Leveraging MLC STT-RAM for Energy-efficient CNN Training

Hengyu Zhao and Jishen Zhao
University of California, San Diego
{h6zhao, jzhao}@eng.ucsd.edu

ABSTRACT

Graphics Processing Units (GPUs) are extensively used in training of convolutional neural networks (CNNs) due to their promising compute capability. However, GPU memory capacity, bandwidth, and energy are becoming critical system bottlenecks with increasingly larger and deeper training models. This paper proposes an energy-efficient GPU memory management scheme by employing MLC STT-RAM as GPU memory to accommodate the image classification training workloads. We propose a data remapping scheme that exploits the asymmetry access latency and energy across soft and hard bits in MLC STT-RAM cells and the memory access characteristics in image classification training workloads. Furthermore, our design enables (i) energy-efficient memory access by leveraging bit-level similarity in training data and (ii) optimal feature map encoding to compress the contiguous 0s in feature maps.[1] Our design reduces VGG-19 and AlexNet training time, GPU memory access energy and capacity utilization by 76% and 70%, 45% and 40%, 26.9% and 26%, respectively.

1 INTRODUCTION

Recent development of convolutional neural networks (CNNs) is radically altering the way we process various applications, such as image classification, speech recognition, object detection, and computer vision. Among these, image classification is one of the most widely targeted application domains in modern CNNs [19, 24, 25]. Software developers strive to improve CNN training performance and accuracy by adopting larger and deeper neural networks with more parameters. As a result, the training of large-scale CNN models is typically performed by graphic processing units (GPUs) with promising compute capability [19].

However, the continuous scaling of training networks makes training workloads increasingly data intensive, exposing GPU memory capacity, bandwidth, and energy as critical system bottlenecks [18, 23]. Figure 1(a) shows the memory capacity demand of recent winners of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [14], including AlexNet [19], GoogLeNet [25], VGG-16 and VGG-19 [24] (experimental setup is described in Section 5). When the batch size of VGG-19 reaches 128, even a single NVIDIA's GTX 1080 Ti (11 GB device memory) cannot meet the memory demand. The memory demand can further increase with the design of recent ILSVRC winners, which adopt more than a hundred convolutional layers [13]. Moreover, the increase of memory capacity demand also increases the bandwidth demand and dynamic energy consumption in GPU memory access. GPUs typically have limited on-chip storage resources (such as caches, register files, and shared memories), which cannot hold the large working set of CNN training workloads. As a result, CNN training can impose high memory bandwidth demand (Figure 1(b)) and memory power consumption (Figure 1(c)).

The goal of this paper is to improve the performance, energy efficiency, and capacity utilization of CNN training for image classification, without sacrificing any of these metrics. Also, this paper will not change the model accuracy, because we do not change the values of training data. To this end, we design a multi-level cell (MLC) STT-RAM-based GPU

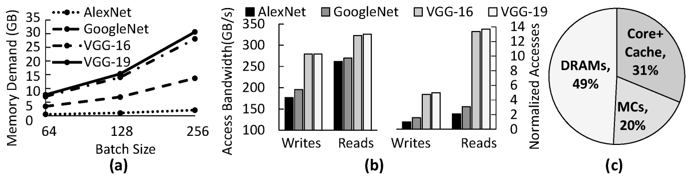


Figure 1: GPU memory demand and system energy breakdown. (a) GPU memory utilization. (b) GPU memory bandwidth demand and accesses. (c) NVIDIA GTX 1080 Ti dynamic power breakdown when executing VGG-19.

memory architecture with lightweight modifications to the memory controllers and memory banks. We propose a data remapping scheme, which reduces memory traffic and access latency by categorizing and mapping different types of data in different manners in MLC STT-RAM. Our scheme further enables two GPU memory optimization mechanisms: (i) an energy-efficient memory access mechanism, which reduces memory access energy consumption by avoiding the unnecessary writes at the bit level; ii) a sparsity-aware data encoding mechanism, which increases effective memory capacity with data encoding by exploiting the sparsity in feature maps. This paper makes the following contributions:

- We propose a data remapping scheme that stores various types of data of training workloads in MLC STT-RAM, based on their different access characteristics. Our remapping scheme enables energy and capacity efficient MLC STT-RAM data access.
- We present two memory access optimization mechanisms based on our data remapping scheme: i) BitLevel-leverage the asymmetric write current and access latency; ii) SparseCode-exploits the sparsity in feature maps to compress the data without quality loss.
- We develop a set of lightweight hardware implementations and software support to facilitate our mechanisms.

2 BACKGROUND AND MOTIVATION

2.1 Convolutional Neural Networks

Deep neural networks(DNNs) have various types, such as recurrent neural networks(RNNs), convolutional neural networks(CNNs), etc. This paper focuses on feedforward-style CNNs commonly used in image classification [19, 24, 25]. CNNs consist several layers, including convolutional layers that perform image convolution, activation layers to make neural networks nonlinear with activation functions, pooling layers to reduce the feature map size by down sampling, and fully connected layers to analyze features and classify input images into groups. CNNs have two phases: training and inference. Training allows CNNs to learn and update weights with multiple layers of neural networks, through forward and backward propagations with opposite traverse directions (Figure 2). Forward propagation generates feature maps using weights and backward propagation updates the weights. Inference employs the trained models to perform new recognitions or classifications. This paper focuses on studying the training phase, which is typically much more compute and data intensive than inference phase.

