CRAY
THE SUPERCOMPUTER COMPANY

Practical Examples for Efficient I/O on Cray XT Systems

Jeff Larkin
<larkin@cray.com>

Motivation:

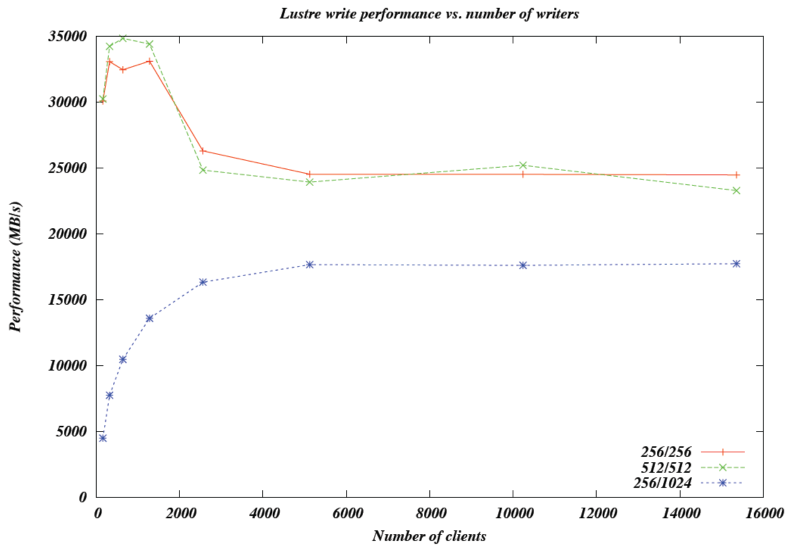
I/O is hard.

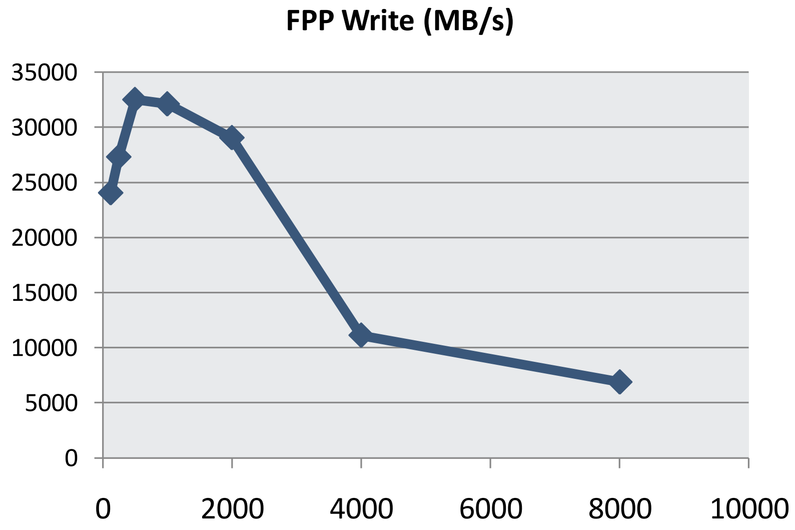
I/O is hard…

Interpreting I/O results is harder.

I/O: Don't over do it.

