Efficient Neural Network Acceleration on GPGPU using Content Addressable Memory

Mohsen Imani‡, Daniel Peroni‡, Yeseong Kim‡, Abbas Rahimi*, and Tajana Rosing‡
‡CSE, UC San Diego, La Jolla, CA 92093, USA
*EECS, UC Berkeley, Berkeley, CA 94720, USA
{moimani, dperoni, yek048, tajana}@ucsd.edu; abbas@eecs.berkeley.edu

***Abstract*—Recently, neural networks have been demonstrated to be effective models for image processing, video segmentation, speech recognition, computer vision and gaming. However, high energy computation and low performance are the primary bottlenecks of running the neural networks. In this paper, we propose an energy/performance-efficient network acceleration technique on General Purpose GPU (GPGPU) architecture which utilizes specialized resistive nearest content addressable memory blocks, called NNCAM, by exploiting computation locality of the learning algorithms. NNCAM stores highly frequent patterns corresponding to neural network operations and searches for the most similar patterns to reuse the computation results. To improve NNCAM computation efficiency and accuracy, we proposed *layer-based associative update* and *selective approximation* techniques. The *layer-based update* improves data locality of NNCAM blocks by filling NNCAM values based on the frequent computation patterns of each neural network layer. To guarantee the appropriate level of computation accuracy while providing maximum energy saving, our design adaptively allocates the neural network operations to either NNCAM or GPGPU floating point units (FPUs). The *selective approximation* relaxes computation on neural network layers by considering the impact on accuracy. In evaluation, we integrate NNCAM blocks with the modern AMD southern Island GPU architecture. Our experimental evaluation shows that the enhanced GPGPU can result in 68% energy savings and 40% speedup running on four popular convolutional neural networks (CNN), ensuring acceptable $< 2\%$ quality loss.**

I. Introduction

The growing *Internet of Things* (IoT) significantly increases the size of applications' datasets and the total amount of computation to be processed. By 2020, the rate of data generation is expected to surpass the capability current computing systems can process [1]. Such large generated data must be classified and learned to serve meaningful results for applications [2], [3]. Neural networks such as convolutional neural networks (CNNs) and deep neural networks (DNNs) are a category of machine learning algorithms inspired by the human brain. In particular, CNNs have been demonstrated to be an effective class of models for image processing, video segmentation, detection and retrieval, speech recognition, computer vision and gaming [4], [5], [6]. CNNs have a capability to exploit learned knowledge to deal with data which they have not encountered. For example, CNNs have been used to identify the nature of images such as stationary of statistics and locality of pixels, and utilize them to recognize the contents in new images. Many CNN algorithms are implemented to run on high-performance computing systems such as GPUs which have highly optimized architecture for parallel processing. However, with the emergence of the large-scaled data of the *IoT* input domain, running neural networks on the general purpose processors is still slow, energy hungry, and prohibitively expensive [8].

Recent research has focused on a great potential to improve energy efficiency of the parallel processors using associative memories [18], [13], [9]. Associative memory, in a form of a look-up table, stores frequent processor operations and exploits them for future computations, thus reducing redundant computation costs. For example, ternary content addressable memories (TCAMs) are considered the building block of associative memories. However, feasible energy saving would be compromised due to the limited number of storable operations and large energy cost for searches. In order to further improve the energy efficiency, earlier efforts introduced the idea of approximation techniques such as voltage overscaling of the associate memory which accepts a small difference in Hamming distances of computations [18], [13]. However, the quality of application outputs for these techniques is fundamentally sensitive to the error of the Hamming distance, and thus may not provide sufficient energy $\times$ quality loss per hit rate ($Energy \times QLoss/HitRate$). In addition, these designs cannot improve the performance of GPGPU computation, since the operations of the associative memory need to be synchronized to the floating point pipeline stages even though the computation results can be retrieved instead of using the actual GPGPUs.