Forward and backward propagation. In forward propagation, the output feature map y can be obtained by multiplying an input feature map x and multiple convolutional filters. Then, y is fed into the next layer as input feature map. When forward propagation of a layer completes, a

[1]This work is supported in part by NSF 1652328 and 1718158, and SRC/DARPA Center for Research on Intelligent Storage and Processing-in-memory.

loss function will generate an output that is calculated by feature maps of that layer and validation dataset. Then, a gradient map is obtained by chain rule:

$$\frac{\partial Error}{\partial Y_{(N-1)}} = \frac{\partial Error}{\partial Y_{(N)}} \cdot \frac{\partial Y_{(N)}}{\partial Y_{(N-1)}}$$

Where *Error* is the sum of the network's prediction error over all training examples, $Y_{(N-1)}$ is the output feature map of layer $(N-1)$, and $Y_{(N)}$ is the output feature map of layer N.

Because the output $\frac{\partial Error}{\partial Y_{(N-1)}}$ is the product of the input $\frac{\partial Error}{\partial Y_{(N)}}$ and $\frac{\partial Y_{(N)}}{\partial Y_{(N-1)}}$, this derivation step can require reading feature maps from memory and writing updated weights into the memory. With the chain rule, layer $(N-1)$ can then get its own gradient map $\partial Y_{(N-1)}$ with its feature map $Y_{(N-1)}$, and pass the $\partial Y_{(N-1)}$ to layer $(N-2)$ as its input. When backward propagation reaches to the first layer, weights of all layers need to be updated with new values for this whole iteration.

Input images and feature maps. Input images of image classification training workloads can be color digital images, which consists of a matrix of pixels represented by three color channels: red (R), green (G), and blue (B). Each channel can be represented by an 8-bit binary. When training a CNN model, the input images are typically divided into several sets ("batches") that are processed independently. Increasing the compute batch size and the number of network layers can typically improve training accuracy [23]. Feature maps are the results after convolution computations on input images; they consist of data blobs that contain multi-dimensional feature information, but stored in memory as 2D arrays. Each input image will generate multiple feature maps after convolution computation in each layer. The number of feature maps equals the number of convolution kernels in each layer.

2.2 MLC STT-RAM

MLC STT-RAM cells store multiple logic bits (typically two bits) in each cell, increasing the density of STT-RAM (Figure 3). Also, MLC STT-RAM has great endurance. MLC MTJ can adopt either series [15] or parallel [10] designs. Series MLC STT-RAM has been demonstrated to be more feasible than the parallel implementation, because series design is compatible with advanced MTJ technologies, such as perpendicular MTJ and has overwhelming advantages in read and write reliability.

Read and write operation of MLC STT-RAM cell. In Figure 3(a), the two MTJs of series MLC MTJ structure have different areas to distinguish the two logic bits. The bit stored in the smaller MTJ is a *soft bit*; the bit in the bigger MTJ is a *hard bit*. Given a constant resistance-area product and a critical switching current density (I_C), the soft bit has a higher resistance than the hard bit. Therefore, the soft bit is typically the more significant bit (MSB) and requires a smaller switching current ($I_{C,soft} < I_{C,hard}$). Both read and write operations of series MLC STT-RAM contain two steps. For example, in the first step of a write operation (Figure 3(b)), a large current I_{WH} ($I_{WH} > I_{C,hard}$) is applied to switch the hard bit; at the same time, the soft bit gets the same value as the hard bit. In the second step, a smaller current I_{WS} ($I_{C,soft} < I_{WS} < I_{C,hard}$) is used to flip only the soft-bit. This step will not disturb the value stored in hard bit. Figure 3(c) illustrates a two-step read operation based on voltage sensing, which requires three reference voltages (Ref-0, Ref-1, and Ref-2) and two comparisons. In the first step, the soft bit is detected by comparing the sensing voltage with Ref-0. In the second step, the hard bit is read by comparing the sensing voltage with either Ref-1 or Ref-2 based on the result of the first step.

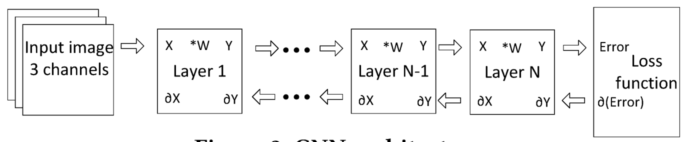


Figure 2: CNN architecture.

