Aaron Duran
DREU project
Final report
Mentor: Dr. Hyesoon Kim
August 2020

High Performance Computing's Impact on COVID-19 Response

Goals of this project.

The original goal for this project was to investigate the uses for High Performance Computing (HPC) in medicine and to apply what was found to aiding in COVID-19 research. Particular interest was shown in how various social interactions influence the spread of COVID-19 and how they will continue to impact the disease.

As existing research was found and time was spent attempting to run their repositories, the timeline of the project drew closer and it was decided to change to scope of this summer project. The refined goal for the rest of the project became to investigate how computers and HPC have impacted how humanity responds to crises, and using the global COVID-19 pandemic as a case study. Interest was placed on how modern computing power has shaped humanity's approach to problems and development of their solutions.

A secondary goal for this project was to analyze the performance of the methods utilized in finding these solutions to determine what components are the limiting factors. This information could potentially be used to direct what future research pursues in order to better prepare humanity for future crises.

Background and prior work

As a result of the global COVID-19 pandemic, countless people are working tirelessly to combat the virus. This is still true In the field of Computer Science, especially among industry leaders, government, and researchers.

The COVID-19 High Performance Computing Consortium is a collection of technology companies, government agencies, and academic institutions pooling their computation and consulting resources to aid researchers in accelerating their work to combat the global COVID-19 pandemic [1].

A lot of interest in this field has been concerning the use of neural networks. Using the molecular structure of millions of known molecules, a neural network is able to identify the shapes that molecules and bonds will form with relative certainty. Using this information the network is able to generate new molecules that could potentially be synthesized for use as drugs. These drugs can then be analyzed to see if they might be effective against a given disease such as COVID-19 [2].

The power of High performance computing can be leveraged to analyze complicated problems relatively quickly. One such program is PyRX, which can be used to analyze how well a drug molecule can bind to a biomolecule [3], this is particularly useful in trying to find molecules and drugs that could combat a disease. When combined with a neural network, thousands of molecules can be evaluated to determine clinical trial candidates.

One open source program was found that made use of both the described neural networks techniques as well as PyRX. This program was developed by Matt O'Connor, a data scientist and startup founder. His program was able to identify Remdesivir as a potentially good treatment for COVID-19 [2]. Today Remdesivir is still undergoing clinical trials, but is showing hopeful results in patients and experiments [4].

Project Process

After defining the original goal for the project, weeks were spent searching the internet for various projects concerning the current COVID-19 pandemic and how computers are being used to model the disease. During this time all found projects were lightly investigated and categorized to determine their usefulness, as well as attempting to find their source code if they were openly available. Initial focus was placed on projects that attempted to model the spread of the disease and how different social behaviors affect these models.

While investigating HPC projects that have impacted the field of medicine, DeepMind's AlphaFold was found to be a major stride in showing how HPC and Artificial Intelligence (AI) could be leveraged to solve problems outside of computer science. While source code for the original program could not be found, a competition to remade AlphaFold as an open source software was discovered [5]. The winner of this competition, Eric

Alcaide, and his repository *MiniFold*, were found and investigated for potential later use in the project [6].

Another, more recent, open-source competition was identified [7]. This competition was primarily focused on prompting the discovery of drugs that had the potential to effectively treat patients with COVID-19. The top three submissions were all looked at to see if they could benefit the goal of this project [2], [8], [9].

With several open-source repositories relating to this project being found. The focus of the project was shifting from finding information to analyzing and using what was discovered. In order to accomplish this, all found repositories were cloned onto a lightly used Georgia Institute of Technology server named Boson. The documentation for these repositories was followed in an attempt to run the programs and verify their results. However, through a combination of less than ideal documentation, numerous errors, and needing to communicate with the Boson administrator to install/update/downgrade several required modules and programs, it took a few weeks to get some of the repositories working.

With time starting to run out to finish this research project, it was decided to reduce the scope and redefine the goal of the project. The new goal for the remaining time of research became to investigate how computers and HPC have impacted humanity's response to the global COVID-19 pandemic. Part of this new goal was to measure how the various systems of HPC are being utilized by the found repositories.

