# GPU-only Unified ConvMM Layer for Neural Classifiers

Syed Tahir Hussain Rizvi, Gianpiero Cabodi
Dipartimento di Automatica e Informatica (DAUIN)
Politecnico di Torino
Torino, Italy
syed.rizvi@polito.it

Gianluca Francini
Joint Open Lab
Telecom Italia
Torino, Italy

***Abstract*—Convolution is most computationally intensive task of Convolutional Neural Network(CNN). It demands both computational power and memory storage of processing unit. There are different approaches to compute the solution of convolution. In this paper, matrix multiplication based convolution(ConvMM) approach is implemented and accelerated using concurrent resources of Graphics Processing Unit(GPU). CUDA computing language is used to implement this layer. Performance of this GPU-only convolutional layer is compared with its heterogeneous version. Further, flow of this GPU-only convolutional layer is optimized using Unified memory by eliminating overhead caused by extra memory transfers.**

***Keywords— Concurrent Computing; General Purpose GPU; Unified Memory; Convolutional Neural Networks; Heterogeneous***

## I. Introduction

Convolutional Neural Network (CNN) exhibits unmatched performance in different computer vision applications [1]. CNN is type of neural classifier that generates features hierarchically using convolutions. Torch or other available frameworks can be used to construct and train a neural classifier for a specific dataset as shown in fig 1. Once training is finished, trained parameters can be imported and used by identical classifier implemented in CUDA, Matlab or in any other framework for further developments. This CUDA based identical classifier can be embed and run on an android phone using its heterogeneous computational resources (CPU-GPU) and can also benefits from built-in sensors like camera for real time classification [2]. Furthermore, an external camera can be attached with desktop system for CUDA or Matlab based real-time classification and based on these results different actuators can be controlled with the help of a microcontroller like Arduino.

Convolutional Neural Network (CNN) is normally composed of convolutional layers, pooling layers, some normalization functions and fully connected layers. However, high accuracy of neural classifiers is mainly dependent on the number of stacked convolutional layers [3]. State of the art deep classifiers are comprised of tens or hundreds of these convolutional layers [3-5]. Where number of convolutional layers increase the accuracy of a classifier, this increased depth also increases the training/classification time and arithmetical complexity. A significant performance improvement can be achieved by reducing computational time of these convolutional layers.

There are several approaches to compute Convolution operation [6-11]. Fast Fourier transformation (FFT), winograd minimal filtering algorithm, look up table and matrix multiplication based convolutions are few of them. In FFT based convolution, signals (image and filters) are transformed and multiplied point wise in frequency domain to reduce the amount of multiplications [6]. However, this transformation between time and frequency domain is computationally intensive and limits the performance gain. Moreover, fft based convolution is fast for large filters but state of the art deep classifiers are using small filters [4]. For networks having small filter, winograd based convolution shows good results [7]. This algorithm reduces the arithmetic complexity of convolutional layer by using minimal filtering technique.

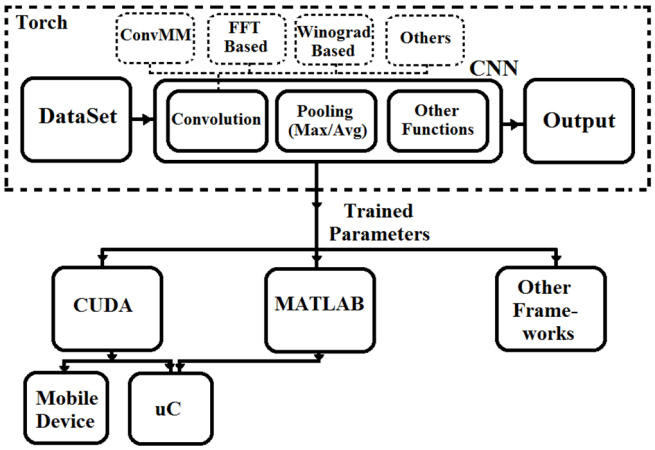


Fig. 1. Block Diagram of a Trained Network based Model

One widely used approach is Matrix multiplication based convolution (ConvMM). Using this approach, first multi dimensional input image and filters are transformed into two-dimensional matrices and then multiplied to achieve result equivalent to traditional convolution operation. Transformation step is performed by copying tiles of pixels from original image/filter to matrices in a specific order. This is a operation where sequence has to be consider and can be performed efficiently using CPU [2]. Once transformed, these matrices can be multiplied efficiently using concurrent resources of GPU. Using heterogeneous resources (CPU-GPU) of a system, an algorithm like this can benefits in terms of execution time [2][15]. However, extra memory transfers

caused by heterogeneous implementation can also break the overall performance.