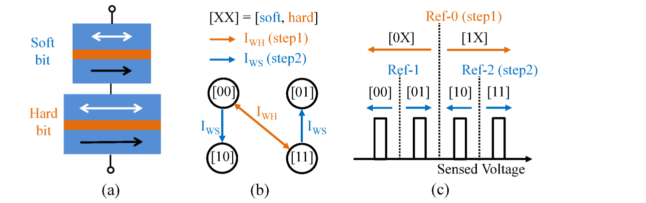


Figure 3: Multi-level cell STT-RAM. (a) Series MLC MTJ structure; (b) Two-step write operation; (c) Two-step read operation.

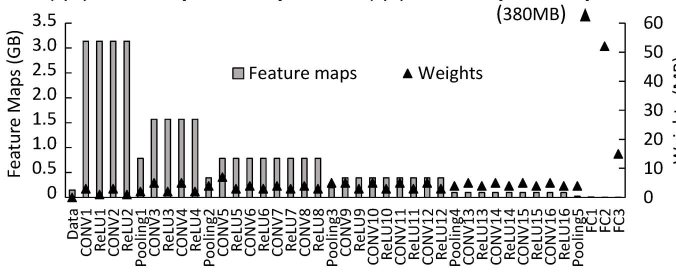


Figure 4: VGG-19 memory utilization in each layer. AlexNet profiling shows similar trends in data access across layers.

MLC STT-RAM as main memory. STT-RAM has been considered as a promising replacement of DRAM in main memory system design. Compared with DRAM, STT-RAM has lower leakage power and no radiation-induced soft errors. With the MLC technology, the density can be doubled compared with SLC STT-RAM. MLC PCRAM also offers high density and low leakage benefits. However, the technology impose much longer access latency and higher dynamic energy than DRAM and MLC STT-RAM. Furthermore, the writing mechanisms of MLC PCRAM cells do not have the asymmetric properties as MLC STT-RAM. Therefore, we adopt MLC STT-RAM as our GPU main memory technology.

2.3 Motivation

We motivate our design based on following observations:

- **Feature maps cost most of memory capacity** (Figure 4). Therefore, improving the capacity utilization and energy efficiency of feature map access and storage can significantly improve GPU memory system performance and energy efficiency.
- **Impact of CNN functions on neighbor data similarity.** ReLU function will generate continuous 0s. Also, a maxpooling function takes the maximum value in each window of certain sizes and generate a new feature map with a smaller size than the original one. As a result, data with large values can be clustered together. It is likely that the higher order bits share the same bit values.
- **CNNs are read-intensive applications** (Figure 1(b)). Therefore, it is critical to optimize read performance and energy consumption.
- **Issues and opportunities with MLC STT-RAM cell.** We observe that, reading the soft bit only takes one step, while writing the soft bit only requires a small switching current that will not flip the hard-bit. Thus, MLC STT-RAM can perform in a similar way as SLC by only accessing the soft-bits, which improves both access speed and the energy efficiency. Furthermore, MLC STT-RAM has a *write disturbance* issue: writing the hard bit with a large write current can also change the soft bit in the memory cell to be the same value.

3 OUR DESIGN

The goal of our design is to reduce the GPU memory capacity utilization and energy consumption, while improving the performance of CNN

Table 1: Overview of categories of data in MLC STT-RAM, remapping schemes, and access mechanisms.

Data Type	Memory Utilization	Reuse	Remapping Scheme	Access Modes
Dense feature map (CONV layers)	High	Write once, multiple reads	Remap across soft and hard bits	BitLevel
Sparse feature map (CONV layers)	High	Write once, multiple reads	Remap across soft and hard bits	SparseCode + BitLevel
Feature map (FC layers)	Low	Write once, multiple reads	Remap to soft bits	Fast soft bit access
Weights (CONV layers)	Low	Multiple reads and writes	Remap to soft bits	Fast soft bit access
Weights (FC layers)	High	Multiple reads and writes	Remap across soft and hard bits	BitLevel
Input images and gradient maps	Low	Write once, read once	Remap to soft bits	Fast soft bit access
Other	Low	Multiple reads and writes	No remapping	Normal access

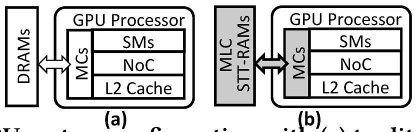


Figure 5: GPU system configuration with (a) traditional DRAM-based device memory and (b) proposed MLC STT-RAM-based device memory (shaded are modified components).

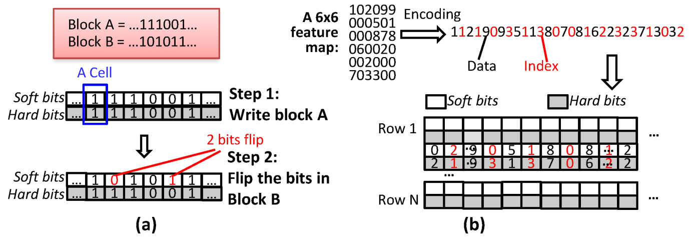


Figure 6: (a) BitLevel and (b) SparseCode mechanisms.

training. To this end, we propose an data remapping scheme that efficiently maps various types of training data across GPU main memory regions and the bits in each memory cell, based on characterization of various types of training data stored in the memory. Furthermore, our design also enables two optimization mechanisms, BitLevel and SparseCode, to improve the performance and energy efficiency of training data access. Hardware implementation and software interface of our design will be discussed in Section 4.

