HMC-MAC: Processing-in Memory Architecture for Multiply-Accumulate Operations with Hybrid Memory Cube

Dong-Ik Jeon, *Student Member*, *IEEE*, Kyeong-Bin Park, and Ki-Seok Chung, *Member*, *IEEE*
Department of Electronic and Computer Engineering, Hanyang University, Seoul, Korea
E-mail: estwingz@naver.com, lay1523@naver.com, kchung@hanyang.ac.kr

Abstract—Many studies focus on implementing processing-in memory (PIM) on the logic die of the hybrid memory cube (HMC) architecture. The multiply-accumulate (MAC) operation is heavily used in digital signal processing (DSP) systems. In this paper, a novel PIM architecture called HMC-MAC that implements the MAC operation in the HMC is proposed. The vault controllers of the conventional HMC are working independently to maximize the parallelism, and HMC-MAC is based on the conventional HMC without modifying the architecture much. Therefore, a large number of MAC operations can be processed in parallel. In HMC-MAC, the MAC operation can be carried out simultaneously with as much as 128 KB data. The correctness on HMC-MAC is verified by simulations, and its performance is better than the conventional CPU-based MAC operation when the MAC operation is consecutively executed at least six times.

Index Terms— Memory Structures, Memory used as logic, Multiple Data Stream Architectures, Parallel processing

1 Introduction

Hybrid Memory Cube (HMC) is one of the most promising next-generation dynamic random access memory (DRAM) systems. HMC is a true 3-dimensional (3D) stacked DRAM that places multiple DRAM dies on the top of a logic die. The stacked DRAM and logic dies in one partition forms a vault, and the logic die has a vault controller which manages all memory operations including automatic refresh control. HMC supports simple atomic commands (arithmetic, Boolean, comparison, and bitwise operations) which involve DRAM read-modify-write operations within the HMC device. By executing a memory-intensive operation within or near the memory, which is commonly called as processing-in-memory (PIM), the amount of data exchange between the host and the memory is significantly reduced. Therefore, many researches about PIM on HMC are being proposed [1], [2], [3]. However, most PIM architectures with HMC modify the conventional HMC structure significantly with the assumption that any computational logic block may be implemented on the HMC logic die. In general, HMC is vulnerable to high temperature because of the 3D stacked structure, and therefore, high temperature causes performance degradation due to throttled DRAM bandwidth [4]. Implementing complex logic circuits on the logic die may cause significant thermal problems due to the 3D stacked structure. Therefore, it is very important to minimize the amount of modification to the conventional HMC structure when we implement additional functionality on the logic die. In this paper, a new PIM architecture for multiply-accumulate operation with HMC called HMC-MAC is proposed with slight modifications to the conventional HMC structure that is in accord with the HMC specification.

- *Dong-Ik Jeon is with the Department of Electronic and Computer Engineering, Hanyang University, Seoul, Republic of Korea 04763 E-mail: estwingz@naver.com.*
- *Kyeong-Bin Park is with the Department of Electronic and Computer Engineering, Hanyang University, Seoul, Republic of Korea 04763 E-mail: lay1523@naver.com.*

2 Motivation

The PIM architecture is proposed to alleviate high throughput requirement between a CPU and a memory, also known as Von Neumann bottleneck [5]. PIM is able to reduce memory traffic by implementing a small amount of processing units near the memory in data-intensive applications. In addition, since simple and repetitive tasks are processed in memory, CPU can process other tasks simultaneously. Because the multiply-accumulate (MAC) is one of the operations with high data access frequencies, it has been widely employed for PIM architectures [3], [6]. A MAC operation carries out the following computation repeatedly: $\mathbf{A} \leftarrow \mathbf{A} + \mathbf{B} \times \mathbf{C}$. $\mathbf{A}$ is commonly called an accumulator where the multiplication result is accumulated. The execution time of the MAC operation heavily depends on the type of the two operands $\mathbf{B}$ and $\mathbf{C}$. The proposed HMC-MAC architecture supports MAC operations with one operand from a memory (memory operand) and the other operand is a result from some computation conducted in the host processor (host operand). This type of MAC operations frequently appears in various applications. For example, in the case of MAC operations for convolutional neural network (CNN), one operand comes from input data such as image, video or audio file, and it

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2017.2700298, IEEE Computer Architecture Letters

is simply read from memory in general. The other operand is a weight value that is calculated by the backpropagation algorithm [3]. Furthermore, all the conventional HMC atomic commands are conducted with one memory operand and one host operand [7]. Therefore, the proposed HMC-MAC architecture can closely follow the conventional HMC structure only with slight modification of the HMC atomic operation logic.

