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

By: Abhishek Kumar

Abstract

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.

Introduction

- Restricted Boltzmann Machines are artificial neural networks that create probability distributions based off inputs
  - The probability distribution makes the algorithm optimal for classification of large sums of data
    - This classification application makes the neural network useful for research labs and hospitals
  - Other applications of the programs include generating new data based off the input

- Data is getting larger and larger with each passing decade
  - Necessary to find ways to compensate for the larger data when dealing with program runtime
    - Offloading the computations to a GPU can significantly increase performance
    - OpenACC provides a solution to this dilemma by allowing for the parallelization of algorithms written in C, C++, or Fortran onto a GPU
      - Allows for a simpler GPU parallelization API relative to CUDA, due to its directive based controls
  - Parallelization would allow for a greater power budget to those working on high performance computers, such as Titan or Blue Gene
    - A smaller runtime would increase the opportunities researchers have with the powerful pieces of technology
      - Would allow for greater strides in research

OpenACC and TAU

- OpenACC is a directive-based parallelization programming model used to run programs in C, C++, and Fortran on GPUs
  - Functions of OpenACC tested include:
    - Scheduling is an important function in the API that allows for the programmer to control the number of threads used
    - `reduction` is a command in the library that divides the work for a looped computation and then sums each of the divisions at the end of the loop
    - `kernel` is another method that can be used to identify if a loop can be parallelized and, if it can be parallelized, parallelize it
  - This can be used instead of CUDA in order to parallelize code in fewer lines and with more ease
  - OpenACC was created by Nvidia and is exclusive to the PGI Compiler

- TAU, or Tuning and Analysis Utilities, is a general computational model for parallel programs used as a compiler for C and Fortran code in order to show how the execution time is spent
  - It is able to create profiles through the use of probes in the program, which reads the program as individual events that can be timed and the characteristics of the run can be shown
  - Using ParaProf, the events can be shown in a graphic representation

Experimentation

A. Environment

- The high performance computer used was the HPC1 at Brookhaven National Labs
  - Each node contained two Intel E5-2670 processors.
  - The GPU in the cluster was the Nvidia Tesla K20Xm, which has 6.04 Gb of RAM and 1024 threads
- Software
  - The PGI Compiler was used for compilation of the program
  - The Restricted Boltzmann Machine was provided by the Open Source Initiative under the MIT Free License
    - It was written in C and was not parallelized when found

B. Methodology

- Testing was done using the 3 binary matrices as different input sizes
  - 200x2, 2,000x2, 20,000x2 matrices
  - Using binary restricted the number of influential variable
- The first group of tests were ran in order to determine the sequential times
  - Each matrix was tested in 400 different configurations and the values found for each configuration was based off the average of 50 runs
- The same tests were ran with the program now parallelized
- The effects of scheduling was then tested
  - Testing of scheduling included the usage of 32, 64, 96, and 128 threads per block
  - The number of thread blocks used in experimentation were 10, 20, 30, 40, 50, and 60 blocks
- All parallelized testing was run twice, once using `reduction` and another time not using the function
- TAU was then used to determine how anomalies in the data occurred

C. Results

- 200x2
  - Sequential averaged .096 seconds, which is .021 more seconds than that of using only OpenACC
  - Using `reduction` decreased the time by .870%
  - Using scheduling increased time
    - At worst it took .146 seconds, which is over 150% of the sequential
    - At best, scheduling took .018 more seconds than sequential
- 2,000x2
  - The sequential runtime took .930 seconds and parallelized code without `reduction` averaged .718 seconds
  - Using `reduction`, it took .699 seconds, allowing for 1.96% decrease
  - Using scheduling took, at best, .830 seconds, and, at worst, .800 seconds
    - This is better than sequential, however, it is still worse than that of the parallelization times
- 20,000x2
  - Sequential times averaged 9.013 seconds, with it's parallelized counterpart averaging 7.126 seconds
    - Including `reduction` further decreased it to 6.898 seconds, or a decrease of over 2.5%
  - Using scheduling finally increased efficiency
    - At its worst, scheduling took 6.073 seconds, and at best, it took 5.776 seconds, which is less than 35% of the sequential time
- TAU
  - It showed that the anomaly towards the end of Figure 2 was due to overhead calculations
  - TAU then showed that scheduling became more effective as size increased because work was being done again on threads that were supposed to be unused
    - As size increased, there was less threads that would loop work, which allowed for the increased performance as size of the matrix increased

