پیشنهاد پژوهشی

ارایه یک شبکه روی حافظه برای شتاب دهی اجرای شبکه های عصبی در حافظه های سه بعدی

Network-on-Memory for Neural Network Acceleration in 3D- Memories

ارائه شده به: پژوهشکده علوم کامپیوتر، پژوهشگاه دانشهای بنیادی

توسط:

مهدی مدرسی

دانشکده مهندسی برق و کامپیوتر، دانشکده فنی دانشگاه تهران، تهران

modarressi@ut.ac.ir

زمستان 1401

چکیده

پردازش در حافظه (PIM) موثرترین روش برای حل و یا تخفیف دادن گلوگاه پهنای باند در شتاب‌دهنده‌های شبکه عصبی عمیق (DNN) است. با این حال، ساختار الگوریتمی و جریان داده‌های شبکه های عصبی همچنان نیازمند جابجایی مقدار زیادی از داده‌ها بین بانک‌ها در داخل تراشه حافظه است تا داده‌های ورودی و پارامترهای مدل متناظر آن‌ها را برای انجام پردازش در کنار هم قرار دهد: این امربخشی از گلوگاه پهنای باند را به زیرساخت‌های ارتباطی داده‌های درون حافظه منتقل کند. برای کاهش این گلوگاه، ما یک معماری درحافظه بسیار موازی برای شتابدهی شبکه های عصبی در حافظه‌های سه بعدی ارایه می دهیم که از یک شبکه داخل حافظه با پهنای باند بالا و مقیاس‌پذیر بهره می‌برد. در حالی که طرح‌های PIM موجود واحدهای محاسباتی و شبکه روی تراشه را بر روی لایه منطقی حافظه سه‌بعدی زیرین پیاده‌سازی می‌کنند، در روش پیشنهادی، وظایف محاسباتی و انتقال داده‌ها در بانک‌های حافظه توزیع می‌شوند. برای این منظور، هر بانک حافظه مجهز به (1) یک واحد پردازش بسیار ساده برای اجرای شبکه های عصبی، و (2) یک روتر برای اتصال بانک های حافظه توسط یک شبکه سه بعدی روی حافظه است. انتظار این است که استفاده از این معماری موازی باعث افزایش پهنای باند بین بانکها در داخل حافظه شده و کارآیی را بهبود بخشد.

Abstract

Processing-in-memory (PIM) is the most promising paradigm to address the bandwidth bottleneck in deep neural network (DNN) accelerators. However, the algorithmic and dataflow structure of DNNs still necessitates moving a large amount of data across banks inside the memory device to bring input data and their corresponding model parameters together, negatively shifting part of the bandwidth bottleneck to the in-memory data communication infrastructure. To alleviate this bottleneck, we present a highly parallel in-memory DNN accelerator for 3D memories that benefits from a scalable high-bandwidth in-memory network. Whereas the existing PIM designs implement the compute units and network-on-chip on the logic die of the underlying 3D memory, in our proposed architecture the computation and data transmission tasks are distributed across the memory banks. To this end, each memory bank is equipped with (1) a very simple processing unit to run neural networks, and (2) a circuit-switched router to interconnect memory banks by a 3D network-on-memory. By providing a highly parallel architecture and higher inter-bank bandwidth, we expect to get considerable performance improvement over state-of-the-art in-memory DNN accelerators.

1 INTRODUCTION

Convolutional Neural Networks (CNNs) have a wide range of applications due to their superior performance in image and pattern classification. However, the performance of CNNs comes at the price of high computational load and memory bandwidth usage. Hardware acceleration has become the primary way to tackle this ever-increasing complexity of CNNs. However, acceleration puts more stress on the already saturated memory bandwidth, since CNNs are now run faster and need to fetch the model parameters and data in a shorter amount of time, hence more bandwidth is needed. As a result, CNN accelerators are typically bottlenecked by memory, primarily due to the need to move large amounts of model parameters and data between memory and processing units [1][2]. The processing-in-memory (PIM) paradigm is generally known as the most promising way to alleviate this bottleneck. PIM suggests moving computation into/near the memory subsystem [3,4]. Most existing DRAM-based PIM proposals target the emerging 3D memory devices, where multiple DRAM layers are stacked on top of a logic layer. Commercial 3D memory products use the logic layer to implement local memory controller, host interface unit, and simple in-memory arithmetic/logic functional units [3]. PIM leverages the logic layer to implement compute units, often in the form of accelerator cores, hence the computation benefits from the high internal bandwidth available within memory devices.

