SCC 110 Not-So-Slow-Slugs *UCSC*

November 4, 2025

Tu Lan Tran <ttran183@ucsc.edu>

Holden Martinez <hmarti24@ucsc.edu>

Tao Sato-Perry <tsatoper@ucsc.edu>

Julien Lee <pijlee@ucsc.edu>

Caleb Lin <clin189@ucsc.edu>

Myles Hallett <mthallet@ucsc.edu>

This is the submission for our Hero Run.

Note: By the time we got our cluster built and HPL compiled during the 24 hour window, we didn't have enough time to complete a full run. We submitted a smaller job hoping it would finish before our allocation was cut but it did not. This is a report on a 3 node cluster run during the 24 hours to follow instead.

Theoretical Peak FLOPS

We used 3 *m3.2xl* instances, each with 64 cores for a total of 192 cores. AMD EPYC 7713s have a base clock frequency of 2 GHz, and boosted frequency of 3.1 GHz. With this in mind, we will be using the base for our theoretical peak calculation.

Each core is capable of 16 double-precision floating point operations/cycle. We justify this considering AMD's AVX2 standard for SIMD width, we operate on 256-bit vectors or 4 double-precision numbers. Per element we have a FMA doing 2 floating point operations and per core there are 2 FMA units so we have 4 x 2 x 2 = 16 floating point operations per cycle.

Putting this all together we have:

64 cores/node x 3 node = 192 cores

Base clock speed = 2 GHz

16 double-precision floating point operations/cycle

Theoretical Peak = 192 x 2 x 16 = **6.144 TFLOPS**

HPL and Cluster Build

We built the cluster with Slurm and used Jetstream2's recommended Manila share for the shared filesystem. We self-compiled OpenBLAS, OpenMPI (and it's dependency pmix) from source with these final configurations and options:

OpenMPI 5.0.0 configuration:

```
./configure
--prefix=/mnt/hpl_manila/opt/openmpi
--with-pmix=/mnt/hpl_manila/opt/pmix
--enable-orterun-prefix-by-default
--with-slurm
```

These options are mostly standard. We specify paths for where things should build, where our source-build pmix is and set this build as the MPI we want to use, and of course include Slurm support.

OpenBLAS v0.36.26 configuration:

```
make TARGET=ZEN DYNAMIC_ARCH=1
CC="gcc -O3 -march=znver3 -mtune=znver3 -fopenmp -funroll-loops"
FC="gfortran -O3 -march=znver3 -mtune=znver3 -fopenmp -funroll-loops"
```

ZEN is the only TARGET available for any Zen architecture on OpenBLAS v0.36.26. We may try experimental branches with later versions in the future. But with our barely getting by runs as is, this was the most stable option.

DYNAMIC_ARCH=1 allows for runtime compilation with architecture considered.

We set a few compiler flags for C and Fortran; obviously -fopenmp for multithreaded code, march and mtune set to zen3 for best instruction and scheduling for our architecture. Optimization 3 is just the highest level of speed optimization without losing numerical precision, at the cost of compile time, which we don't really care about anyway. The added -funroll-loops flag is suited for SIMD instructions; it potentially unrolls loops further than -O3 already does, making it even more parallelizable for vector data.

Initial Failed Approaches on 9 nodes:

We initially used spack and specified Intel C compiler and Intel MKL libraries, but for some reason that ended up giving us numerical issues as the HPL runs would complete (and with higher GFLOPS than the OpenBLAS counterpart), but the runs would fail to validate, meaning we couldn't use it. Upon installing aocc, then AMDblis, then AMDScaLAPACK, and trying to compile HPL with openmpi, the SAT solver inside spack would fail, and would refuse to compile HPL. So, in the end, we stuck with the basic build of HPL with gcc and OpenBLAS.

At the same time, we were trying to figure out how to build a cluster, through Magic Castle. At some point we scratched our whole setup to reboot on Rocky Linux 9 to try to follow a tutorial that worked on Rocky Linux 8 and unfortunately Jetstream 1 it seems. We ran into python errors so then ended up going back to the latest Ubuntu but that burned a lot of time.

From here we rushed building OpenBLAS generically and got extremely poor performance and a very slow run which makes sense because it didn't use any of the architecture features like AVX2. Had we built OpenBLAS correctly the first time, we might have had decent FLOPS that might've had it finish in time, though still not finely tuned.

Optimization and Tuning

Final HPL.dat tunings:

```
290048 Ns
256    NBs
6      Ps
32     Qs
```

Everything else remained standard.

SLURM batch script nuances:

```
export OPENBLAS_NUM_THREADS=1

export OMP_NUM_THREADS=1

export OMPI_MCA_btl_tcp_if_include=enp1s0

export OMPI_MCA_btl=self,vader,tcp


/mnt/hpl_manila/opt/openmpi/bin/mpirun -np 192 /mnt/hpl_manila/opt/hpl/bin/xhpl
```

We were having tcp issues on initial runs that were fixed by those OMPI exports for specifying ethernet and communication types. We explicitly set num_threads as well as include the full mpi path for safety.

Process and Calculations:

N ≈ sqrt(250GB*3nodes/8bytes) ≈ 306,186

We take the square root of our total ram divided by the size of a double-precision floating point. To leave ram space for the OS, we initially used N = 290,000, then changed to 290,048, the nearest multiple of 256, our NBs. That put it at 94.7% or RAM capacity.

PxQ=192

Initially we tried P=12, Q=16. This was chosen as the most square while multiplying to 192, and it was a bonus that P was divisible by 3 (nodes). We then read that smaller Ps should be chosen for slow networks to reduce communication. Considering we are on Ethernet, we tried 3x64. This improved GFLOPs slightly. A more balanced 6x32 worked even better.

Performance Analysis

FINAL RESULT:

```
T/V                N    NB     P     Q               Time
Gflops
--------------------------------------------------------------------------------
-
```

```
WR11C2R4      290048   256     6    32            2921.59
5.5680e+03
```

We ended up at 5.5680/6.144 = 90.625% of theoretical peak. There are a few more things that could have been done. We can look into all the settings in HPL.dat, or use vendor provided modules. Considering the most square PxQ wasn't the fastest, the network, Ethernet, is definitely a bottleneck that could be preventing achieving a higher GFLOPS. We as a team also believe we should have better prepared to build a cluster from scratch and have a more organized 24 hour run, as much of our time was wasted trying new things. Inevitably, we are satisfied with our final run and have learned so much from this.

Thank you IndySCC!

Submission Requirements

2. Environment Details
    - System specifications and configuration
    - BLAS library choice and rationale
    - Compiler selection and version
    - MPI implementation used
3. Binary and Tuning Information
    - Source of binary (self-compiled vs. vendor-provided)
    - Optimization techniques applied
    - Performance tuning parameters
4. Performance Analysis
    - If performance is not within 5% of theoretical peak:
        - Identify potential bottlenecks

We scrapped the node twice so in total there were 3 clusters…

> we wonder if tasks run by other teams could significantly affect the network throughput of our cluster

Yes, this is possible; if two teams' (or just other Jetstream2 users') instances are both scheduled on the same hypervisor, they'll be splitting the 100 Gbps physical uplink to their leaf switch. We are aware that this introduces variability into network performance, and thus inconsistency in large-scale, multi-node benchmarking. This is unfortunately a limitation of using Jetstream2 outside of its primary use case that we are forced to work around.

    - Suggest areas for further investigation

    - Recommend performance improvement strategies

5. ,

Identification

- Include team name or individual names
- Specify team affiliation