HPL Hero Run Report

ShanghaiTech University, GeekPie_HPC

Cluster

Setup

In the miniVite assignment, we set up the head node and wrote some simple shell scripts to set up the cluster. However, in the HPL hero run, we had to maintain a cluster with over 300 nodes. To make things easier, we developed a Python script to execute commands on multiple nodes asynchronously.

The head node cannot connect to some compute nodes for unknown reasons. So we need to pick out the bad nodes. Thanks to the `async_ssh.py` script we developed, we could do this quickly.

We used NFS to share files between the head node and the compute nodes. Considering that there are many compute nodes, we increased the number of `threads` in `/etc/nfsd.conf` to avoid potential performance issues.

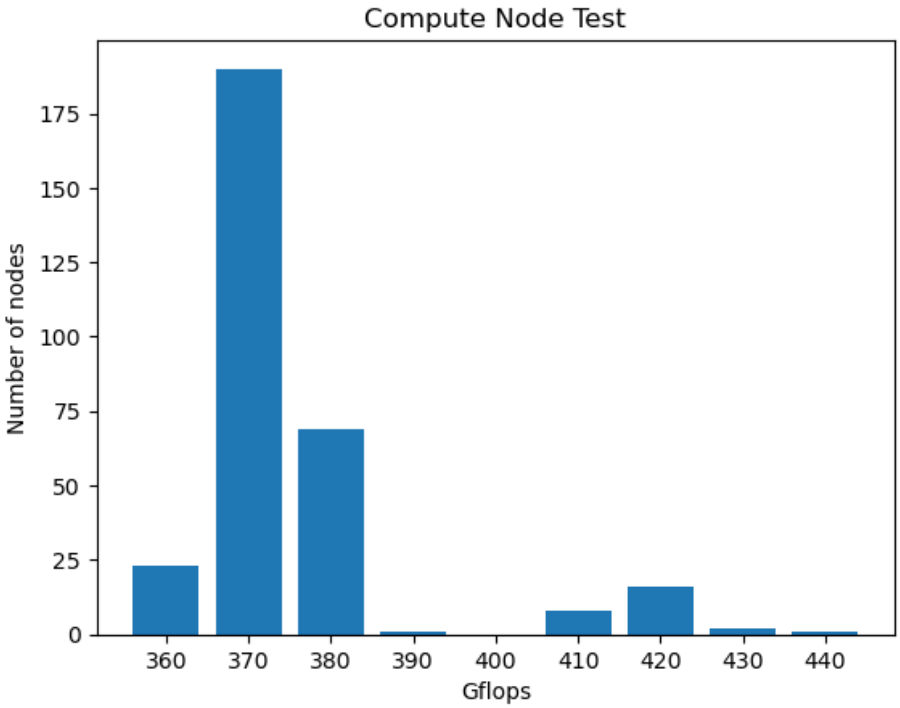
To use InfiniBand, we packed an image with the InfiniBand driver installed and launched instances with this image.

To get better performance, we turned off Hyper-threading.

Test

In the beginning, we tested the performance of each node using HPL. Here's the result:

```
mean: 380.7211612903226 std: 14.213937274553915
max: 441.0 min: 366.03
```



InfiniBand in some compute nodes is not stable. We had to pick out the bad nodes.

Software stack description

We set up a software stack containing an operating system (CentOS), HPC programming tools (GCC, Open MPI), and package management tools (Spack) to help us with the HPL Hero Running. Details are as follows.

OS

We packed an image of **CentOS 8** with InfiniBand support.

Compiler

We use **gcc-12.2.0** to compile HPL, the latest GCC version released in August 2022.

MPI

With several MPI implementations available, we compiled **openmpi-4.1.4** with `fabrics=ucx` enabled to use InfiniBand. Open MPI is an open-source Message Passing Interface implementation used as a communication protocol for parallel

and distributed computers. The version we used was released in May 2022 and is currently stable.

Spack

Spack is an incredible multi-platform package manager that can install and load multiple versions and configurations of software, enabling users to utilize up-to-date compilers and tuning options optimized for specific hardware. We use spack to build HPL 2.3 and run it with the corresponding compiler and MPI version.

HPL Tuning Steps

We started the HPL Hero Run from 16 nodes, as it will form a 4x4 grid. We tested the following variants:

- One MPI process per node (with OpenMP).
- One MPI process per node (with Intel TBB).
- 20 MPI processes per node (one process per core, bind core).

