

HECC User meeting Nov 7<sup>th</sup>

The bi-weekly tip:

Running on Pleiades: Let's Talk Performance

This is really the "mother of all performance tips". We go over what services the Apps group can provide and give an overview on some performance related topics. The plan is to dive into details on each item in future meetings.

What the Applications Group does:

- **Help with application-related issues, for example**
  - Porting your code to Pleiades/Electra from other systems
  - Basic performance analysis and optimization
  - Setup for parallel execution of numerous serial jobs
- **In-house developed tools for dealing with applications and job scheduling in /u/scicon/tools/bin/...**
  - Info about your running jobs: **qtop.pl**, **qps**, **qsh.pl**
  - Node availability: **qs**, **node\_stats**
  - Job startup: **mbind.x** (process binding),  
**several\_tries** (resilience)
  - Memory monitoring: **gm.x**, **vnuma**

**Helpful starting point to look for information:**

<https://www.nas.nasa.gov/hecc/support/kb/160/>

Choosing the right node type/number of nodes:

- Communication requirements?
  - See below
- Computational requirements?
  - See below
- Memory requirements?  
<https://www.nas.nasa.gov/hecc/support/kb/128/>
- I/O requirements?  
<https://www.nas.nasa.gov/hecc/support/kb/using-the-iot-toolkit-for-io-and-mpi-performance-analysis-546.html>
- Process/Thread Placement
  - See below
- ... much, much more

### **Process Placement**

- Why is this important?
  - o Resource sharing among cores on various levels: caches, sockets, memory access paths, ... A best practice is to **distribute threads/processes evenly across both sockets** on a node.

- There are various ways to accomplish this:  
<https://www.nas.nasa.gov/hecc/support/kb/113/>
  - Example 1: hybrid run on 1 Sandybridge node  
`qsub -l select=2:ncpus=16:model=san  
mpiexec -np 16 mbind.x -n 8 -t 2 ./a.out`
  - Example 2: OpenMP run on Bro. Note that using mbind yields a lower execution time and less variations in timings.

```
#!/bin/bash
#PBS -l walltime=00:30:00
#PBS -l select=1:ncpus=28:model=bro
#PBS -N NPB-OMP-C
#PBS -j oe
#PBS -q devel
. /usr/share/modules/init/sh
module load comp-intel
module list
make clean
make sp CLASS=C
for t in 16
do
  rm -f sp-C-out$t.txt sp-C-out$t-mbind.txt
  export OMP_NUM_THREADS=$t
  for i in `seq 1 5`
  do
    mbind.x -t 16 bin/sp.C.x >> sp-C-out${t}t-mbind.txt
    bin/sp.C.x >> sp-C-out${t}t.txt
  done
done

grep "Time in" sp-C-out16t-mbind.txt
Time in seconds = 50.59
Time in seconds = 51.14
Time in seconds = 49.12
Time in seconds = 48.85
Time in seconds = 48.88
pfe20.gjost 212> grep "Time in" sp-C-out16t.txt
Time in seconds = 56.81
Time in seconds = 57.80
Time in seconds = 57.36
Time in seconds = 57.82
Time in seconds = 55.74
```

#### Quick overview on MPI performance:

- MPI profiling using mpiprof:  
<https://www.nas.nasa.gov/hecc/support/kb/using-mpiprof-for-performance-analysis-525.html>

```
module load comp-intel/2015.3.187 mpi-sgi/mpt
module load /u/scicon/tools/modulefiles/mpiprof
```

easy to use:  
mpixexec -n \$NP mpiprof ./a.out

**(Example output)**

```
==> Summary of this run
Number of nodes           = 3
Number of MPI ranks      = 64
Number of inst'd functions = 16

Total wall clock time     = 182.711 secs
Average wall clock time  = 182.708 secs
Average computation time  = 135.856 secs (74.36%)
MPIProf overhead time    = 0.63683 secs ( 0.35%)

Average communication time = 46.1045 secs (25.23%)
  collective              = 46.1043 secs (25.23% or 100.00%Comm)
Total message bytes sent  = 1.7465T
  collective              = 1.7465T
Total message bytes received = 1.7465T
  collective              = 1.7465T
Gross communication rate  = 39.1685 Gbytes/sec
  collective              = 37.8806 Gbytes/sec
Communication rate per rank = 612.007 Mbytes/sec
```

... much much more

There are also **other tools**  
<https://www.nas.nasa.gov/hecc/support/kb/138/>

**MPI optimizations**

- few large messages vs many small messages, ... case specific. Problems figuring in out? Ask your friendly apps team ☺

**OpenMP optimizations**

- thread placement, see above
- diverse OpenMP considerations, case specific. Problems figuring it out? Ask your friendly apps team ☺

**CPU optimizations**

- Best practice compiler flags
- <https://www.nas.nasa.gov/hecc/support/kb/recommended-compiler-options-99.html>
- Different applications require different flags
- vectorization:
  - o Does the code vectorize?
    - For icc use the flag **-qopt-report=5** and inspect optimization reports
  - o Using the right instruction set:

- Wes has sse (16 bytes), San, Ivy, Has and Bro have avx2 (32 bytes) instruction sets, Skylake has avx512 (64 bytes)
- Compiler might only generate sse by default, to get e.g. avx2 instructions set `-xcore-avx2`
- To generate a "fat" executable use `-xcore-avx512,avx2,avx -xsse4.2`

- Memory layout: Achieve stride 1 access
- Memory bandwidth bound applications: Consider GPU programming? Problems tracking all this down? Ask your friendly Apps Team ☺

• Other items to consider: Memory requirements (e.g. `gmx.x`, `qps.sh` to find out), I/O (`mpiprof` tells something) Need help with this: Ask your friendly apps team ☺

### **Putting it all together:**

Example from NASA's Gravity Recovery and Interior Laboratory mission code

Originally: Ran on 900 Ivy nodes, left idle cores, due to memory requirements (32 GB memory per Ivy node), no process binding was used. Program hung early on.

Apps team helped with:

- Debugging: The code was opening a file on all ranks just to read some header information (not a good thing on Lustre). The user modified the code so it would open the file on at most `sqrt(ncpus)` thus avoiding possible Lustre contention.
- Process binding helped decreasing the runtime
- Optimization:
  - Switch to Has nodes (128 GB)
  - Use only 700 Has nodes, fully packed. This reduced internode communication and reduced vulnerability on network congestion (resilience)
  - Use the `-xcore-avx2` flag to take advantage of the avx2 instruction set => 2X Speed-up
  - Overall speed-up: Not known, but considerable.