Master of Science Thesis

Bit Transpose Unit Design for Approximate Memory Architecture

August 2018

Graduate School of Engineering
Seoul National University
Electrical and Computer Engineering Major
Nguyen Huy Hung

Bit Transpose Unit Design for Approximate Memory Architecture

Supervisor: Prof. Hyuk-Jae Lee

This work is submitted as a Master of Science thesis

August 2018

Graduate School of Engineering
Seoul National University
Electrical and Computer Engineering Major

Nguyen Huy Hung

Confirming the master's thesis written by
Nguyen Huy Hung
August 2018

Chair	Prof. Choi, Kiyoung
Vice Chair	Prof. Lee, Hyuk-Jae
Examiner	Prof. Kim, Jangwoo

Abstract

Deep Learning is one of the most successful methods in the artificial intelligence field. Both mature of machine learning algorithm and the development of GPU contribute to the triumph of many deep learning applications. When the deep learning applications became more heavily, the more requirement for the accessing in the DRAM also increase. Recent DRAM generation met the problem to control the influence of refresh power in the total power consumption of the DRAM. A novel idea [8] proposed a method to sacrifice a little accuracy in Convolutional Neural Network but it helps to save at least 70% percent of refresh power consumption. However, in the paper, the authors only show the prediction of the amount of saving power without real estimation. In this thesis, the real estimate of Bit Transpose Unit, which is an important part to work with approximate data. The modification in the DRAM architecture also displayed and estimated in the real simulation. A solution also proposed to overcome some problem of Bit Transpose Unit. The real estimation received from the simulation in both McSimA+ and DRAMSim2 simulations give an evidence for the real operation of the idea in [8].

Keyword: Deep Learning, Power Saving, Refresh Rate, Convolutional Neural Network

Student Number: 2015-22136

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

In the recent year, deep learning proves the outstanding ability in many difficult problems. One of the prevalent problems is image classification problem. Many researchers tried to propose various kind of neural networks to solve these problems. The most successful network is Convolutional Neural Network (CNN), which is proposed by Yann LeCun in 1998 in computer vision community. The network achieved the very high accuracy in MNIST dataset. The successful of CNN is extend with the work of Alex Krizhevsky in 2012. He proved deep neural network is possible to train and the network gains the highest accuracy in ImageNet competition at this time. The research community is inspired by the success of the deep neural network, so many different kinds of the neural network are presented to improve accuracy in the ImageNet dataset. For example, VGG and GoogleNet in 2014 and 2015 subsequently.

With the mature of deep learning application, the accessing to the DRAM also increase with very high speed. More accessing on the DRAM brings more power for ACTIVATE/PRECHARGE and dynamic power. The power for these operations accounts for 50% power for the DRAM operation [8]. In [8], the authors proposed the novel idea to save power in the DRAM by using different refresh rate.

My thesis is organized as follow. Chapter 2 summarized all necessary background for my work. It includes background about Convolutional Neural Network, basic knowledge of DRAM and the architecture of McSimA+ and DRAMSim2 simulations. In chapter 3, I explain how to evaluate the actual running power of DRAM with the Bit Transpose Unit for the idea in [8]. Many various version to work with Bit Transpose Unit is proposed. Chapter 4 is the simulation result and the explanation for the result in each case. Chapter 5 is the conclusion and future works.

Chapter 2: Background

In this chapter, I will cover some important background which is crucial to understand my work in next chapter. The core topic in the chapter includes the background of Convolutional Neural Network (CNN), Dynamic Random Access Memory (DRAM) architecture and Mcsim, DRAMSim2 simulators.

2.1. Convolutional Neural Network

In the section, the background of CNN will be summarized. The CNN is introduced first time by Yann LeCun in 1998. In the paper [1], he proposed a network to classify handwriting digit numbers (MNIST dataset) in ten classes from zero to nine. The structure of the network is shown in the figure below.

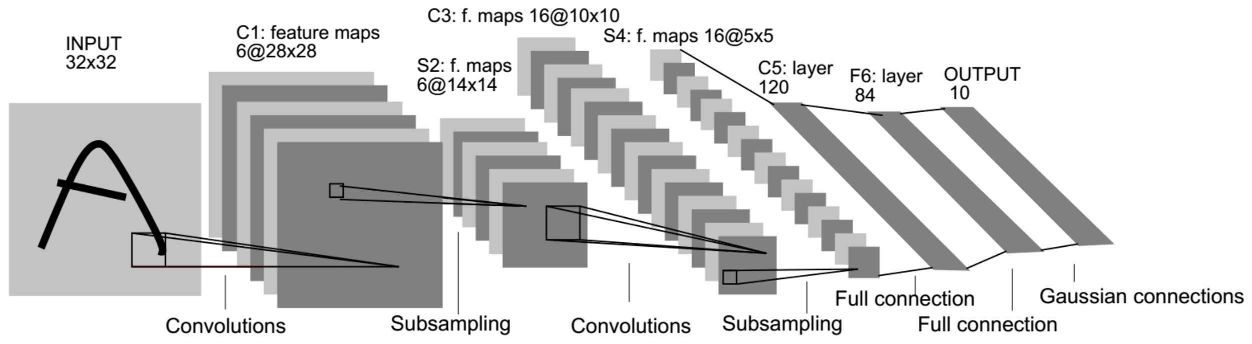


Figure 1: Architecture of LeNet-5, a Convolutional Neural Network [1]

From figure 1, the architecture of a CNN use an image as input with the size is 32x32. This layer also calls visible layer. In the next hidden layer, the author used six different kernels to create six feature maps. Each feature map is created by multiply a kernel (5x5) with input images. The output from convolution layer becomes input for next subsampling (pooling) layer. Most popular pooling method is max-pooling or average-pooling. The output of pooling layer is six feature maps with the size of 14x14. The similar process is applied with one convolutional and one subsampling layer subsequently. Two next layers are two fully connected layers. Output layer is the fully connected layer with ten neurons corresponding to ten classes which are needed to classify.

Three important characteristics contributed to the success of Convolutional Neural Network is:

- Use one kernel for all receptive region in each input feature maps. It helped to reduce the number of parameters, so reducing the training time. Furthermore, using one kernel have to increase the stability of network because we want the convolutional network to classify correctly when the object is translated, rotated.
- The appearance of pooling layer also helps to increase the stability of the network. Then, it contributed to the success of CNN.
- CNN consists of many hidden layers, so it can attract the high abstract level of the object, which has a major influence on the final result.

2.2. AlexNet, VGG, and GoogleNet

In the previous section, we have introduced a very famous Convolutional Neural Network (LeNet5). The network is very successful in MNIST dataset. However, in ImageNet competition, the network is not enough and a new requirement of a very deep network appears. ImageNet dataset includes over 1.3 million images in training set belong to 1000 classes. Therefore, a good neural network classifier must have the ability to capture the complex distribution of data in the dataset. This requirement leads to the appearance of the deep neural network. In theory, the deep neural network is the potential solution for the complicated problem such as ImageNet competition. But in practice, training a deep neural network is very slow and spend a lot of time.

Researchers also met an obstacle when they trained the deep neural network is the overfitting problem. All of these problems prevent the development of the deep neural network for a long time.

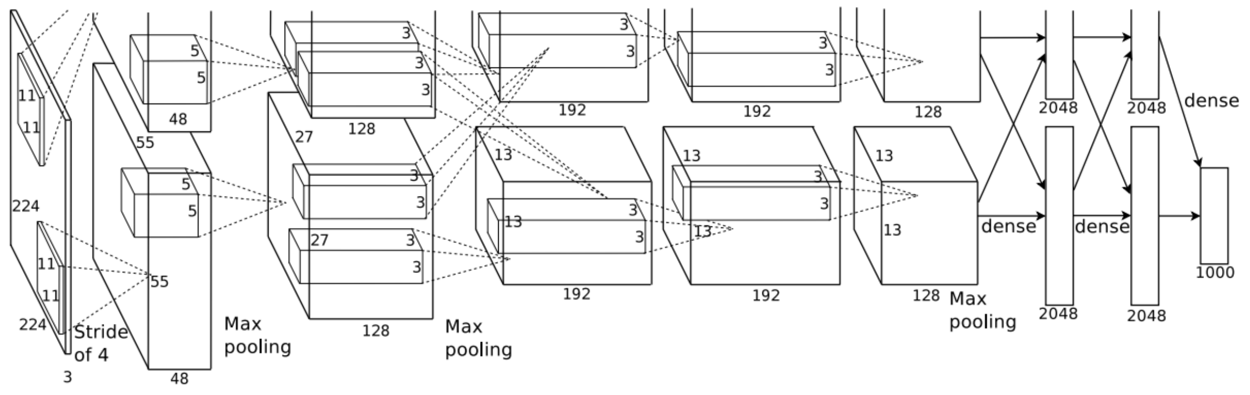


Figure 2: The architecture of AlexNet [2]

Fortunate, in 2012, Alex Krizhevsky has proved that a deep neural network (AlexNet) can be trained in an acceptable time. He used two synchronized GPU to accelerate training speed. Furthermore, he also proposed to use Local Normalization which helps to overcome overfitting problem. An output from the network became the winner in ImageNet competition in 2012 with very high accuracy. The success of the network shocked research community and it is the start event for the explosive in the growing of using the deep neural network in many research field, especially in image classification problem.

In 2014, Karen Simonyan presented a highest deep neural network at this time. The architecture of the network is shown in figure 3. A visible characteristic of the VGG network is the increasing many hidden layers when compared with the network of Alex Krizhevsky. Nevertheless, the important attribute of the VGG net is that its use 3x3 kernel for all convolutional layers. VGG received accuracy higher than AlexNet and one critical conclusion from the achievement of VGG is that the success of Convolutional Neural Network depends on the number of hidden layers in the network.

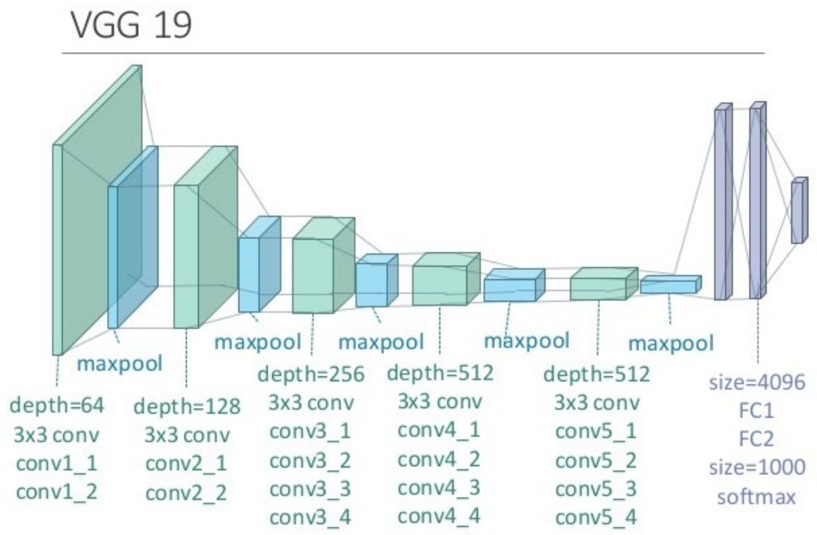


Figure 3: VGG19 Architecture [5]

Following the success of VGG net, Christian Szegedy introduced a very impressive network in 2015, which is called GoogleNet. The network has a strange structure when comparing with all convolutional neural network before. The structure of the network can see in [4]. The core item in the GoogleNet is the inception module, which is displayed in figure 4. Inception module is a combination of various kernel sizes. The advantage of inception module is the invariant of the classifier with the changing of scale, position in the image because the module uses different size of the kernel, so the inception module can attract the same feature with varying size in the same or different location. GoogleNet is the accumulation of many inception modules and it has 22 hidden layers. It is very impressive when comparing with VGG which has only 12 hidden layers. However, the number of GoogleNet's parameters is fewer than the VGG net. The evidence from

GoogleNet proves that deep architecture plays an important role in the success of Convolutional Neural Network.