3.1 Data Remapping and Access Modes

Table 1 lists major categories of data stored in GPU memory based on our workload profiling. Feature maps and weights are two critical categories of data stored in GPU memory (Figure 4). In particular, feature maps are the primary memory consumer in convolutional layers, while weights utilize the most of the GPU memory space in fully connected layers. Note that high memory capacity utilization typically leads to high GPU memory bandwidth utilization [23]. Table 1 also illustrates our proposed data remapping scheme and optimized data access modes of each type of data.

Feature map. Feature maps are the results after computations of the same set of input images. We observe that neighbor data blobs (groups of 32-bit floating point values) in feature maps can share similar or even the same bit values. Based on this observation, we propose the following options for remapping and accessing feature maps:

- **Option 1:** Dense feature maps (without many contiguous 0s) in convolutional layers (including ReLU and pooling layers) – Remap the feature map data in a manner, where every two corresponding bits in a pair of neighbor data blobs are stored in the soft and hard bits in each memory cell. As a result, the soft and hard bits of the same memory cell can have the same bit value, which enables our BitLevel optimization mechanism (Section 3.2).
- **Option 2:** Sparse feature maps (with a large number of contiguous 0s) in convolutional layers – Encode the feature maps with SparseCode (Section 3.2), and then remap the feature map data the same manner as Option 1.
- **Option 3:** Feature maps in fully connected layers – Write them into soft bits to reduce the access energy and latency, as feature maps have insignificant memory utilization in fully connected layers.

In convolutional layers, we employ the Option 1 by default. In case the memory controller identifies substantially low number of contiguous 0s in the first 65536 data blocks in feature map writing flow (details discussed in Section 4), our design will switch to Option 2. The switching of remapping options between convolutional and fully connected layers does not require data migration – we simply write the output feature maps of a convolutional computation to the soft bits.

Weights. Weights have low memory utilization in convolutional layers. Therefore, we remap them in the soft bits to reduce access latency and energy. In fully connected layers, where weights are the primary memory space consumer, we observe that weights also have substantial neighbor bit-level value similarity. Therefore, we remap the weights to soft and hard bits of MLC STT-RAM cells in a similar way as feature maps in convolutional layers. The difference is that neighbor weights are simply a pair of 32-bit floating point values, instead of larger-granularity data blobs.

Input images and gradient maps. Input images and gradient maps have much lower memory utilization than feature maps. In particular, input images and gradient maps of VGG-19 consume up to 6% and 7% the memory space used to store feature maps, respectively. Furthermore, input images are only accessed at the beginning of the workload; the corresponding memory space is recycled once the images are read. In backward propagation, the generated gradient map for layer N+1 is no longer useful once the backward propagation reaches layer N. GPU systems can potentially recycle the corresponding memory space. Therefore, we remap these data in soft bits of memory cells to ensure low access latency and energy.

Other data. CNN workloads can also generate other types of data, such as parameters and intermediate results of compute kernels. However, these data consumes insignificant space and access energy in GPU memory compared to the aforementioned types [23]. Therefore, we store such data in normal manner without data remapping.

3.2 Energy-efficient Memory Access

Our data remapping scheme allows us to adopt the following memory access mechanisms to improve memory capacity utilization, access performance, and energy consumption.

BitLevel: Energy-efficient access enabled by bit-level remapping. Figure 6(a) illustrates an example employing this mechanism. Assuming contiguous data blocks A and B – e.g., data blobs in feature maps of convolutional layers or 32-bit floating point values in weights of fully connected layers – our remapping scheme will store the neighbor data blocks A and B in the soft and hard bits of the same memory cells. Due to write disturbance in MLC STT-RAM, a single write to the hard bit (i.e., a bit in Block A) will also write the same value to the soft bit

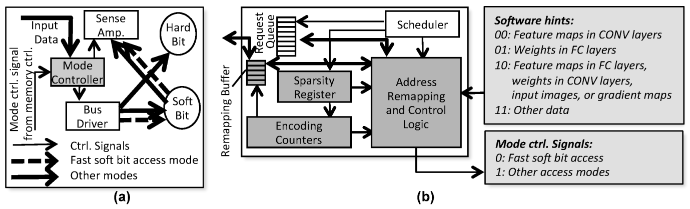


Figure 7: (a) MLC STT-RAM bank organization and (b) memory controller design.

(i.e., corresponding bit in Block B) in the same cell [3]. Therefore, a single step write to the hard bits will write all the bits of Block A, along with substantial amount of bits in Block B. We only need to perform a second-step write to flip a small amount of soft bits with different values from the corresponding hard bits in the same cell. While conventional DRAM-based memory requires to write each bit value, our design allows us to eliminate the writes to a substantial portion of bits, significantly reducing the latency and energy consumption of GPU memory access.