These approaches to compute convolution can further be optimized using different techniques and schemes[12-14].

In this paper, ConvMM layer is optimized by parallelizing the transformation step and eliminating extra data transfers required by heterogeneous approach. This paper presents the GPU-only ConvMM layer by performing both transformation and multiplication steps on GPU. Overhead caused by explicit data movements is also eliminated using unified memory and performance of optimized layer is compared. The remaining sections of this paper are arranged as follows: Section II discusses the flow of Heterogeneous and GPU-only ConvMM layer. Optimization of GPU-only ConvMM layer using unified memory is presented in Section III. Section IV provides an evaluation of GPU-only Unified ConvMM layer . Finally, Section V concludes the research.

# II. Heterogeneous and GPU-only ConvMM

Detail description of heterogeneous matrix multiplication based convolutional layer is presented in [2]. Fig 2 illustrates the flow of heterogeneous ConvMM layer where transformation of data (input maps and filters) is done using CPU and multiplication of this transformed data is performed using GPU.

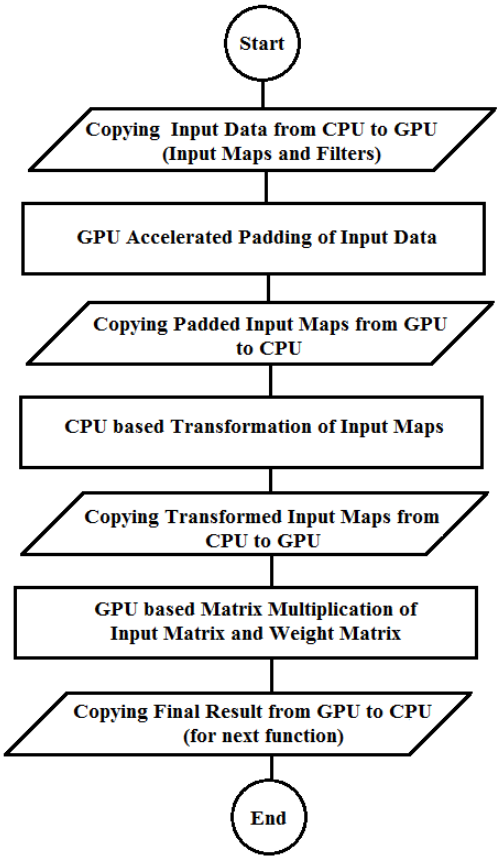


Fig. 2. Flow of Heterogeneous ConvMM Layer

In this transformation step, multi-dimensional input maps and filters are converted to 2-dimeniosnal matrices. Data transformation is performed by copying tiles from multi-dimensional maps and placing them in a specific order in the matrices as shown in Fig 3.

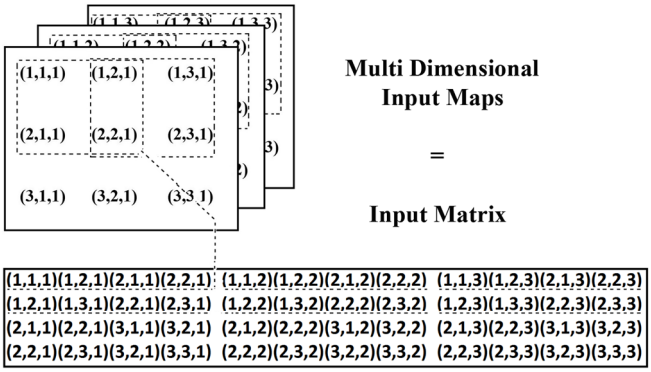


Fig. 3. Transformation of Multi-Dimensional Input Maps to 2-Dimensional Input Matrix

There are four data transfers between CPU and GPU as shown in Fig 2. Out of four, two data transfers are made to perform CPU based transformation which is a sequential operation and can be performed efficiently using high operational frequency of CPU [2].

