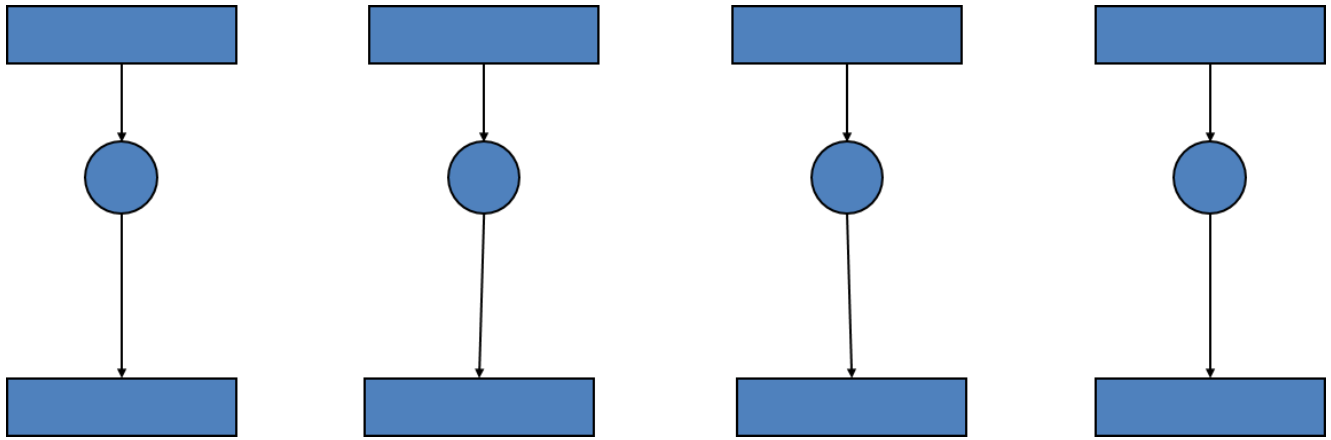
big blocks improve performance

short distance from original, serial code

Cons:

lack of parallelism limits scalability, performance (single node bottleneck)

How About Each Process Writes to a Separate File?



Pros:

parallelism, high performance

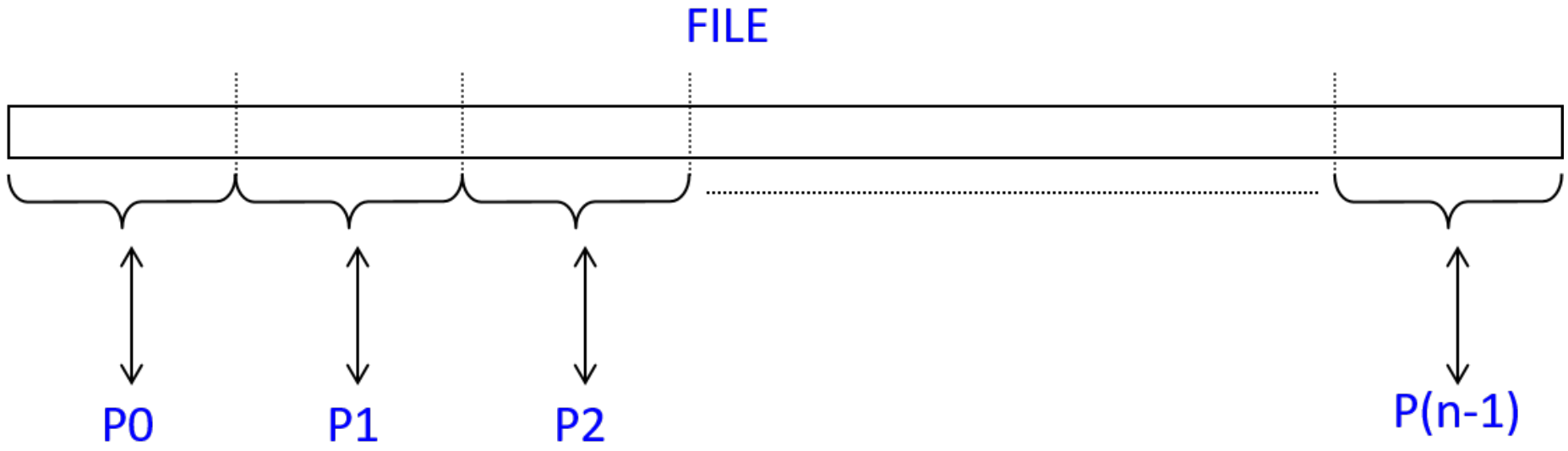
Cons:

lots of small files to manage

difficult to read back data from different number of processes

MPI-IO Approach

- What is Parallel I/O?
  - Multiple processes of a parallel program accessing data (reading or writing) from a common file



Why Parallel I/O?