SparseCode: Sparsity-aware feature map encoding. SparseCode exploits the sparsity in feature maps yielded by ReLU functions to encode the large amount of 0s. We observe that the number of 0s in feature maps is significantly increased after applying ReLU functions. Furthermore, the 0s tend to gather in contiguous memory areas. Based on these observations, we compress feature maps by adopting two data structures, *index* and *data* (similar to previous studies [8, 21]), to store feature maps in the memory. *Data* stores each non-zero values (32-bit floating points), while *index* records the number of contiguous 0s between two non-zero values. Figure 6(b) shows an example of SparseCode encoding. Each data (black) is followed by an index (red) in the encoded format. The encoding mechanism does not require extra metadata to identify across index and data when reading the feature maps, because the every 32-bit data is uniformly followed by an index (4 bits in our evaluation). Note that weights can be compressed with the same encoding policy. However, we do not observe substantial energy and performance improvement by compressing weights, due to their low memory utilization in convolutional layers. Therefore, our design only employ the SparseCode on feature maps.

To increase the bit-level similarity in soft and hard bits the same cells, we store neighbor indexes in soft and hard bits of one set of cells, while storing neighbor data blocks in another set. In fact, the encoded data can have substantial neighbor block value similarity. For example, the indexes are 4-bit integers. The neighbor indexes are highly possible to share substantial amount of the same bit-level values.

4 IMPLEMENTATION

4.1 Memory Bank Organization

Figure 7 (a) illustrates an MLC STT-RAM bank incorporated with the components that implement our design mechanisms. We implement two modifications to the memory banks. First, we add a mode controller implemented by a set of switches to distinguish the fast soft bit access mode with other modes that access both soft and hard bits. Second, we adopt an optimized MLC STT-RAM peripheral circuit design [4] to implement the two-step writes in our BitLevel mechanism. By adjusting the biases on the word line and bit line, the circuit design allows a writing current with different amplitudes and directions to be applied to an MTJ. It is also possible to avoid writing a memory cell, if the original soft and hard bit values of the cell are the same as the new values. However, doing so can introduce extra read operations to compare the original with the new values. Therefore, our design does not consider such optimizations. The rest of the memory bank organization stays the same as conventional MLC STT-RAM designs.

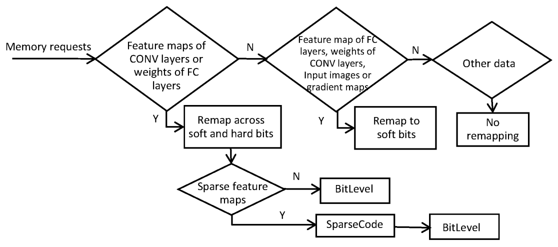

Memory requests
Feature maps of CONV layers or weights of FC layers
N
Feature map of FC layers, weights of CONV layers, Input images or gradient maps
N
Other data
Y
Remap across soft and hard bits
Y
Remap to soft bits
No remapping
Sparse feature maps
N
BitLevel
Y
SparseCode
BitLevel

Figure 8: Control flow of logic design.

4.2 Memory Controller Design

Our design requires the memory controller support to 1) perform physical address remapping, 2) generate control signals of various access modes, 3) choose between Option 1 and Option 2 of feature map access, and 4) encode/decode the data flow of feature maps accessed by the SparseCode mechanism. Figure 7 (b) illustrates our modification to memory controllers.

Address remapping and control logic. We add an address remapping logic in the memory controller to calculate the new physical address of data accesses, based on our address remapping scheme and software hints of data categories (Section 4.3). The address remapping logic also generates access mode control signals, which distinguish between access to the soft bits only and those to both bits in memory cells. In addition, we implement encoding and decoding logic used by SparseCode. The encoding logic inserts the indexes generated by encoding counters among non-zero data values; the decoding logic inserts 0s into the request queue according to the indexes. Figure 8 shows the control flow of the address remapping and control logic.

Remapping buffer. We adopt a buffer to store the remapped memory requests (output to or input from the MLC STT-RAM). The size of each buffer entry is the same as the request queue entry. The buffer also stores encoded feature map data, when accessed by SparseCode mechanism. We implement a 16-entry buffer, which is sufficient based on our performance evaluation.

Sparsity register. We employ a 16-bit register to determine whether a feature map is sufficiently sparse for adopting the SparseCode mechanism. The register counts the number of non-zero values with a index that is smaller than a predefined threshold. After investigating the first 65536 (2^{16}) accessed data values, we identify that a feature map is dense if the register value is larger than certain threshold. In our evaluation, most of the feature maps are identified as sparse when we set the thresholds of index and register values to be eight and 32768, respectively.

Encoding counters. We add a 4-bit counter to count the number of 0s between two non-zero values in the data flow sent to each memory bank. Our SparseCode mechanism employs the counter results to generate index values. Whenever reaching a non-zero value in the data flow, we simply reset the counter. If there are more than 15 continuous 0s, the counter not only should be reset but also change the following data value to 0.

