**Parallel Systems - Assignment 1**

**Exercise 1:**

*qsub –* submit a job

*qstat –* info about queued/running jobs

*qdel –* delete job

*qsh –* run an interactive job, but make sure to cancel any interactive job after you are done

submission script that starts an arbitrary executable: */01/code/job.script*

5 most important parameters:

-pe parallel-environment number-of-slots

🡪 use *openmpi-Xperhost* to specify how many instances will be used per node

🡪 use *openmpi-fillup* to use the least number of nodes possible

essential parameter for parallel processing

-l

🡪 h\_rt=[hours:minutes:]seconds – stop job if it takes too long

🡪 h\_vmem=size[M|G] – limit per slot memory size

Both seem useful, if you want to save resources by trying to estimate/calculate how much time and memory your program should use before you start it!

-hold\_jid job-id

🡪 wait for job with job-id to finish.   
Could be useful for job that depends of the result of a previous job!

-w v

🡪 checks for syntax error in the job but does not submit the job.   
Remove parameter before submitting!

-o

🡪 specify the destination of the output file  
use -j yes to also join the error stream to this file

You need to specify the *-pe* parameter to run a program in parallel and tell the system the number of total slot as well as how many slots should be handled by each node/socket (*openmpi-Xperhost/openmpi-fillup).* Of course, the hardware needs to comprise multiple cores in order to run a program in parallel.

**Exercise 2:**

Tables & Figures of the OSU Benchmark: */01/ benchmark\_results.xlsx*

As expected, out **latency** performance results show that two processes communicating between two different nodes takes longer than it would on processes on the same node with communication on the same socket being the fastest. The latency imbalance of the different ranks decreases with an increase in size. One interesting observation was that the latency on data sizes >2GB was best on two different nodes (maybe due to memory limitations for processes on the same node)!

**Bandwidth** performance results showed that node to node communications was the slowest again, but both one-node communications where on par until a data size of 2MB. For larger data sizes the bandwidth performance dropped significantly, especially for the same socket benchmark and dropped a second time after 2GB (possibly due to cache and memory limitations respectively)!

You could verify rank placement without looking at the performance by executing the */bin/hostname* (for the nodeID) and *cat /proc/self/stat | awk '{print $39}'* (for the cpuID) commands at the start of each benchmark.

After comparing multiple benchmark runs, the results seem fairly stable except for occasional latency/bandwidth spikes for smaller data sizes within the same benchmark. Only during a single run did we get significantly higher latencies (for all data sizes) for the *different cores/same socket* benchmark, but this appears to be an outlier.   
MPI rank placement might depend on compute node utilization of the lcc2 supercomputer, because there might not be enough slots on the same socket/node, thus it is important to verify rank placement (with the method mentioned above).