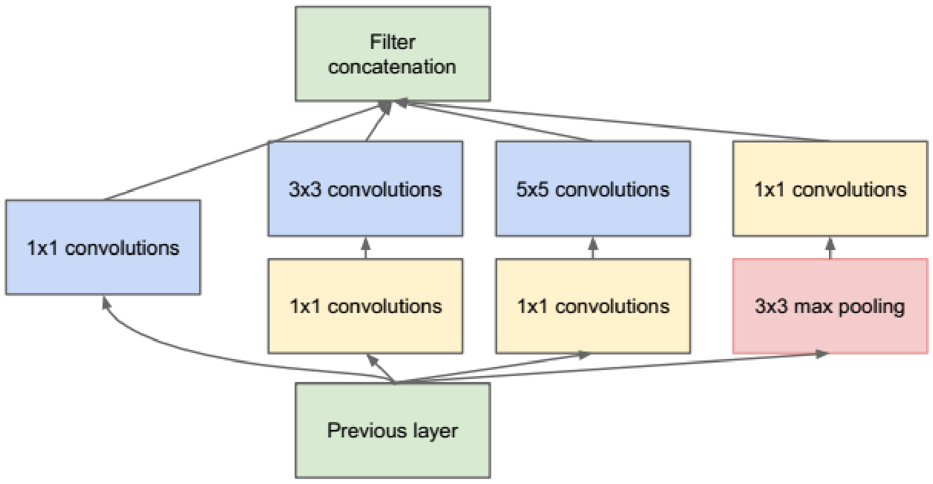


Figure 4: Inception Module [4]

2.3. Different Refresh Rate Solution

In [8], authors proposed a new memory architecture to reduce the power consumption by refresh operations by slowing down the refresh rate. The data is transposed to a two-dimensional array with the first row contains all the highest bit of all elements in the array. The significance of each row decreases from row 0 to the row 31. The refresh period (in milliseconds) is increased linearly as the row number increases:

$$RP(n) = \begin{cases} 64 & for\ 0 \leq n \leq 8 \\ (n-9) * incr + offset & for\ 9 \leq n \leq 31 \end{cases}$$

where RP(n) represents the refresh period of the n-th row and two parameters, incr and offset, are chosen experimentally.

2.4. McSimA+ and DRAMSim2 Simulations

In this section, we cover briefly some important information about McSimA+ and DRAMSim2 Simulations.

2.4.1. McSimA+ Simulation

McSimA+ is an application-level+ simulator, offering a middle ground between a full-system simulator and an application-level simulator. It is an event-driven many core application-level simulator. The simulation uses Intel's PinTool to interpret the binary instructions of any custom application running on the system. Binary instructions extracted by PinTool are used as input for a so-called PinTool Simulator (PTS). A PTS simulates an entire system, including processor cores, L1/L2 cache, directory, memory controller, etc. Many parameters of the system like queueing modes etc. can be modified. McSimA+ as PTS does not model the entire main memory hardware part but assumes fixed latencies for memory transactions and thus power/timing estimates are not as accurate as DRAMSim2.

The execution of the program in McSimA+ includes:

- A shell file links the program to be executed (stream.cc) to the interpreter (PinTool)
- McSimA+ simulates the program's execution
- McSimA+ and DRAMSim2 provide log files

2.4.2. DRAMSim2 Simulation

The simulation use clock-cycle simulator of the entire hardware architecture of DDR2/DDR3 main memory including the memory controller. Input for the simulation can either be trace files or other system/application simulators. Memory transaction processing is based on a general model of DDRx DRAM memory controller (patent). Datasheet parameters of existing DDR2/DDR3 chips can be used for simulation. Power/timing estimates are more accurate than McSimA+ since all transactions are processed entirely and no fixed bandwidth or timing is assumed.

Chapter 3: Bit Transpose Unit

In chapter 2, we have summarized all necessary background. In this chapter, I will show detail about how to implement bit transpose unit (BTU) and how to use BTU in McSimA+ and DRAMSim2 simulations.

3.1. Overall Computer Architecture with Bit Transpose Unit.

In the section, we introduce the position of bit transpose unit in general computer architecture and the function of bit transpose unit, which is shown in figure 5

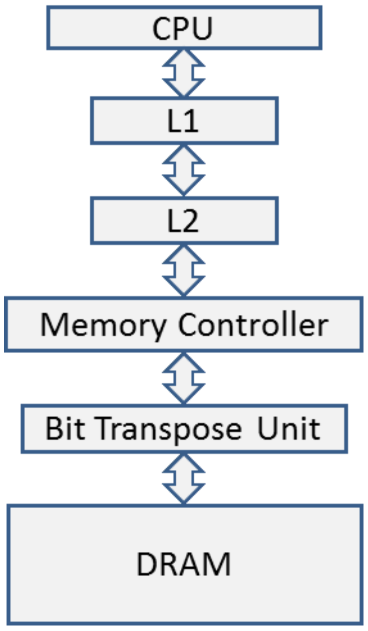


Figure 5: Overall Architecture with Bit Transpose Unit

First, CPU sends an instruction. If the L2 cache is miss, data will be read from DRAM. In this situation, Memory Controller will detect the requirement data from the L2 cache to know what the kind of data is. If the data is approximate, the reading and writing operation to DRAM will execute through Bit Transpose Unit. From the above figure, the location of bit transpose unit is between Memory Controller and DRAM. In the next section, the detail operation of Bit Transpose Unit will be introduced.

To more details, the hybrid architecture for approximate memory is shown in figure 6.

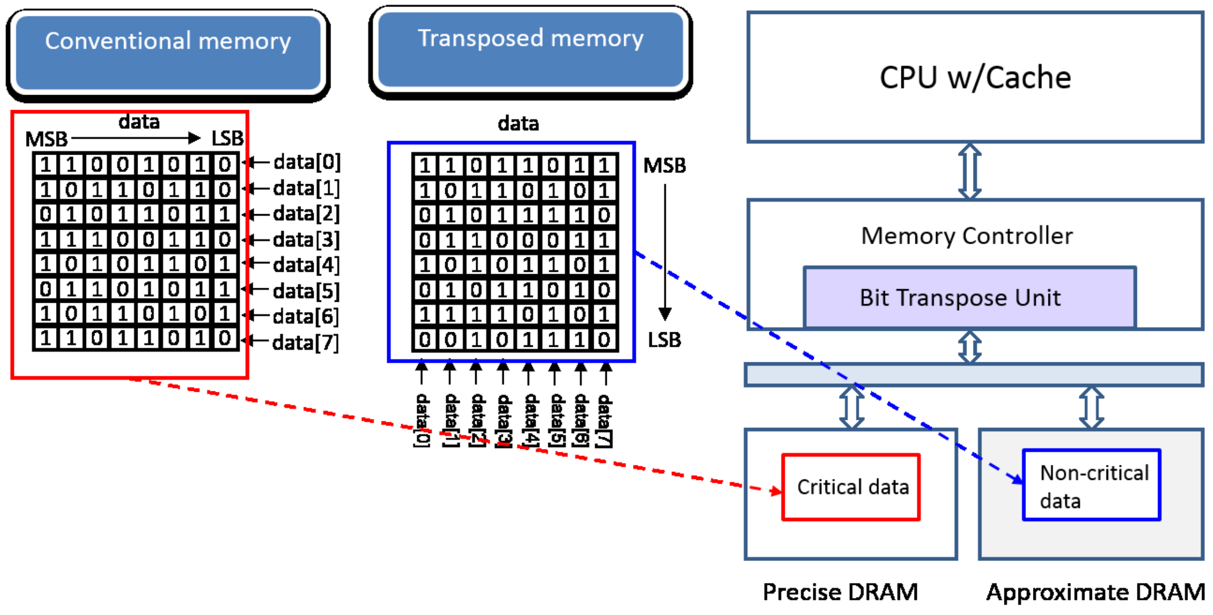


Figure 6: Hybrid Architecture for Approximate Memory [8]

3.2. Bit Transpose Unit

The detailed structure of Bit Transpose Unit (BTU) is displayed in the section. Firstly, we will concentrate on the relation between L2 cache and Bit Transpose Unit to understand why BTU's size is $32x512$. Second, the condition which is used to classify data between precise and approximation by Memory Controller also explains in detail.

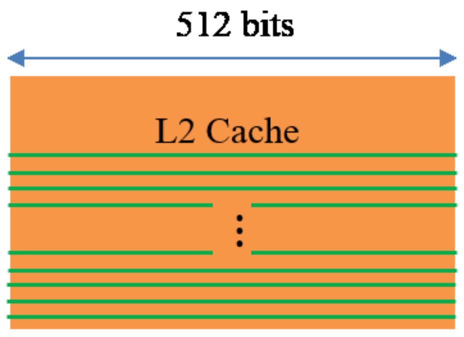


Figure 7: L2 Cache

As we know, each cache line has size is 512 bits or 64 bytes. Therefore, if we use 32 bits data, then each cache line contains 16 data elements. For simplicity, we will illustrate the reorganization with one cache line as in figure 7. Each cache row will transpose to become a column.