4.3 Software Support

Our design requires software hints on the categories of data being accessed. We implement the hints as annotations added in in CNN frameworks, such as Caffe [16] and TensorFlow [1]. For example, before a convolutional layer outputs a feature map, we mark first data blob with `begin_feature(i, N)` and define the size (the number of data blobs) of the feature map. Based on such information, the address remapping logic in the memory controller can calculate the address range of the feature map.

Table 2: System configuration.

CPU	Intel Xeon E5-2620 V3@2.4GHz
Main memory	16GB DDR4
Operating system	Ubuntu 16.04.2
GPU	NVIDIA GeForce GTX 1080 Ti (Pascal)
GPU cores	28 SMs, 128 CUDA cores per SM, 1.5GHz
L1 cache	24KB per SM
L2 cache	4096KB
Memory interface	8 memory controllers, 352-bit bus width
GPU main memory	11GB GDDR5X

Table 3: Memory access parameters comparison.

	MLC STT-RAM	DRAM
Read energy (pJ/bit)	0.51	1.39
Write energy (pJ/bit)	Soft Transition: 1.81 Hard Transition: 2.69	0.33
Read speed (ns/bit)	3.2	7.1
Write speed (ns/bit)	Soft Transition: 9.3 Hard Transition: 19.2	7.1

5 EVALUATION

5.1 Experimental Setup

VGG-19 and AlexNet. Our experiments evaluate VGG-19 [24] and AlexNet [19], two of the latest winners of ILSVRC [14]. The applications perform image classification and localization. The workloads are widely used in recent studies on CNN training and image classification [18, 23].
Datasets. We employ ImageNet as our training datasets. It is a huge image dataset which contains millions of images belong to thousands of categories. Beginning at 2010, ILSVRC [14] has been held annually. ILSVRC exploits a subset of ImageNet with 1.3 million training images, 50000 validation images, 150000 testing images in 1000 categories. Every category has about 1300 of training images and 50 validation images.
Training framework. We adopt Caffe [16] as our training framework. Caffe is a widely-used deep learning framework developed by Berkeley AI Research.
Real machine configuration. Table 2 lists the details of our system configuration.

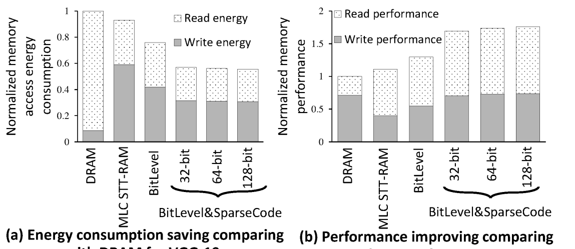
5.2 Performance and Energy Modeling

Our evaluation adopts the MLC STT-RAM cell performance, energy, and area parameters provided by prior MLC STT-RAM designs [2, 7]. We scale the parameters to 32nm technology node the same way as previous studies [4]. We employ NVSim [9] to calculate the required MLC STT-RAM parameters as listed in Table 3. To evaluate the performance of GPU systems with MLC STT-RAM, we developed an in-house script with a performance model incorporated with MLC STT-RAM-based GPU memory and memory controllers. We obtain memory access statistics from real-machine profiling and feed the statistics into our performance model to estimate the memory access performance of our proposed design. To evaluate GPU processor power and energy, we employ the power statistics obtained by NVIDIA profiler [20]. We calculate the dynamic energy of MLC STT-RAM using our energy parameters listed in Table 3 and our performance results.

5.3 Results

In the following result analysis, we compare among the following memory configurations:

- DRAM – traditional GDDR5X-based GPU memory.
- MLC STT-RAM – MLC STT-RAM-based GPU memory without our data remapping, BitLevel, and SparseCode mechanisms.

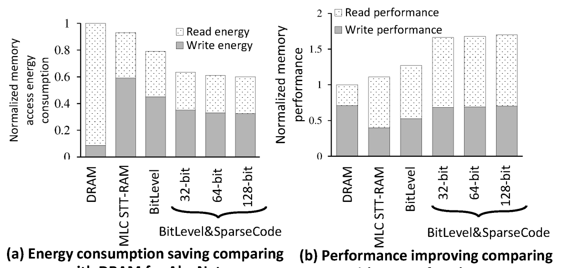


(a) Energy consumption saving comparing with DRAM for VGG-19. **(b) Performance improving comparing with DRAM for VGG-19.**

Figure 9: Energy consumption saving and performance improving comparing with DRAM for VGG-19.

- BitLevel – MLC STT-RAM-based GPU memory with our data remapping and BitLevel mechanisms.
- BitLevel&SparseCode – MLC STT-RAM-based GPU memory with all our proposed mechanisms.