In this work, we propose a novel neural network acceleration technique for GPGPU using resistive content addressable memory blocks, called NNCAM, which is specialized for the neural network computation. Instead of performing energy intensive precise computations, NNCAM stores highly frequent patterns and approximately searches for a row with the closest Hamming distance to the input data. To fully exploit computation locality in each neural network layer, we propose a *layer-based associative update* technique that fills the NNCAM with highly frequent computation patterns corresponding to each neural network layer. We also propose a *selective approximation* technique which puts each neural network layers on different approximate levels, depending on their accuracy sensitivity. The proposed NNCAM memory architecture is designed to efficiently support the two techniques while addressing the aforementioned issues of the existing GPU-based approximation. Our NNCAM design

978-3-9815370-8-6/17/$31.00

can configure the computation efficiency and the quality of outputs at runtime by adaptively assigning input data to either NNCAM or GPU floating point units according to the required accuracy. We show the efficiency of integrating NNCAM on modern AMD southern Island GPUs. Our evaluation on four CNN designs shows that the enhanced-GPGPU can result in an average energy savings of 68% and 40% speed up, while providing acceptable quality loss below 2%.

II. Related Work

Modern neural network algorithms are executed on diverse types of processors such as CPU, GPGPU, FPGAs and ASIC chips [14], [15], [16], [17]. Running neural network on traditional computers results in a large computation cost. This has motivated attempts to fully utilize current computing systems to run the algorithms in an efficient manner [7]. For example, for neural networks performing image classification applications, GPU-based approaches show up to two orders of magnitude improvement over CPU implementations. In addition, CNN and DNN designed for GPU allow faster training time [15][16].

Ternary content addressable memories (TCAMs) can be applied to several domains including database engines, data compression and recently as computational reuse for approximate computing [19], [18], [21]. In CMOS technology, TCAMs can be designed using SRAM cells, but they consume high energy for each search operation [20]. In contrast, the high density, CMOS compatibility, and nearly zero energy consumption of non-volatile memories (NVMs) make them viable for emerging memory or memory-based computing units. in diverse application domains. In particular, the spin transfer torque RAMs (STT-RAMs) have been diversely explored to replace with caches [10], [11], [24], [25]. From other side, resistive and ferro-electric non-volatile memories have shown great opportunities to be used as computing units, beside their memory functionality [22], [23], [12].

For example, in the GPU architecture, associative memory allows for computational reuse [18] and memoization for error free execution [26]. However, as mentioned before, the low hit rates and large search energy limit associative memories potential for computational reuse. Several techniques were proposed to improve the energy efficiency and hit rate of associative memories [18], [13], [27]. These designs exploit circuit- and architecture-level techniques to reduce the switching activity [13] and increase inexact hit rate by voltage overscaling of TCAMs [18], [27]. Voltage overscaled TCAMs improves both energy and hit rates by searching to find data while accepting a small difference in modifiable Hamming distances, thus allowing for inexact matching. However, the Hamming distance is not an appropriate metric in finding similarity for floating-point binary representations, as it incurs high possibility of multiple matches, creating an additional decision issue to choose a single final solution, potentially degrading the computation accuracy. The other issue of the existing techniques is their inability to accelerate the GPU computation, as the execution time of the GPUs enhanced with the associative memory is still limited by the pipeline stage of the floating point units (FPUs).

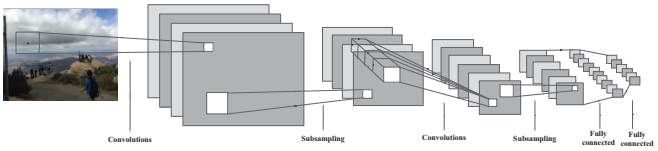

Fig. 1. An illustraion of k-layer convolutional neural network

To address these issues and enable the approximate computing for the neural network, we use a resistive content addressable memory (CAM) with the capability of performing all neural network computations inside the memory. We model the basic computation of each neural network layer so that the NNCAM can search for the memory row whose value difference from the neural network operations is smallest. To enable accurate approximate computing, our design selectively assigns a part of operations to the GPUs at runtime if the expected computation accuracy is not acceptable. In addition, our design significantly accelerates the GPU computation by assigning a majority of computations to fast and efficient CAMs for memory-internal processing.