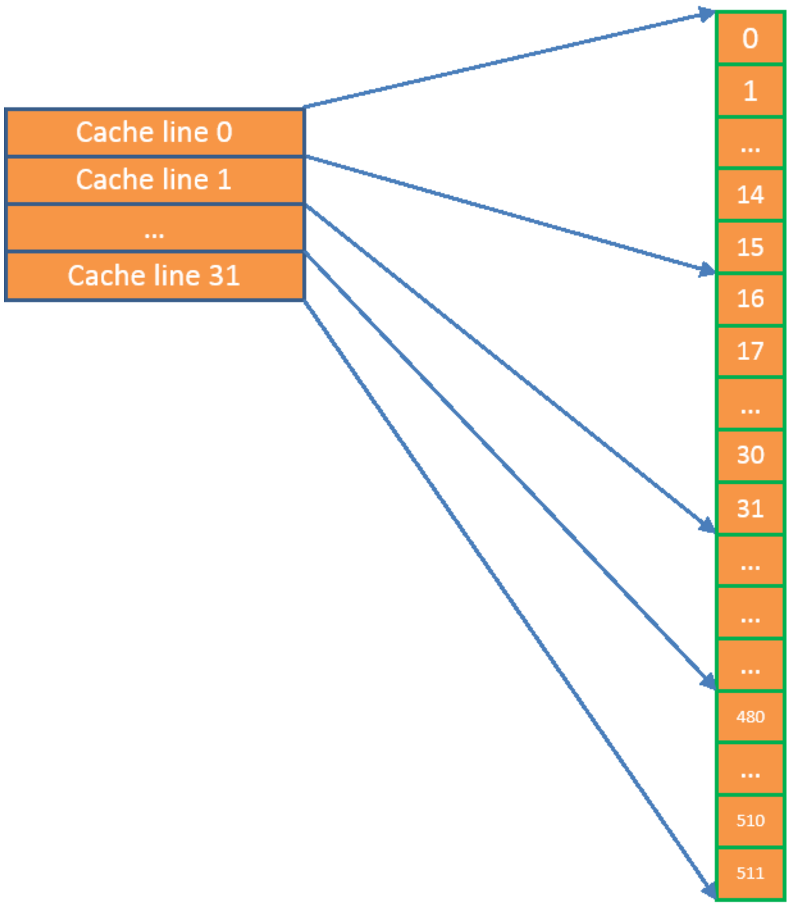


Figure 8: Convert 32 Cache Lines to Two-Dimensional Data

In the next step, we use the reorganization method in figure 8 for 32 subsequent cache lines. With cache line 0, we have data in a column with index from 0 to 15. Data in cache line 1 have the indices from 16 to 31. The same situation for other cache lines. After the process is finished, we have a column with the indices from 0 to 511. The column is two-dimensional array and size is

512×32. The explanation is summarized in figure 8. The two-dimensional array is transposed to another 32×512 two-dimensional array (figure 10). The new 32×512 array has the property that the row 0 is the highest bit of all 512 elements in the before array. The row 1 has the second highest bit and so on. The $32x512$ array has a critical advantage is that all equal high priority bit is put in the same row, so we can use different refresh rate for different rows depend on the important properties of each row.

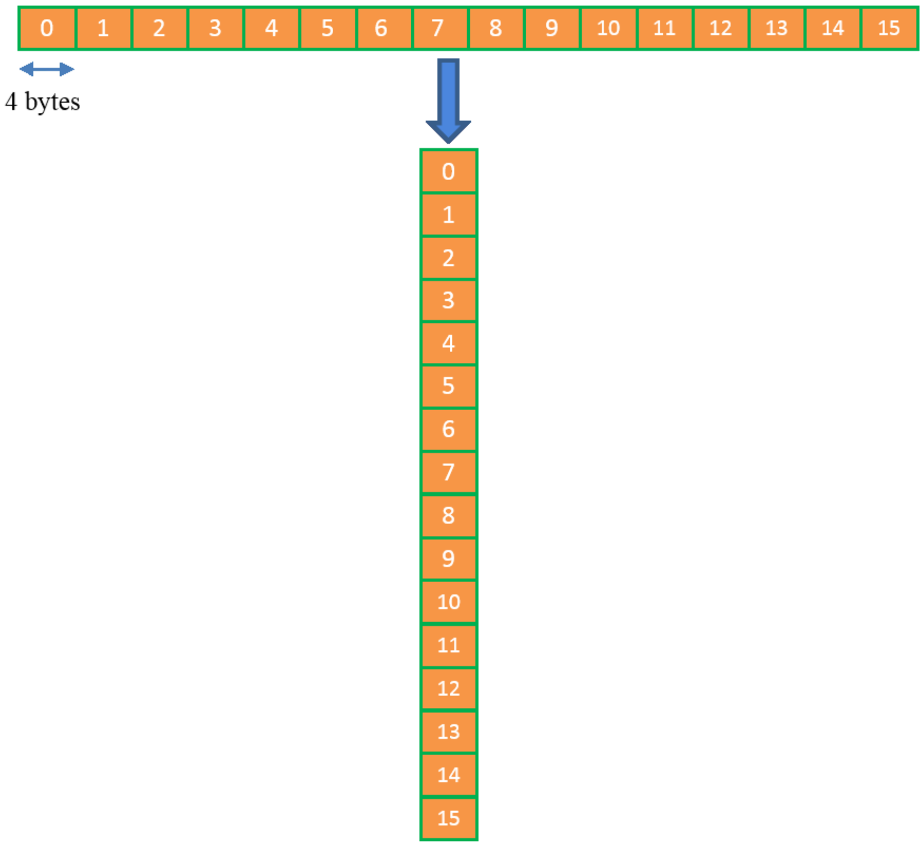


Figure 9: Convert Data in a Cache Row to a Column

From now, we will call the 32×512 array is a block for simplicity. When the data is approximate data, if L2 need to store data in DRAM, it will send data to bit transpose unit before sending data to DRAM. Similarly, when L2 need some approximate data from DRAM, data must be read to bit transpose unit before transferring data to L2 cache.

So, we have completed explaining how data from the L2 cache is transferred to Bit Transpose Unit. In the next step, we will focus on how Memory Control separates precise and approximate data. Actually, Memory Controller depends on the highest bit in the address of data to classify which kind of data. If highest bit is 1, then data is approximate. Otherwise, if highest bit is 0, data belongs the precise. From figure 11, it is obvious that if data is precise, the L2 cache will contact directly with the DRAM. However, if data is approximate, L2 cache need to contact with DRAM through Bit Transpose Unit. In the next section, we will introduce specifically the structure of DRAM to store both approximate and precise data.

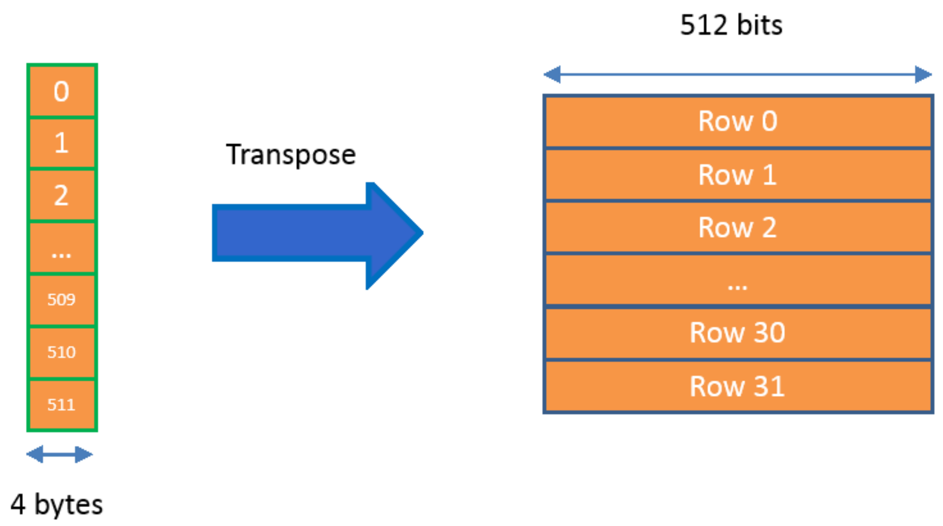


Figure 10: Transpose Data from 512x32 to 32x512

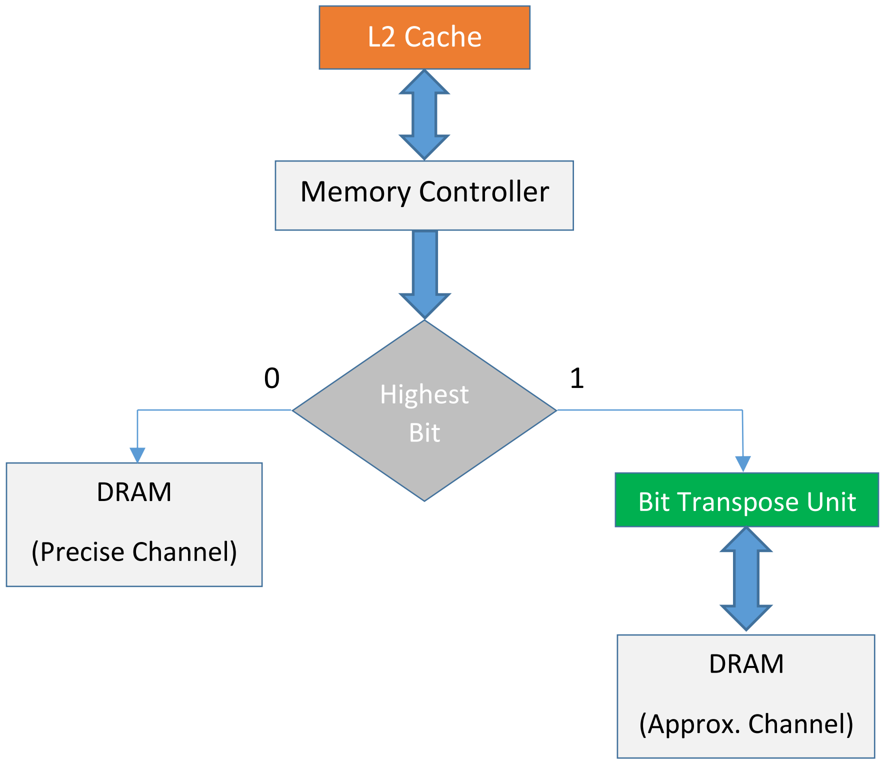


Figure 11: Memory Controller Control the Interaction between L2 Cache and DRAM

3.3. DRAM Architecture with Approximate and Precise Data

To suit the requirement of using both approximate and precise data in DRAM, we allocate data in DRAM in two channels. Channel 0 stores approximate data and channel 1 will be used to store precise data. Hence, L2 will read and write with approximate data only on channel 0 of DRAM. The main reason why we need to separate in two different channels because we can use different refresh rate in the different channel. By using different refresh rate in approximate data, we can save a lot of power when DRAM operates. If we use only one channel to store both approximate and precise data, it will become laboriously to control refresh rate in each row of DRAM. Another advantage of using two channels in DRAM is that we can exploit the locality of data in the precise data channel. This attribute is very useful characteristic of the much deep learning application.