If this CPU based transformation step is parallelized and performed using GPU, so these extra two transfers can be avoided and all tasks can be performed using GPU as shown in Fig 4. This multi-dimensional input map can be concurrently converted to input-matrix using following formula

$$Input\_matrix_{(row,col)} = \sum_{i=1}^{oRow} \sum_{j=1}^{oCol} \left( \sum_{k=1}^{iDim} \sum_{h=1}^{kh} \sum_{w=1}^{kw} Input\_map_{(i+h,j+w,k)} \right)$$

where

oRow = iRow-kh+1
oCol = iCol-kw+1
row = i * (oCol)+j
col = (k*kw*kh+(h*kw)+w)

If input map is of size 3x3 (iRow x iCol) and size of the filter is 2x2 (kh x kw), so output map after convolution would be of size 2x2 (oRow x oCol) with padding and stride of 0 and 1 respectively.

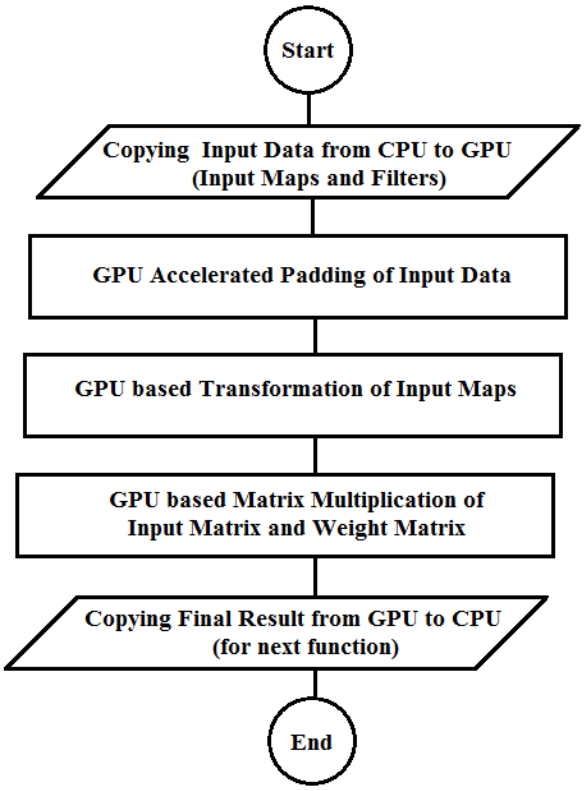


Fig. 4. Flow of GPU-only ConvMM Layer

## III. GPU-only ConvMM with Unified Memory

Typical desktop and embedded systems have separate host(CPU) and device(GPU) memories as shown in Fig 5. Data has to be allocated in both CPU and GPU memories. And this allocated data has to be transferred to GPU for concurrent operations and after computations, results have to be transferred back to CPU. Speed of these data transfers is dependent on PCI express bus. These transfers add overhead and a lot of complexity to flow of an algorithm as shown in Fig 2 and 4. A programmer has to consider these separated memories to define flow of an algorithm.

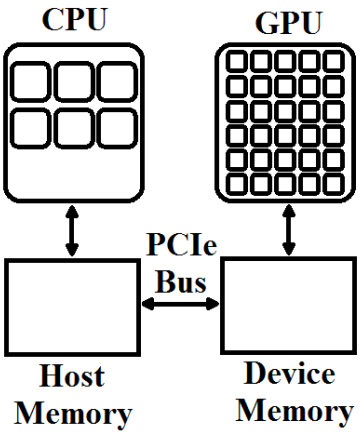


Fig. 5. Host(CPU) and Device(GPU) Memories

With CUDA 6.0, Unified memory(UM) is introduced which defines a pool of shared memory for CPU and GPU as shown in Fig 6. This shared memory can be accessed by both CPU and GPU. Same allocated variable can be used by a CPU function and a GPU kernel according to the requirement of program. But data synchronization is essential after every kernel execution.

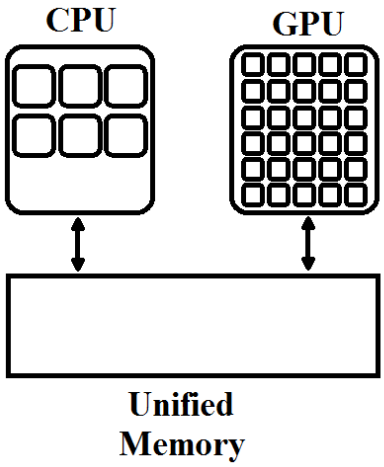


Fig. 6. Unified Memory

At least, two data transfers were essential for every layer as shown in Fig 4 to bring the input data to GPU for computations and to move back the results from GPU to CPU for next operation/layer. Using unified memory, flow of this GPU-only ConvMM layer can be further simplified and optimized as shown in Fig 7. This is done by allocating inputs and outputs in unified memory space from main program. By using unified memory in ConvMM and other layers, output of one layer can be directly fed to the next layer by eliminating data exchanges between CPU and GPU.