- Non-parallel I/O is simple but
  - Poor performance (single process writes to one file) or
  - Awkward and not interoperable with other tools (each process writes a separate file)
- Parallel I/O
  - Provides high performance
  - Can provide a single file that can be used with other tools (such as visualization programs)

Why is MPI a good setting for Parallel I/O?

- Writing is like sending a message and reading is like receiving
- Any parallel I/O system will need a MPI-like mechanism to
  - define collective operations (MPI communicators)
  - define noncontiguous data layout in memory and file (MPI datatypes)
  - test completion of nonblocking operations (MPI request objects)

How Does it Work?

- Four stages
  - **Open File**
  - **Set File View (optional)**
  - **Read or Write Data**
  - **Close File**
- All the complexity is hidden in setting the file view
- Write is probably more important in practice than read

Opening a File (C Syntax)

- int **MPI_File_open**(MPI_Comm comm, const char *filename, int amode, MPI_Info info, MPI_File *fh)
  - amode: File access mode (integer), e.g., :
  - info: Info object (handle)
  - fh: New file handle (handle)

MPI_MODE_APPEND
MPI_MODE_CREATE -- Create the file if it does not exist.
MPI_MODE_DELETE_ON_CLOSE
MPI_MODE_EXCL -- Error creating a file that already exists.
MPI_MODE_RDONLY -- Read only.
MPI_MODE_RDWR -- Reading and writing.
MPI_MODE_SEQUENTIAL
MPI_MODE_WRONLY -- Write only.
MPI_MODE_UNIQUE_OPEN

Set File View (C Syntax)

- int **MPI_File_set_view**(MPI_File fh, MPI_Offset disp, MPI_Datatype etype, MPI_Datatype filetype, const char *datarep, MPI_Info info)
  - disp: Displacement (integer).
  - etype: Elementary data type (handle).
  - filetype: File type (handle).
  - datarep: Data representation (string).
  - info: Info object (handle).

Reading a File (C Syntax)

- int **MPI_File_read**(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)
  - fh: File handle (handle).
  - count: Number of elements in buffer (integer).
  - datatype: Data type of each buffer element (handle).
  - buf: Initial address of buffer (integer).
  - status: Status object (status).
- int **MPI_File_seek**(MPI_File fh, MPI_Offset offset, int whence)
  - fh: File handle (handle).
  - offset: File offset (integer).
  - whence: Update mode (integer).
    - MPI_SEEK_SET - The pointer is set to offset.
    - MPI_SEEK_CUR - The pointer is set to the current pointer position plus offset.
    - MPI_SEEK_END - The pointer is set to the end of the file plus offset.

Closing a File (C Syntax)

- **MPI_File_close**(MPI_File *fh)

mpiio_seqwrite.py

```python
#!/usr/bin/env python
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
amode = MPI.MODE_WRONLY|MPI.MODE_CREATE
comm = MPI.COMM_WORLD
fh = MPI.File.Open(comm, "./datafile.contig", amode)

buffer = np.empty(10, dtype=np.int)
buffer[:] = rank
print(buffer)

offset = comm.Get_rank()*buffer.nbytes
fh.Write_at_all(offset, buffer)
fh.Close()

if (rank == 0):
    print (np.fromfile("./datafile.contig", dtype="int"))
```

```
[jin6@node0378 15_parallel-IO]$ mpirun -np 4 --mca mpi_cuda_support 0 python mpiio_seqwrite.py
[0 0 0 0 0 0 0 0 0 0]
[1 1 1 1 1 1 1 1 1 1]
[2 2 2 2 2 2 2 2 2 2]
[3 3 3 3 3 3 3 3 3 3]
[0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3
 3 3]
```

mpiio_circwrite.py

```python
#!/usr/bin/env python
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank();size = comm.Get_size();

amode = MPI.MODE_WRONLY|MPI.MODE_CREATE
fh = MPI.File.Open(comm, "./datafile.noncontig", amode)
item_count = 10
buffer = np.empty(item_count, dtype='i')
buffer[:] = rank
print (buffer)
filetype = MPI.INT.Create_vector(item_count, 1, size)
filetype.Commit()

displacement = MPI.INT.Get_size()*rank
fh.Set_view(displacement, filetype=filetype)

fh.Write_all(buffer)
filetype.Free()
fh.Close()

if (rank == 0):
    print (np.fromfile("./datafile.noncontig", dtype="i"))
```

```
[jin6@node0378 15_parallel-IO]$ mpirun -np 4 --mca mpi_cuda_support 0 python mpiio_circwrite.py
[0 0 0 0 0 0 0 0 0 0]
[1 1 1 1 1 1 1 1 1 1]
[2 2 2 2 2 2 2 2 2 2]
[3 3 3 3 3 3 3 3 3 3]
[0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0
 1 2 3]
```