3 HMC-MAC Architecture

3.1 HMC-MAC Instruction Set

Since a memory is a passive device that responds to external requests from host components (e.g. CPU), the host should initiate a PIM operation. To fully take advantage of the PIM architecture of HMC, once a PIM operation is initiated by the CPU, the PIM operation should be executed repeatedly without any intervention from the CPU. Therefore, HMC-MAC proposes six new instructions with a combination of the data size and the data type as shown in Table 1. The data size denotes the operand size. The data type defines the number representation of the operand. The HMC-MAC instruction set has two registers and one immediate number, as in

$$MAC\_X4B \quad Rd \quad Rn \quad \langle MAC\# \rangle \tag{1}$$

where *Rd* indicates the destination register where the final MAC operation result will be stored, *Rn* is the register that contains the start address of the memory operand, and <*MAC*#> is an immediate number that indicates how many times the MAC operation will be repeated. This number will be called as the execution count. HMC is capable of processing in parallel at the vault, bank and memory block access levels. In order to maximize performance, it is desirable that memory operands for HMC-MAC operations are consecutively located so that computing the new memory address would not be necessary.

TABLE 1
HMC-MAC Instruction Set.

Data size	Data type	Instruction format
4 bytes	Fixed	MAC_X4B Rd, Rn, <MAC#>
	Floating	MAC_L4B Rd, Rn, <MAC#>
8 bytes	Fixed	MAC_X8B Rd, Rn, <MAC#>
	Floating	MAC_L8B Rd, Rn, <MAC#>
16 bytes	Fixed	MAC_X16B Rd, Rn, <MAC#>
	Floating	MAC_L16B Rd, Rn, <MAC#>

Rd : Destination register
Rn : Register on which the memory operand starting address is based
MAC# : The execution count of MAC operations

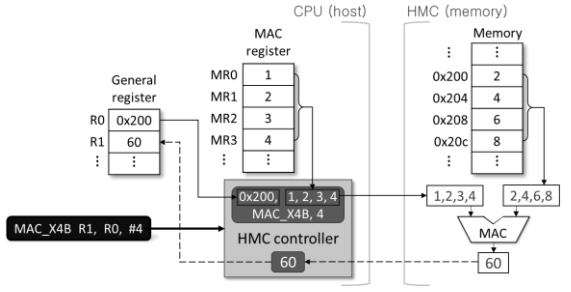


Fig. 1. An example of how to process the HMC-MAC instruction.

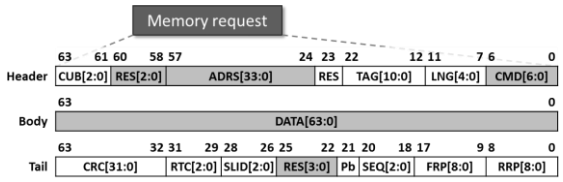


Fig. 2. HMC memory request packet layout which is composed of header, body and tail [6].

Fig. 1 shows an example of how an HMC-MAC instruction is processed in the CPU and HMC side. A set of special registers, called MAC register, is added for the host operand in the HMC-MAC architecture. CPU stores host operands in the MAC register in the order of execution. As a result, the HMC-MAC instruction loads the MAC register as many as the <MAC#> value. In the example shown in Fig. 1, the HMC controller converts the instruction into HMC-MAC memory requests where the start address of memory operands is given in R0. The HMC-MAC memory request reads four memory operands from memory address 0x200, 0x204, 0x208, 0x20C, respectively. Then, the MAC operation is carried out with host operands (1, 2, 3, 4) and memory operands (2, 4, 6, 8). The operation result (60) is returned to the HMC controller, and then, the result will be eventually stored in R1.

