Cluster performance, how to get the most out of Abel

Ole W. Saastad, Dr.Scient
USIT / ITF / FI
November 3\textsuperscript{rd} 2017
Introduction

• Smarte queueing
• Smart job setup
• Scaling
• Effective IO
• Big data handling
• Compiler usage
• Programming for the future
Smart queueing
Smart queueing

- Exclusive or none-exclusive?
- Exclusive
  --ntasks-per-node=16 (or 20 for rack 31)
  --nodes=n

- None-exclusive
  --ntasks=m
Smart job setup

• Ask only for what you really need
• Stage data smart
• Carefully consider the IO load
• Have you control over scaling and input?
• What happen if your job crashes or is killed?
• What about log files?
Ask only for what you need

• Queue system allocate and prioritise based on your footprint
• Footprint is based on
  – number of cores/nodes
  – Memory
  – Wall time

• Large footprint means longer wait time in queue
Stage data smart

• /work is faster and larger than /cluster
• There is NO backup on /work
• $SCRATCH is on /work
• $SCRATCH is deleted when job is finished
• /work/users/<your user name> is available
  – Files are kept for 45 days, no backup
• /project is slow, stage files on /work
Carefully consider the IO load

• /project is slow and not part of Abel
• /cluster is backed up and high performance
• /work is not backed up and has even higher performance
• Backup cannot tackle millions of files
• Small log files are ok on /cluster/home
• Large files should be on /work
  – $SCRATCH for temporary scratch files
  – /work/user/<username> for log files
• Files on /work/users are deleted after 45 days
  – Touching the files does not prevent this!
Carefully consider the IO load

• /cluster and /work file system are unhappy with millions of small files

• If you use millions of temporary files ask for local scratch named $LOCALTMP
  --gres=localtmp:x (x is in GiB)

• Backup system are unhappy with millions of small files
  – Use a directory called nobackup or /work
Have you control over scaling and input?

- Does your application scale?
- Does it scale to the core count you request?
- Does it scale with the given input?

- OpenMP threads
- MPI ranks
- Hybrid (multi threaded – MPI executables)
MPI Vs OpenMP

- MPI stands for Message Passing Interface. It is a set of API declarations on message passing (such as send, receive, broadcast, etc.), and what behavior should be expected from the implementations.
  - Many nodes many threads
  - OpenMPI is MPI (not OpenMP)

- OpenMP is an API which is all about making it (presumably) easier to write shared-memory multi-processing programs.
  - One node many threads
What happen if your job crashes or is killed?

• If your application crashes everything works as if it terminated normally

• If your job is killed (memory, time limit etc)
  – `chkfile <file>` work as expected
  – Directory `$SCRATCH` is removed
  – Files on `$SCRATCH` are lost
What about log files?

- Log files can reside on /cluster/home or /work
  - No need to use $SCRATCH for them
  - Logs of reasonable size on $SUBMITDIR

- If log files are written to $SCRATCH remember to use chkfile

- Log files are required when reporting problems (RT)
Effective IO

• How much IO is your application doing?
• Few large files or millions of small files?
• Where are the files placed?
• Copying of files
• Archiving many files to one large
• Do the files compress?
  – Not all files compress well
Effective IO

• /work and /cluster are global parallel file systems, BeeGFS.
• In order to make it coherent locks and tokens are needed.
• Any operation must be locked to prevent corruption if more than one writer want to update a file.
• All parallel file systems struggle to cope with huge amount of metadata updates.
• They love few large files!
Effective IO

• Use $SCRATCH or /work/users/ during runs
• Use $LOCALTMP for millions of files
  – Make a tar archive of these before moving them to /work or /cluster
• Stage files from /projects to /work/users
  – /project has limited bandwidth
Big data handling

• Few large files or millions of small files?
• BeeGFS can handle large files easy
  – Bandwidth like 8-10 GiB/s
  – /work have higher bandwidth than /cluster

• Use infiniband if possible
  – scp file.ext <host ib ip number>:
  – Doubles the transfer speed
Big data handling

- Many small files are problematic
- Backup system struggle with millions of files
- BeeGFS struggle with metadata of millions of entries
- Make archives of directories with large number of files
- Use local scratch or temporal storage of possible
  - These sizes are relatively small
  - Tar directory to a single large file

- Mount an image on /work as a file system
  - Require help from support team
Big data handling

• Copying of millions of files are slow
• Rsync may take hours
• Option to use parallel rsync

1: rsync
2-32: parsync
Scaling
Scaling

The graph shows the speedup of a program as the number of cores increases. The red bars represent the measured speedup, and the blue bars represent the perfect speedup. As the number of cores increases from 4 to 32, the measured speedup approaches the perfect speedup, indicating good scaling performance.
Scaling

• How does your application scale?
• Is it dependent on input?
• Are you running at the sweet spot?
• Threads?
• MPI?
• Hybrid?
• Will your application scale to more than 120 threads to use the new Intel Knights Landing processor?
Compiler usage

• Parallel – 16 to 20 cores in a node
  – Vectorisation
  – OpenMP
  – MPI

• More and more focus on vectorisation
Programming for the future

• Pure MPI may yield too many ranks
  – Large memory demand
• Pure threads may scale weakly
  – OpenMP scaling is often limited
• Hybrid multi threaded MPI

• One MPI rank per core, 2-32 threads per core
Programming for the future

• Vectorization vital for performance
  – Scalar code perform at 1/16 performance

• OpenMP 4.0 introduces SIMD directives

• All OpenMP programming facilities now apply to SIMD

• Vectorization is made easier with OpenMP, just like threading was made simpler.
Programming for the future

- OpenMP SIMD - vectorization
- Single program, multiple data
- Vector units are becoming increasingly more important
- AVX / AVX2 256 bits = 4 dp and 8 sp floats
- AVX512 has 512 bits vector = 8 dp and 16 sp floats
- Vector fused multiply add double vector performance
Programming - Intel tuning tools

• Tuning tools
• Compiler – *only time the source code is read*
• MPI library – *can provide vital statistics*
• Vector Advisor - *vectorization*
• Thread Inspector - *Threads*
• Vtune Amplifier – *CPU / memory performance*
• Trace Analyser – *MPI calls*
Questions ?