The remainder of the time allocated for this research project was dedicated to collecting profiling data of the repositories that were running. This was accomplished by utilizing the perf command for CPU monitoring and the nvprof command for GPU metrics. These gathered metrics would then be analyzed and evaluated to determine what systems were bounding the programs. This information could then be used to further optimize said programs and/or invest in improving the responsible hardware and hardware interfaces.

Due to a few errors and the large amount of time it took to profile the GPU, focus was placed on only the most promising of the cloned repositories. This repository was for Matt O'Conner's *Deep Learning Coronavirus Cure* program [2]; this program was chosen because it was one that revealed that the existing drug Remdisiver was a highly potential treatment for COVID-19, which has science been undergoing clinical trials and is still showing promise [4]. The profiling data collected from the *Deep Learning Coronavirus Cure* program is analyzed in the Results and Analysis section of this report

The last few days of the summer research term was spent preparing for someone else to take over this research topic and constructing this report.

Results and Analysis

The full files produced by the profiling of the *Deep Learning COVID Coronavirus* program can be found on this project's website at https://amduran2017.github.io/**.**

Due to not having weeks of compute time. All profiling was done on a scaled down single generation of the *Evaluation and Refinement* jupyter notebook provided in Matt O'Connor's original repository. The steps to repeat these profiles can be found in the Appendix of this report.

The hardware specifications of the Boson server are as follows
CPU: Intel Core i7-4820K CPU @ 3.70GHz
RAM: 8 x 8GB DDR3 @ 1333 MT/s
GPU: NVIDIA GeForce GTX TITAN X

This remainder of this section will analyze the metrics generated by above mentioned CPU and GPU profiling.

GPU profiling

The GPU multiprocess usage typically idles around 1.5%, and maxes out at 82%. However for most of the time the GPU is in use, the multiprocess usage is in between 40% and 70%. The Instructions per Cycle (IPC) of the GPU was between 0.5 - 1.8 IPC under typical load while it peaked at 3.6 IPC when under the maximum load provided by the program. This low Multiprocess utilization and relatively low IPC metrics indicate that the parts of *Deep Learning Coronavirus Cure* program that utilize the GPU are not GPU compute bound.

The GPU metrics also reveal that this program makes heavy use of memory. When in use, the Global Load, Global Store, Device Memory Read, Device Memory Write, Local Memory Load, Local Memory Store, Shared Memory Load, and Shared Memory Store each individually have throughputs ranging from as 20GB/s to upwards of 190GB/s. Out of the listed memory types, the most utilized memory is the device memory. Other metrics revealed by nvprof are the percentages of stalls that were caused by various issues. The three most common reasons for stalls were instruction fetches, data

requests from memory, and immediate/constant requests from memory (these three issues often accounted for more than 70% of the stalls created during the execution of the program). With the above mentioned metrics it is highly likely that this program is memory bound in its current form.

CPU profiling (when GPU is in use)

When running the GPU alongside the CPU, the CPU has an average of 0.75 IPC. In this mode of running, the CPU misses less than 2.1% of its cache references. For about 71% of its fronted cycles, the CPU was idle. Overall, generating 100 molecules, selecting unique and valid molecules, and adding these selected molecules to the model took 567 seconds to run 2.0 trillion instructions over 2.7 trillion cycles.

CPU profiling (Cpu only)

When disabling the GPU and only allowing the program to run on the CPU, the CPU has an average of 0.52 IPC. In this mode of running, the CPU misses about than 22.4% of its cache references. For about 82% of its fronted cycles, the CPU was idle. Overall, generating 100 molecules, selecting unique and valid molecules, and adding these selected molecules to the model took 376 seconds to run 2.5 trillion instructions over 4.9 trillion cycles.

CPU vs CPU + GPU

Comparing the CPU performance of running this program with and without a GPU realvers some rather interesting results. When running without the GPU the program is a lot less efficient in most gathered metrics. The only notable metric where this was not the case was in the time it took to execute the program, when running without the GPU, the program was actually faster, this could be because GPU multiprocessing may be interacting badly with how tensorflow is being used within this program. Another cause could be that with such a relatively short test, the typical time benefits of running on a GPU were unable to manifest themselves, this might be remedied by running full length tests across multiple data generations. This process would take several hours each on the current hardware installed on Boson. However this slower execution time does not subtract from the fact that the CPU ran less instructions and had a much better cache hit ratio when working with the GPU.