It is possible that addresses of consecutive memory data may cross beyond the page boundary. In this case, the HMC controller breaks one instruction into two with multiple memory requests in accordance with the page boundary. This might consume a little time to analyze and regenerate HMC-MAC instruction, but the total execution time of the regenerated MAC operations decreases a little because the two regenerated memory requests are processed in parallel.

3.2 HMC-MAC Memory Request

HMC is linked to the host component through high speed serial links (SerDes). All in-band communication across links is packetized. According to the HMC specification, the packetized memory request is composed of a header, a body and a tail, as shown in Fig. 2. An HMC-MAC instruction has four kinds of information: the instruction type, the start address of the memory operand, the execution count of MAC operations and the host operand. By utilizing the unused value of the CMD (command) field, the proposed HMC-MAC instruction type is encoded. The start address of the memory operand is encoded in the ADRS (address) field. The execution count employs RES (reserved) fields with a combination of 3 bits in the header and 4 bits in the tail. The host operand is encoded in the DATA (data block) field. Since HMC-MAC fields utilize the conventional HMC packet layout, the HMC-MAC memory request can be implemented without modifying a packet layout.

3.3 Memory Request Control Logic

HMC supports read and write block accesses up to the predefined maximum block size (32B, 64B, 128B or 256B) in one vault [7]. The consecutive memory data of one HMC-MAC memory request may access more than one vault when the execution count is relatively big. In this case, one memory request should be regenerated into multiple memory requests because a memory request should access

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2017.2700298, IEEE Computer Architecture Letters

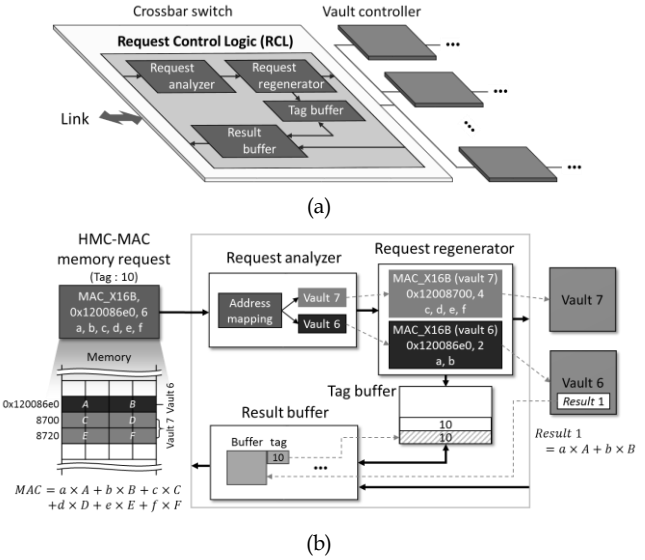


Fig. 3. Memory request handling of HMC-MAC architecture (a) the Request Control Logic (RCL) structure (b) an example of how to process HMC-MAC memory request.

a single vault. When a single memory request is regenerated into multiple memory requests, results from multiple vaults should be accumulated together before the final result is returned to the host. To deal with both request regeneration and accumulation over multiple vaults, a new control logic called "Request Control Logic (RCL)" is added as shown in Fig. 3 (a). Fig. 3 (b) shows an example of HMC-MAC memory request processing. The request analyzer determines which vaults should respond to the HMC-MAC memory request through the address decoding logic. The request regenerator produces new memory requests, and the regenerated requests are sent to the corresponding vaults. Detailed explanation on the processing of a single memory request in the vault controller is given in the next section.

When an HMC-MAC memory request is mapped to more than two vaults, multiple results from the respective vaults should be accumulated together to get the final result. Besides, the execution completion time from respective vaults will be different due to the fact that each vault is independently operated. In order to obtain the final MAC result, the regenerated request results are accumulated in the corresponding result buffer. All HMC packets are uniquely identified by a tag according to the HMC specification. Therefore, as many tag values as the number of the regenerated requests are copied in the tag buffer. One of the copied tags is removed from the tag buffer when the execution from a vault is finished until all tags are removed from the buffer. In this way, the final MAC result can be derived from multiple vaults by adding simple control logic without any significant structural modification.