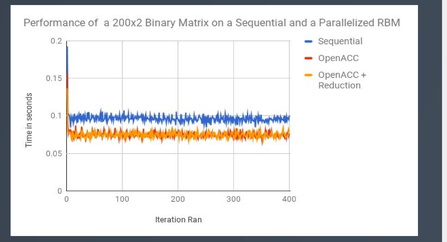


Figure 1

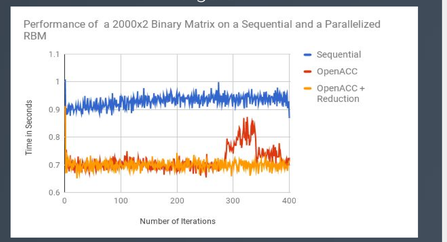


Figure 2

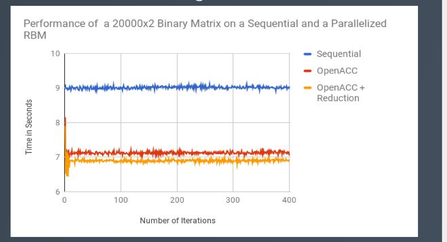


Figure 3

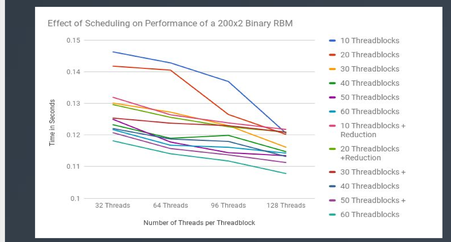


Figure 4

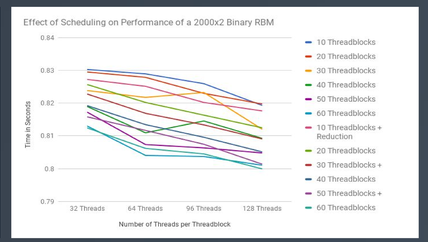


Figure 5

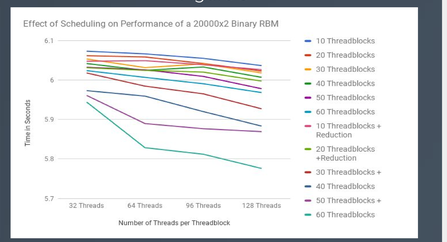


Figure 6

Restricted Boltzmann Machine

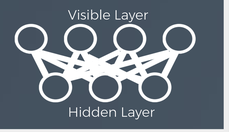
- Restricted Boltzmann Machines, or RBMs, create probability distributions based off the connection weight between nodes of the hidden and visible data
  - This distribution allows for fast classification of data, which can be applied to in many fields
- A connection weight is the strength of the connection between two nodes, which can be determined by the input in the node of the visible layer used for that connection
  - The more values in the node, the stronger the weight
- The program uses the formulas below in order to create the distribution
  - W is the connection weight between 2 specific nodes
  - h is the hidden layer; v is the visible layer
  - j and i are the location of specific nodes in the hidden and visible layers, respectively
  - a and b are the specific nodes from i and j
  - Z is the normalizing function used in order to make sure that the distribution sums to 1

Energy Function:

$$E(\mathbf{v},\mathbf{h}) = -\left(\sum_i\sum_j w_{ij}h_j v_i + \sum_i v_i a_i + \sum_j h_j b_j\right)$$

Probability Distribution:

$$P(\mathbf{v},\mathbf{h}) = \frac{1}{Z}e^{-E(v,h)}$$



Conclusion

- The parallelization of a RBM onto a GPU has been shown to be significant
- It was found that as data input size increases, OpenACC becomes more effective
  - As seen in the results, it increased performance in this specific research by over 35%
- An increase in thread blocks is more effective than an increase in threads than thread blocks
- Future research includes using deep learning algorithms instead of a 2 layer neural network
  - Perhaps using a Boltzmann Machine, which is an RBM with more than 2 layers
  - Or use a non binary input and see how performance gains change then

References

Training of Restricted Boltzmann Machines. (2007, June 21). Retrieved August 7, 2017, from http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DBNPseudoCode

OpenACC Programming and Best Practices Guide. (n.d.).

Bhimani, J., Leeser, M., & Mi, N. (2015). Accelerating K-Means Clustering with Parallel Implementations and GPU Computing. IEEE. Retrieved August 7, 2017.

(n.d.). Retrieved August 7, 2017, from https://www.openacc.org/

Elfwing, S., Uchibe, E., & Doya, K. (2015). Expected energy-based restricted Boltzmann machine for classification. Neural Networks, 64, 29-38. doi:10.1016/j.neunet.2014.09.006

Mohr, B., Malony, A. D., & Cuny, J. E. (1995). Tau - Tuning and Analysis Utilities for Portable Parallel Programming (1995). Retrieved August 8, 2017.

U.S. DEPARTMENT OF ENERGY | Office of Science