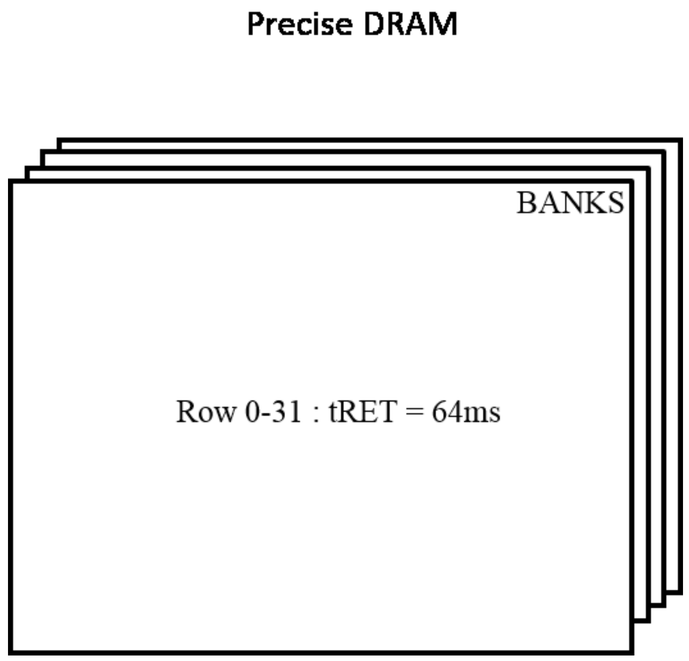


Figure 12: Precise Memory in DRAM

Note that because our target is to reduce power with reasonable accuracy, so in my work, I decided to employ approximate data channel for feature map in CNN network and the precise data channel store weight value of kernel. It is motivated by the fact that weight value in the kernel is applied for many positions in feature map, so it has a lot of influence on the output of the convolutional operation. Nonetheless, the value of feature map has smaller effect on the output's result than weight's value because CNN network has the ability to tolerate some inaccurate value from input feature maps. Furthermore, some pooling network, which usually uses max pooling, also helps the CNN network again unpredicted data created by using very high refresh rate on the lowest bit of data.

In figure 14, the architecture of DRAM is displayed. In channel 1, the structure of DRAM is similar to the normal DRAM. It also uses normal refresh rate (64ms) for every row in DRAM. However, in channel 0, the structure of DRAM is very different when compares with the normal

DRAM. DRAM is stored by each block (32×512) and each row in each block has different refresh rate followed by the ideal in [8].

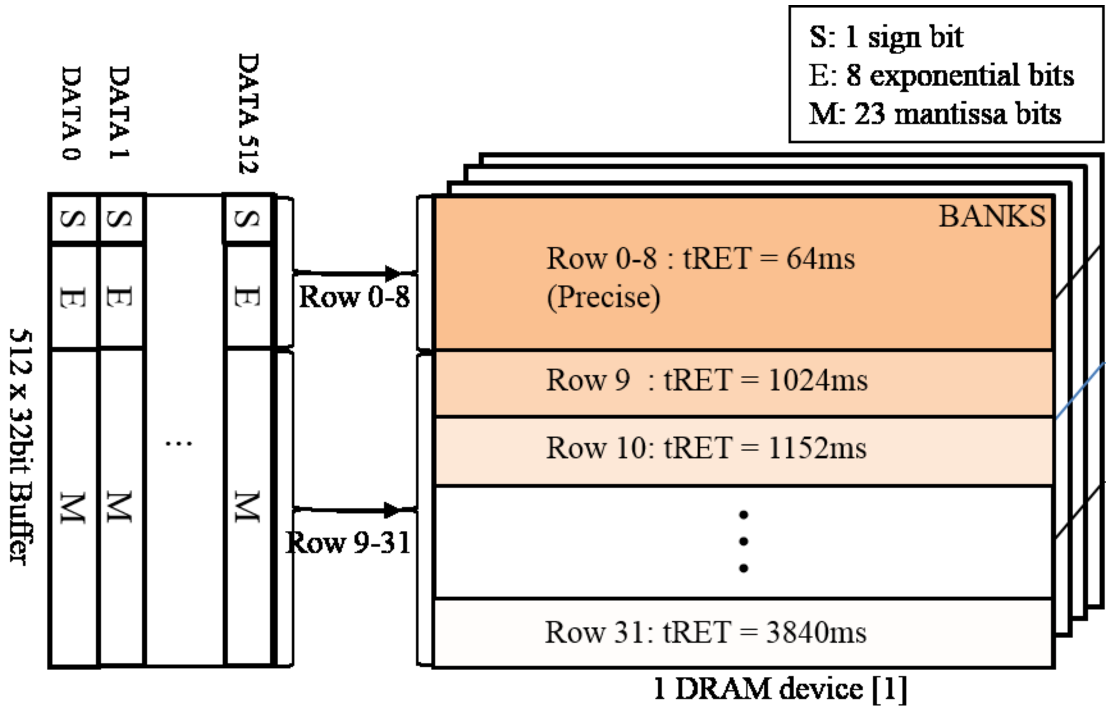


Figure 13: Approximate Memory in DRAM [8]

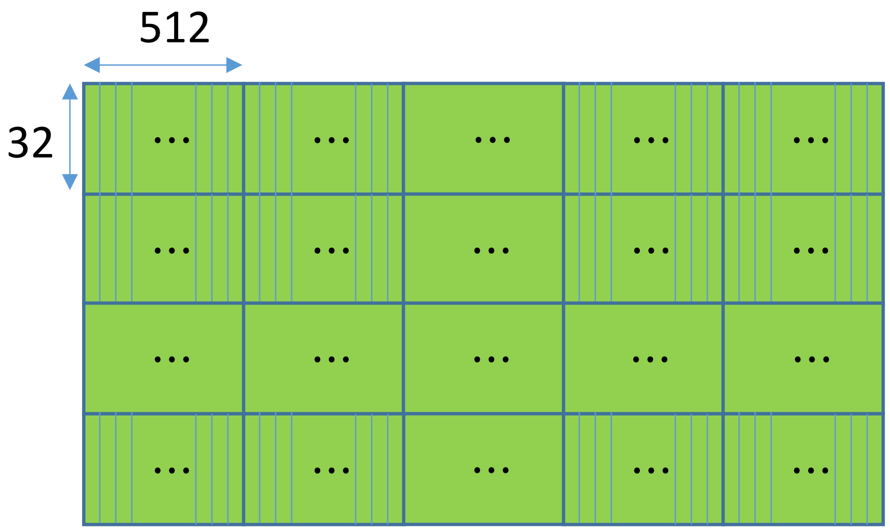


Figure 14: Detail of DRAM Architecture with Approximate Data

By using block structure to store approximate data, DRAM can use different refresh rate for each row. Hence, DRAM can save power with the method. However, the organization method also has another disadvantage is that when only one L2 cache line needs to read data from DRAM, DRAM still need to read all 32 rows in a corresponding block. It is obvious that when the situation occurred, DRAM will waste a lot of power in the ACTIVATE/PRECHARGED operations.

3.4. Hit/Miss Criterion of Bit Transpose Unit.

In this section, the criterion of accessing hit/miss operation of Bit Transpose Unit is established. Because the condition has a direct effect on evaluation the operation of Bit Transpose Unit, we will explain in detail how the criterion is created.

First, we recall that data, which is stored in Bit Transpose Unit, has related directly with the data in the L2 cache, so to understand about how data is ordered in Bit Transpose Unit, we need to know about the position of data in L2 Cache. For example, assume that data is read from row 0 in DRAM to L2 cache. Then, cache line 0 consists 16 subsequent data elements with column address from 0 to 15. Similarly, cache line 1 also includes next 16 elements with column address from 16 to 31 in the same row 0. The same process is applied to other cache lines 2 to 31 and finally, we have a two-dimensional array (or block) with size 32×512. Figure 15 illustrates the above idea.

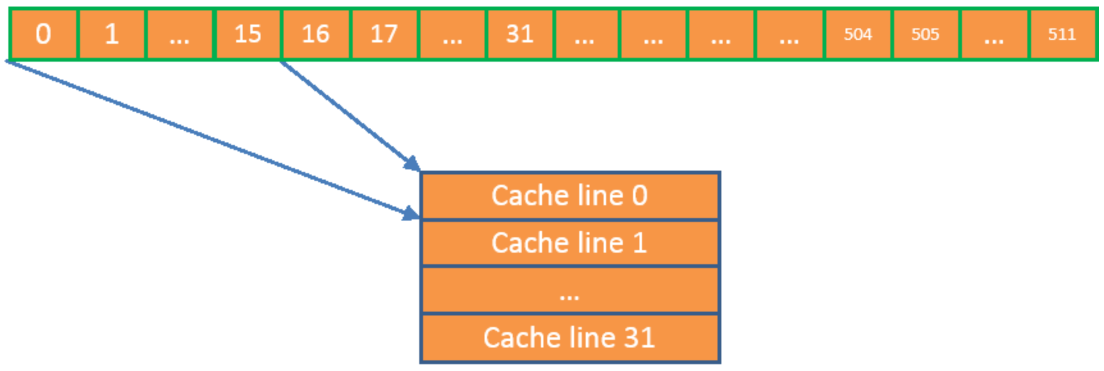


Figure 15: Store Data from DRAM to L2 Cache

It is obviously 32 consecutive cache lines include 512 subsequent elements in the same row in the DRAM. Therefore, after transpose operation, Bit Transpose Unit still includes 512 successive elements. Trivially, when the L2 cache is a miss, it will need to read data from DRAM. If data is precise, DRAM will return directly data to L2 cache. However, if data is approximate, we need to check whether or not data is stored in Bit Transpose Unit. If Bit Transpose Unit consists necessary data, it will return data to L2 cache. In this case, we call Bit Transpose Unit is hit. Otherwise, we need to read data from DRAM to Bit Transpose Unit and return data to L2 cache, so we will call Bit Transpose Unit is miss in this situation. From above reasons, we need a condition to check whether or not requirement data from L2 cache exists in Bit Transpose Unit. We use a global variable called buffer to store the address of the first element in Bit Transpose Unit. Then, we compare requirement data's address with the address of the first element. The row address and column address is used to compare. The comparing condition is shown below.

$$row_{address_{data}} = row_{buffer}$$

$$column_{buffer} \leq column_{address_{data}}$$

$$\&\&\ column_{address_{data}} < column_{buffer} + 512$$

Why we have 512 in our condition? The reason is that Bit Transpose Unit includes 512 subsequent elements from the same row in DRAM. The row compares condition detects the new necessary element belong to the same row with other elements in Bit Transpose Unit. Hence, if a new requirement element has the address satisfies the above condition, the Bit Transpose Unit is hit. Otherwise, Bit Transpose Unit will miss and it needs to read data from DRAM to return value to L2 cache.

3.5. Reading/Writing Operation with Bit Transpose Unit

The section will clarify the operation of Bit Transpose Unit. Furthermore, we also show the advantage and disadvantage of the proposed method. Based on the disadvantage of this method, we propose a different way in section 3.6 to overcome the problems.

Bit Transpose Unit includes two main operations: Reading and Writing operation. First, we talk about reading operation of Bit Transpose Unit. Second, the writing operation will be introduced.

The reading operation occurs when L2 cache needs to read data from DRAM. When L2 cache needs to read data, we have three cases need to control.

