Performance Optimization on GPGPU & Multicore CPU Using Roofline Model

Noor M. Allayla\textsuperscript{1*} and Shefa A. Dawwd\textsuperscript{1}

\textsuperscript{1}Computer Engineering Department, College of Engineering, University of Mosul, Mosul, Iraq.

\*E-mail: noor.mowafeq@uomosul.edu.iq

Abstract. The roofline model introduced in this paper to evaluate the best optimized platform for training the neural network that used to recognize handwritten digits under multicore CPU and general-purpose GPU (GPGPU) as hardware environment. The pattern parallel training technique for MNIST dataset is applied. The parallel network training of MNIST using different data layout of multicore CPU and GPGPU is presented. Different bottlenecks have been explained by applying the roofline model. The most suitable platform is selected according to layouts and constrains either for memory or computation bounds. The computational intensity of all rooflines is moved toward right, then the performance is increased. As a result of optimization, and with the diversity of the available data size, core number, operational strength, the most suitable hardware platform is selected.

1. Introduction

Applying parallelism on the systems becomes widespread to increasing the performance of computer architectures. The Graphics Processor Unit (GPU) is suitable to be used in a different type of applications, while other applications are more suitable to be implemented on a multicore processor. So, which one is a more suitable in a specific application? One of the methods to choose is a roofline model.

The roofline model is a performance model used in recent years to understand and analyse the performance and bottlenecks in varying platform [1]. The roofline model is a very simplified model, so it is executed very fast. The roofline model could be applied on the CPU, GPU and the memory architectures [2]. This gives a multiple option for computing on varied platforms. Applying the performance on specific application depends on difficulty of the factors that need to be taken when using a multicore. Since the exclusiveness properties of the processor architecture vary for each processor [3], the roofline model gives the performance valuation of a specific application depending on the postulate that performance is bounded by the bandwidth between the processor and the memory or by the maximum computational performance of the specific architecture. The roofline model firstly designed to rating the CPU execution, but can easily applied on the GPU [4]. Some works use the roofline are presented: Yu Jung Lo and others, measured sustained PCIe (peripheral component interconnect express) throughput GPU, based on architectural specifications. The achieved results from the properties of the GPU architecture aided with the calculations using roofline model, guided them to select the appropriated platform [5]. Aleksandar Ilic and others, analyses and applied the roofline model, and proposes a new approach to provide a more sight performance modelling of modern architectures [4]. Nugteren and others, trying to improve performance before implementation using algorithm classification to predict performance prior the implementation by modifying the roofline model to enable architectural option through performance forecasting prior to the development of architecture specific code by the boat hull model. For six example algorithms using a GPU, this model proved performance predict without requiring code [6]. Choi and others, also applying the roofline model on the time, energy, and power, the roofline model applied on the features of an algorithm such as locality and concurrency with time and energy costs. The difference in an algorithm optimization for time versus an algorithm optimization for energy is explained, and the time-energy trade-offs is achieved [7].
The roofline model gives a perfect performance estimation of the neural network training process under CPU and GPU environments. A neural network is applied to exceeded problems inclusive extraction the feature, noise reduction, classifications and image matching, it is effective to exceed the problems where the mathematical solution is difficult to apply. Typically, large data pattern is desired to train the neural networks. Whenever increasing the size of the neural network pattern, the time required to train the network is increased exponentially [8] [9]. In this research, we used the multicore Central Processing Units (CPUs), and GPU with shared memory as a cost-effective way to increasing the performance. Then we introduce the roofline model to show the CPU, GPU, and the memory bottlenecks and trying to improve the performance regard to each designed model.

2. The Roofline Model
The visual representation of the roofline model concludes two roofs: firstly, memory roof, limited by the bandwidth of external memory, the second is a computational roof, limited by the maximum computational performance of the processor [10]. The roofline rates the kernel of a specific application as a data transfer from one or more memories to a processor where it can be used for the computation. The results are written back to the memories. The movements of data between the memory and the processor are bounded by the characteristics of the interconnection between them. The memory traffic with the processor has the greatest latency, and it is a form of bottleneck. For example, when data is transfer to the processor from the memory, the caches make sure this data is kept locally if it is re-used (unless there is a great amount of data used that could cause the data to be overwritten in any layer of the cache hierarchy). The maximum computational performance consists of single precision floating-point operations (such as compares, add, subtract or multiply) [11].

