Performance Analysis of a Parallelized Restricted Boltzmann Machine Artificial Neural Network Using OpenACC Framework and TAU Profiling System

ABHISHEK KUMAR, The Wheatley School

Restricted Boltzmann Machines are stochastic neural networks that create probability distribution based off connection weight between nodes of the hidden and visible layer. The distribution makes the program optimal at classifying large amounts of data, which could be useful in work settings, such as a research lab. The parallelization of these neural networks would allow for the classification of data at a much faster rate than before. Using a high-performance computer it was determined that parallelizing the neural networks could decrease the runtime of the algorithm by over 35% when offloading the work to a GPU through OpenACC. Using Tuning and Analysis Utilities Profiling Systems, it was found that scheduling the program would only be effective if the data size was large enough and an increase in the number of thread blocks used for scheduling would allow for greater performance gains than the number of threads in each thread block.

CCS Concepts: • **Computing methodologies → Parallel programming languages**; *Feature selection*; • **Computer systems organization →** *Multicore architectures*;

Additional Key Words and Phrases: Restricted boltzmann machine, performance analysis, parallelization, neural network, openacc

ACM Reference format:
Abhishek Kumar. 2017. Performance Analysis of a Parallelized Restricted Boltzmann Machine Artificial Neural Network Using OpenACC Framework and TAU Profiling System. 1, 1, Article 1 (October 2017), 2 pages.
https://doi.org/0000001.0000001_2

1 INTRODUCTION

A RBM is a generative neural network that uses probability distributions to learn and complete whatever task is given to it, whether it be classification, feature learning, regression computing, etc.[Tra 2007; Elfwing et al. 2014] OpenACC is a directive-based programming framework used for the parallelization of sequential code on GPUs.[Ope 2007] Applying OpenACC to RBMs would allow for faster classification that would be useful for researchers, as they need to organize large sums of data quickly.

2 RESTRICTED BOLTZMANN MACHINES, OPENACC, AND TAU

2.1 Restricted Boltzmann Machines(RBM)

RBMs are used for the classification of data across multiple fields. They are commonly seen sorting data at labs, hospitals, and other locations with large sums of information. They are based off an âĂIJenergy functionâĂİ which is dependent on connection weight. For the purpose of keeping this research focused on the increase in performance, we decided to have the RBM have a binary input.

© 2017 Association for Computing Machinery.
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in , https://doi.org/0000001.0000001_2.

2.2 OpenACC and Tuning and Analysis Utilities(TAU)

OpenACC is an API that simplifies CUDA and allows parallelization to be done similarly to OpenMP. It allows the programmer to control the number of threads used, to use reduction calculations, control the types of loops used, etc.

TAU is a profiling system that shows how execution time is spent by deploying probes into a program and tracking the time of commands. With Paraprof, it can be visualised into 2- or 3-d graphs, allowing for greater overall comprehension.

3 EXPERIMENTATION

3.1 Environment

For the testing, we used the HPC1 at Brookhaven National Labs, which uses Intel E5-2670s, allowing for 16 threads at 2.594 GHz. The system uses 128 Gb of RAM and has an Nvidia Tesla K20Xm GPU. The GPU has 6.04 Gb of memory and 1024 threads[HPC [n. d.]]. I used the PGI Compiler and MIT Free License RBM.

The data presented will be the average of 50 runs for each data point. In order to prevent optimization due to saved runs, we removed all data saved after each run.The inputs tested were a 200x2, 2000x2, and 20000x2 binary matrix. Each point on Figures 1-3 represents a different configuration of the matrix input. At x=0, the matrix only has 0s. As the iteration increases, a 0 becomes a 1.

3.2 Results

The RBM took an average of .096 seconds for an input of 200x2. Running the code in parallel decreased the time taken by 22.085%. Using the reduction function, it further decreased the time taken by the program by .008 seconds. Using scheduling, the neural network took over 150% of the sequential time to run, and at best, it was still less efficient.

When running the 2000x2 matrix sequentially, it was found to average .93 seconds. Using solely OpenACC, I saw time spent decrease to approximately .72 seconds, allowing for a 23% decrease. Adding the reduction function, time further decreased to about .7 seconds, or a decrease of almost 25%. We can now see how reduction allows for a significant drop in time, having it cause a 2% decrease. However, some of that decrease may be attributed to the spike in time towards the end of the testing. It initially assumed that this was due to another user on the cluster, but when checking if anyone else was using that node, I found that we were the only ones using it. After using TAU for further investigation, it was found that overhead calculations were being done, which increased time. Scheduling with this matrix input put results between that of sequential and parallelization. It took anywhere from .830 to .800 seconds per run.