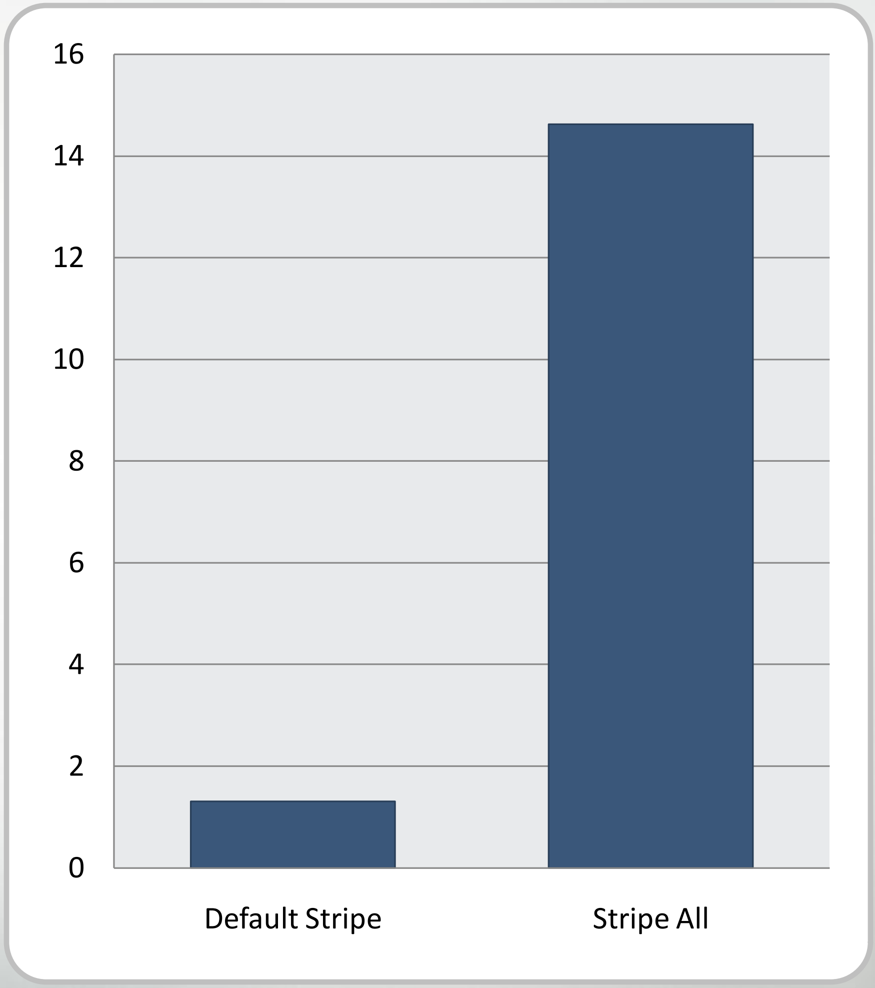
- Countless techniques for doing I/O operations
  - Varying difficulty
  - Varying efficiency
- All techniques suffer from the same phenomenon, eventually it will turn over.
  - Limited Disk Bandwidth
  - Limited Interconnect Bandwidth
  - Limited Filesystem Parallelism
- Respect the limits or suffer the consequences.





Lustre Striping

- Lustre is parallel, not paranormal
- Striping is critical and often overlooked
- Writing many-to-one requires a large stripe count
- Writing many-to-many requires a single stripe