III. Neural Network Acceleration on GPU

A. Overview of Proposed Architecture

Convolutional neural networks (CNN) consist of convolutional layers (subsampling step) and fully connected layers as multilayer neural networks as shown in Fig. 1. This structure is adequate to process 2-dimensional input signals, e.g. image, speech, and video sequences. Each layer receives a set of floating point numbers as the input from the previous layer and delivers another set of numbers as the output to the next layer. Thus, once the output of a layer is fully computed, the next layer is ready to compute sequentially using the output of the previous layer. The computation of each layer is processed by a group of basic computing units, called neurons, using learned weight values, thus connectivity of the neurons of adjacent layers increases correlation of spatial locality of CNN computations.

CNNs are usually processed on multicore processors such as GPUs. In GPU architectures, a larget portion of energy is consumed by floating point units (FPUs) in streaming cores. For instance, in the AMD southern Island GPU architecture, adder (ADD), multiplier (MUL), and multiply-accumulator (MAC) are the main energy-intensive FPU components. In order to reduce the energy consumption of the FPUs, our proposed architecture exploits CAM blocks, called NNCAM (Sec. III-D), beside each FPU. Figure 2 illustrates the overview of the proposed architecture which integrates the associative memories, which include the NNCAM blocks, into each FPU of the GPU streaming cores. Once a FPU operation is issued with input operands, the proposed architecture approximately checks whether similar input operands are stored in the associative memory block, while the pipeline stages of the FPU are processed in parallel. The associative memory stores a set of frequent patterns and their corresponding outputs on a TCAM and a memory block respectively. When a hit occurs in associative memory (i.e., if the similar input operands exist

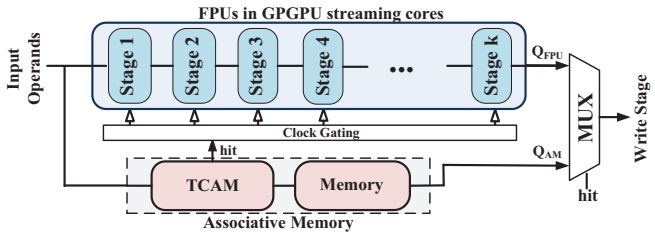

Fig. 2. Integration of the associative memory beside each floating point unit in a GPGPU streaming core

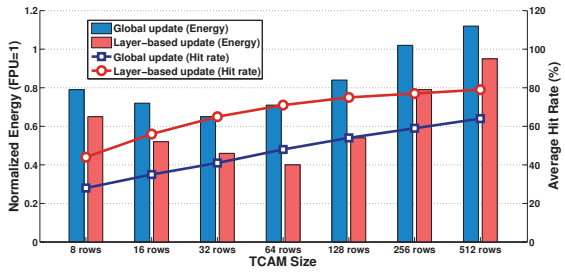

Fig. 3. Average hit rate of associative memory and normalized GPU energy saving using uniform and layer-based associative update.

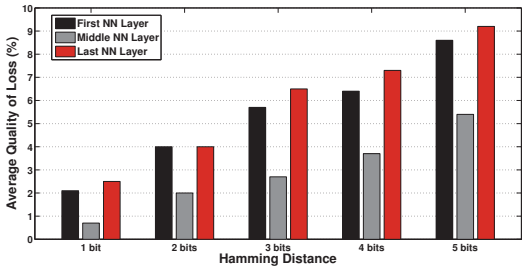

Fig. 4. Sensitivity of different CNN layers to inexact matching of 16 rows CAM in different Hamming distances.

in the TCAM block), the processor computation is clock-gated, and instead the FPU can retrieve the preprocessed computation result from the memory. Because neural network algorithms are processed fundamentally based on add and multiplication operations with the learned weights, a large amount of repeated computations happens necessarily. Thus, looking up these highly frequent patterns approximately in the small-size associative memory can provide significant computation reduction and energy saving.

B. Layer-Based Associative Update

During the approximate searches in the TCAM blocks, stored patterns have a high impact on hit rates and computation accuracy. For example, a naive approach to fill the TCAM blocks is to consider global patterns of the entire neural network execution. However, this may degrade the hit rates, missing potential energy saving, since each layer has data and computation locality due to the specific abstraction of the input data. For example, in the case of image processing, the neurons of each layer are responsible to handle different information of the input image, e.g., edge, background, etc. This motivates us to update the TCAM adaptively according to the layer which is currently processing. We implement the neural network software usually called as GPU kernel so that it creates a notification to the GPU to update associative memories with most frequent operand patterns for the running layer.