2.1 Computational Roofline
For a specific architecture, the computational roofline model is achieved by applying equation:

\[ C_{\text{roof}} = \#\text{cores} \times C_f \times C_{\text{ops}} \]  \hspace{1cm} (1)

The computational roofline is represented by \( C_{\text{roof}} \). The \( C_{\text{roof}} \) can be calculated by multiplying the number of cores (\( \#\text{cores} \)) by the clock frequency of the cores (\( C_f \)), and the floating-point number of the operations that can be performed per core, per clock cycle (\( C_{\text{ops}} \)). The \( C_{\text{ops}} \) take multiple function units per core; it is calculated as shown ‘in equation (2)’ by summing the number of the function units and their respected width:

\[ C_{\text{ops}} = \sum_{0}^{q-1} \text{FU width} \]  \hspace{1cm} (2)

The number of function units per core represented by \( q \) [12]. In this paper, we used different platforms, after applying the above equations. The results are show in table 1:

| The Platform   | #cores | \( C_f \) | \( C_{\text{OPS}} \) | \( C_{\text{roof}} \) |
|----------------|--------|-----------|---------------------|----------------------|
| CPU3210        | 2      | 2.5       | 4                   | 20                   |
| CPU4200        | 2      | 2.5       | 4                   | 20                   |
| CPU2670        | 4      | 2.2       | 8                   | 70.4                 |
| GeForce610     | 48     | 0.81      | 2                   | 77.76                |
| GeForce740     | 384    | 0.98      | 2                   | 752.64               |
2.2 Memory Bandwidth

The memory bandwidth $B_{\text{roof}}$ can be calculated using equation:

$$B_{\text{roof}} = M_f \cdot M_{\text{dt}} \cdot M_{\text{width}} \cdot \#\text{channels}$$  \hspace{2cm} (3)

Where $M_f$ is the memory frequency at which a memory unit moved the data, $M_{\text{dt}}$ is the size of data moved per clock cycle, $M_{\text{width}}$ specify the width in bytes of a single memory moved and $\#\text{channels}$ is the number of memory channels [12]. The roofline model (Figure 1) can be obtained by using equations (1) to (3). The $B_{\text{roof}}$ of our platforms according to platforms specification is shown in table 2; at this point the ceiling point is calculated as $C_{\text{root}} / B_{\text{roof}}$.

| The Platform   | $B_{\text{roof}}$ | $C_{\text{roof}} / B_{\text{roof}}$ |
|---------------|------------------|-----------------------------------|
| CPU3210       | 25.6             | 0.78                              |
| CPU4200       | 25.6             | 0.78                              |
| CPU2670       | 21.3             | 3.3                               |
| GeForce610    | 14.4             | 5.4                               |
| GeForce740    | 80               | 9.4                               |

2.3 Operational Intensity

When the computations are performed a large amount of data are required to move from the memory, the computational performance is expressed in floating point operations per second (flops/sec). The operational intensity is expressed in floating point operations per byte (flops/byte). Increasing the operational intensity leads to bounding the kernel by the computational roofline of a specific architecture, which performs a large number of floating-point operations per byte moved from the memory. While decreasing operational intensity leads to bounding the kernel by the memory bandwidth. The operational intensity can be controlled by the software programmer.

2.4 Computational Performance

After estimating the operational intensity of a specific kernel, the application could be drawn as a vertical line on the roofline, as shown in Figure 1.

![Figure 1. Application on Roofline Model.](image-url)

It is possible to achieve the maximum performance of a specific application on the specific architecture using this model. For example, when a specific kernel is considered, it can be deduced when the kernel optimally uses the resources of that architecture, it could change to hit the memory bandwidth roof. When conclude the time required to executing the kernel, it can be drawn as a point on the roofline graph and the computational performance of the kernel can be extracted by calculating the number of floating-point operations [10].
3. The Case Study and Mapping
The MNIST (Mixed National Institute of Standards Technology) handwritten digit database (showed in figure 2) is applied on the neural network. The MNIST sample can be founded in [5].

![Figure 2. The MNIST Database.](image)

It contains 60,000 patterns; each one represents a digital image used to train the classifier. 10,000 patterns are used for testing. All images are black and white. The size is normalized and centred in a fixed-size image where the centre of the intensity lies at the centre of the image of 28 × 28 pixels. The dimensions of each sample vector are 28 * 28 = 784, where each element is binary one or zero [11]. The reason behind choosing this database is: it is a standard and simple database for fast-testing theories and algorithms. Also testing neural networks for actual issue in the real world. The MNIST database have already been pre-processed including segmentation and normalization, so it could be a good start for us with minimal efforts on pre-processing.

The PTT (Pattern Parallel Training) technique is implemented for parallel training on MNIST database as a 60000 pattern for training and as 10000 patterns as a network tester. A network of 784 neurons in the input layer, a 350 neuron in the hidden layer (this number of neurons is selected by trial and error), and 10 neurons in the output layer is designed. A 28 × 28 pixels pattern is applied on the input layer for the network and the output from 0 to 9 is produced. The neural network structure is shown in Figure 3.

![Figure 3. The ANN Architecture.](image)

4. Results and Discussion
The neural network is trained on MNIST using different specification platforms as shown in table 3. Then different bottlenecks have been explained by applying the roofline model as presented later.
Table 3. Platform Specifications.

| The Platform | RAM  | The Freq. | Core NO. | GPU Type         | GPU NO. | Memory of GPU |
|--------------|------|-----------|----------|------------------|---------|---------------|
| i7-2670QM    | 8GB  | 2.2GHZ    | 7        | Intel® HD Graphics3000 | -       | -             |
| i5-3210M     | 6GB  | 2.5GHZ    | 5        | GeForce610       | 48      | 1GB           |
| i5-4200M     | 6GB  | 2.5GHZ    | 5        | GeForce740       | 384     | 1TB           |

4.1 CPU Bottlenecks
By determining the operation intensity of the training program which equals to 1.7869, the result on the first platform that contains five cores, is shown in Figure 4. The CPU roof is 20, then increasing the CPU cores to seven cores by using another platform, the roof becomes 70.4.