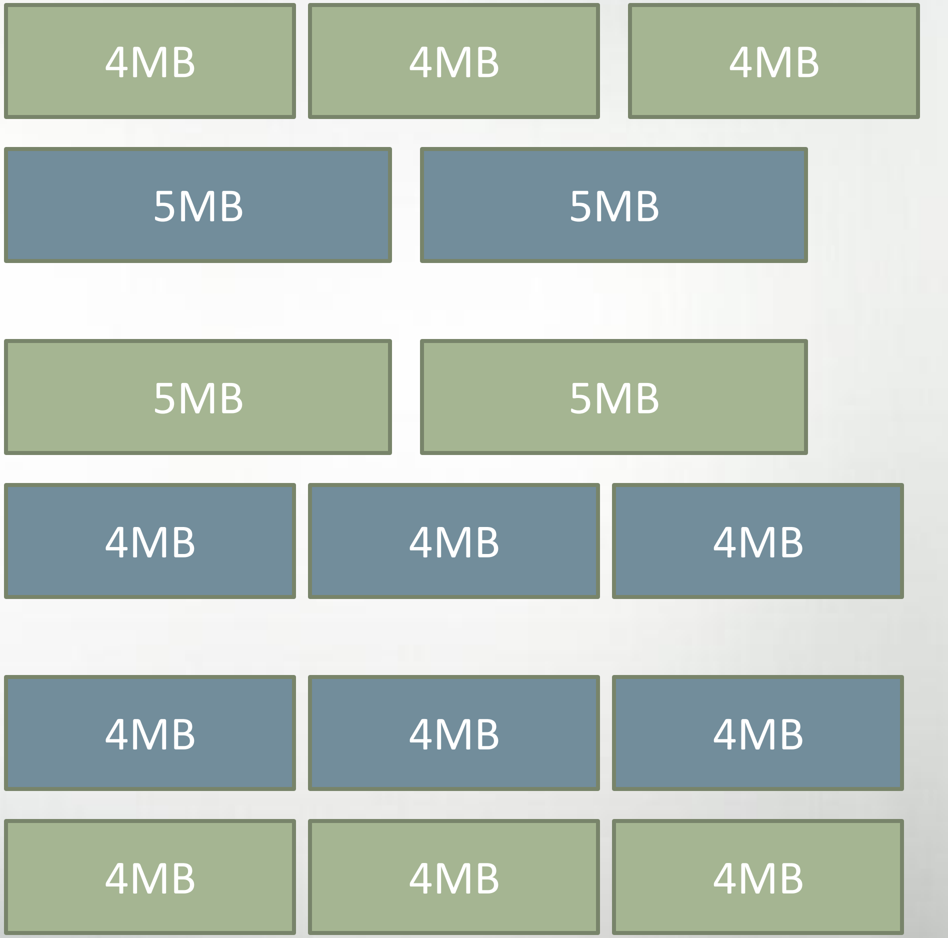
Lustre Striping: How to do it

- Files inherit the striping of the parent directory
  - Input directory must be striped before copying in data
  - Output directory must be striped before running
- May also “touch” a file using the lfs command
- An API to stripe a file programmatically is often requested, here’s how to do it.
  - Call from only one processor
- New support in xt-mpt for striping hints
  - striping_factor
  - striping_size

```
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <lustre/lustre_user.h>
int open_striped(char *filename,
  int  mode, int  stripe_size,
  int  stripe_offset, int  stripe_count)
{
  int fd;
  struct lov_user_md opts = {0};
  opts.lmm_magic = LOV_USER_MAGIC;
  opts.lmm_stripe_size = stripe_size;
  opts.lmm_stripe_offset = stripe_offset;
  opts.lmm_stripe_count = stripe_count;

  fd = open64(filename, O_CREAT | O_EXCL
| O_LOV_DELAY_CREATE | mode, 0644);
  if ( fd >= 0 )
    ioctl(fd, LL_IOC_LOV_SETSTRIPE,
&opts);

  return fd;
}
```

Lustre Striping: Picking a Stripe Size

- We know that large writes perform better so we buffer
- We can control our buffer size
- We can ALSO control our stripe size
- Misaligning Buffer and Stripe sizes can hurt your performance



