Speeding up DNNs using HPL based Fine-grained Tiling for Distributed Multi-GPU Training

Kaustubh Shivdikar, Kaushal Paneri and David Kaeli
Department of Electrical and Computer Engineering
Northeastern University
Email: kaustubhcs@ece.neu.edu, paneri.k@husky.neu.edu, kaeli@ece.neu.edu

***Abstract*—With the advent of hardware accelerators, it is possible to carry out massive computations on petascale-class problems using graphics processors. One of the challenges when designing programs to exploit a GPU is the strong relationship between performance and the mapping/utilization of GPU memory. This paper considers how to map a Deep Neural Network (DNN) application to the GPU memory systems. We consider an optimized version of a DNN model leveraging coarse-grained parallelism.**

This work presented includes two studies. First, we consider the performance of of High Performance Linpack benchmark, a highly optimized implementation of Linpack. Second, using the insights derived from first study, we optimize the performance of DNNs by altering its batch size. We consider two different types of DNNs, presenting different forms of parallelism in terms of data-level and model-level characteristics. The HPL study for 8 NVIDIA K80 GPUs yielded an optimized Block size of 160 which amounted to batch size value of 64 images. Comparing this method to a heuristic approach, a speedup of 1.46x for CNN was achieved.

I. Introduction

A modern-day system hosting multiple NVIDIA DGX servers could easily have been in top 100 systems on the Top500 list of 2012. The raw compute compute capability of today's GPU system has spurred on renewed interest by industry and academia in deep learning applications. In this work, we consider how best to map DNNs to a GPU's memory hierarchy.

An important factor to consider while implementing the high-level software models on any cluster is how it maps on to the underlying hardware. Fatahalian et al. [2] discuss the challenges of programming a memory hierarchy using a programming language they developed called Sequoia. They expand further talking about the problems associated with increased latency as a direct result of multi-level memory model. The paper provides an example of mapping a 1024x1024 matrix-matrix multiplication to different caches. Achieving the right tile can impact performance significantly. If the problem is written in a way that maps perfectly to the memory hierarchy, long memory latencies can be hidden.

A typical approach to understand the performance of a memory system is to perform benchmarking [1]. The High Performance variant of the Linpack benchmark named HPL [3], commonly used to benchmark supercomputers, provides a good starting point. Moreover, there is a lot of similarity between Deep Neural Nets and Linpack, since both involve solving linear systems of equations.

II. Experimental Approach

We start with developing a Convolutional Neural Network model, and use the MNIST handwriting dataset on Keras, with the Tensorflow-GPU backend and CUDA, cuDNN and cuBLAS integrated [4]. Categorical cross-entropy as loss function and the Adam Optimizer [5] was used for training. The total number of parameters to be trained were 1,192,042. The colored digits in Fig 1 indicates the number of nodes in each layer.

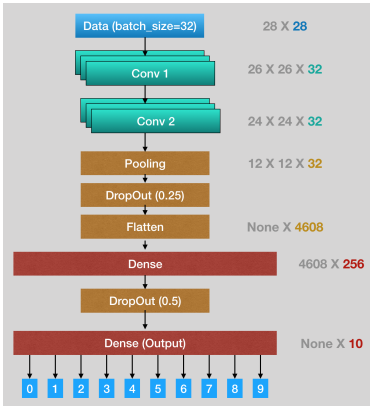

Fig. 1. Single GPU CNN Model for MNIST dataset.

Our experiments were performed using multiple GPUs [6] and applying coarse-grained parallelism techniques (i.e., model-level and data-level parallelism [7], [8]) to the same CNN model. In model-level parallelism, the first dense layer is distributed to multiple GPUs. To achieve data parallelism, we used a mini-batch rmsprop algorithm [9] for distributed learning, and applied an asynchronous weight update for robustness and ease of implementation. In order to evaluate the speed of the DNNs, we record the epoch time.

Since the first epoch takes the most time to run since it randomly initializes weights, we focus on the average time of the first epoch collected across multiple runs.

The dimensions of an image in the MNIST database is 28x28, with each image being single precision, resulting in a image size of $25,088\ bits$. To map these onto a GPU memory hierarchy, the output from the HPL library [10] is used. HPL

divides the problem set in to smaller matrices (called Blocks), with each dimension NB-by-NB. We start with tuning the tile size parameter "NB" for different iterations of same version of HPL, and identify the parameter configuration that obtain the best speedup.

III. Results

After running multiple iterations of HPL over a single node comprising of 8 Tesla K80 GPUs by NVIDIA, we found the fastest block size to be 160 (see Fig 2). Thus, the minimum memory requirements for this tile can be computed using Equation 1.

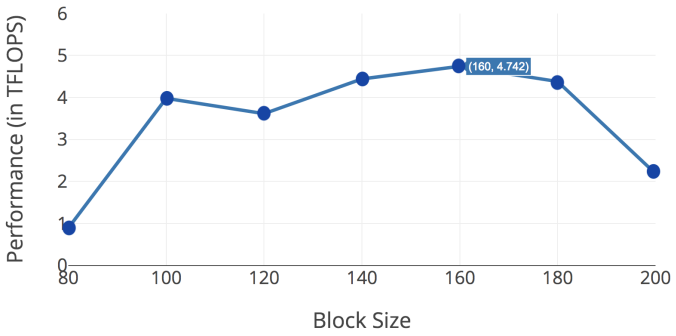

Fig. 2. Performance plot in 100 GFLOPS with varying block sizes of HPL.

$$HPL\ Tile\ Size = 160{*}160{*}64_{(double\ \ precision)} = 1,638,400 \quad (1)$$

As the number of bits in a single image is known (i.e., 25,088 bits), the batch size (the total images in every epoch) leading to the best performance can be derived from Equation 2 - 65.3.

$$Batch\ Size = \frac{HPL\ Tile\ Size}{Image\ Size} \quad (2)$$

Since the pooling layer in a CNN reduces the window size in powers of two, we explore powers of two while selecting the batch size. In this case, Equation 3 produces the best batch size value of 64.

$$Batch\ Size_{Optimal} = 2^{nint(log_2(\frac{HPL\ Tile\ Size}{Image\ Size}))} \quad (3)$$

Where $nint()$ is nearest integer function.