Fig. 3 shows the energy saving and the hit rates of associative memories in a 7-layer convolutional neural network on the AMD GPU architecture. In order to show the impact of the layer-based update on the hit rates, we compare it to the global updates which store the input operands frequently observed through the entire CNN computations. The result shows the layer-based update provides higher average hit rates than the global update strategy. The hit rates of the layer based updates approach saturates for larger CAM sizes, as the CAM already sufficiently stores frequent patterns. However, for the global updates, the CAM needs to be larger to store the highly frequent patterns for all layers. The bars in Fig. 3 show the normalized energy consumption of the GPGPU which exploits associative memory blocks. Since a large-size CAM consumes higher search energy, the advantage of high-level hit rates is compromised. Thus, the maximum energy saving is observed with 64 rows and 32 rows for the layer-based and global update cases respectively. For these configurations, the layer-based associative update technique achieves more energy savings by 26% compared to the global update policy.

C. Selective Approximation

The proposed architecture allows to configure the level of approximation in processing the neural network. A design issue related to this functionality is how to determine the depth of approximation to ensure enough accuracy of the results. For example, if high energy efficiency is required, it may set a high level of approximation at the expense of the quality loss. To ensure enough accuracy with high energy saving, we consider the sensitivity of different layers based on the observation that each layer has different levels of sensitivity for approximation. Fig. 4 shows the error increase of a neural network, which recognizes digits from images, when the layer computations are approximated by accepting inexact input operands. In this experiment, utilizing an associative memory beside each floating point unit (i.e., ADD, MUL and MAC), we accept a small number of bit differences in the Hamming distance when searching for the input operands of each individual layer. The result shows the first and last layer of the neural network have higher sensitivity to approximation due to their direct impacts on input and output signals. In contrast, the middle layer is less sensitive for the approximation. Based on this observation, our design allows to apply different levels of approximation for each layer, so that it maximizes energy saving by better trading with accuracy.

D. Resistive NNCAM Design

In this section, we describe the details of the proposed NNCAM block which approximately performs the neural network computation by looking up the closest stored patterns

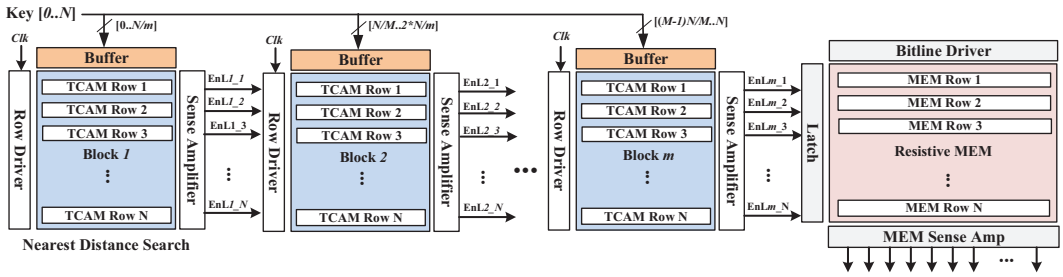

Fig. 5. Details of NNCAM structure in m pipeline stages.

to input operands. Fig. 5 shows the structure of the proposed NNCAM consisting of m pipeline stages.

Assume that we have l input operands to be computed, say $I_1, I_2, \cdots, I_l$. The bit representation of an input operand of N bits, I_i, is divided into m blocks of N/m bits, and the search operation starts from the most significant block, i.e., the first block. If the search operation for the first block of I_1 is hit, the rows of the next CAM stages are selectively activated. Then, in the next cycle, the second stage performs the search operation for the second block of I_1. At the same cycle, the search for the first block of I_2 is issued in the first CAM. Similarly, in the third cycle, the first block of I_3, the second block of I_2, and the third block of I_1 are processed individually in each CAM through the pipeline stages. The search continues until it finds a single active row at the final CAM stage. Finally, the hit on the last stage activates a row of the resistive memory which stores the preprocessed computation results updated by the layer-based associative update technique (Sec. III-B). This selective row activation of the pipeline stages reduces the number of active TCAM rows, thus significantly saving the energy consumption of the NNCAM blocks. In addition, since multiple input operands can be processed in the pipeline by up to m instructions, it also improves the search performance.