4MB 4MB 4MB

5MB 5MB

5MB 5MB

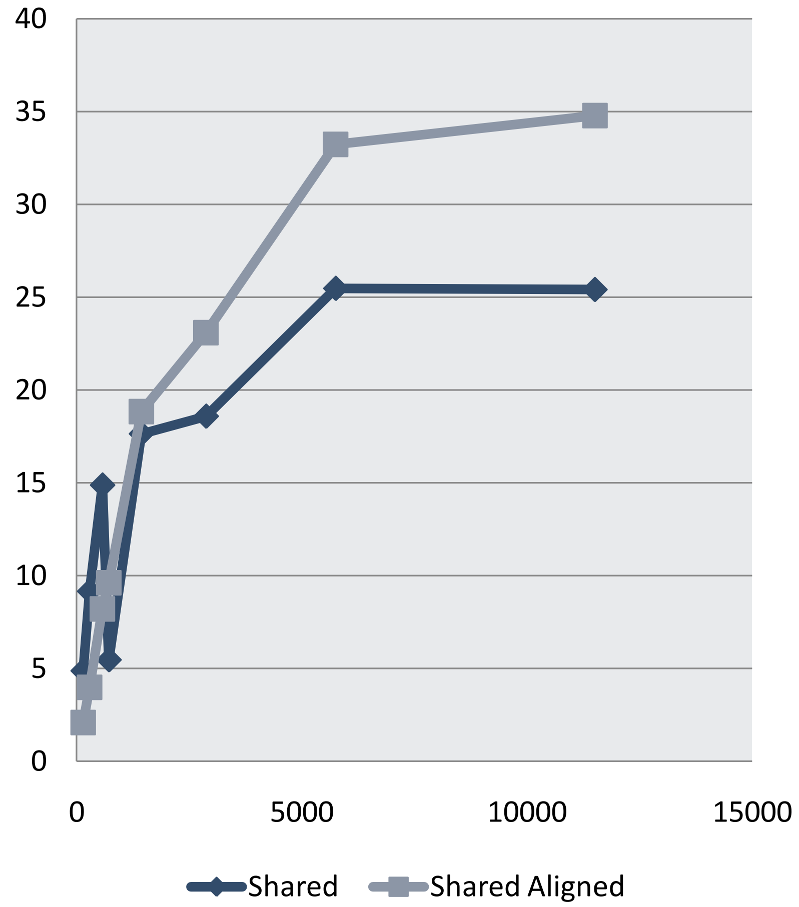
4MB 4MB 4MB

4MB 4MB 4MB

4MB 4MB 4MB

Memory Alignment: An Interesting Side Effect

- In order to use O_DIRECT, data buffers must be aligned to page boundaries
  - O_DIRECT is rarely a good idea
- Memory alignment can be done by:
  - C: posix_memalign instead of malloc
  - FORTRAN: over-allocation and the loc function
- Aligning I/O buffers on page boundaries can improve I/O performance.