The 20000x2 matrix showed how OpenACC would affect performance the most drastically. The sequential execution time took

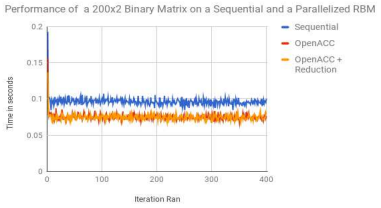

Fig. 1. Time Required to Run a Sequential and Parallelized RBM for a 200x2 Input

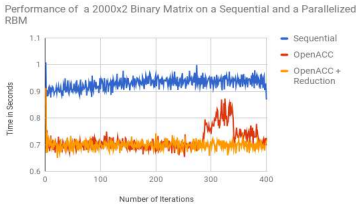

Fig. 2. Time Required to Run a Sequential and Parallelized RBM for a 2000x2 Input

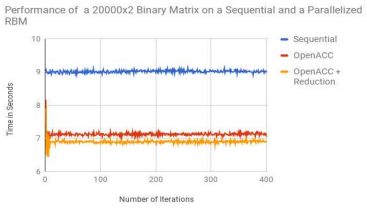

Fig. 3. Time Required to Run a Sequential and Parallelized RBM for a 20000x2 Input

9.013 seconds and parallelized without reduction took 7.126 seconds. Using reduction, there was a drop by 2.5%. Scheduling, at its worst significantly outperformed the parallelization, taking a mere 6.073 seconds. At its best, the algorithm took 5.776 seconds, which is a decrease by over 35

When analyzing the scheduling through TAU, I determined that unused threads were being used for the repetition of code. The 200x2 and 2000x2 matrix were able to be done without using all threads, however, unused threads were redoing calculations. Because the 20000x2 matrix utilized all threads, there were no loops, allowing for the significant performance gains.

4 CONCLUSION AND FUTURE WORKS

Applying OpenACC to the neural network has allowed for drops in runtime by up to 35%. The results found show that as data input size increases, the effects of scheduling and reduction becomes more apparent. This is because of the overhead calculations done by unused threads. Beyond that, an increase in thread blocks is more significant than an increase in threads per block.

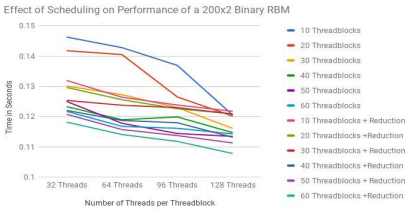

Fig. 4. Time Required to Run a Sequential and Parallelized RBM for a 200x2 Input

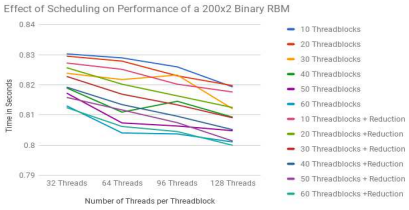

Fig. 5. Time Required to Run a Sequential and Parallelized RBM for a 200x2 Input

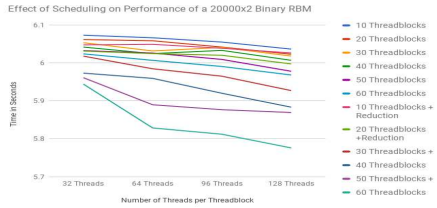

Fig. 6. Time Required to Run a Sequential and Parallelized RBM for a 20000x2 Input

In the future, research analyzing nonbinary inputs or Boltzmann Machines could lead to greater strides in the research. Analyzing how deep learning is impacted is another area that could prove to be useful.

ACKNOWLEDGMENTS

The author would like to thank Dr. Abid Malik of Brookhaven National Labs for assisting and guiding us through our research.

REFERENCES

[n. d.]. High Performance Computing Cluster. ([n. d.]).
2007. OpenACC. (2007). Retrieved August 7, 2017 from https://www.openacc.org/
2007. Training of Restricted Boltzmann Machines. (June 2007). Retrieved August 7, 2017 from http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DBNPseudoCode
2015. *OpenACC Programming and Best Practices Guide.* OpenACC.
S. Elfwing, E. Uchibe, and K. Doya. 2014. Expected energy-based restricted Boltzmann machine for classification. *Neural Networks* 64 (September 2014), 29–38. https://doi.org/10.1016/2014.09.006
Bernd. Mohr, Allen D. Malony, and Janice E. Curry. 1985. Tau - Tuning and Analysis Utilities for Portable Parallel Programming (1995). *ACM Trans. Program. Lang. Syst.* 7, 3 (July 1985), 359–379.