Mini-App Analysis on Polaris and ThetaGPU

Hunter Negron and Pranjal

Introduction

This semester was a continuation of the research under supervision of Dr. Zhiling Lan and Stefan Muller that started in the spring and the summer. During the first few weeks of the semester, I finished collecting data, creating charts, and writing the research paper for our submission to the conference. The rest of the semester was focused on Polaris and collecting more Mini-App data on Polaris. Particularly, most of the work was centered on getting NCU to work with Mini-App, which was a non-trivial task. The queues were long and the jobs took many hours to run, making the debugging process very slow. However, we did conclude the semester with lots of data and steps to take going forward.

Learning Polaris

For the first few weeks after submitting the summer paper, I spent quite a bit of time getting familiar with ALCF's new high performance compute system. Since the Mini-App is enabled to take advantage of both CPU and GPU processing power simultaneously, Polaris is the perfect system for analyzing its performance. There are many similarities to ThetaGPU, but it has plenty of notable differences. It is equipped with one AMD EPYC "Milan" processor and four NVIDIA A100 GPUs, compared to ThetaGPU's eight A100s [1]. Polaris uses PBS for job scheduling, but the qsub command will look very similar. The tooling on Polaris is very similar to ThetaGPU, so the Mini-App code only required minimal changes to CMakeLists.txt to configure it properly for this platform.

Here is an example of a qsub command that I used frequently to request an interactive job, which was very helpful for learning the tooling, compiling the Mini-App, and testing small runs.

```
qsub -A SEEr-Polaris -l walltime=01:00:00 -l filesystems=grand -I -q debug
```

Notice that the specified queue is "debug" instead of "single-gpu" or "full-node". One of the major differences on this system compared to ThetaGPU is the queuing structure. When submitting a job via qsub, one of three queues must be specified. These are "debug", "debug-scaling", and "prod". "Debug" will give you 1-2 nodes for a maximum of 1 hour. Since the jobs are small, it is much easier to get an interactive node here. On the other hand, "prod" is meant for long-running, multi-node, non-interactive jobs. The maximum here is 496 nodes and 24 hours. However, the "prod" queue redirects the job to one of three more queues depending on its size ("small", "medium", "large"). It is important that we understand the differences between these queues because I spent a while testing which problem sizes would fit on which of these three queues, especially when collecting data. “Small” has a maximum of 6 hours, “medium” has a maximum of 12 hours, and “large” has a maximum of 24 hours [2]. Since the Mini-App is small in terms of resources, 1 node is sufficient at any scale.

Developing a Job Script

When running larger jobs or collecting data, it was necessary for me to write a job script and submit it as a non-interactive job. The script that I used to run the Mini-App and collect data on Polaris is miniapp_polaris.sh, which can be found in the SEEr-Planning-IIT Google drive [3]. The PBS parameters are placed at the top. The “walltime” parameter needs to be set to 24 hours to fully handle size S Nsight Compute data collection, so this must go to the large queue. The “select” option specifies the number of nodes. Even though we only need 1, the minimum for the large queue is 50. The “place” option is set to “exclhost” so that the Mini-App has exclusive control over the memory of the nodes that it is running on. That is, it controls all the CPUs and all the GPUs. This is to ensure that the data collection process is not impeded by resource sharing. After that, the script loads and activates the Conda module and navigates to the preferred problem size. The last command actually calls Nsight Compute (NCU) with the list of desired metrics. We run this with the “time” command so that we can have a rough estimate of the wallclock runtime of the data collection at the end. Here’s an example of how I submit the job on Polaris: `qsub -q prod ./miniapp_polaris.sh`

Data Collection

We are interested in the wall clock runtime of the data collection because NCU is known for incurring a massive overhead. For example, a typical uninstrumented run of the size A Mini-App takes about 45 seconds. When collecting a single metric using NCU, the runtime is now over 38 minutes. For Mini-App size S, an uninstrumented run is only a few minutes, whereas NCU data collection typically falls between 18 and 20 hours (1100 and 1200 minutes). The highly compressed ncu-rep data files were all between 6 and 7 GB and can be found in the “miniapp polaris data” folder in the SEEr-Planning-IIT Google drive. Details for all the data collection runs, including Polaris and ThetaGPU, sizes, metrics, runtimes, and parameters, are logged in the Google sheet titled “sheet for miniapp”, which is also in the drive [3].

One difference that you may notice between this new job script and the job script for summer data collection on ThetaGPU is the absence of the Python virtual environment. Upon further inspection, the Conda module provides all the libraries that the Mini-App needs, so this portion of the SDL AI Workshop tutorial from which the Mini-App originates is redundant, so for uniformity we no longer load it when running the Mini-App. A second difference is that I am no longer collecting data using Mantis. It took quite a while to configure Mantis to handle the demands of profiling the Mini-App, so I opted to move straight into data collection manually. In the end it seems this route is preferable, given that we would only be able to collect data on one size at a time anyway because of the overhead.

Issues Experienced

In the early time of testing NCU data collection on Polaris, I experienced an issue where the program would hang and not do anything. This issue was soon resolved, but we are uncertain whether this was because of the virtual environment and/or a versioning issue, or whether it was a recent bug fix in NCU. However it happened, we moved on to collecting actual data soon after.