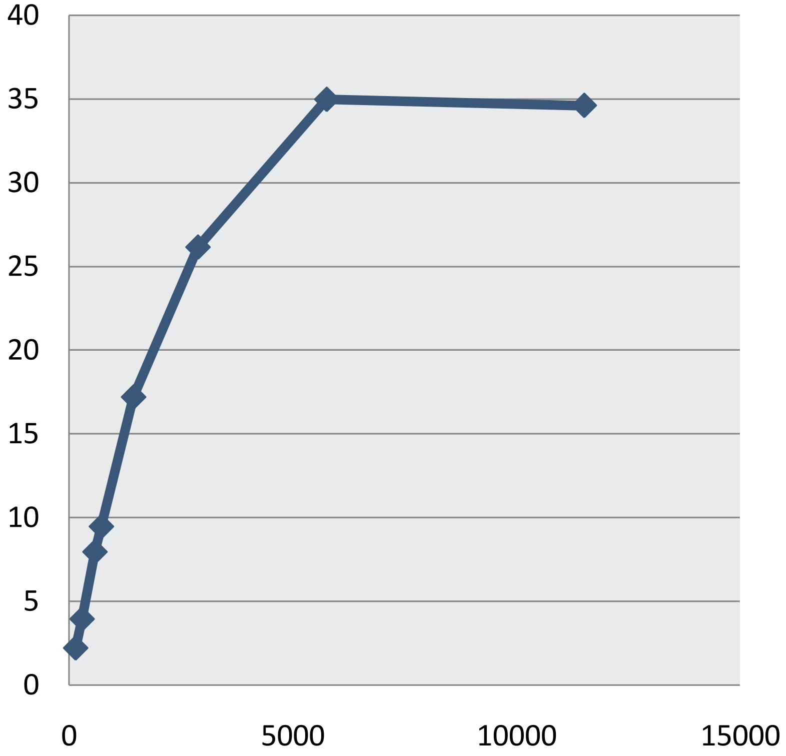


File-per-process

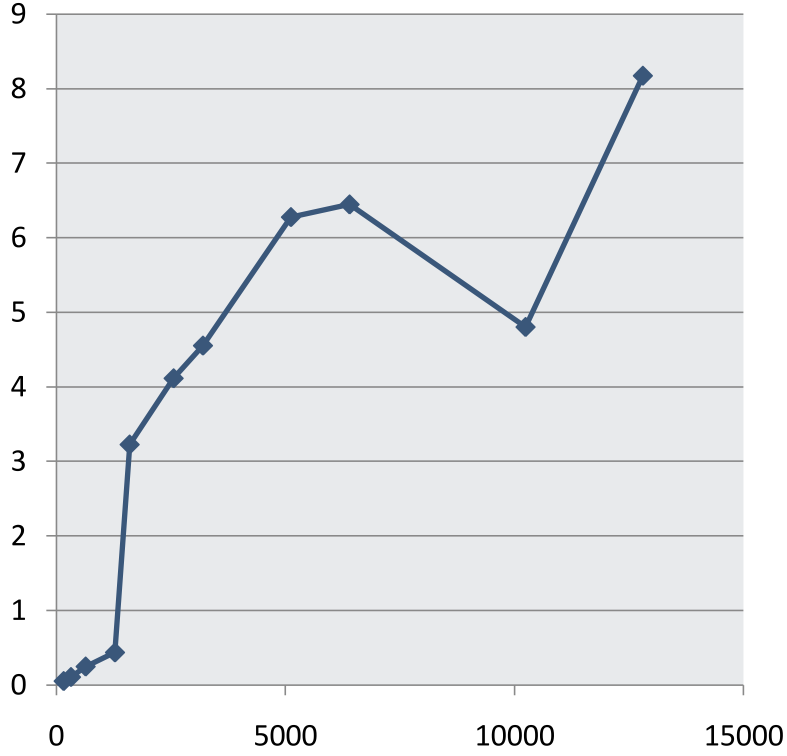
- This method is simple to implement and can utilize > 160 OST limit
- This method is also very stressful on the FS and inconvenient with thousands of clients
  - Too many opens at once floods the MDS
  - Too many concurrent writers can stress the OSTs
  - Too small writes kills performance
  - Too many files stresses user

File-per-process

Performance Results



Open Time



POSIX Shared File & Fortran Direct Access

- Slightly more difficult to implement than fpp
  - still fairly easy
- Generally slightly less efficient than fpp
- More convenient than many files
- Nicer to the MDS? Maybe marginally.
- Still can overload OSTs from many writers
- Try to make sure that two processors don’t need to write to same stripe

POSIX Shared File & Fortran Direct Access: The Pseudo-Code

POSIX Shared

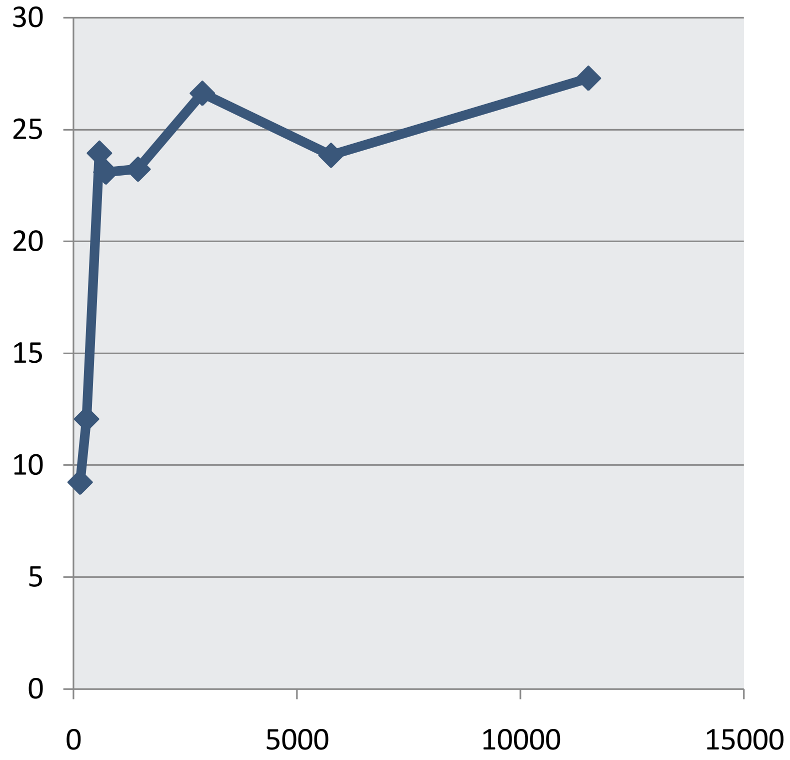
```
fd = open64("test.dat", mode, 0644);
/* Seek to start place for rank */
ierr64 = lseek64(fd, commrank*iosize,
   SEEK_SET);
remaining = iosize;
/* Write by buffers to the file */
while (remaining > 0)
{
  i = (remaining < buffersize ) ?
      remaining : buffersize;
  /* Copy from data to buffer */
  memcpy(tmpbuf, dbuf, i);
  ierr = write(fd, tmpbuf, i);
  if (ierr >= 0) {
    remaining -= ierr;
    dbuf += ierr;
  } else
  {
    MPI_Abort(MPI_COMM_WORLD, ierr);
  }
}
close(fd);
```