3.4 Memory Request Processing in a Vault

The HMC-MAC memory request deals with reading memory operands, calculating multiplication and accumulation for the MAC operation in a vault. Fig. 4 shows an example of the regenerated request processing. The MAC unit, which is composed of an operand buffer and a MAC operator, is added in the HMC-MAC architecture. In the

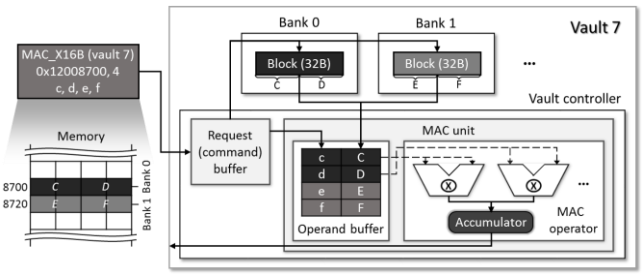


Fig. 4. An example of how to process the HMC-MAC memory request in Vault 7 from the example in Fig. 3 (b).

current implementation of the MAC operator consists of four multipliers and one accumulator. When an HMC-MAC memory request arrives at the vault controller, the memory request is stored in the request buffer and converted into a DRAM command (activate, read/write, precharge) for a bank. The host operands are extracted from a memory request and stored in the operand buffer. The vault controller issues a DRAM command in accordance with the DRAM timing parameter and arbitration, like a conventional memory controller. Reading data for a HMC-MAC memory request exactly follows the block access process of the conventional HMC. All the read memory operands are stored in the operand buffer after they are paired with host operands. If the MAC operator is ready for a new execution, multiplication is conducted by fetching a data pair from the operand buffer and then the result is accumulated. When the last MAC operation is finished, the final result is returned to RCL. The subsequent process is already explained in the previous section. The atomicity of the HMC-MAC operation should be guaranteed so that the value of a memory operand of an HMC-MAC operation should not be modified accidently by other memory requests. In order to guarantee the atomicity, the vault controller sets a request lock during the HMC-MAC operation. In addition, memory requests that access the same address are processed in the order of arrival to guarantee the memory data coherency, similarly to the First Ready, First Come First Serve (FR-FCFS) scheduling [8].

Compared with the conventional HMC architecture, the HMC-MAC architecture needs multipliers and an accumulator for the MAC operation additionally. However, the hardware overhead for the HMC-MAC is similar to what the conventional HMC needs to add for atomic command processing. In other words, the HMC-MAC structure and the control logic in the vault do not require any considerable amount of modification.

4 Experiments

In order to verify the proposed HMC-MAC architecture, a cycle-accurate simulator for HMC called CasHMC is used [9]. CasHMC is modified in order to verify the functionality of HMC-MAC. The modified simulator source code is available at http://github.com/estwings57/HMC-MAC. The proposed architecture is compared with the CPU-based MAC operation that is implemented by the gem5 simulator [10] and DRAMSim2 [11]. The CPU-based MAC operation is conducted on the x86, Alpha, two ARMs (Cortex-A8 and Cortex-A8 with NEON) architectures by gem5.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2017.2700298, IEEE Computer Architecture Letters

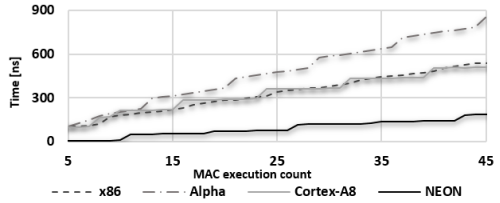
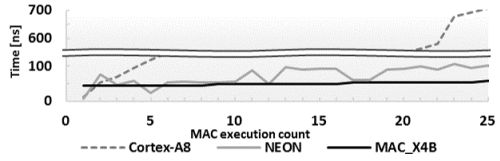
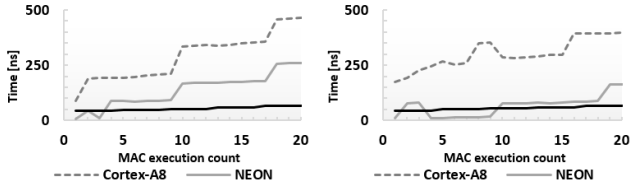


Fig. 5. Execution times of four CPU-based architecture.



(a)



(b) (c)

Fig. 6. The execution times of MAC operations of Cortex-A8, NEON and the proposed HMC-MAC architecture with respect to various data types (a) int (b) long long (c) double

The HMC-2500x32 DRAM timing parameter defined in the gem5 memory modeling is employed for simulations.