![Figure 4. CPU Roof Representation.](image4.png)

When increasing the cores number, the line of application going toward CPU roof, and approaching from the ceiling point, while it was overtaken the ceiling point. This means that the result is improved by increasing the cores number.

4.2 Memory Bottlenecks
After applying the training on multicore CPU platforms, the memory roof 0.7514 is achieved. Then when trying to decrease the operation (by simplifying some operations), this leads the memory roof to be equals to 1.7869. Figures 5, shows the memory roof of two different platforms (5 cores CPU and 7 cores CPU respectively).

![Figure 5. Memory Roof of Core i5 and Core i7.](image5.png)

From Figure 5, one can see that, approaching the ceiling point is achieved in the normal operation intensity when core i5 is used, while approaching the ceiling point at core i7 is achieved when reducing the operation intensity.

After applying the roofline calculation of different GPU platforms, a memory roof at 1.8869 is achieved. Then when trying to reduce the operation intensity, the memory roof equals to 6.9184. Figures 6, shows the memory roof with two different platforms: 48 GPU core and 384 GPU core.
Figure 6. Memory Roof of 48 and 384 Cores GPU.

As shown in the Figure 6, approached to the ceiling point in both GPU types are achieved when the operation intensity is reducing.

4.3 GPU Bottlenecks

After applying the training on the neural network of data MNIST on two different platforms, GPU roof at 77.76 in 48 cores GPU is achieved. Then, when applying the training on 384 core GPU, GPU roof at 752.4 is achieved, as shown in Figure 7.

Figure 7. GPU Roof Representation.

The Figure 7; show that approaching the ceiling point at the GeForce 610 is closer than that of GeForce 740.

5. Training Time

The processing time of different platforms that used in training before and after roofline optimization is show in Figure 8.

Figure 8. Training Time Before and After Roofline Optimization.
The benefit of using the optimizations is clearly shown due to reducing the training time in all types of the platforms.

6. Conclusion
The neural network based handwritten digit recognition system on MNIST database is presented in this work. The parallel training using different types of platforms is achieved. Then the roofline model is applied to investigate bottlenecks of both CPU and GPGPU for the performance optimization purposes. The platform of best specification is selected according to layouts and constrains either for memory or computation bounds. Suitable results are achieved when the data size, operational intensity and the core numbers are suitably identified.

7. References
[1] Nugteren, C. and Corporaal, H., 2012, February. The boat hull model: adapting the roofline model to enable performance prediction for parallel computing. In Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (pp. 291-292).
[2] Cabezas, V.C. and Püschel, M., 2014, October. Extending the roofline model: Bottleneck analysis with microarchitectural constraints. In 2014 IEEE International Symposium on Workload Characterization (IISWC) (pp. 222-231). IEEE.
[3] Denouyelle, N., Goglin, B., Ilic, A., Jeannot, E. and Sousa, L., 2017, November. Modeling large compute nodes with heterogeneous memories with cache-aware roofline model. In International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (pp. 91-113). Springer, Cham.
[4] Ilic, A., Pratas, F. and Sousa, L., 2013. Cache-aware roofline model: Upgrading the loft. IEEE Computer Architecture Letters, 13(1), pp.21-24.
[5] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp.2278-2324.
[6] Nugteren, C. and Corporaal, H., 2012, February. The boat hull model: adapting the roofline model to enable performance prediction for parallel computing. In Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (pp. 291-292).
[7] Choi, J.W., Bedard, D., Fowler, R. and Vuduc, R., 2013, May. A roofline model of energy. In 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (pp. 661-672). IEEE. [8] Wang, Y.E., Wei, G.Y. and Brooks, D., 2019. Benchmarking tpu, gpu, and cpu platforms for deep learning. arXiv preprint arXiv:1907.10701.
[9] Alzantot, M., Wang, Y., Ren, Z. and Srivastava, M.B., 2017, June. Rstensorflow: Gpu enabled tensorflow for deep learning on commodity android devices. In Proceedings of the 1st International Workshop on Deep Learning for Mobile Systems and Applications (pp. 7-12).
[10] Cabezas, V.C. and Püschel, M., 2014, October. Extending the roofline model: Bottleneck analysis with microarchitectural constraints. In 2014 IEEE International Symposium on Workload Characterization (IISWC) (pp. 222-231). IEEE.
[11] Lorenzo, J.A., Pichel, J.C., Pena, T.F., Suárez, M. and Rivera, F.F., 2011. Study of Performance Issues on a SMP-NUMA System using the Roofline Model. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA) (p. 1). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).
[12] van Doormaal, R.P.M., Corporaal, H. and Peemen, I.M., 2012. Parallel Training of Large Scale Neural Networks: Performance Analysis & Prediction.