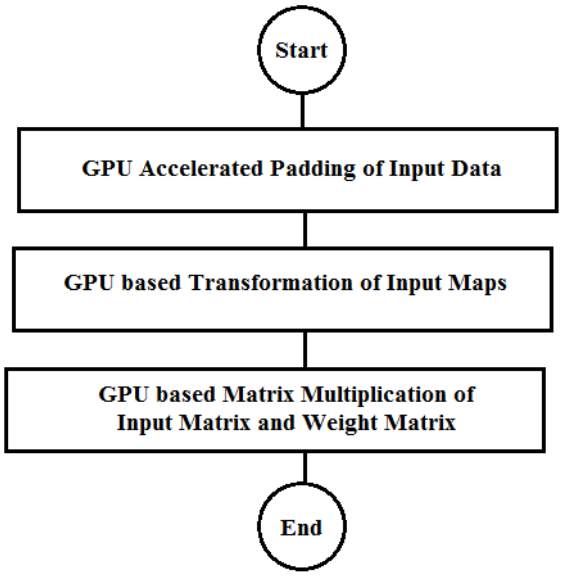


Fig. 7. Flow of GPU accelerated Unified ConvMM Layer

# IV. Experiments and Results

In this work, CUDA computing framework is used to implement hetero ConvMM layer and its different versions (GPU-only ConvMM and GPU-only Unified ConvMM). Nvidia Quadro k2200 is used to perform experiments and evaluate results of implemented functions.

Execution times of all three ConvMM layers for different image sizes are listed in Table I. Both modified versions (GPU-only and GPU-only UM ConvMM layers) are taking less time to compute convolution for various image sizes.

TABLE I. COMPARISON OF HETERO CONVMM WITH OTHER VERSIONS

| 16 Output Maps of Same Size | | | |
|---|---|---|---|
| **Size of Input Image** | **Hetero ConvMM** | **GPU-only ConvMM** | **GPU-only Unified ConvMM** |
| | (Milli-Seconds) | | |
| Cifar (32x32x3) | 2.49 | 2.18 | **1.06** |
| Imagenet (224x224x3) | 29.54 | **21.70** | 21.82 |
| VGA (480x640x3) | 170.07 | **122.84** | 128.58 |
| SVGA (600x800x3) | 251.95 | **179.97** | 191.29 |

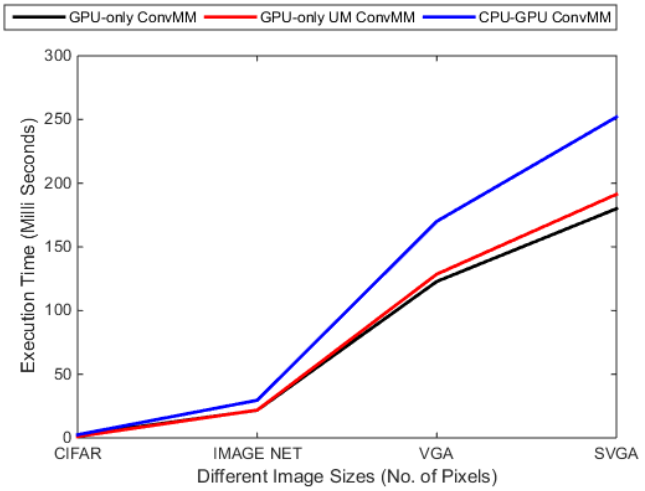


Fig.8. Comparison of different ConvMM Layers

However, it can be noted that a single Unified memory based ConvMM layer is taking more time than GPU-only ConvMM layer, it is because of data synchronization required by unified memory. But this same GPU-only unified ConvMM layer outperforms the other versions for deeper networks as shown in Table II.

It is because of large data transfers performed by Hetero and GPU-only ConvMM layers in deeper networks. These data transfers add overhead more than synchronization (required by unified memory). However, results achieved by Unified memory depends on the specifications of host system as well as GPU.