Another small problem experienced with Polaris and ThetaGPU was with the limitations of the filesystem. The home directory has a quota smaller than what I required to profile the Mini-App, so have several failed attempts to debug, I moved all my code and run scripts into the project directory located at /lus/grand/projects/SEEr-Polaris/. Also, our ThetaGPU ALCF project, SEEr-planning, expired, so I was blocked from writing to any files in that project directory. Therefore, after several more failed attempts, I moved all the ThetaGPU code and scripts into the SEEr-Polaris project directory, too. We have also been the victim of a server fault roughly halfway through the semester, which caused several days of delays.

Lastly, I was getting issues with collecting multiple metrics at once. It would just crash nearly immediately on ThetaGPU. I edited the final line of the ThetaGPU job script (which is nearly identical to the the Polaris script but configured for ThetaGPU, and can be found in the Google drive as miniapp_thetagpu_nov_30.sh) to include the NCU parameter "--sampling-max-passes n", where n is the number of metrics I would like to collect at once. It worked fine after doing this.

Metrics

Pranjal and I split the work of collecting Mini-App data on Polaris and ThetaGPU. You'll notice Dr. Muller's document "CUDA cost" lists the many NCU metrics that we are interested in. In the miniapp_polaris.sh job script, these are labeled A-M, respectively (the last two, N and O, were excluded later on). We collected these for sizes C and S, the two largest sizes that would complete in a manageable amount of time.

Although we did not do this for every single metric of every single size, once the ncu-rep data file is properly collected, this command can be used to convert it to a CSV file: ncu --import <filename>.ncu-rep --csv > <filename>.csv. I hesitated to do this for all the data at first since the CSVs would be quite large and take a lot of time to convert. Therefore, I prioritized the data collection first.

Problem Sizes

Important to note is how the problem size scales. Sizes A, B, and C all have DT=0.001. Once we get to size S, this is changed to DT=0.0001. You'll notice in the "sheet for miniapp" that the wall clock times for NCU profiling on these three sizes are very similar. Once we get to size S does this statistic drastically change. I suspect that change is related to how the problem sizes are determined. The original six sizes were established with the wisdom of Dr. Maulik, who understands the domain of the problem that the Mini-App solves. To get a clear picture of what's going on here, it may be worthwhile to once again consult Dr. Maulik and ask for clarification or a new set of problem sizes.

Known Issue with Mini-App

I should also reiterate that on a normal run of the Mini-App, it will fail with an ABORT due to a double-free after program exit. This is expected and does not affect the actual work that the Mini-App performs. The reason for this is the Horovod functionality that was added to the Mini-App at the beginning of the summer. In particular, the horovod.init() that is called at the beginning of the ml_module.py script is the source. According to Hannah, the Horovod functionality doesn't actually do anything since the C++ code is not equipped to handle multi-GPU run cases. Therefore we can only run it on a single GPU anyway. Therefore, it is likely that reverting to the pre-Horovod version of the Mini-App will perform exactly the same as it does right now. This restoration can be done by navigating to the app_build subdirectory and copying ml_module.py.original to ml_module.py.

Future Work

Now that all the data collection is done, there are several steps that we can take from here. Firstly, the data collected should be analyzed and organized into charts so that Dr. Muller can apply it to his research on static analysis. Also, it would be beneficial to have a conversation about Dr. Maulik about possibly picking different, more manageable problem sizes. No data has been collected for sizes M or L since those are much larger than S and will surely take more than the maximum time of 24 to collect NCU data for. Since the Mini-App uses TensorFlow to perform the machine learning, the thorough profiling of NCU must profile TensorFlow itself, which is quite a large library. Also, NSYS power and timing data should be collected for the Mini-App on Polaris to complete the data collected on ThetaGPU over the summer.

The details about the Mini-App, how to set up its environment and how to run it and collect data on it are all written down in a document called “Mini-App on Polaris”, which can be found in the Google drive [5]. Only minor adjustments need to be made to make this work on ThetaGPU as well. Although the original SDL AI Workshop already makes these details explicit. All the code for the Mini-App as well as job scripts can be found in the Github repository [4].

Action Items

- Mini-App Polaris power and timing data using NSYS, sizes A,B,C,S
    - Need Melanie’s help
    - The data in the Google Drive is probably not good data; needs more attention

References

1. Polaris information: https://www.alcf.anl.gov/polaris
2. Polaris Queueing: https://www.alcf.anl.gov/support/user-guides/polaris/queueing-and-running-jobs/job-and-queue-scheduling/index.html#queues
3. SEEr-Planning-IIT Google Drive: https://drive.google.com/drive/folders/0AIk-MUMFdOrEUk9PVA
4. SEEr Github Repository: https://github.com/SPEAR-IIT/SEEr
5. Mini-App on Polaris Document: https://docs.google.com/document/d/1JhYndjMpK3J9Ph2DMNtBe2uBfpNgPq0Yk2yx8SGeDYA/edit?usp=sharing