Fortran Direct

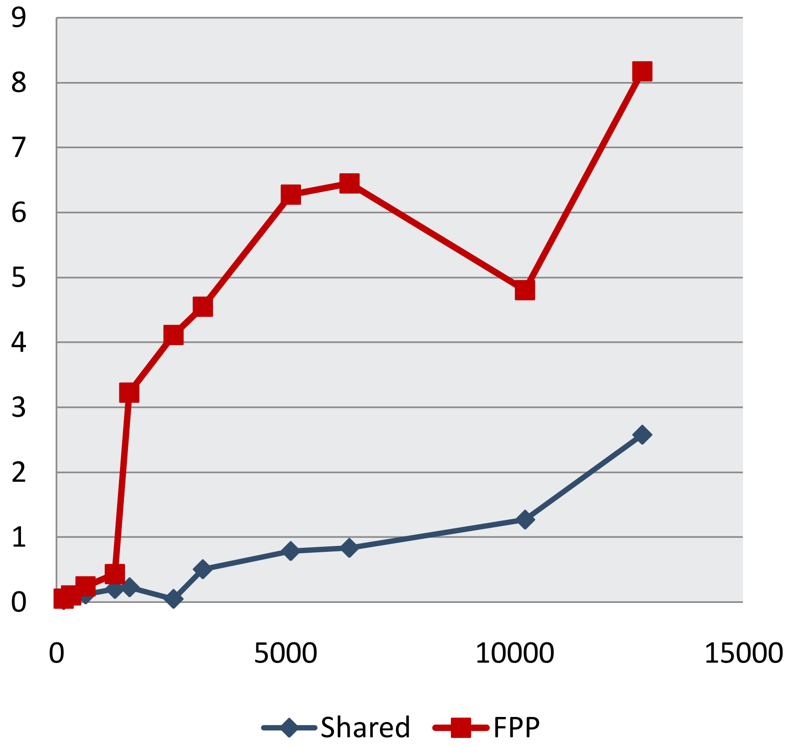
```
! Establish Sizes
reclength = 8*1024*1024
iosize = reclength * 10
! Starting Record For Rank
recnum = (iosize * myrank)/reclength
recs = iosize/8
numwords = recs/10

open(fid, file='output/test.dat',
   status='replace', form='unformatted',
   access='direct', recl=reclength,
   iostat=ierr)
! Write a record at a time to the file
do i=1,recs,numwords
  write(fid, rec=recnum, iostat=ierr)
    writebuf(i:i+numwords-1)
    recnum = recnum + 1
end do
close(fid)
```