We used a CAM proposed in [21] to search for the nearest Hamming distance value to the input operands of each neural network computation. In particular, neural networks typically exploit floating point values as input operands, and the inaccuracy of the binary representation may significantly reduce the quality of the neural network computations. For example, the exponent part of the floating point values has more sensitivity than the fraction part. In addition, for each part which is represented as an unsigned integer value, the most significant bits have higher impact on computation than other bits. Thus, our first design goal is to find the row which has the smallest difference from the given operands of each neuron. The second design challenge is that, although we could always find the nearest Hamming distance value, the approximate computation sometimes needs to stop to avoid high error, with consideration of the sensitivity of each layer. For example, in an extreme case, we can compute all operations approximately without using the FPUs. However, it may significantly degrade the accuracy of the application. The accuracy mainly depends on the size of the TCAM which stores the frequent patterns. For example, increasing the NNCAM size improves computation accuracy, but an NNCAM with many rows consumes more energy and slows down the search operation due to the delay of the input buffers and interconnections. In order to guarantee enough accuracy with a reasonable TCAM size, our design selectively assigns operations to the precise FPUs by disabling the clock gating when the given operands are far from all patterns stored in NNCAM. Our design enable tuning accuracy by allowing the hits which are close enough in the first NNCAM stage. In case of the hit within a threshold distance, the input operands are processed on NNCAM. Otherwise, the search operation stops in the next NNCAM stages and it is processed by the FPUs [21].

IV. Experimental Results

A. Experimental Setup

We integrated the proposed NNCAM with the floating point units of an AMD Southern Island GPU which has been commercially used, e.g., Radeon HD 7970 device. Our design can be also implemented in other general GPU architectures. We run neural network implementations using Open*CL* to evaluate the efficiency of NNCAM. We simulate the proposed GPGPU architecture based on Multi2sim, a cycle accurate CPU-GPU simulator [28]. We also implemented the kernel code of the neural network implementation which works with NNCAM by supporting the runtime layer-based updates and selective approximation. The NNCAM block has been integrated beside each FPU in NNCAM on four pipeline stages. We extracted the most frequent patterns for ADD, MUL, and MAC FPUs for each neural network layer. ADD and MUL have 2 input operands while MAC accepts three inputs. Therefore, their corresponding NNCAM blocks handle 64 bits and 96 bits respectively. We evaluated energy of FPUs using Synopsys Design Compiler and optimized for power using Synopsys Prime Time for 1 ns delay in 45-nm ASIC flow [30]. The circuit level simulation of the CAM design has been performed on HSPICE simulator in 45-nm technology.

B. Benchmark and Framework

We evaluate the efficiency of the proposed design for a CNN implementation, *LeNet-5* [31], where each layer has trainable weights. We use `MNIST` dataset [29] as input data which are 32x32 pixel images of hand-written digit characters. We trained *LeNet-5* with 60*K* training images, and it recognizes the digits with about 95% accuracy for 10*K* tested image samples. The execution flow of NNCAM has two main steps:

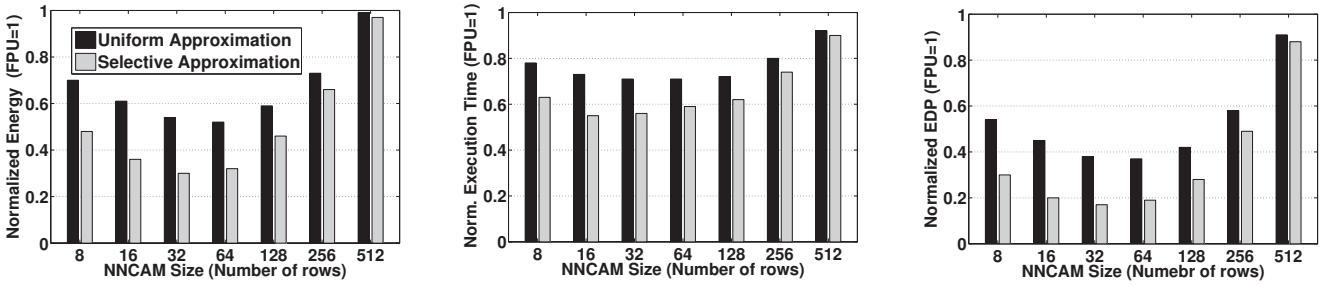

Fig. 6. Normalized energy consumption, execution time and energy-delay product of partially running CNN on FPU and NNCAM.

design time profiling and runtime reuse. We first execute an Open*CL* kernel and the C++ based host code with trained network coefficients to profile the input operands and their corresponding results for NNCAM based on the input dataset. We used 10% random images from the `MNIST` dataset. Then, the host code saves and ranks the input patterns for each FPU based on their occurrences. To evaluate the accuracy of the proposed design, we compare with the results exactly computed on the original GPU. The accuracy of neural network algorithms depends on the ratio of running application on NNCAM and FPUs, which is tunable by the *ST* threshold values used for each layer. In our evaluation, we determined the threshold values based on the sensitivity analysis discussed in Sec. III-C, so that the neural network is accelerated with less than 2% loss in quality, i.e., the degraded percent in the number of recognized images, for all evaluated configurations.

C. Evaluation of GPU acceleration

We first evaluate the energy and performance efficiency of the proposed design. Fig. 6 compares the energy consumption, execution time and energy-delay product. In order to understand the impact of the selective approximation discussed in Sec. III-C, we compare to the uniform approximation, which applies the same level of approximation for all layers, by using the same *ST* signals with the given quality loss of 2%. For small TCAM sizes from 8 rows to 32 rows, increasing the size saves more energy and improves performance of the enhanced GPGPU architecture. This is because larger sizes can increase the hit rates in the CAMs, thus assigning more instructions to the NNCAM. Table I shows the ratio of the instructions which run on NNCAM instead of using FPUs, on average for the three FPUs. For example, the selective approximation of the 32-row case can assign more instructions by 23% compared to the 8-row case. However, from 64 rows, the energy and the performance starts degrading because the highly frequent patterns are already stored. One way to determine the best NNCAM size is to use energy-delay product (EDP). In this criteria, using 32 rows results in 6× better EDP improvement compared to traditional GPU. In this NNCAM size, the proposed GPU design can show 70% energy savings and 44% speedup. In addition, our evaluation also shows that, with the same level of the CNN accuracy, the selective approximation achieves higher energy saving and speedup than the uniform approximation. For example, for the same level quality of loss of 2%, the selective approximation technique, which differently relaxes the level of approximations for each neural network layer, achieves higher energy and performance efficiency by 1.8× and 1.3× respectively, compared to the maximum case of the uniform approximation.

TABLE I
THE RATIO OF RUNNING CNN ON NNCAM IN DIFFERENT SIZES, ENSURING LESS THAN 2% QUALITY LOSS.

Approximation	8 rows	16 rows	32 rows	64 rows	128 rows	256 rows	512 rows
Uniform	68%	58%	48%	41%	32%	25%	17%
Selective	46%	31%	22%	16%	10%	6%	4%

TABLE II
ENERGY SAVING AND PERFORMANCE IMPROVEMENT OF ENHANCED GPGPU IN DIFFERENT NEURAL NETWORKS

CNN	Energy Saving	Performance Improv.	EDP Improv.
Convolutional LeNet-5	70%	44%	5.9×
Convolutional net Boosted LeNet-4	73%	39%	6.0×
Conv4x4-pool-380-40-10	67%	42%	5.2×
Conv7x10-pool-1210-120-10	62%	33%	3.9×

Table II summarizes energy savings, performance, and EDP improvement of four other CNN implementations running on the proposed GPGPU. We use *LeNet-4*, *LeNet-5* and two other listed CNN designs [31] with the `MNIST` database. For each CNN, we apply the layer-based update and selective approximation techniques to ensure the same amount of quality loss by 2%. Our evaluation shows running these four neural networks on the proposed hardware can achieve 68% of energy savings, 40% of speed up and 5.3× of EDP improvement compared to the traditional GPGPU on average.