Although PIM significantly reduce data movement between the memory and computation host, CNN accelerators still need to move large amount of data inside the memory between banks [1,3,5]. Most of this data movement is generated when the output feature map of a layer is sent as input feature map to some remote bank, where the weights of the next layer are stored.

In the existing PIM designs, the data is first moved to the logic layer at the source column and traverses the inter-column network-on-chip (NoC) on the logic layer to reach the destination column. There, the data can be either placed in temporal buffers or written to the memory bank by the local memory controller. Although a good data mapping strategy can cut down part of the traffic on the logic die's NoC, the algorithmic and dataflow structure of CNNs still necessitates transferring large data sizes across different DRAM banks on the logic die to bring together the required operands (input feature map and weights) of different layers. This shifts part of the memory bandwidth bottleneck inside the memory device and to the logic die's NoC. Our observations show that the collective throughput of a PIM-based CNN accelerator is highly sensitive to the bandwidth of this NoC.

To achieve peak performance, this proposal presents a 3D PIM architecture that integrates very simple CNN processing logic to each memory bank and directly connects the banks via a network-on-memory (NoM). NoM implements a scalable and high bandwidth topology to directly transfer data within the memory across DRAM banks, effectively alleviating the internal memory bandwidth bottleneck. In our previous work [6], we showed that a circuit-switched inter-bank network-on-memory improves the performance of data copy-intensive applications (e.g. operating system routines) considerably. This research will show that PIM-based CNN accelerators can also take advantage of NoM to alleviating the internal bandwidth bottleneck of conventional designs. We base the architecture of the processing units of the proposed architecture on the Eyriss accelerator [7]. It uses a row-stationary dataflow design to maximize data reuse.

2 PRELIMINARIES

This section introduces the Eyeriss and Network-on-Memory architectures that are used as the base of the proposed accelerator design.

Eyeriss. We base the architecture of proposed accelerator on the Eyeriss design presented in [7]. Eyeriss uses an array of PEs as functional units and an off-chip DRAM for storing CNN weights and input activations. To minimize input image and weight movement, Eyeriss minimizes three types of data movement such as convolutional reuse, filter reuse, and weight reuse. Eyeriss uses a CNN dataflow called Row Stationary (RS). In RS dataflow each row remains stationary in PEs. In RS dataflow, Filter rows in each line of PE array are the same, and input image rows moving diagonally in PE array. In Eyeriss architecture, one row of the filter is multiplied by one row of the input image. This multiplication is the multiplication of peer-to-peer elements. After multiplying the peer-to-peer elements, the product obtained is multiplied by the input partial sum obtained and then added by the product of the partial sum stored from the previous step to obtain the partial sum of the output of the processing element. Eyeriss minimizes off-chip DRAM accesses. It uses a network-on-chip architecture with multicast and point-to-point support. To support Multi-cast and Point-to-Point, Eyeriss uses a flat multicast NoC. Each PE in Eyeriss consists of a scratchpad, a multiply and Accumulate unit, and a control unit. The main problem of Eyeriss is the PE utilization problem especially when there are multiple PE sets in the PE array. In this case, some PEs remain Idle during CNN computations. Eyeriss benefits from compression and zero data gating techniques for energy efficiency. Figure 1 shows how filter rows are reused in a PE set in Eyeriss (green arrows). All PEs in a row have the same filter row. Figure 1 also shows data reuse of Input image in a PE set in Eyeriss. Each row of the input image is reused across PEs diagonally. Partial sums accumulate vertically across PEs. Psum of each row finally will ready on PEs on the top side of the PE set.