POSIX Shared File & Fortran Direct Access: The Results

Performance Results



Open Time



Subgrouping

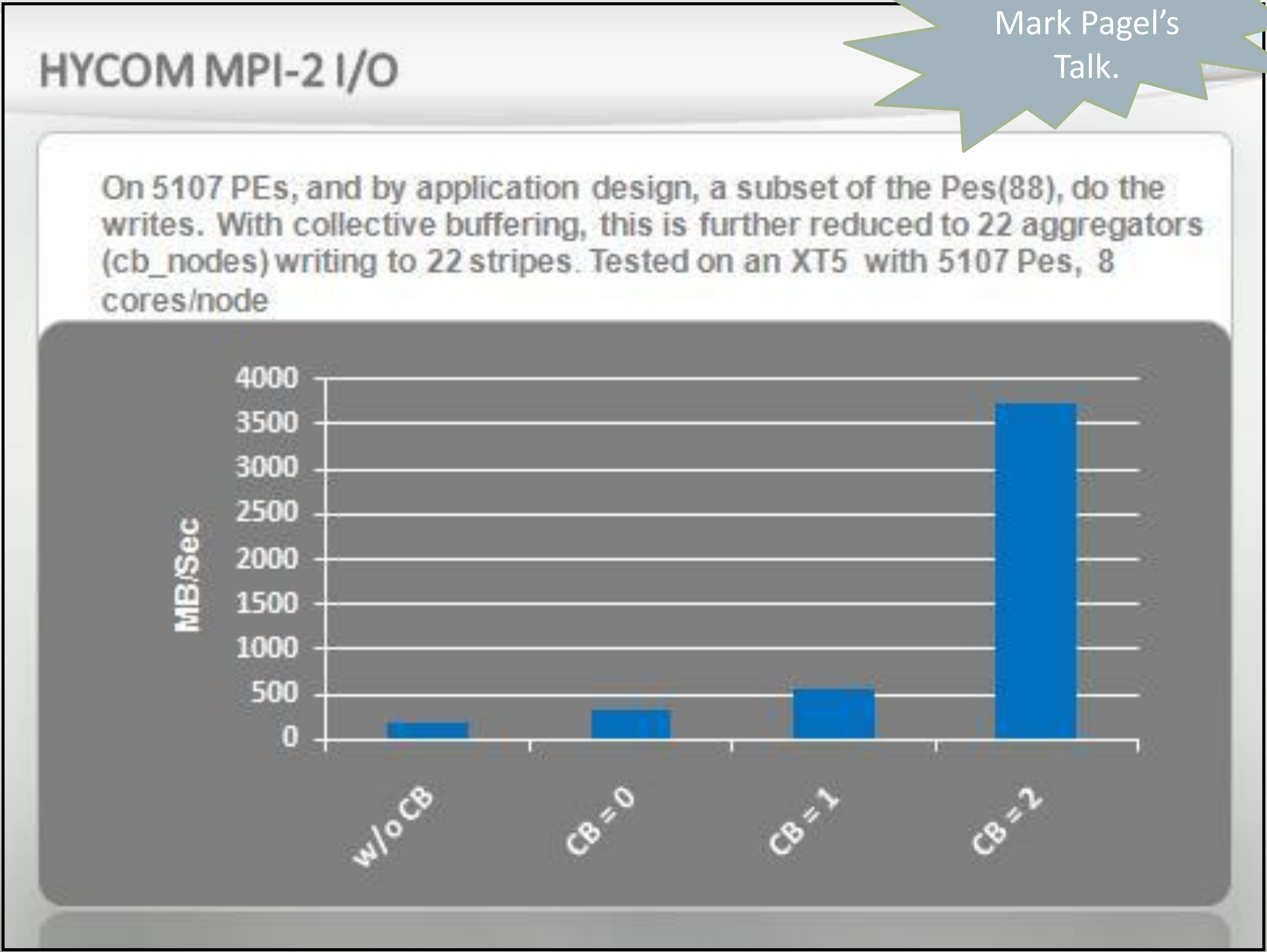
- I/O Scaling Limitations
  - Turns over above some number of clients
  - Shared files are limited to 160 OSTs, but some filesystems have more
- Can we use this knowledge to improve I/O performance?
- Aggregate I/O via sub-grouping to
  - Reduce number of clients using the FS
  - Aggregate into larger I/O buffers
  - Potentially cover > 160 OSTs via multiple shared files
- We can do this
  - Via MPI-IO Collective Buffering
  - By hand (many different ways)

Subgrouping: MPI-IO Collective I/O

- MPI-IO provides a way to handle buffering and grouping behind the scenes
  - Advantage: Little or No code changes
  - Disadvantage: Little or No knowledge of what's actually done
- Use Collective file access
  - MPI_File_write_all – Specify file view first
  - MPI_File_write_at_all – Calculate offset for each write
- Set the cb_* hints
  - cb_nodes – number of I/O aggregators
  - cb_buffer_size – size of collective buffer
  - romio_cb_write – enable/disable collective buffering
- No need to split comms, gather data, etc.

Subgrouping: MPI-IO Collective I/O – Does it work?

Yes! See Mark Pagel’s Talk.