D. Pure NNCAM Computation

One feasibility for the proposed NNCAM is that we may accelerate the entire computation of the neural network without using any processing cores on GPU. For example, we may place NNCAM close to main memory in order to locally process data without data movement between GPU cores

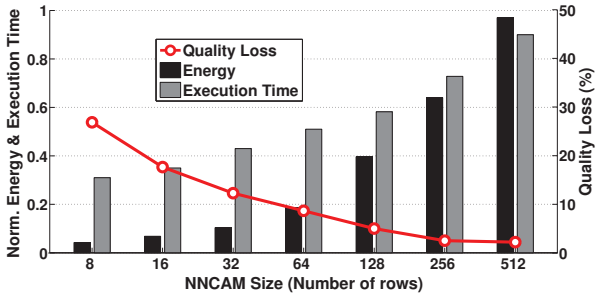

Fig. 7. Normalized energy consumption and execution time of NNCAM computing in different sizes.

and memory. The data movement cost is one main concern for emerging BigData applications such as *IoT* streaming applications [1], [2]. Fig. 7 shows the energy and execution time of the proposed design running the CNN for different row sizes. For each size, both energy and execution time are normalized to the original GPU which does not use any associative memory. The result shows that increasing the NNCAM size improves computation accuracy by storing more patterns. However, NNCAM, which uses many rows, degrades the energy and latency, because the large NNCAM consumes higher energy and has longer execution time due to the large buffer size needed to distribute input data among all rows simultaneously. Therefore, the size of NNCAM shows a trade-off relationship for the energy/performance and the accuracy. To ensure the quality loss below 2%, our design requires more than 512 rows, decreasing energy by 3% and reducing execution time by 10%, compared to the original GPU. However, although the improvement is less than the proposed GPU-based acceleration for this configuration, the pure NNCAM acceleration may have several application domains. For example, if the application can allow less accurate results, e.g., around 5%, our design requires an NNCAM block with 128 rows, and it can save energy by 60%. Thus, some application domains which may accept less accurate results can exploit the pure NNCAM acceleration, e.g., IoT embedded systems which energy efficiency is more crucial.

V. Conclusion

In this paper, we propose a neural network acceleration technique on GPU using a resistive CAM accelerator, NNCAM, which has the capability of performing memory-based computation. NNCAM approximately models the basic computations of neural networks, stores frequent patterns and searches for the closest similarity to reuse the results. Our design selectively assigns a block of computations to FPUs to guarantee the accuracy of the computed results. To further improve the efficiency of the proposed design in running neural networks, we propose layer-based updates and selective approximation techniques. The layer-based updates improve computation locality by filling the NNCAM adaptively for each neural network layer, thus increasing the hit rate in CAMs. The selective approximation technique relaxes computations by considering the sensitivity of each layer. Our evaluation shows that the proposed design integrated with the AMD Southern Island GPU can achieve 68% of energy savings and 40% of speed up, while providing less than 2% accuracy loss for four CNN implementations. The application of the proposed NNCAM is not limited to the neural network applications. For other applications, which are executed with multiple phases, our platform can be co-designed with the GPU software to dynamically update the frequent patterns and select the proper approximation of search layer to find the optimal trade-off of energy and performance with accuracy.

VI. Acknowledgment

This work was supported by NSF grant #1527034 and Jacobs School of Engineering UCSD Powell Fellowship.