1. If data is precise, DRAM will return data immediately to L2 cache.
2. If data is approximate and Bit Transpose Unit is hit. In the case, Bit Transpose Unit will return data directly to the L2 cache.
3. This is the worst case, data is approximate and Bit Transpose Unit is miss. In this situation, DRAM will send data to Bit Transpose Unit and Bit Transpose Unit will transfer data to the L2 cache.

In our explanation, we will focus only on case 2 and case 3. Case 1 is obviously for every normal DRAM, so we will omit it in our clarification. Of course, cases 1 still exists in my implementation, which is demonstrated in Chapter 4.

When Bit Transpose Unit operates for the first time, the buffer always misses. This is obviously because at this time, we did not have any data in the buffer and we need to read data from DRAM to transfer it to the L2 cache. In the next operation, we based on the hit/miss condition in section 3.4 to detect whether or not Bit Transpose Unit is hit or miss. If the condition is satisfied, Bit

Transpose Unit returns data immediately to L2 cache. However, if the Bit Transpose Unit is a miss, it is the worst case and we need to execute following steps:

1. Step1: Write back all data from Bit Transpose Unit to the corresponding location in the DRAM. The step is required because we need to release data in the Bit Transpose Unit to have space to store new data from DRAM. Of course, we write entire a block data (32x512) to DRAM.
2. Step2: Read the corresponding block data from DRAM to Bit Transpose Unit. We read a block (32x512) from DRAM. After finishing reading data to Bit Transpose Unit, the necessary data will be transferred to L2 cache.

Figure 16 shows the operation of step1.

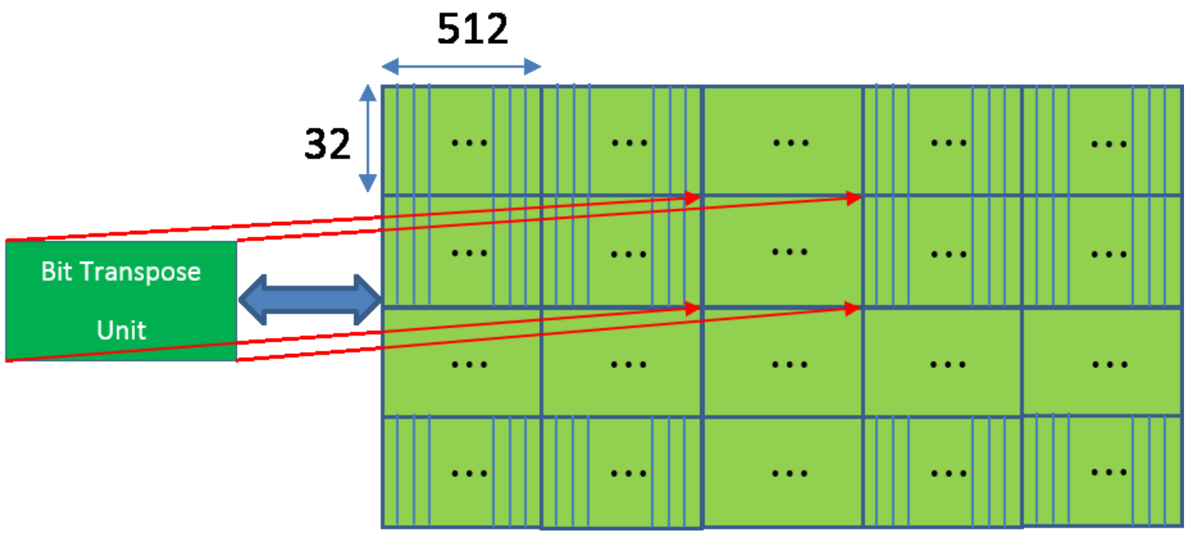


Figure 16: Step1 in Reading Operation

Data in the Bit Transpose Unit will be written to the corresponding address location in the DRAM.

In figure 17, we use the address of required data to determine the necessary block data in DRAM. After knowing the exact position of the block, we will transfer data in the block to the Bit Transpose Unit.

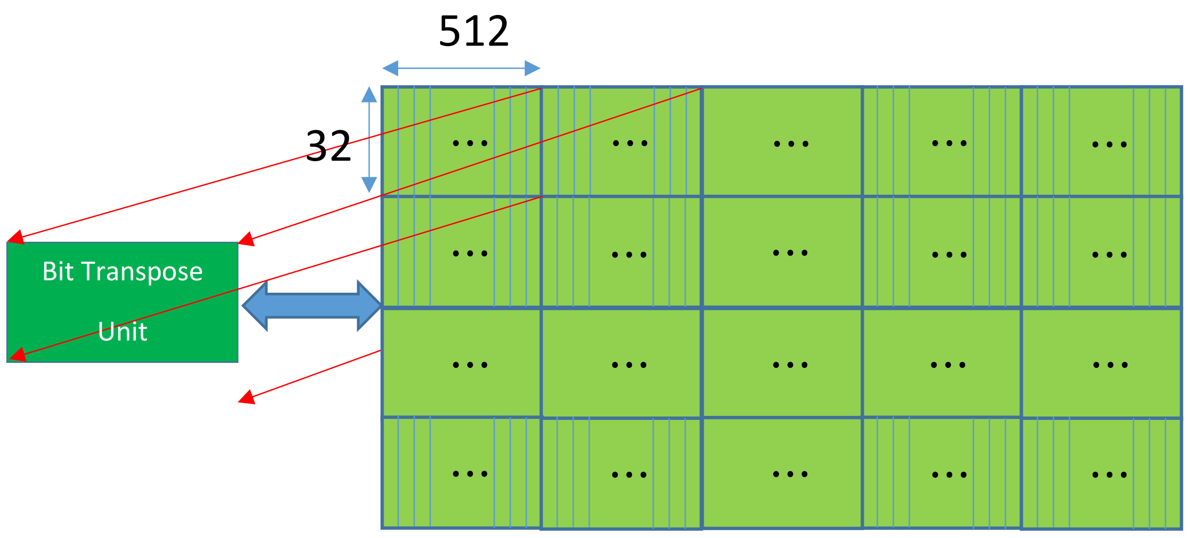


Figure 17: Step2 in Reading Operation of Bit Transpose Unit

Until now, we completed explaining the reading operation of Bit Transpose Unit. In the next clarification steps, we will concentrate on exemplifying the writing operation of the Unit. Likewise reading operation, writing operation also includes 3 situations as following:

1. Writing precise data from the L2 cache to DRAM. In the situation, data will be written normally to the normal location of the DRAM, which is the channel 1 in the DRAM.
2. Writing approximate data and Bit Transpose Unit is hit. In this case, we only write data from the L2 cache to the suitable position in the Bit Transpose Unit.
3. The worst case when data is approximate data and Bit Transpose Unit is miss. We must execute two steps in the case. First, we need to write back all data from Bit Transpose Unit to DRAM. The step is the same step1 in the reading operation. Second, we will write data

from L2 cache to the Bit Transpose Unit. Note that we need to write entirely a block, which consists the writing requirement data to the Bit Transpose Unit.

Obviously, for the first time of the operation, if the reading operation is activated, the following reading or writing operation will check the hit/miss condition as section 3.4. However, if the writing is the first operation of the Bit Transpose Unit, then the writing operation is miss trivially and in this case, we only need to write data directly from L2 cache to the Bit Transpose Unit. Because case1 and case2 in the writing operation are very easy to understand, so we illustrate only the case3 in figure 18. The step1 of the writing operation is similar with the step1 of the reading operation (figure 16). Therefore, we will not display it in here. Figure 18 shows approximate data wrote from L2 cache to the Bit Transpose Unit. In figure 18, we assumed that we need to write data in the cache line 50 in the L2 cache. However, because our proposed architecture for the Bit Transpose Unit, we need to write all data from cache line 32 to the cache line 63 to the Bit Transpose Unit. Of course, it is the waste operation but it helps to preserve the consistency of data between the Bit Transpose Unit and the data architecture in the DRAM (channel 0).

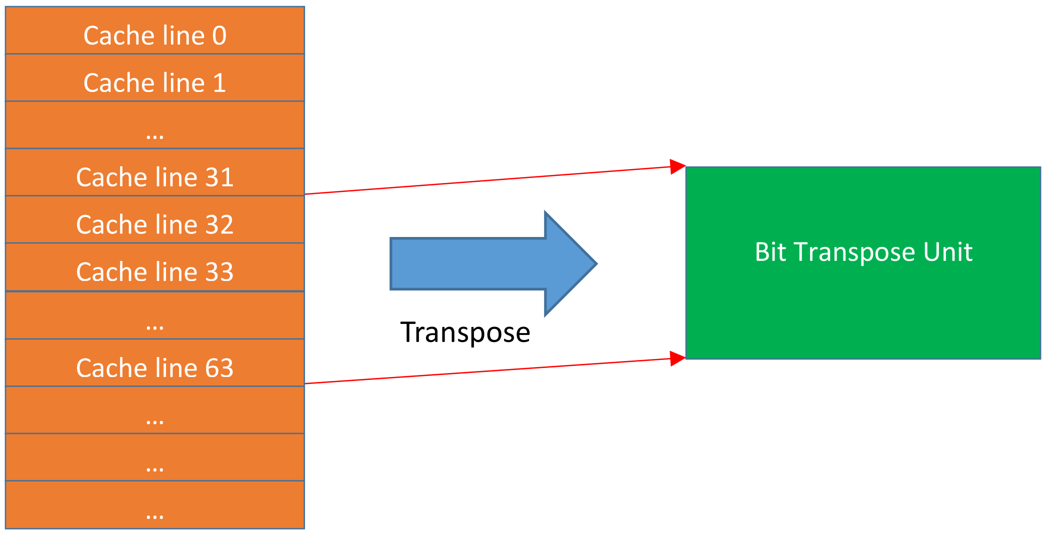


Figure 18: Step2 in the Writing Operation of Bit Transpose Unit

Note that in our implementation, we assume that the transpose data from 32 cache lines in the L2 cache to Bit Transpose Unit is completed. We did not implement the step in our simulation for simplicity. From the above presentation of the reading/writing operation of the Bit Transpose Unit, we summarize the advantage and disadvantage when using Bit Transpose Unit.

- First, the advantage of Bit Transpose Unit is that if the Unit is hit in the reading or writing operation, we can save power, reduce bandwidth, latency, and also increase IPC because we don’t need to go to the DRAM to access necessary data. Moreover, Bit Transpose Unit is the intermediate steps which help to store data in a kind suitable for the approximate data block in the DRAM because data from the L2 cache is transformed before sending to the Bit Transpose Unit.
- Second, the disadvantage of Bit Transpose Unit is that when the buffer is a miss, both the reading and writing operations need to write back data from the Bit Transpose Unit to the appropriate location in DRAM before reading necessary data from DRAM or writing data from L2 cache to the Bit Transpose Unit. The obvious consequence is that both the latency and the bandwidth increase because we need to write back data from the Bit Transpose Unit to the DRAM.

In the next section, we proposed a solution to overcome the drawback of using one Bit Transpose Unit.