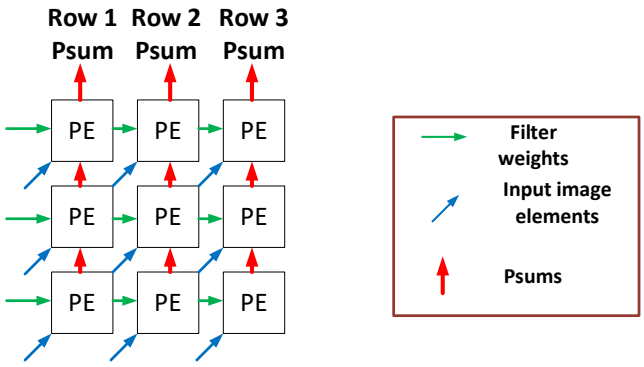

Figure 1. Eyeriss dataflow

Eyeriss v2 is a new version of the original Eyeriss. It has a new NoC architecture in comparison to the original Eyeriss. A Hierarchical Mesh network is used for supporting a wide range of bandwidth requirements. Three types of convolutional layers can be supported by Eyeriss v2. First Conventional layers with high data reuse, depth-wise convolutional layers with no reuse for iacts and only reuse for weights, and finally fully connected layers that have little reuse. For each data type such as iacts, Psum, and weights a separate hierarchical mesh NoC is created. Eyeriss v2 optimized for compact DNNs. When required data reuse is low then its network provides high bandwidth via unicast. In cases in which data reuse is high, it uses spatial data reuse by using Multicast or Broadcast. Eyeriss v2 uses router clusters arranged as a mesh network. Each router cluster is connected to a GLB cluster and PE cluster. Each router cluster consists of four Psum routers, three weight routers, and 3 iact routers. Each PE cluster consists of 12 PEs arranged in a 3×4 manner. Eyeriss v2 has a Global buffer for adding an extra level memory hierarchy. There are two control mechanisms in Eyeriss v2 similar to original Eyeriss. First is a top-level control mechanism controlling off-chip data accesses and traffic between Global Buffer and PEs and second is a control mechanism in each PE that controls the processing process in PE. Input activations read from off-chip into the GLB cluster. Partial sums always store in the GLB cluster. Eyeriss v2 can be scaled with more PEs but the performance may not be scaled due to PE utilization issues. Hierarchical Mesh NoC in Eyeriss v2 uses a circuit-switched routing with only the use of multiplexers.

Network-on-Memory (NoM). We use NoM to interconnect memory banks [6]. NoM is a 3D mesh in which, each memory bank is connected to four adjacent banks in its layer, as well as to the vertically adjacent banks in the upper and lower layers. In [6], a NoM-light design is introduced that uses the already existing vertical links (made by TSV links) of the memory columns to transfer NoM data in the third dimension, effectively eliminating the need for extra vertical links. NoM adopts a circuit-switching scheme with time-division multiplexing (TDM). TDM shares the link bandwidth among multiple packets in time: it divides link bandwidth into some time slots and allocates slots to different circuits. Network switches keep a time slot table for each output link, through which the circuit paths are kept in a distributed way. The content of each slot table entry of an output link is the input port that should be connected to that output at that time slot. Circuit-switched switches are actually simple logic that read the slot table entries at each cycle to make input-output connections. Time slot allocation is a complex task, since it should guarantee that (1) each time slot on a link is not allocated to more than one circuit, and (2) circuits must use increasingly numbered slots in successive switches to avoid the need for buffering along the circuit's path.

In NoM, the memory banks are the network nodes. A centralized circuit setup unit (CSU) on the logic layer receives circuit setup requests and reserves a circuit between the source and destination nodes. CSU is a hardware unit that implements a fast parallel circuit setup algorithm. It explores all shortest paths between the source and destination nodes of the circuit and finds a consecutive sequence of time slots along one of the paths with the earliest starting time. It then (1) returns the starting time of the circuit to the node, when it should deliver its data to the first switch of the circuit, and (2) fills the slot table entries of the switches along the circuit path to set up the circuit. This is done through direct narrow links that connect CSU to each individual NoM switch. CSU needs two cycles to reserve a circuit; one cycle to find a path and one cycle for configuring the slot tables along the path. If NoM adopts B-bit links, the circuit is kept reserved for V/B time windows. The details of SCU circuit setup algorithm is as follows.

The NoM circuit setup is done in the CCU in a centralized manner. The CCU keeps the state of all reserved time slots across the network and services a new request by finding a sequence of time slots along one of the paths between the source and destination banks. The TDM slot allocation is a complex task, as it should guarantee collision-free movement of data.