To confirm that our solution for DNN was the best, we implemented the same network with other batch sizes to determine their epoch run times, which are plotted in Figure 3 and Figure 4.

With a heuristic approach, the batch size value for DNN would start computation with 32 images per epoch, and gradually increase to find the best batch size. Although a batch size of 32 would produce lower loss, the time taken to complete 1 epoch would be considerable.

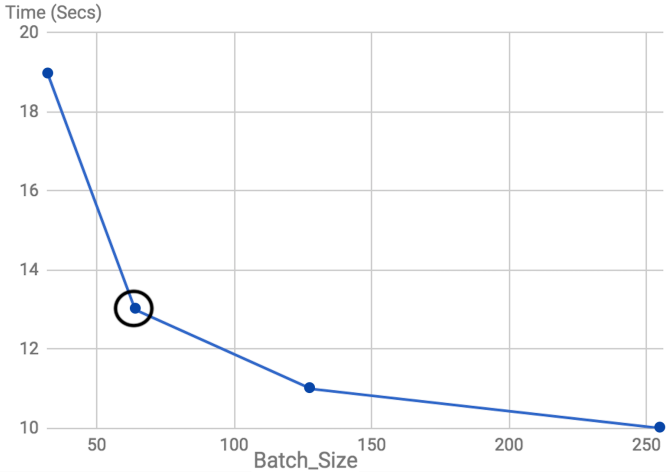

Fig. 3. SingleGPU model on Tesla K80 GPU: Execution time(Seconds) Vs. Batch.

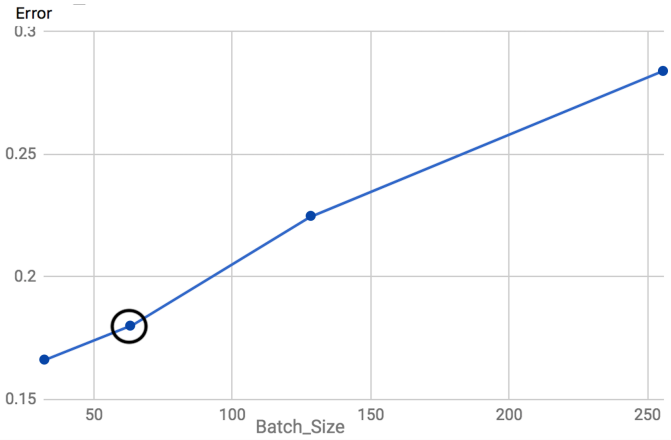

Fig. 4. SingleGPU model on Tesla K80 GPU: Categorical cross-entropy vs Batch size.

The suggested method of obtaining Speedup for DNN using HPL has shown to be superior to tuning DNNs manually using the ideal batch size obtained through heuristics. By using a batch size of 64, we obtain a 1.46x speedup. Since we would improve DNN performance by this rate over a thousand epochs, the performance gain should be significant.

IV. Conclusion

Tuning a large number of hyper-parameters for neural networks has always been a time-consuming iterative process. This work provides a method to speedup DNNs by selecting the best batch size. We consider selection from a hardware perspective versus a brute force approach. Since the focus of our work here is to consider batch size as a function of memory model and architecture of GPU, we did not consider the speed of convergence for a particular optimization problem. Speedups for DNNs can also be achieved by simply allocating more GPUs to the process. This is where convergence-invariance comes into consideration. Tallada in his work [8] describes how doubling the number of GPUs and halfing the

batch size, does not guarantee a 2x speedup. The convergence of a network is a area for future work. This work can also be expanded to take into consideration the overall network structure parameters, such as the number of nodes and number of layers.

REFERENCES

[1] Wang E. et al., "High-Performance Computing on the Intel Xeon Phi", Springer 2014, ISBN 978-3-319-06486-4.

[2] Fatahalian K. et al.,"Sequoia: programming the memory hierarchy", SC 2006, Proceedings of the 2006 ACM/IEEE conference on Supercomputing.

[3] Petitet A. et al., "HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers" Innovation Computing Laboratory, Feb 2016, University of Tennessee.

[4] Szegedy C. et al. "Going Deeper with Convolutions", Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference.

[5] Kingma D. and Ba J., "ADAM: A Method for Stochastic Optimization", International Conference on Learning Representations (ICLR) 2015.

[6] Schaa D. and Kaeli D., "Exploring the multiple-GPU design space", Parallel & Distributed Processing, 2009, IPDPS.

[7] Dean J. et al., "Large Scale Distributed Deep Networks", Neural Information Processing Systems, NIPS 2012.

[8] Tallada M. "Coarse grain parallelization of deep neural networks" in PPoPP '16 Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming.

[9] Tieleman T., Hinton G., "Lecture 6.5 RmsProp: Divide the gradient by a running average of its recent magnitude", COURSERA: Neural Networks for Machine Learning.

[10] LeCun Y., Cortes C. and Burges C, "The MNIST Database of Handwritten Digits", Google Labs New York and Microsoft Research, Redmond.