Energy consumption. We only show the memory access energy consumption reduction and performance improvement in the following sections, because memory access is the main bottleneck for energy consumption and performance (Figure 1), also our proposed modifications to memory controller only introduces negligible overhead. We observe that naïvely replacing main memory with MLC STT-RAM can hardly improve energy efficiency and performance from Figure 9 10 Although MLC STT-RAM can lead to lower read energy than DRAM-based memory, it significantly increases write energy because substantial amount of writes need to be performed in two steps. For VGG-19, with our data remapping scheme and BitLevel mechanism, we reduce 26% write energy compared with the naïve replacement. Combining our SparseCode mechanism with BitLevel can reduce memory dynamic energy consumption by 45% compared to the baseline DRAM-based design, with further energy savings in both reads and writes. For AlexNet, our design can reduce memory dynamic energy consumption by 40% compared to the baseline (Figure 10). We also perform a sensitivity study on the size of data blocks in BitLevel mechanism, ranging from a single 32-bit floating point value to 128-bit floating point values. As shown in Figure 9(a), the increase of the data block size will further reduce the memory energy consumption.
Memory access performance. We evaluate the memory access performance of our design by taking into account the latency of both memory access. Again, figure 9(b) shows that naïve memory replacement with MLC STT-RAM only introduce 11% speedup on average write latency of VGG-19 memory access. Yet, the direct replacement can improve read performance by 1.4×, despite the 47% write performance degradation. Our data remapping with BitLevel mechanism improves both read and write performance, leading to 30% improvement of overall memory access speedup. Combining SparseCode can further improve access performance. Overall, our design results in 76% and 70% speedup of memory access for VGG-19 and AlexNet, respectively.
In-memory data characteristics. Figure 11 shows the bit-level neighbor data similarity for VGG-19. Overall, the average similarity across all layers can reach up to 67.3%. As shown in Figure 4, between two maxpooling layers, convolutional layers and ReLU layers have almost same memory capacity utilization. Therefore, the number of write operations in order to store the feature maps can also be similar. Figure 12 illustrates the compression ratio of each ReLU layer with the SparseCode mechanism for VGG-19. Across all layers, the mechanism can up to 26.9% compression ratio with 128-bit value (data block) size. This leads to substantial reduction in memory accesses and capacity consumption. AlexNet also has the same trend in bit-level similarity and ReLU layer compression ratio, so we do not show them in this section.
Memory capacity utilization. In Figure 12, we present the compression ratio with three different bit-width: 32-bit, 64-bit and 128-bit, which



(a) Energy consumption saving comparing with DRAM for AlexNet. (b) Performance improving comparing with DRAM for AlexNet.

Figure 10: Energy consumption saving and performance improving co[illegible]

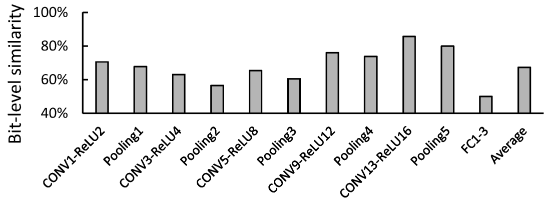


Figure 11: Bit-level similarity of every layer for VGG-19.

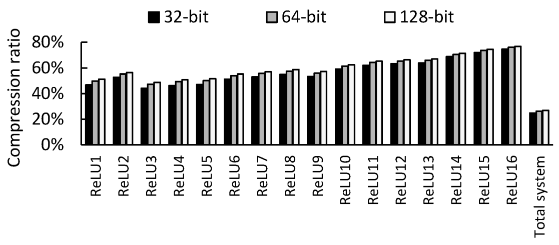


Figure 12: Compression ratio of each ReLU layer for VGG-19.

can accommodate various requirements for precisions of different networks. When the bit-width is increasing, the compression efficiency will also be improved, because SparseCode uses a 4-bit index to represent data with longer bit length. With our proposed SparseCode mechanism, the total system's feature map compression ratio of VGG-19 and AlexNet can reach to 26.9% and 26% with 128-bit length, respectively. What is more, due to the unique cell architecture of MLC STT-RAM, with same quantity of memory cells, the memory capacity of MLC STT-RAM main memory will be doubled than traditional DRAM main memory.

Memory bandwidth. Figure 1(b) shows that memory read bandwidth is higher than memory write bandwidth, so in this section we will analyze read and write bandwidth respectively. With our proposed scheme, for VGG-19, memory read bandwidth can be reduced to 33% of DRAM based main memory bandwidth. However, due to the poor write performance of MLC STT-RAM, memory write bandwidth increases by 14% comparing with DRAM based main memory bandwidth. Considering CNNs are memory read intensive applications (Figure 1(b)), total memory bandwidth of VGG-19 and AlexNet will be still lower than DRAM based main memory bandwidth with 45%, 40%, respectively.

Memory area efficiency. We use NVSim [9] to obtain MLC STT-RAM area model and area efficiency. With the same size of 2GB, DRAM costs area of 419 mm^2 and MLC STT-RAM only costs 372 mm^2. Also, the area efficiency of MLC STT-RAM is 37.8%, is higher than DRAM's 25.2%.