TABLE II. COMPARISON FOR DEEP CLASSIFIERS

| **Model** | **Layers** | **Hetero ConvMM** | **GPU-only ConvMM** | **GPU-only Unified ConvMM** |
|---|---|---|---|---|
| | | (Milli-Seconds) | | |
| Alexnet | 5 | 339.02 | 109.37 | **87.20** |
| OverFeat | 8 | 2650.93 | 2318.84 | **1917.30** |
| ResNet-34 | 34 | 3739.44 | 3556.51 | **3277.42** |

# V. Conclusion

This paper presents the implementation of GPU-only Unified ConvMM layer. Matrix Multiplication based convolutional layer is accelerated purely using GPU. Furthermore, Unified memory is used to optimize flow of this GPU-only ConvMM layer. Unified Memory benefits the implementation in two ways. It simplifies the flow of model and increases the speed of data access by migrating allocations into managed memory space due to which allocated data can be accessed directly by both CPU and GPU functions according to requirement. These results are compared with heterogeneous ConvMM layer. Comparative analysis concludes that the performance of heterogeneously implemented algorithm can be improved by using GPU-only implementation even if get little or no performance enhancement on a specific hardware. Because GPU-only implementation minimizes the amount of data transfers between host and device and can provide higher performance where PCIe bus is not fast enough .

Future work would be to perform same experiments on mobile phone and to test other memory types and optimization techniques to accelerate the real time classification.

## *Acknowledgment*

This research work is funded by Telecom Italia and performed at Joint Open Lab, Telecom Italia, Torino, Italy.

## *References*

[1] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "Imagenet classification with deep convolutional neural networks," In Proceedings of The Twenty-sixth Annual Conference on Neural Information Processing Systems (NIPS), Lake Tahoe, NV, USA, December, 2012, pp. 1097–1105.

[2] S. T. H. Rizvi, G. Cabodi, D. Patti, and G. Francini, "GPGPU Accelerated Deep Object Classification on a Heterogeneous Mobile Platform," Electronics, vol. 5, no. 4, p. 88, Dec. 2016.

[3] C. Szegedy et al., "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 1-9.

[4] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778.

[5] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks" In International Conference on Learning Representations (ICLR2014), CBLS, April 2014.

[6] M. Mathieu, M. Henaff, and Y. Lecun, "Fast training of convolutional networks through FFTs," In International Conference on Learning Representations (ICLR2014), CBLS, April 2014.

[7] A. Lavin and S. Gray, "Fast Algorithms for Convolutional Neural Networks," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 4013-4021.

[8] K. Chellapilla, S. Puri, and P. Simard, "High Performance Convolutional Neural Networks for Document Processing," in Proc. of the Int. Workshop on Frontiers in Handwriting Recognition (IWFHR'06), 2006.

[9] J. Cong and B. Xiao "Minimizing Computation in Convolutional Neural Networks,"In: S. Wermter et al. (Eds.): ICANN 2014, LNCS 8681, pp. 281–290, 2014.

[10] J. Wang, J. Lin and Z. Wang, "Efficient convolution architectures for convolutional neural network," 2016 8th International Conference on Wireless Communications & Signal Processing (WCSP), Yangzhou, 2016.

[11] W. Jiang, Y. Chen, H. Jin, B. Luo and Y. Chi, "A Novel Fast Approach for Convolutional Networks with Small Filters Based on GPU," 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, New York, NY, 2015, pp. 278-283.

[12] H. Park, D. Kim, J. Ahn and S. Yoo, "Zero and data reuse-aware fast convolution for deep neural networks on GPU," 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Pittsburgh, PA, 2016, pp. 1-10.

[13] K. Li, H. Shi and Q. Hu, "Complex convolution Kernel for deep networks," 2016 8th International Conference on Wireless Communications & Signal Processing (WCSP), Yangzhou, 2016.

[14] P. K. Tsung, S. F. Tsai, A. Pai, S. J. Lai and C. Lu, "High performance deep neural network on low cost mobile GPU," 2016 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, 2016, pp. 69-70.

[15] V. H. Naik and C. S. Kusur, "Analysis of performance enhancement on graphic processor based heterogeneous architecture: A CUDA and MATLAB experiment," 2015 National Conference on Parallel Computing Technologies (PARCOMPTECH), Bangalore, 2015, pp. 1-5.