References

[1] L. Atzori, et al., "The internet of things: A survey." *Elsevier Computer networks*, Vol. 54, no. 15, pp. 2787-2805, 2010.
[2] M-D. Assuno, et al., "Big data computing and clouds: Trends and future directions," *Elsevier Journal of Parallel and Distributed Computing*, pp. 3-15, 2015.
[3] M. Imani, et al., "Maximum-Likelihood Adaptive Filter for Partially Observed Boolean Dynamical Systems," *IEEE Transactions on Signal Processing*, vol. 65, no. 2, pp. 359, 2017.
[4] M. Oquab, et al., "Learning and transferring mid-level image representations using convolutional neural networks," *IEEE CVPR*, pp. 1717-1724, 2014.
[5] Y. LeCun, et al., "Convolutional networks and applications in vision," *IEEE ISCAS*, pp. 253-256, 2010.
[6] S. Ji, et al., "3D convolutional neural networks for human action recognition," *IEEE TPAMI*, pp. 221-231, 2013.
[7] B. Rouhani, et al., "DeLight: Adding Energy Dimension To Deep Neural Networks," *ACM ISLPED*, pp. 112–117, 2016.
[8] A. Krizhevsky, et al., "Imagenet classification with deep convolutional neural networks," *NIPS Proceedings*, pp. 10971105, 2014.
[9] M. Imani, et al., "Exploring Hyperdimensional Associative Memory," *IEEE HPCA*, 2017.
[10] N. Khoshavi, et al., "Read-Tuned STT-RAM and eDRAM Cache Hierarchies for Throughput and Energy Enhancement," *arXiv:1607.08086*, 2016.
[11] N. Khoshavi, et al., "Bit-Upset Vulnerability Factor for eDRAM Last Level Cache Immunity Analysis," *IEEE ISQED*, 2016.
[12] N. Mozaffari, et al., "More Efficient Testing of Metal-oxide Memristor-based Memory," *IEEE TCAD*, 2016.
[13] M. Imani, et al., "MASC: Ultra-low energy multiple-access single-charge TCAM for approximate computing," *IEEE/ACM DATE*, pp. 373-378, 2016.
[14] M-A. Bhuiyan, et al., "Acceleration of spiking neural networks in emerging multi-core and gpu architectures," *IEEE IPDPSW*, pp. 1-8, 2010.
[15] D-C. Ciresan, et al., "Flexible, high performance convolutional neural networks for image classification," *IJCAI*, vol. 22, no. 1, 2011.
[16] J. Schmidhuber, et al., "Multi-column deep neural networks for image classification," *IEEE CVPR*, pp. 3642-3649, 2012.
[17] R. Andri, et al., "YodaNN: An Ultra-Low Power Convolutional Neural Network Accelerator Based on Binary Weights," *arXiv:1606.05487*, 2016.
[18] M. Imani, et al., "Resistive configurable associative memory for approximate computing," *IEEE DATE*, pp. 1327-1332, 2016.
[19] T. Kohonen, et al., "Content-addressable memories," *Springer Science & Business Media*, vol. 1, 2012.
[20] A. Goel, et al., "Small subset queries and bloom filters using ternary associative memories, with applications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 1, pp. 143154, 2010.
[21] M. imani, et al., "Resistive CAM Acceleration for Tunable Approximate Computing," *IEEE TETC*, 2016.
[22] X. Yin, et al., "Exploiting ferroelectric FETs for low-power non-volatile logic-in-memory circuits," *IEEE/ACM ICCAD*, 2016.
[23] X. Yin, et al., "Design of latches and flip-flops using emerging tunneling devices," *IEEE DATE*, pp. 367–372, 2016.
[24] M. Valad Beigi, et al., "Tapas: Temperature-aware adaptive placement for 3d stacked hybrid caches," *ACM MEMSYS*, 2016.
[25] M. Valad Beigi, et al., "TESLA: Using Microfluidics to Thermally Stabilize 3D Stacked STT-RAM Caches," *IEEE ICCD*, 2016.
[26] J-M. Arnau, et al., "Eliminating redundant fragment shader executions on a mobile GPU via hardware memoization," *ACM/IEEE ISCA*, pp. 529-540, 2014.
[27] M. Imani, et al., "Approximate Computing using Multiple-Access Single-Charge Associative Memory," *IEEE TETC*, 2016.
[28] R. Ubal, et al., "Multi2Sim: a simulation framework for CPU-GPU computing," *ACM PACT*, 2012.
[29] Y. LeCun, et al., "MNIST handwritten digit database," Available: http://yann.lecun.com/exdb/mnist, 2010.
[30] Synopsys User Guide, See: "http://www.synopsys.com".
[31] Y. LeCun, et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 22782324, 1998.