Specifically, the allocation must guarantee that (1) no time slot of a link is shared by multiple circuits, and (2) a circuit must use increasingly numbered slots in consecutive routers to avoid data buffering. For example, if time slot m is allocated to a circuit in a router, time slot m+1 should be allocated to it in the next router to have the data advance one hop per cycle without any need for buffering, arbitration, and hop-by-hop flow control.

NoM utilizes the fast and efficient centralized circuit setup mechanism presented in prior work [10]. NoM relies on a hard-ware accelerator to explore all the possible paths from the source to the destination in parallel. This hardware accelerator is com-posed of a matrix of simple processing elements (PEs), each associated with a network node. Assuming that the circuit reservation is done on n-slot windows and each router has p output ports, each accelerator PE keeps the occupancy state of its corresponding network node in a 2-D matrix V of size p×n. V[i][j]=1 indicates that the jth slot of the ith output port of the

corresponding network router is reserved. To find a path, the source PE (associated with the source node of the circuit) uses an n-bit vector with all elements initialized to zero. This bit vector keeps track of the empty time slots of different paths and is propagated through all the shortest paths to the destination. If time slot x is free in this router, we need time slot x+1 to be free in the next router. Hence, in each PE, the bit vector is first rotated right and then ORed with the vectors corresponding to the output ports along the shortest path (to mask busy slots). It is then passed to the next PE towards the destination cell. Available circuit paths (sequence of time slots) appear as zero bits in the vector at the destination PE. The circuit path and time slots can be reserved by tracing back the path towards the source PE.
With B-bit links, B bits of data is transferred on NoM link in each cycle. If a circuit has V bits to transfer, the time-slots remain reserved for V/B time windows. The data transfer can be accelerated by reserving multiple slots, provided that the algorithm returns more than one free slot. After that, the algorithm is allowed to use the time-slot for the next requests.

3 THE PROPOSED DESIGN

The proposed design equips the baseline 3D memory with (1) one circuit setup unit (CSU) in the logic layer, (2) one column controller unit in each logic layer slice, (3) one neural network processing unit in each memory bank, and (4) one Network-on-Memory (NoM) switch in each memory bank.
In this design, each DRAM bank comes with a Neural Processing Unit (NPU) that runs part of the CNN mapped to the bank. The banks are interconnected by the circuit-switched NoM, with each bank is a NoM node. In the logic layer, each column has (1) a conventional local memory controller, which generates the required signalling to read/write memory for the banks of its column, and (2) a *Column Controller Unit* (CCU), which controls the NPUs of the memory slices of the column. This column controller, together with a program at the host and a sequencer at each bank, comprise the control plane of the proposed architecture.
Local memory controllers serve three types of requests: regular memory accesses and DRAM refresh requests, memory accesses made by CCU to fetch data for NPUs, and write requests made by CSU. In contrary with the regular memory accesses, memory accesses made by CCU will not transfer data between the bank and logic layer, rather CCU appropriately configures the target bank to direct the data to its local NPU.
NPU implements the architecture of the Eyriss [7]. Datapath is an array of functional units. Each functional unit consists of a multiplier, register, and accumulator. The functional units process an array of input pixels in parallel. Input Buffer (IB) and Weight Buffer (WB) keep the input and filter elements. Output Buffer (OB) is a temporal buffer that keeps the results.
The calculated results should be sent over the network to be written to the right memory bank. To this end, each block of OB has a counter to keep track of the accumulated partial sums. Once this counter indicates that the accumulation is completed, the index of the OB is sent to the CCU. The block is then written to a temporary buffer to be sent over the network when its circuit is set up. CCU maps the output buffer index to the memory address (bank number and the address inside the bank) and sends a circuit setup request to the circuit setup unit (CSU).
CSU queues the data transfer requests coming from all columns and takes the following sequence of actions for each request. (1) CSU invokes the circuit setup algorithm to find a path (in the form of a sequence of time slots in a chain of switches) between the source and destination banks. CSU fills the slot tables along the path using dedicated links to set up the circuit. It also returns the starting time of the circuit to CCU. Starting time is the cycle at which the circuit is available to the source node and data should be delivered to the first switch of the circuit. (2) On the starting time, the NoM switch of the bank read the data and injects it to the network. (3) The data traverses the pre-established circuit one hop per cycle to reach and get buffered at the destination. CSU knows when the data arrives at the destination bank because circuit-switching has deterministic transmission time. (4) Finally, the CSU sends a write request to the destination memory controller to write the received block to the right address at the destination bank. The memory controllers prioritize CSU write requests over other requests, since the read/write operation of the data traveling on the network is predetermined and the corresponding signals should be generated according to the CSU's schedule made during circuit setup.