Fig. 5 shows the execution times of the four CPU-based MAC operations. All the CPU-based MAC execution times increase in proportion to the consecutive execution count of the MAC operation. In addition, the MAC execution time sharply increases at the end of every 8th execution of the MAC operation because of data cache misses. According to the results, the ARM Cortex-A8 with NEON turns out to be best for repetitive MAC operations, because NEON is specialized in parallel processing with multiple data. Thus, the proposed HMC-MAC architecture is mainly compared with ARM processors. Fig. 6 shows the execution time comparison results of ARM (Cortex-A8, Cortex-A8 with NEON) and the proposed HMC-MAC architecture with respect to three operand data types of MAC operations: int, long long and double. The MAC execution time of Cortex-A8 with respect to three data types show similar tendencies. In case of NEON, even though the simulation results show different tendencies due to different parallelization characteristics, the MAC execution time increases. However, the HMC-MAC instruction consistently shows stable execution times. This is because the MAC operation is carried out in parallel in the HMC-MAC architecture with parallel vault operations, bank interleaving and data block accesses. In theory, up to 128 kilobytes of data, which is the product of the number of vaults (32), the number of banks (16) and the maximum block size (256B) according to HMC specification, can be processed in parallel. As a result, as long as the MAC operation is repeated at least six times or more, the CPU-based MAC execution takes longer than the HMC-MAC execution. In conclusion, it is claimed that HMC-MAC is a very effective PIM architecture for MAC operations.

5 Conclusions

Many researches for processing-in-memory (PIM) architectures on Hybrid Memory Cube (HMC) have been conducted due to a logic die. However, most of them have tried to embed complicated logic circuits in the logic die by proposing a far different controller than the conventional HMC. Since HMC has a stacked structure, it is very vulnerable to thermal problem when a complicated logic block is implemented. Thus, in this paper, a new PIM architecture for HMC called HMC-MAC is proposed. Since HMC-MAC makes the best use of the conventional HMC structure, the amount of modification to the logic die is minimal. Furthermore, HMC is very effective structure to process PIM operations in parallel. It is confirmed that the proposed architecture is much faster than the conventional MAC execution by CPUs.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (NRF-2015R1D1A1A09061079). Ki-Seok Chung is the corresponding author.

References

[1] J. Ahn, et al., "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," *Proc. International Symp. on Computer Architecture (ISCA '15)*, pp. 336-392, 2015.

[2] L. Nai, et al., "GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks," *International Symp. on High-Performance Computer Architecture (HPCA '17)*, 2017.

[3] D. Kim, et al., "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," *Proc. International Symp. on Computer Architecture (ISCA '16)*, pp. 380-348, 2016.

[4] M.J. Khurshid and M. Lipasti, "Data compression for thermal mitigation in the Hybrid Memory Cube," *International Conference on Computer Design (ICCD '13), pp. 185-192,* 2013.

[5] N. Venkateswaran, et al., "Memory In Processor: A Novel Design Paradigm for Supercomputing Architectures," *Proc. the 2003 workshop on MEmory performance: DEaling with Applications, systems and architecture (MEDEA '03)*, pp. 19-26, 2003.

[6] B.S.-H. Kwan, B.F. Cockburn and D.G. Elliott, "Implementation of DSP-RAM: an architecture for parallel digital signal processing in memory," *Electrical and Computer Engineering*, pp. 341-346, 2001.

[7] Hybrid Memory Cube Consortium (HMCC), "Hybrid Memory Cube Specification 2.1," available at http://www.hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.1_20151105.pdf. 2016.

[8] S. Rixner, et al., "Memory Access Scheduling," *Proc. International Symp. on Computer Architecture (ISCA '00)*, pp. 128-138, 2000.

[9] D.I. Jeon and K.S. Chung, "CasHMC: A Cycle-accurate Simulator for Hybrid Memory Cube," *IEEE Computer Architecture Letters*, 2016.

[10] N. Binkert, et al., "The gem5 Simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp, 1-7, 2011.

[11] P. Rosenfeld, E. Cooper-Balis and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16-19, 2011.