6 RELATED WORK

A large body of previous works endeavor to reduce memory intensity and energy consumption of neural networks. Network pruning [11, 12] aims to reduce the memory consumption through pruning small valued weight connections in neural networks. Other redundancy alleviating methods exploit reduced precision [17] to reduce the number of bits required by corresponding neural networks. In addition, recent works proposed a variety of CNN inference accelerators [5, 6, 21, 22]. These designs improve the energy efficiency and performance of neural networks. However, none of these studies focuses on the memory issues of CNN training. Moreover, most of previous works focus on improving the efficiency of storing the weights in neural networks. However, in training phase, most of the memory space is used by storing feature maps. Most previous works do not explore the impact of feature maps on performance and energy of CNNs. Finally, prior design with reduced precision policies can result in the decrease of CNN accuracy. Our design does not sacrifice precision, when improving memory access performance and energy efficiency.

Rhu *et al.* demonstrated that the increase of memory demand of neural network training phase can outpace the development of commodity GPU systems. Their study proposed a runtime memory manager to virtualize the memory usage of DNNs and schedule CPU memory and GPU memory simultaneously [23]. This study reduces the average GPU memory usage, by trading off performance, energy consumption and memory bandwidth. However, our design simultaneously improves memory energy, performance, and capacity utilization.

7 CONCLUSION

GPU memory capacity, bandwidth, and energy consumption are becoming critical system bottlenecks with increasingly larger-scale CNN trainings. In this paper, we propose an energy-efficient GPU memory management scheme that leverages the access asymmetry of soft and hard bits in MLC STT-RAM technology and the characteristics of image classification training workloads. The evaluation results shows that our proposed scheme can improve the performance of VGG-19 and AlexNet training by 76% and 70%, respectively. Meanwhile, 45% and 40% GPU memory access energy and 26.9% and 26% capacity utilization of VGG-19 and AlexNet have been reduced, respectively.

REFERENCES

[1] Martin Abadi, Paul Barham, and Jianmin Chen et al. 2016. TensorFlow: A system for large-scale machine learning. In *OSDI 16*. 265–283.

[2] Xunchao Chen, Navid Khoshavi, Ronald F DeMara, et al. 2017. Energy-Aware Adaptive Restore Schemes for MLC STT-RAM Cache. *IEEE Trans. Comput.* 66, 5 (2017), 786–798.

[3] Xunchao Chen, Navid Khoshavi, and Ronald F DeMara et al. 2016. Energy-Aware Adaptive Restore Schemes for MLC STT-RAM Cache. *IEEE Trans. Comput.* (2016).

[4] Yiran Chen, Xiaobin Wang, et al. 2010. Access scheme of multi-level cell spin-transfer torque random access memory and its optimization. In *MWSCAS*. 1109–1112.

[5] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ISCA*. IEEE, 367–379.

[6] Ping Chi, Shuangchen Li, et al. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ISCA*. 27–39.

[7] Ping Chi, Cong Xu, and Tao Zhang et al. 2014. Using multi-level cell STT-RAM for fast and energy-efficient local checkpointing. In *ICCAD*. IEEE, 301–308.

[8] Minsoo Rhu et al. 2017. Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks. (2017).

[9] Xiangyu Dong et al. 2012. NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. *TCAD* (2012).

[10] Xiaohua Lou et al. 2008. Demonstration of multilevel cell spin transfer switching in MgO magnetic tunnel junctions. *Applied Physics Letters* (2008).

[11] Song Han, Huizi Mao, and William J Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR* (2016).

[12] Stephen José Hanson et al. 1989. Comparing biases for minimal network construction with back-propagation. In *Advances in neural information processing systems*. 177–185.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.

[14] ImageNet Large Scale Visual Recognition Challenge(ILSVRC). 2017. (2017). http://www.image-net.org/challenges/LSVRC/.

[15] T. Ishigaki et al. 2010. A multi-level-cell spin-transfer torque memory with series-stacked magnetotunnel junctions. In *VLSIT*.

[16] Yangqing Jia, Evan Shelhamer, and Jeff Donahue et al. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093* (2014).

[17] Patrick Judd et al. 2015. Reduced-precision strategies for bounded memory in deep neural nets. *arXiv preprint arXiv:1511.05236* (2015).

[18] Heehoon Kim, Hyoungwook Nam, Wookeun Jung, and Jaejin Lee. 2017. Performance Analysis of CNN Frameworks for GPUs. In *ISPASS*.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.

[20] NVIDIA, Profiler user's guide. 2017. (2017). http://docs.nvidia.com/cuda/profiler-users-guide/.

[21] Angshuman Parashar, Minsoo Rhu, and Anurag et al. Mukkara. 2017. SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. In *ISCA*.

[22] Brandon Reagen, Paul Whatmough, Robert Adolf, et al. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *ISCA*. 267–278.

[23] Minsoo Rhu and Natalia Gimelshein et al. 2016. vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design. In *MICRO*.

[24] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556* (2014).

[25] Christian Szegedy, Wei Liu, and Yangqing Jia et al. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.