3.6. Reading/Writing Operation with Two Bit Transpose Units

In section 3.5, we have seen that when the Bit Transpose Units missed, we incurred the disadvantage from the writeback data from the Bit Transpose Unit to the DRAM. To overcome the problem, we proposed a solution by using two isolated Bit Transpose Units. One Bit Transpose

Unit uses for the reading operation. Another uses for the writing operation. When we use two Bit Transpose Units, we need some modifications for the operation of each buffer. In addition, we also need to control the coherence between the reading buffer and the writing buffer. To easy for understanding, the structure of two Bit Transpose Unit is displayed in figure 19. Comparing with the figure 13, an additional buffer is attached in the architecture. The reading and writing operation is divided into the two isolated buffers. First, we talk about the reading operation. From the figure 19, the reading operation still includes three cases as explaining in section 3.5. The case1 and case2 are similarly with case1 and case2 of the reading operation in section 3.5, so we will not mention it again in the section. We will focus on the case3 because of some modification with the operation of Bit Transpose Units. For simplification, we annotate the Bit Transpose Unit with the reading operation is Reading Buffer and another is Writing Buffer. Now, in cases3 of the reading operation, the Reading Buffer is a miss, so we need to read approximate data from the DRAM. In section 3.5, because we have only one buffer working for both reading and writing, hence we need to release the space of the buffer before we read or write data to the buffer. However, in this section, we use a separate buffer for reading operation and we don't need to do the work. But two cases still exist and need to control by our implementation.

- Case1: New requirement reading data doesn't exist in the Writing Buffer. In the case, the Reading Buffer plays a simple by reading directly necessary approximate data from the DRAM to the buffer without worry about the correctness of the data.
- Case2: It is not the expectation cases. In this case, the reading data exist in the Writing Buffer, so to preserve the consistency of the data, we need to execute the two following steps:

1. Step1: Writing the data from the Writing Buffer to the correct position in the DRAM.
2. Step2: Reading the necessary data from the same location in the DRAM to the Reading Buffer and send this data from the buffer to the L2 cache.

The above implementation protects the coherence in the operation between the two buffers. In the next steps, we explain the operation of the Writing Buffer in the writing work. In the writing operation, we also have three cases as in section 3.5. All the three cases are same with three cases in the writing operation of section 3.5. Therefore, we did not talk again about it.

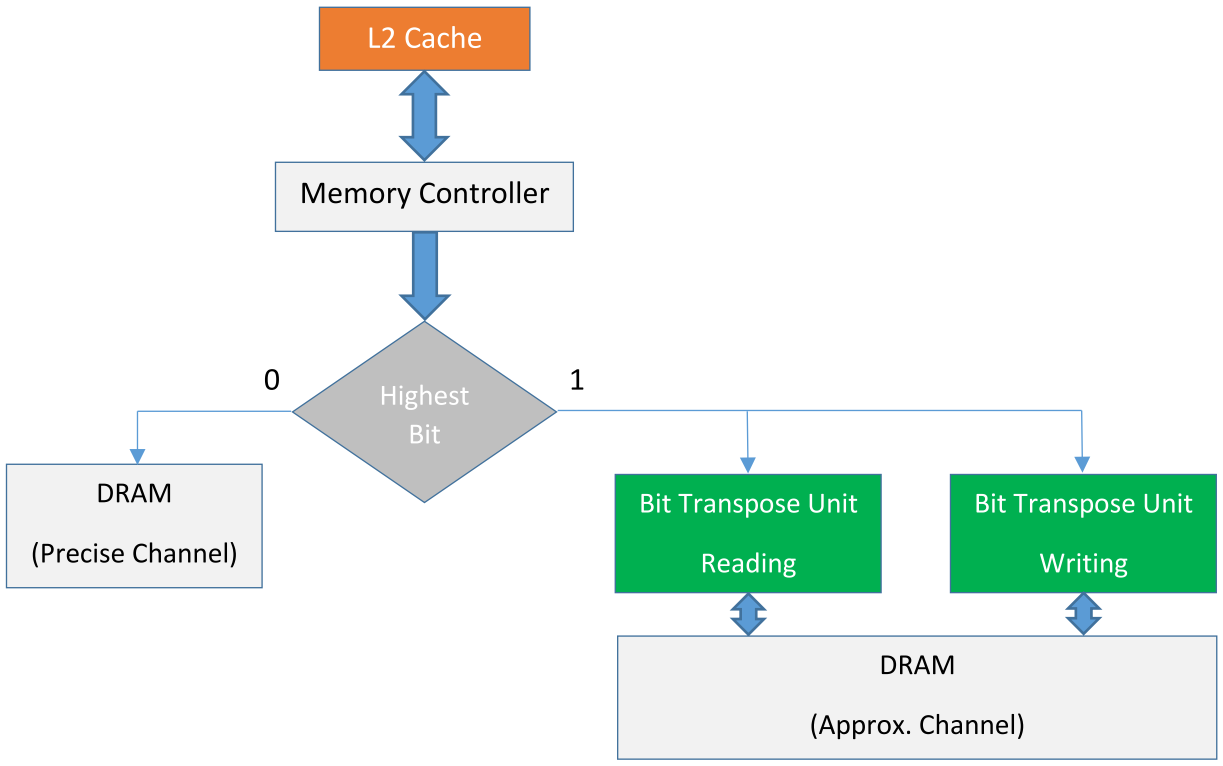


Figure 19: Two Bit Transpose Units Architecture

Now, we sum up the advantage of the operation with the work of two buffers.

- It is easy to see that with the work of two buffers, we can save a lot of active power because we did not need to release the buffer space every time the buffer is missed. We only incurred small drawback when the data need to move in the Reading buffer is coherent with the data in the Writing Buffer.
- The proposed two buffers solution increases the IPC and reduce latency and bandwidth.

All of the above advantages of two buffers solutions will be proved in Chapter 4.

3.7. Two-Bit Transpose Units with Truncated Data

In all previous section, we use data with 32 bits both for precise and approximate data. However, in many application, 32 bits are too big and we did not need very high accuracy data. For this reason, in the section, we will two Bit Transpose Units with truncated data. We have used the various number of the truncated data from 8 bits data to the 20 bits data. The accuracy and the power of our simulation with the truncated data will be shown in Chapter 4. The important core in the section is that we will concentrate on the effect of truncated data on the structure of the Bit Transpose Unit and the DRAM architecture.

First, we will talk about the effect on the size of the Bit Transpose Unit. For easy to image, we assumed that truncated data is 16 bits. Then, the size of Bit Transpose Unit became 16×512. The reduction in the size of the Bit Transpose Unit caused by the decreasing of the number of bit in the truncated data. In this situation, we only care about the 16 highest bits in the data and omit other lower bits. The size of old and new Bit Transpose Units are presented in figure 20. When data from L2 cache need to store in the Bit Transpose Unit, L2 cache data still need to be reorganized to a two-dimensional array with size 512×32. After the transpose operation, the array has the size of 32×512. However, in the current, the size of Bit Transpose Units is only 16×512, so we only transfer only first 16 rows in the 32×512 array. Because first 16 rows consist of 16 highest bits for all 512 elements in the array, the data in the Bit Transpose Units include all 16 highest bit data.

Another effect of the truncated data is that it also changes the architecture of the DRAM. In figure 14, the data is stored in the DRAM architecture by each block 32×512 to match with the size of data in the Bit Transpose Units. Nonetheless, the current size of the buffer is only 16×512, so we need to reduce the size of the data block in the DRAM. The data block in DRAM has size only 16×512. The changing of DRAM architecture is shown in figure 21.

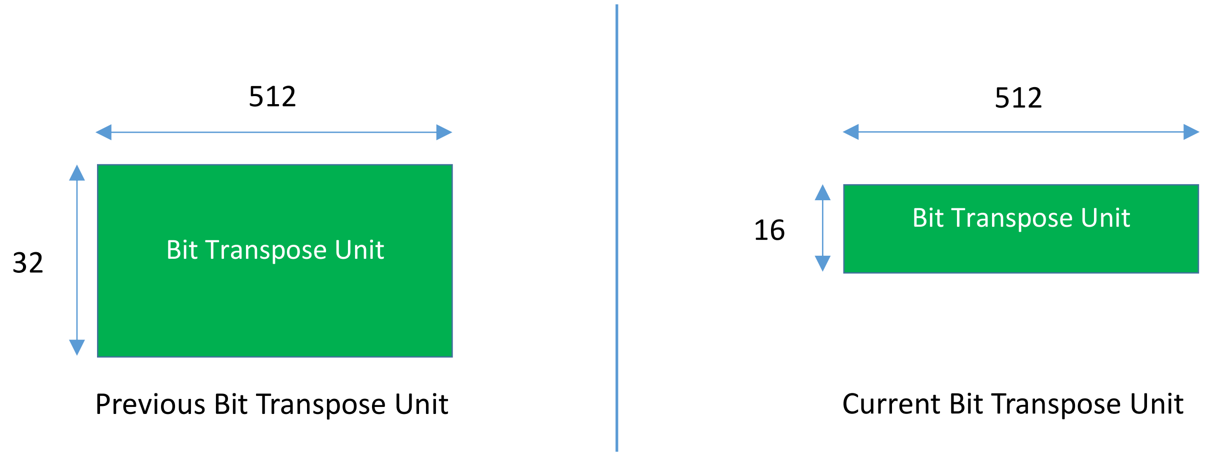


Figure 20: Bit Transpose Unit w/ and w/o Truncated Data

Overall, the reduced number of bits in truncated data has the direct influence on the size of Bit Transpose Units and the size of the data block in the DRAM.

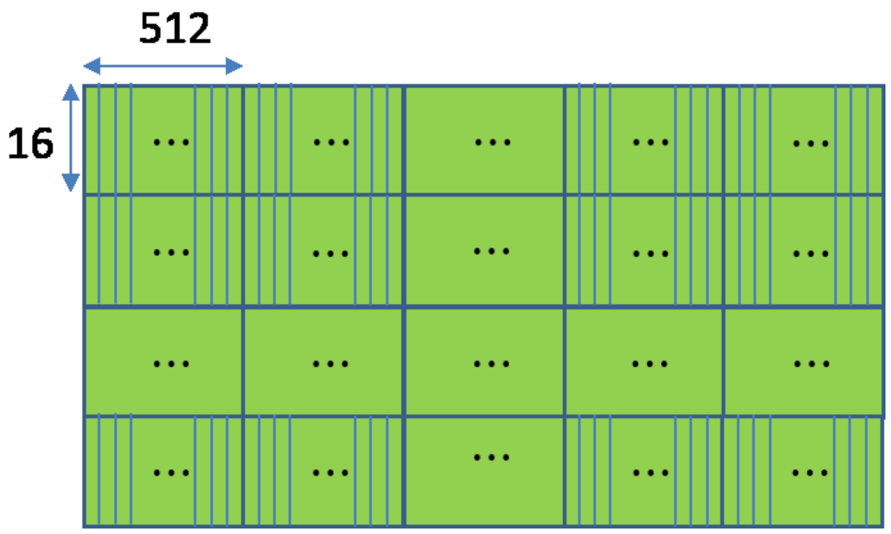


Figure 21: DRAM Architecture with Truncated Data

By using the truncated data, we have following advantages:

- Reduce power which is used by Bit Transpose Units.
- Reduce latency, bandwidth, and dynamic power when the Bit Transpose Units are miss

Chapter 4: Experiments and Results

The chapter has a mission to check the results of simulation for different proposed methods. It is also the evidence to prove the advantage of using two Bit Transpose Units with or without truncated data. In section 4.1 and 4.2 we explain how to modify the McSimA+ simulation and DRAMSim2 simulation to work with precise and approximate data. The next section 4.3, shows the result of running 3D_CNN network and VGG network for only precise data. Section 4.4 deserves for the running the two above networks with the one Bit Transpose Unit. Section 4.5 displays a different kind of powers for the two Bit Transpose Units solutions. The subsequent section 4.6 concentrates on the results for two Bit Transpose Units with the various number of truncated data. Section 4.7 compares the simulation results for open-page policy and close-page policy in the DRAM. Finally, we introduced the estimation for the power, which is used by Bit Transpose Units and also the area of the corresponding Bit Transpose Unit in section 4.8.