Accomplishments and Reflections

Extensive investigation has been conducted into what people have done with HPC and computers during this trying time in the world. What has been found so far is that companies, governments, and individuals alike have all invested heavily into HPC in order to better combat COVID-19.

Several open-source repositories from various individuals and organizations have been found and analyzed to find out what systems are in place to take advantage of modern computing power. What was found was that most of these programs utilized AI training and predictions to accomplish their goals, from modeling proteins structures [6], to discovering drugs to combat the virus [2]. As a result of the CPU and GPU profiling done during this project, it can be concluded that leveraging HPC computing can be very effective at improving the processing efficiency of programs, assuming they are properly optimized and are designed around multiprocessing.

As a result of these projects, humanity is already finding ways to combat this terrible disease and potentially even cure it or vaccinate against it. These are typically tasks that take years or decades to accomplish, and with the power of high performance computing, people are rapidly accelerating when these treatments could be made available.

Future work that can be done

The research started here can be greatly expanded upon in the future. The remaining open source projects should all be profiled and analysed. This gathered data could then be compared to the results stated here to see if there is a trend in what systems are being utilized the most and making performance recommendations to their original authors (or simply contributing improvements to the projects directly). In addition, the repositories could be further analysed to the point of taking some of their features to develop or improve a program that fits the original goal of this project. If profiling was done on a computer server updated with the latest hardware, the gathered profiling data could also be used to spur further focused development in associated hardware technology and or optimization techniques to help humanity be better prepared for the next global crisis.

References

[1] COVID-19 High Performance Computing Consortium. *Who We Are,* COVID-19 HPC Consortium. Apr. 2020. Accessed on May 19, 2020. [Online]. Available: https://covid19-hpc-consortium.org/who-we-are

[2] M. O'Connor. *Deep Learning Coronavirus Cure,* GItHub. Mar. 2, 2020. Accessed on June 8, 2020. [Online]. Available: https://github.com/mattroconnor/deep_learning_coronavirus_cure

[3] S. Dallakyan. *PyRX*. SourceForge. Accessed on June 15, 2020. [Online]. Available: https://github.com/mattroconnor/deep_learning_coronavirus_cure

[4] U.S. Food and Drug Administration. *Fact Sheet for Health Care Providers Emergency Use Authorization (EUA) OF Veklury (Remdesivir),* U.S. Food and Drug Administration. p. 9. July 2020. [Online]. Accessessed June 31, 2020. Available: https://www.fda.gov/media/137566/download

[5] S Raval. *DeepMind AlphaFold,* YouTube. Jan. 15, 2019. Accessed on June 3, 2020. [Online]. Available: https://www.youtube.com/watch?v=cw6_OP5An8s

[6] E. Alcaide. *Minifold,* GitHub. May 10, 2020. Accessed on June 5, 2020. [Online]. Available: https://github.com/EricAlcaide/MiniFold

[7] S Raval. *Coronavirus Competition Results (Remdesivir),* YouTube. Mar. 7, 2020. Accessed on June 8, 2020. [Online]. Available: https://www.youtube.com/watch?v=EVoZMRmtBkY

[8] T. MacDougall. *2019-nCov,* GitHub. Mar 2, 2020. Accessed on June 8, 2020. [Online]. Available: https://github.com/tmacdou4/2019-nCov

[9] T Vidovic. COVID *competition,* GitHub. Mar 23, 2020. Accessed on June 8, 2020. [Online]. Available: https://github.com/tinkavidovic/competition

Appendix: Steps to Recreate Experiment

Install following software:

Python 3.7.7
Pip
CUDA 10.1 (this is the only version compatible with the pip package of tensorflow 2.2 as of the time of writing)
cudNN 7.6.5
perf
nvprof
git
Miniconda

Setup:

1. Create environment
   conda create -c rdkit -n environmentName rdkit python=3.7
2. Activate virtual environment
   conda activate environmentName
3. Clone original repo:
   git clone
   https://github.com/mattroconnor/deep_learning_coronavirus_cure.git
4. Get requirements
   wget -O requirements.txt
   https://raw.githubusercontent.com/AMDuran2017/AMDuran2017.github.io/master/files/deepLearnCovidCure_updatedRequirements.txt
   python -m pip install -r requirements.txt
5. Download molecule data
   The original repo says it used both MOSES and ChEMBL datasets, however for the testing in this project, only the MOSES dataset was used because the requested ChEMBL csv file was not easily found.
   wget -O ./datasets/moses_dataset_v1.txt
   https://media.githubusercontent.com/media/molecularsets/moses/master/data/dataset_v1.csv
6. If not using the ChEMBL dataset, remove/comment out all of cell 3 and cell 6. Also change the fist line in cell 8 to the following:
   smiles = moses_smiles
7. Run all cells in data prep notebook
8. Close jupyter and run the cleaning script

python cleanup_smiles.py ./datasets/all_smiles.smi
./datasets/all_smiles_clean.smi

9. Open Evaluation and Refinement jupyter notebook
10. Add this line to cell 1:
    from rdkit.Chem import Descriptors
11. Change Global generation to *2* (can be found in cell 16)
12. In the cell 40 (sixth cell from the bottom), change sample number from *5000* to *100* (this is so that way the program will finish in than 12 minutes instead of several hours)
13. To automatically terminate the python process once the notebook finishes, add a cell at the end of the notebook that contains these lines
    %%javascript
    Jupyter.notebook.session.delete();
14. Run the entire notebook
15. In cell 27, make sure that *print(tensorflow.test.is_gpu_available())* prints "True"
16. Make sure notebook runs all included cells

Running Experiments:

Cpu+Gpu: no setup changes

CPU only: before the notebook imports tensorflow (cell 27) insert a cell (recommend putting this at the top of cell 1) with these lines:

import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

GPU profiling:

1. Start the jupyter notebook server, open up the "Evaluation and Refinement" jupyter notebook
2. In a second terminal, run nvprof, and allow it to profile all new processes with this command:
   nvprof --profile-all-processes --log-file log%p.log --metrics alll
   a. You can change the metrics measured by listing the ones you want instead of using "all" (this is highly recommended as it appears that for each metric that is being measured, nvprof has to run a copy of the active kernel to gather the metric data from)
3. In the "Evaluation and Refinement" notebook restart the kernel (kills existing python process and starts a new one) and then run all the cells in the notebook
4. Once the python process is killed/terminates the results with be output

5. Make sure to kill the nvprof process after it has finished writing the file for the python process
6. The results gathered are the profiling metrics of one generation of smile refinement

Cpu profiling:

1. Start the jupyter notebook server, open up the “Evaluation and Refinement” jupyter notebook (skip this if attempting to profile the GPU at the same time)
2. Run *top -u <your username>* in a second terminal, copy the PID of the python process
3. Run perf with the -e flag followed by all the events to be monitored and the -p flag followed be the copied PID. The command that was used in this report’s experiments is as follows:

    perf stat -e

    branch-instructions,branch-misses,bus-cycles,cache-misses,cache-references,cycles,instructions,ref-cycles,idle-cycles-frontend,topdown-fetch-bubbles,topdown-recovery-bubbles,topdown-slots-issued,topdown-slots-retired,L1-dcache-load-misses,L1-dcache-loads,L1-dcache-prefetch-misses,L1-dcache-store-misses,L1-dcache-stores,L1-icache-load-misses,LLC-load-misses,LLC-loads,LLC-prefetch-misses,LLC-prefetches,LLC-store-misses,LLC-stores,branch-load-misses,branch-loads,dTLB-load-misses,dTLB-loads,dTLB-store-misses,dTLB-stores,iTLB-load-misses,iTLB-loads,node-load-misses,node-loads,node-prefetch-misses,node-prefetches,node-store-misses,node-stores -p 17539

    Where 17539 was the PID of the python process

4. Once the python process is killed/terminates the results with be output