mpiio_bigwrite.py

```python
#!/usr/bin/env python
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
amode = MPI.MODE_WRONLY|MPI.MODE_CREATE
comm = MPI.COMM_WORLD
fh = MPI.File.Open(comm, "/scratch1/jin6/cpsc4770/datafile.contig", amode)

local_count = (int)(1600000000 / size)

buffer = np.empty(local_count, dtype=np.int)
buffer[:] = rank

offset = comm.Get_rank()*buffer.nbytes
fh.Write_at_all(offset, buffer)
fh.Close()
```

```
[jin6@login001 ~]$ qsub -I -l select=2:ncpus=16:mpiprocs=16:mem=16gb,walltime=8:00:00
qsub (Warning): Interactive jobs will be treated as not rerunnable
qsub: waiting for job 5391359.pbs02 to start
qsub: job 5391359.pbs02 ready

[jin6@node1597 ~]$ module load gcc/5.3.0 openmpi/1.10.3 python/3.4
```

```
[jin6@node1597 15_parallel-IO]$ time mpirun -np 8 --mca mpi_cuda_support 0 python mpiio_bigwrite.py

real    1m46.466s
user    0m14.508s
sys     0m20.896s
[jin6@node1597 15_parallel-IO]$ ls -lh /scratch1/jin6/cpsc4770/datafile.contig; rm /scratch1/jin6/cpsc4770/datafile.contig
-rw-r--r-- 1 jin6 cuuser 12G Feb 13 13:00 /scratch1/jin6/cpsc4770/datafile.contig
[jin6@node1597 15_parallel-IO]$ time mpirun -np 16 --mca mpi_cuda_support 0 python mpiio_bigwrite.py

real    1m46.504s
user    0m29.257s
sys     0m24.284s
[jin6@node1597 15_parallel-IO]$ ls -lh /scratch1/jin6/cpsc4770/datafile.contig; rm /scratch1/jin6/cpsc4770/datafile.contig
-rw-r--r-- 1 jin6 cuuser 12G Feb 13 13:03 /scratch1/jin6/cpsc4770/datafile.contig
[jin6@node1597 15_parallel-IO]$ time mpirun -np 32 --mca mpi_cuda_support 0 python mpiio_bigwrite.py

real    0m57.621s
user    1m9.806s
sys     0m13.403s
```

mpiio_bigwrite_2.py

```python
#!/usr/bin/env python
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
amode = MPI.MODE_WRONLY|MPI.MODE_CREATE
comm = MPI.COMM_WORLD
fh = MPI.File.Open(comm, "/home/jin6/datafile.contig", amode)

local_count = (int)(1600000000 / size)

buffer = np.empty(local_count, dtype=np.int)
buffer[:] = rank

offset = comm.Get_rank()*buffer.nbytes
fh.Write_at_all(offset, buffer)
fh.Close()
```

```
[jin6@node1597 15_parallel-IO]$ time mpirun -np 8 --mca mpi_cuda_support 0 python mpiio_bigwrite_2.py

real    1m52.092s
user    0m52.265s
sys     0m32.361s
[jin6@node1597 15_parallel-IO]$ ls -lh ~/datafile.contig; rm ~/datafile.contig
-rw-r--r-- 1 jin6 cuuser 12G Feb 13 13:08 /home/jin6/datafile.contig
[jin6@node1597 15_parallel-IO]$ time mpirun -np 16 --mca mpi_cuda_support 0 python mpiio_bigwrite_2.py

real    1m49.016s
user    4m46.431s
sys     0m41.144s
[jin6@node1597 15_parallel-IO]$ ls -lh ~/datafile.contig; rm ~/datafile.contig
-rw-r--r-- 1 jin6 cuuser 12G Feb 13 13:12 /home/jin6/datafile.contig
[jin6@node1597 15_parallel-IO]$ time mpirun -np 32 --mca mpi_cuda_support 0 python mpiio_bigwrite_2.py

real    0m54.710s
user    1m1.237s
sys     0m24.195s
[jin6@node1597 15_parallel-IO]$ ls -lh ~/datafile.contig; rm ~/datafile.contig
-rw-r--r-- 1 jin6 cuuser 12G Feb 13 13:13 /home/jin6/datafile.contig
```

Under the Covers of MPI-IO

- MPI-IO implementation is given a lot of information in this case:
  - Collection of processes reading data
  - Structured description of the regions
- Implementation has some options for how to obtain this data
  - Noncontiguous data access optimizations
  - Collective I/O optimizations

General Guidelines for Achieving High I/O Performance

- Buy sufficient I/O hardware for the machine
- Use fast file systems, not NFS-mounted home directories
- Do not perform I/O from one process only
- Make large requests wherever possible
- For noncontiguous requests, use derived datatypes and a single collective I/O call