HYCOM MPI-2 I/O

On 5107 PEs, and by application design, a subset of the Pes(88), do the writes. With collective buffering, this is further reduced to 22 aggregators (cb_nodes) writing to 22 stripes. Tested on an XT5 with 5107 Pes, 8 cores/node

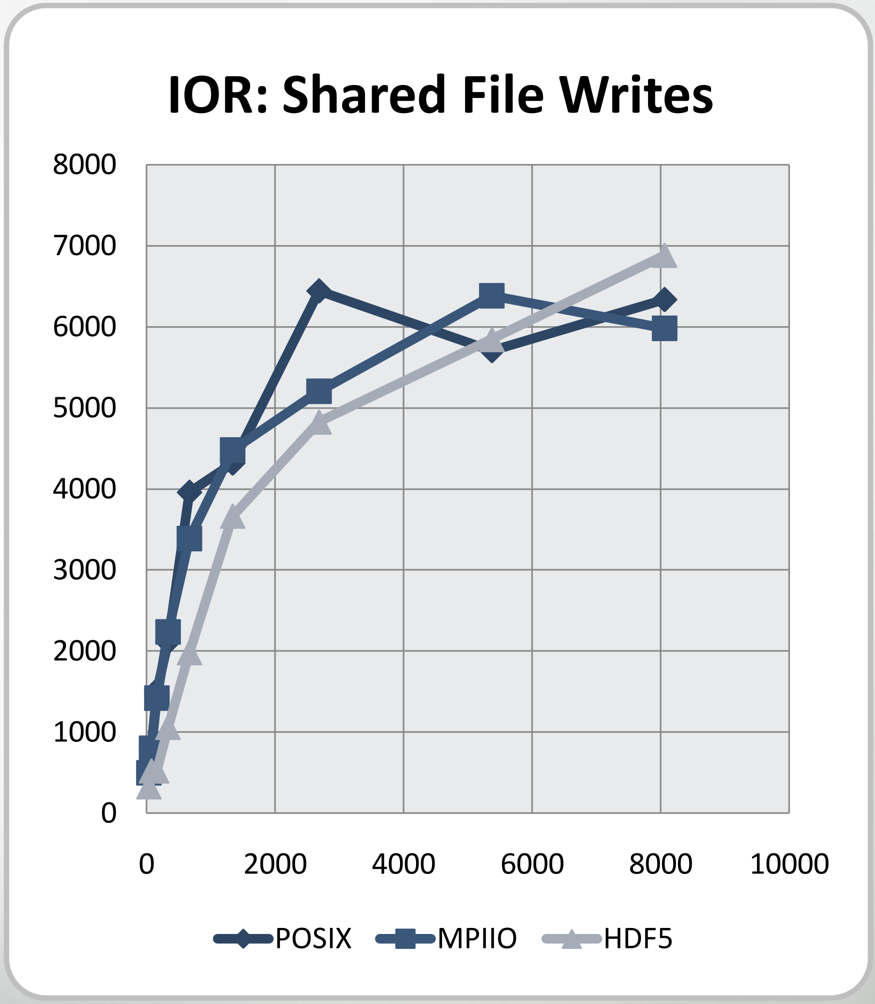
Subgrouping: By Hand

- Lose ease-of-use, gain control
- Countless methods to implement
  - Simple gathering
  - Serialized Sends within group
  - Write token
  - Double Buffered
  - Bucket Brigade
  - …
- Look for existing groups in your code
- Even the simplest solutions often seem to work.
  - Try to keep the pipeline full
  - Always be doing I/O
- Now we can think about multiple shared files!

I find your lack of faith in ROMIO disturbing.

What about HDF5, NetCDF, Etc?

- Every code uses these very differently
- Follow as many of the same rules as possible
- It is very possible to get good results, but also possible to get bad
- Because Parallel HDF5 is written over MPI-IO, it's possible to use hints



Thank You

Related CUG Talks/Papers

- *Performance Characteristics of the Lustre File System on the Cray XT5 with Regard to Application I/O Patterns*, Lonnie Crosby
- *Petascale I/O Using The Adaptable I/O System*, Jay Lofstead, Scott Klasky, et al.
- *Scaling MPT and Other Features*, Mark Pagel
- MPI-IO Whitepaper, David Knaak, ftp://ftp.cray.com/pub/pe/download/MPI-IO_White_Paper.pdf

Thank You

- Lonnie Crosby, UT/NICS
- Mark Fahey, UT/NICS
- Scott Klasky, ORNL/NCCS
- Mike Booth, Lustre COE
- Galen Shipman, ORNL
- David Knaak, Cray
- Mark Pagel, Cray

CRAY

THE SUPERCOMPUTER COMPANY