4 Research Plan

In this research, we will answer to two questions: (1) how much performance improvement will be achieved by using a network-on-memory to interconnect banks and functional units inside the memory device and (2) how much overhead does the proposed architecture impose and how feasible is the circuit-level implementation of the proposed accelerator architecture inside the memory.
To evaluate the performance of proposed architecture, we will implement it on Ramulator [8] by adding the required functions for circuit-switching and Eyeriss-based computation to memory banks. Circuit-level parameters and memory timing parameters will be set based on DDR4 DRAM. Then, the impact of network-on-memory on pushing the bandwidth bottleneck inside the memory device will be evaluated.
We expect to get considerable performance improvement, mainly due to providing more parallelism in data communication inside the memory device. Note that the proposed architecture adopts large 3D mesh that directly connect memory banks and associated NPUs. However, conventional PIM designs, like NeuroCube, adopts an 2D mesh network that only connect the NPUs which are implemented on the logic layer. This reduces the traffic load on the proposed architecture links, thereby the blocking latency is reduced. Although the smaller network of Neurocube can cut off the network diameter, the potential latency

reduction is expected to be nullified and overcome by the higher traffic on NoC links and injection ports. Note that in NeuroCube, all data (input feature map, output feature map, and filter elements) may potentially be sent over the net-work to reach to the column where they should be processed. However, in our design, most read transactions are performed locally from the local memory bank and only writing calculated output feature map elements are sent over NoM to be written in the right destination memory bank.

The required tasks to complete this research are as follows.

1- Implementing the NPU architecture based on the Eyeriss architecture

2- Designing a data mapping and scheduling for data movement and computation based on the data flow of Eyeriss. Mapping deals with placing input image, intermediate feature maps, and CNN filter weights into the physical address space of the memory. Scheduling algorithm specifies where and when the computation of a CNN layer is carried out and how the data is transferred between the banks.

3- implementing the design at the circuit level to measure the circuit-level specifications, namely power consumption, area, temperature, and critical path. If the design violates the timing of the 3D memories or the temperature constraint of the logic or memory layers, the architecture must be revised. For the timing, many prior work, set the working frequency of the logic layer of 3D memories to 1.25GHz [3,5]. In order for our design to keep pace with this speed, NoM, circuit setup unit, and NPUs should have critical paths below 0.8ns.

References

[1] G. Oliveira, J. Gomez-Luna, S. Ghose, A. Boroumand and O. Mutlu, "Accelerating Neural Network Inference with Processing-in-DRAM: From the Edge to the Cloud" in *IEEE Micro,* 2022.

[2] M. Daneshtalab and M. Modarressi. *Hardware Architectures for Deep Learning*. IET publishers, (2020).

[3] O. Mutlu, et al., “A Modern Primer on Processing in Memory”, in *Emerging Computing: From Devices to Systems. Computer Architecture and Design Methodologies- Chapter 4*, Springer Pubs, 2023.

[4] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna and O. Mutlu, "Processing-in-memory: A workload-driven perspective," in *IBM Journal of Research and Development*, 2018.

[5] D. Kim, J. Kung, S. Chai, S. Yalamanchili and S. Mukhopadhyay, “Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory", *in Proc. of ISCA*, 2016.

[6] S. H. S. Rezaei, M. Modarressi, R. Ausavarungnirun, M. Sadrosadati, O. Mutlu and M. Daneshtalab, "NoM: Network-on-Memory for Inter-Bank Data Transfer in Highly-Banked Memories," in *IEEE Computer Architecture Letters*, 2020.

[7] Y.-H. Chen, J. Emer, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 367–379, 2016.

[8] Y. Kim, et al., "Ramulator: A Fast and Extensible DRAM Simulator," in *IEEE Computer Architecture Letters*, 2016.

[9] Micron, DDR3 SDRAM system-power calculator, 2018.

[10] F. Pakdaman, A. Mazloumi, M. Modarressi, “Integrated Circuit-Packet Switching NoC with Efficient Circuit Setup Mechanism”, in *Springer Journal of Supercomputing*, vol. 71, no. 8, 2015.