In each section, we will explain what the reason is for the many different types of powers.

Figure 18 shows the overview of the integration between two simulators. Application in the figure is some Deep Learning network such as GoogleNet, VGGNet, AlexNet and ResNet. The Bit Transpose Unit is added in McSimA+ simulator and contribute as a bridge between L2 cache and DRAMSim2 simulator. Hit condition block will check whether data exists in the Bit Transpose Unit or approximate region in DRAM. If hit condition is miss, data will be transferred between Bit Transpose Unit and approximate memory through addTransact() function.

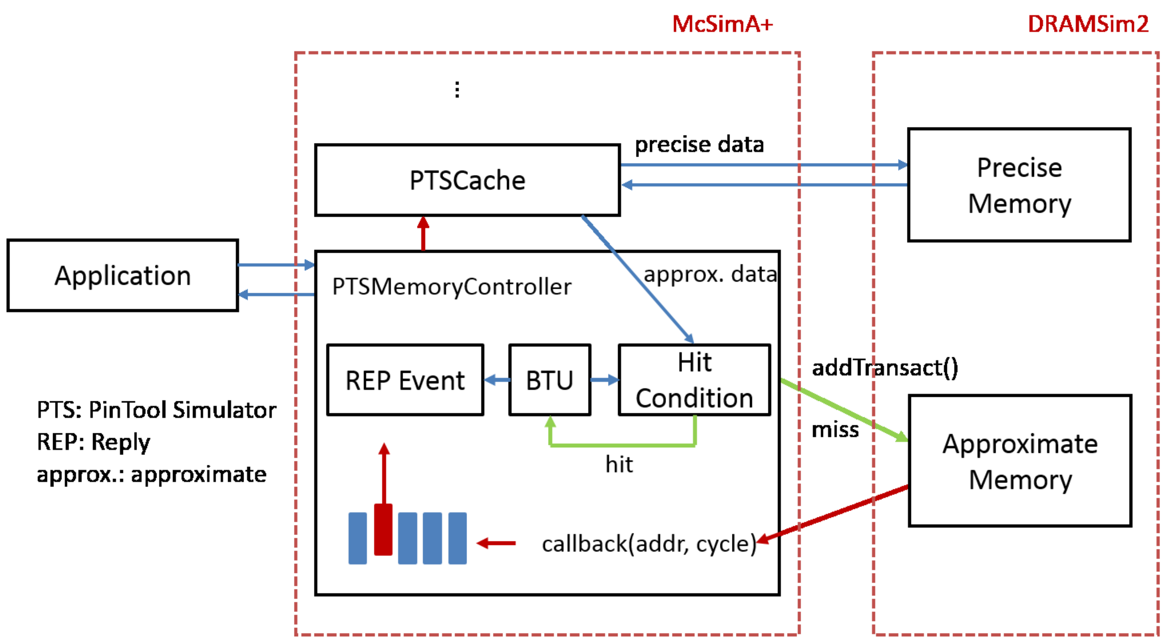


Figure 22: McSimA+ and DRAMSim2 Integration

4.1. Modification in McSimA+

Before going further in the section, we must recall from the section 3.2 that Memory Controller classifies the kind of data based on the highest bit in the correlated data's address. To achieve the target is separating between precise and approximate data, we must change the malloc function in the McSimA+ simulation. By changing the malloc function, we imply the position of data will be stored in the DRAM. The function has the mission to decide which part of the DRAM is chosen to store data. The implementation of new malloc function (we call it to distinguish with the original malloc function in the McSimA+ simulation) includes two part. In the first part, malloc function will allocate data normally for precise data and data is stored in the channel 1 of the DRAM. In the second part, it needs to allocate memory for approximate data, so the highest bit in the data's address need change from 0 to 1. We use a simple Macro function for the mission of activating the highest bit in the address. Parallel with the malloc function, we also modify the free function to match with the change in the in malloc function. In the new free function, we will deallocate memory both for precise and approximate data.

Another modification in the McSimA+ is that we need to create a link between the McSimA+ simulation and DRAMSim2 simulation. Recall from the section 2.4, McSimA+ based on the events to run the simulation. Events are processed whenever they occur (timestamps). On the other hand, DRAMSim2 simulation depends on the clock to run. Commands are processed cycle-by-cycle. Because the difference in the simulation method, when we tried to integrate both the simulation, we met the problem that clock information is missing: transactions could be added to DRAMSim2, but they would not be processed without the clock. For this reason, to make the McSimA+ work together with DRAMSim2, we used a solution to overcome the problem by the Memory Controller in McSimA+ has to create events constantly to emulate clock cycle. Transactions are forwarded to DRAMSim2 and a callback function fires every time a transaction is completed. Because callbacks will only return address and type of data, a Local Queue Element (LQE) needs to be saved in the PTS Memory Controller. Using the address field (and write/read Boolean), the corresponding LQE will be searched for and sent back to the PTS Directory and deleted.

4.2. Modification in the DRAMSim2 Simulation

To use the DRAM with both precise and approximate data in the simulation, we need many modifications in the DRAMSim2 simulation. Here, it is the necessary modification in the DRAMSim2 simulation.

- Separate two channel in the DRAM. Channel 0 is used for the approximate data and precise data deploy the channel 1 in the DRAM.

- Modification in PTS Memory Controller to add Bit Transpose Unit and make it work with the new architecture of the DRAM.
- Change the write and read functions in the DRAMSim2 simulation to work in both case with precise and approximate data.
- Implement condition for hit/miss state of Bit Transpose Unit.
- Aligned the address, which is sent from McSimA+ simulation.

We will explain each modification in the DRAMSim2 simulation in the detail. First, we need to isolate a different kind of data in the different channel of the DRAM. To achieve our target, we need to control the channel bit in the data's address. Because we want channel 0 to store approximate data and channel 1 for precise data, so I wrote a function to set the channel's bit in the data's address before the address is sent to the Memory Controller in the DRAM. Then, Memory Controller can choose the suitable location for each kind of data.

Second, the core of my implementation is the modification in the DRAM Memory Controller. In the DRAM Memory Controller, I changed the process_event function in the Memory Controller. The operation of Bit Transpose Unit is added in the process_event function. Depending on one or two Bit Transpose Units in our simulation, we have written the different version of modification to match with the requirements. In the original version, the process_event function only works with the precise data. But, at the current time, the function must control its operation with both approximate and precise data. Its operation also relied upon on the state of the Bit Transpose Unit is hit or miss. The detail of Bit Transpose Unit is introduced in detail in section 3.5, 3.6, and 3.7.

Two functions read_complete and write_complete also need to modify. In the normal operation, these functions will return data from DRAM to the Directory when the reading or writing operation is completed. However, because now we need to aligned data's address before sending the command to the DRAM, we must check the existence of address in the Local Queue Event. If the address exists in the Queue, the value of data will be returned to the Directory. Otherwise, the program will appear errors and stop working.

Another contribution to my work is to implement condition to check when the Bit Transpose Unit is hit or miss. Two functions are used to detect the state of the Bit Transpose Unit. The first function is is_approx function. If the return value of the function is 1, the data is approximate and will use the is_valid function to know whether the Bit Transpose Unit is hit or miss with the reading/writing operations. When the return value of the is_approx function is 0, data will be read or written directly with the DRAM through the channel 1. The condition for checking the status of the Bit Transpose Unit is described in section 3.4.

Finally, the data's address must be aligned before sending to the DRAM Memory Controller. It became an advantage when we need to write or read a block of data (32x512) to/from the DRAM.

4.3. Simulation with Precise Data

In the section, the simulation result for 3D_CNN and VGG networks is collected. We use the results in the section as the baseline for comparing with the operation of one or two Bit Transpose Units and with two Bit Transpose Unit plus truncated data. The environment of our simulation is Centos7. Every simulation is run on the computer core i5 3.2GHz and 4Gb memory.

First, it is the result of running 3D_CNN network.

Network	Average Power (mW)	Background	Dynamic	Refresh	IPC	Bandwidth (GS/s)	Buffer Hit Rate
Precise Data	2083	1491	167	425	1.795	0.244	0%

Table 1: 3D_CNN Power with Precise Data

Second, the running result of VGG is shown in the figure below

Network	Average Power (mW)	Background	Dynamic	Refresh	IPC	Bandwidth (GS/s)	Buffer Hit Rate
Precise Data	1778	1273	80	425	1.283	0.15	0%

Table 2: VGG Power with Precise Data

4.4. Simulation with Bit Transpose Unit

In the section, the result for running two above network with the Bit Transpose Unit is summarized.

- 3D_CNN

Network	Average Power (mW)	Background	Dynamic	Refresh	IPC	Bandwidth (GS/s)	Buffer Hit Rate
Precise Data	2083	1491	167	425	1.795	0.244	0%
1 Bit Transpose Unit	1835	1540	166	129	1.786	0.243	94.88%

Table 3: 3D_CNN Power using Single Bit Transpose Unit

- VGG

Network	Average Power (mW)	Background	Dynamic	Refresh	IPC	Bandwidth (GS/s)	Buffer Hit Rate

Precise Data	1778	1273	80	425	1.283	0.15	0%
1 Bit Transpose Unit	1634	1371	134	129	1.226	0.275	96.72%

Table 4: VGG Power using Single Bit Transpose Unit

The results from two above tables show that all aspect operation of the DRAM is reduced with the operation of the Bit Transpose Unit. The reduction in the quality of DRAM's operation caused by the overhead writing operation from the Bit Transpose Unit to the DRAM every time the buffer is miss. This is the drawback of using the Bit Transpose Unit. Even only one element from L2 cache needs to be read or write, all elements in the buffer block must write to the DRAM. Furthermore, the structure of approximate data in DRAM require will must open all 32 consecutive rows in the DRAM to write a block of data to the DRAM. All these above reasons lead to the bad result of the DRAM's operation with the Bit Transpose Unit

4.5. Simulation with Two Bit Transpose Units

To overcome the problem with one Bit Transpose Unit, we proposed to use two separate Bit Transpose Units for the reading and writing operations. The detail of the solution depicted in section 3.6.