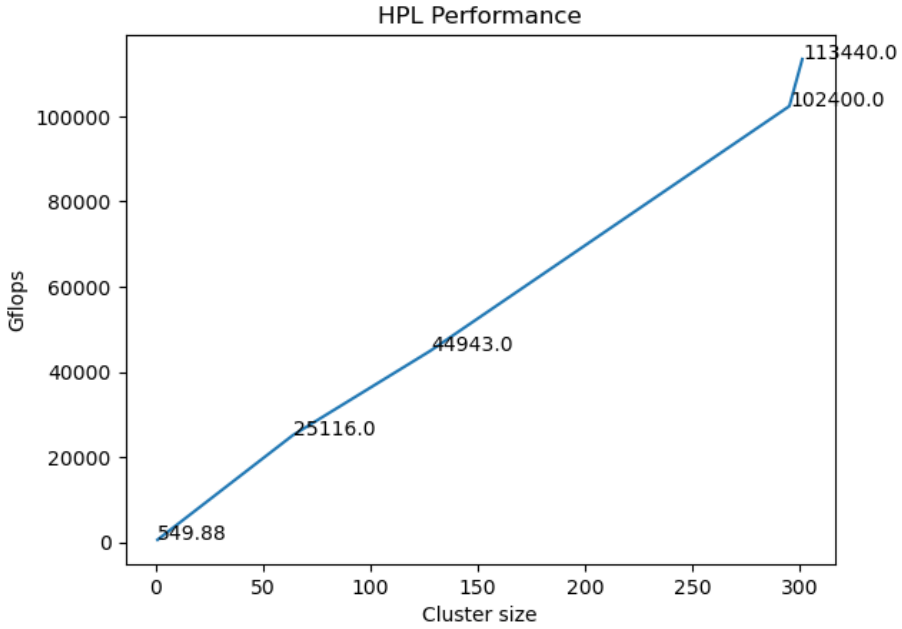
To our surprise, we found that the HPL that uses 20 MPI process per node get the maximum performance. We wanted to ensure that the policy that we run one MPI process per core is best, so we checked other options.

- Two MPI processes per node (with OpenMP, one MPI process per socket).
- Two MPI processes per node (with Intel TBB, one MPI process per socket).
- Ten MPI processes per node (with OpenMP, one MPI process per two core).
- Ten MPI process per node (with Intel TBB, one MPI process per two core).

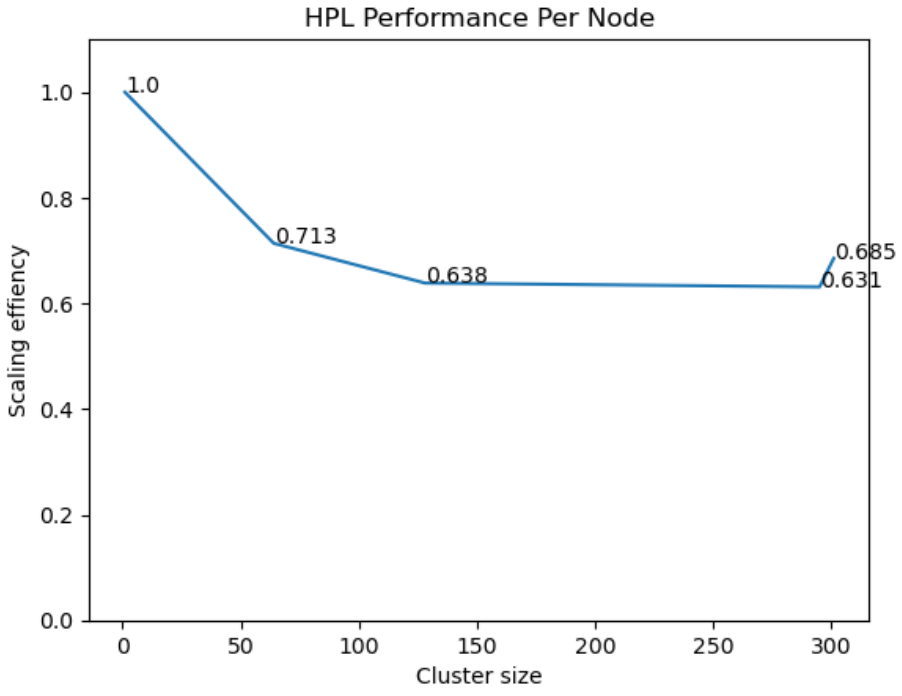
We checked one MPI process per two cores because of the ring bus design of the Intel Processor. 20 MPI processes per node still performed best, so we decided to run one MPI process per core, and the math library didn't involve any threads (no OpenMP & no Intel TBB).

Then, we scaled the nodes from 16 nodes to 64 nodes, 128 nodes, 256 nodes, and 300 nodes. The performance scaled well, basically linearly. We tried to use huge pages to accelerate, but the performance decreased.

Finally, we started to tune the HPL.dat. We followed the official manual, turned P into a power of 2, and made Q slightly bigger than Q ($P = 64, Q = 92$). We found $Q = 93$ gets lower Flops than $Q = 92$, and $Q = 94$ is much faster, but it needs slightly more nodes than 300. Then we tuned NB, we tried $NB = 16 \times 2^m \times 5^n$, where n and m were positive integers because P was a power of 2 and we had 20 processes per node, where $20 = 4 \times 5$. We tried many NBs and found $NB = 320$ is optimal. Finally, we chose N as a common multiple of NB, P and Q, and gradually increased N (problem size), which resulted in better HPL performance but longer run times.



Scaling Efficiency



As the number of nodes increases, the scaling efficiency becomes progressively lower, and the rate of decrease tapers off to about 0.63 at 300 nodes. in our experiments, an unexpectedly high scaling efficiency occurs when used a specific number of nodes, such as 301. we believe this is related to the topological of the nodes (P, Q, NB).

Challenges faced

We can't start as many instances as we want in Chameleon Cloud at the same time. And building instances may fail. It takes a long time to start instances.

During our tuning, one of the nodes went down when running HPL, and several became slow. We wrote a script to find the slow nodes and grouped the nodes by eight and ran HPL benchmark each group (8 nodes). Then we found one group that is significantly slower than other groups (about 20x slower). By this means, we narrowed the suspicious nodes from 300 to 8. We examined these 8 nodes one by one and tested the suspicious node with 7 good nodes by running HPL. Finally, we removed the slow node from our node list.