- 3D_CNN

Network	Average Power (mW)	Background	Dynamic	Refresh	IPC	Bandwidth (GS/s)	Buffer Hit Rate
Precise Data	2083	1491	167	425	1.795	0.244	0%
1 Bit Transpose Unit	1835	1540	166	129	1.786	0.243	94.88%

2 Bit Transpose Units	1834	1540	166	129	1.786	0.243	94.88%

Table 5: 3D_CNN Power using Double Bit Transpose Units

- VGG

Network	Average Power (mW)	Background	Dynamic	Refresh	IPC	Bandwidth (GS/s)	Buffer Hit Rate
Precise Data	1778	1273	80	425	1.283	0.15	0%
1 Bit Transpose Unit	1634	1371	134	129	1.226	0.275	96.72%
2 Bit Transpose Units	1507	1290	88	129	1.286	0.156	96.72%

Table 6: VGG Power using Double Bit Transpose Units

It is obvious to see that when using two Bit Transpose Units, all characteristics of DRAM's operation is improved. The reason is that when using two separate buffers, we don't need to write data from the buffer to the DRAM every time the buffer is miss. We only need to write buffer when requirement data from the Reading Buffer exists in the Writing Buffer. In this situation, we need to write data from the Writing Buffer to the DRAM before reading the corresponding data to the Reading Buffer to protect the consistency of data.

4.6. Simulation Two Bit Transpose Units plus Truncated Data

As explained in section 3.7, using the truncated data will help to save both the power and improve the operation of the DRAM. The prediction is proved by the result in the section. In the section, we run our simulation with various kind of truncated data. Truncated data includes 8 bits, 9 bits, 12 bits, 16 bits, 20 bits. The result is similar our prediction in section 3.7. When we use the truncated data, we reduce the number of writes and read with the approximate data from the DRAM. Furthermore, if the Bit Transpose Units are a miss, the number of ACTIVATE/PRECHARGED also reduced because we don't need to open 32 rows as in the normal two Bit Transpose Units simulation.

- VGG

Network	Average Power (mW)	Background	Dynamic	Refresh	IPC	Bandwidth (GS/s)	Buffer Hit Rate
Precise Data	1778	1273	80	425	1.283	0.15	0%
1 Bit Transpose Unit	1634	1371	134	129	1.226	0.275	96.72%
2 Bit Transpose Units	1507	1290	88	129	1.286	0.156	96.72%
2 Bit Transpose Units 20 bits Data	1445	1253	64	128	1.313	0.108	96.72%
2 Bit Transpose Units 16 bits Data	1423	1240	56	128	1.320	0.092	96.72%
2 Bit Transpose Units 12 bits Data	1402	1227	47	127	1.332	0.076	96.72%
2 Bit Transpose Units 9 Bits Data	1377	1218	41	119	1.338	0.064	96.72%
2 Bit Transpose Units 8 Bits Data	1359	1214	39	106	1.341	0.059	96.72%

Table 7: VGG Power using Double Bit Transpose Units and Truncated Data

4.7. Open-page Policy versus Close-page Policy

The section shows the influence of two kind of polices in the operation of the DRAM. The results are shown in the following tables.

- Two Bit Transpose Units Simulation

Network	Average Power (mW)	Background	Dynamic	Refresh	IPC	Bandwidth (GS/s)	Buffer Hit Rate
2 Bit Transpose Units (Open-Page)	1507	1290	88	129	1.286	0.156	96.72%
2 Bit Transpose Units (Close-Page)	1563	1335	99	129	1.232	0.149	96.72%

Table 8: Compare VGG Power between Two Policies

- Two Bit Transpose Units with 9 bits truncated data

Network	Average Power (mW)	Background	Dynamic	Refresh	IPC	Bandwidth (GS/s)	Buffer Hit Rate
2 Bit Transpose Units 9 bits Data (Open-Page)	1377	1218	41	119	1.338	0.064	96.72%
2 Bit Transpose Units 9 bits Data (Close-Page)	1376	1215	43	119	1.334	0.064	96.72%

Table 9: Compare VGG Power between Two Policies and 9 bits Truncated Data

- Two Bit Transpose Units with 8 bits truncated data

Network	Average Power (mW)	Background	Dynamic	Refresh	IPC	Bandwidth (GS/s)	Buffer Hit Rate
2 Bit Transpose Units 8 bits Data (Open-Page)	1359	1214	39	106	1.341	0.059	96.72%
2 Bit Transpose Units 8 bits Data (Close-Page)	1355	1209	40	106	1.338	0.059	96.72%

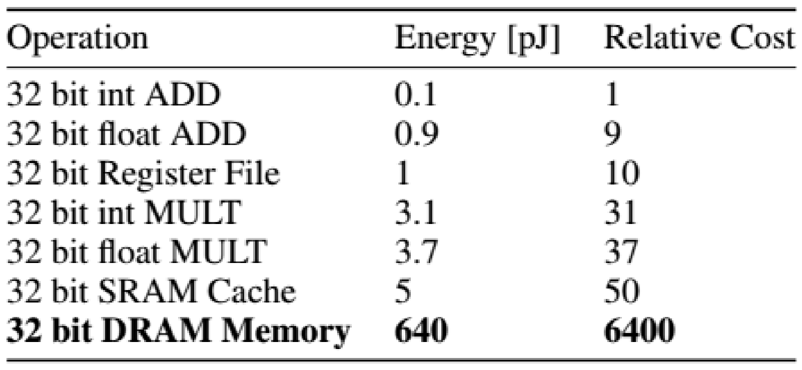
Table 10: Compare VGG Power between Two Policies and 8 bits Truncated Data

The results prove that almost aspect of the DRAM's operation is the same for both open-page and close-page policies. Note that the average power and background power in close-page policy is smaller than open-page policy because, in the close-page policy, a row is closed after write or read data from it. It is matched with our DRAM architecture, which requires opening all rows in a block to read data from a block to the Bit Transpose Unit.

4.8. Estimate Bit Transpose Power and Area

In the section, we will estimate the Bit Transpose Unit power and area for all cases in section 4.4, 4.5, and 4.6. First, the simulation condition is collected as following:

- CPU frequency is 3.0 GHz
- 30 timestamp for one clock
- # of time stamp: 36866978940
- #of hit/miss: 3677967/124557



Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**

Figure 23: Energy for 45nm CMOS process [9]

Based on the above simulation conditions, the second step is estimate the power of Bit Transpose Unit.

- Time for one stamp is:

$$t_{timestamp} = \frac{\frac{1}{3}.10^{-9}}{30} = \frac{1}{9}.10^{-10}(s)$$

Therefore, the total running time is:

$$t = \ t_{timestamp} * \#timestamp$$

$$= \frac{1}{9}.10^{-10} * 36866979840 = 0.4096(s)$$

- The number of accessing Bit Transpose Unit is:

$$3677967 * 1 + 124557 * 65 = 11774172$$

- The power for reading and writing operations of Bit Transpose Unit is:

$$P_{BTU} = 32 * 11774172 * 10^{-12} * \frac{16}{0.4096} = 14.72(mW)$$

Next, the area of Bit Transpose Unit is estimated. Estimated gate count of Bit Transpose Unit is implemented in TSMC 65nm library. I coded a simple version of Transposed Unit by Verilog then do synthesis. The result shows that Bit Transpose Unit requires 16384bits plus $512x32$ MUX. Hence, it need 243K gates.

When comparing the power of Bit Transpose Unit with average power of 3D_CNN or VGG network in section 4.4, 4.5, 4.6, it is clear that power of Bit Transpose Unit only increases less than 1% total power. It is a worth prize when comparing with the reducing of refresh power of DRAM.

Chapter 5: Conclusion and Future Work

This paper present the implementation of the Bit Transpose Unit and the modification in DRAM architecture to work with both the precise and approximate data. It also proposed a solution with two Bit Transpose Unit to overcome these problems which is met when using only one Bit Transpose Unit. The combination between the two Bit Transpose Unit and the truncated data have proved the advantage of saving the power and the other operating aspects of the DRAM. The additional power using by Bit Transpose Unit and its area also estimated in this paper.

In the future, we will modify the operation of the DRAM with the Half Page Row Accessing method to improve the quality of DRAM.

Reference

[1] Yann LeCun, et al. Gradient-Based Learning Applied to Document Recognition. Proc. of the IEEE, November 1998.

[2] Alex Krizhevsky, et al. ImageNet Classification with Deep Convolutional Neural Networks.

[3] Karen Simonyan, et al. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR 2015.

[4] Christian Szegedy, et al. Going Deeper with Convolutions CVPR 2015.

[5] https://telecombcn-dl.github.io/2017-dlai-team4/

[6] Yoongu Kim, et al. A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM

[7] Ishwar Bhati, et al. Flexible Auto-Refresh: Enabling Scalable and Energy-Efficient DRAM Refresh Reduction ISCA 2015.

[8] Duy Thanh Nguyen, et al. A Transposed Memory Architecture for Low Power Deep Learning ISCAS2018.

[9] Mark Horowitz. Energy table for 45nm process, Stanford VLSI Wiki.

Abstract in Korean

Nguyen Huy Hung

Electrical and Computer Engineering

The Graduate School

Seoul National University

딥러닝은 인공 지능 분야에서 가장 성공적인 방법 중 하나이다. 머신러닝 알고리즘의 발달과 GPU의 개발은 모두 다양한 딥러닝 응용 프로그램의 성공에 기여 해 왔다. 딥러닝 응용 프로그램이 점점 커지게 되면 DRAM으로의 접근도 많아진다. 현대의 DRAM은 DRAM의 전체 전력 소비에서 리프레시 전력의 영향을 조절해야 하는 문제가 있다. [8]은 컨볼루션 신경망에서 약간의 정확도를 희생하는 대신 리프레시 전력의 70 % 이상을 절약하는 방법을 제안했다. 그러나 이 논문은 실제로 리프레시 전력을 측정하지 않고 예측된 전력의 감소만을 보여준다. 본 연구에서는 근사 데이터를 이용하기 위한 중요한 부분인 Bit Transpose Unit을 실제로 측정한다. DRAM 아키텍처의 변경 사항 또한 실제 시뮬레이션에서 표시되고 측정된다. 또한 Bit Transpose Unit의 문제를 해결하기 위한 방안을 제안한다. [8]의 방식을 실제로 구현 했을 때의 결과를 McSimA +와 DRAMSim2 시뮬레이션 결과로부터 구할 수 있다.

Keyword: Deep Learning, Power Saving, Refresh Rate, Convolutional Neural Network

Student Number: 2015-22136