4-2019

A Novel Processing-In-Memory Architecture for Dense and Sparse Matrix Multiplications

Andrew Robert Bear
arb8434@rit.edu

A Novel Processing-In-Memory Architecture for Dense and Sparse Matrix Multiplications

ANDREW ROBERT BEAR

A Novel Processing-In-Memory Architecture for Dense and Sparse Matrix Multiplications

ANDREW ROBERT BEAR

April 2019

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

R·I·T | KATE GLEASON
College of ENGINEERING

Department of Computer Engineering

A Novel Processing-In-Memory Architecture for Dense and Sparse Matrix Multiplications

ANDREW ROBERT BEAR

Committee Approval:

Abstract

Modern processing speeds in conventional Von Neumann architectures are severely limited by memory access speeds. Read and write speeds of main memory have not scaled at the same rate as logic circuits. In addition, the large physical distance spanned by the interconnect between the processor and the memory incurs a large RC delay and power penalty, often a hundred times more than on chip interconnects. As a result, accessing data from memory becomes a bottleneck in the overall performance of the processor. Operations such as matrix multiplication, which are used extensively in many modern applications such as solving systems of equations, Convolutional Neural Networks, and image recognition, require large volumes of data to be processed. These operations are impacted the most by this bottleneck and their performance is limited as a result.

Processing-in-Memory (PIM) is designed to overcome this bottleneck by performing repeated data intensive operations on the same die as the memory. In doing so, the large delay and power penalties caused by data transfers between the processor and the memory can be avoided. PIM architectures are often designed as small, simple, and efficient processing blocks such that they can be integrated into each block of the memory. This allows for extreme parallelism to be achieved, which makes it ideal for big data processes. An issue with this design paradigm, however, is the lack of flexibility in operations that can be performed. Most PIM architectures are designed to perform application specific functions, limiting their widespread use.

A novel PIM architecture is proposed which allows for arbitrary functions to be implemented with a high degree of parallelism. The architecture is based on PIM cores which are capable of performing any arbitrary function on two 4-bit inputs. Nine PIM cores are connected together to allow more advanced functions such as an 8-bit Multiply-Accumulate function to be implemented. Wireless interconnects are utilized in the design to aid in communication between clusters. The architecture will be applied to perform matrix multiplication on dense and sparse matrices of 8-bit values, which are prevalent in image and video formats. An analytical model is proposed to evaluate the area, power, and timing of the PIM architecture for both dense and sparse matrices. A real-world performance evaluation will also be conducted by applying the models to image/video data in a standard resolution to examine the timing and power consumption of the system. The results are compared against CPU and GPU results to evaluate the architecture against traditional implementations. The proposed architecture was found to have an execution time similar to a GPU implementation while requiring significantly less power.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Modern computing systems are designed to perform tasks as quickly and efficiently as possible. Performance improvement is normally seen through microprocessor advancements such as architecture modifications, increased clock frequencies, and increased core counts. This allows modern processors to perform exponentially better than previous generations. The computing systems as a whole, however, are not seeing the same rapid improvements due to a large bottleneck that exists between the processing units and the main memory. Dynamic Random Access Memory (DRAM), which is used as main memory in most computing systems, has seen limited generational improvement when compared to processors. Processor performance has been scaling with Moore's law at a rate of around 70 percent improvement per year [1]. In contrast, DRAM speeds have been increasing by 7 percent per year [1]. This performance gap is caused by divergent goals between the microprocessor and memory fabrication industries. Microprocessor production focuses on developing increasingly fast devices while memory production is focused on creating high capacity memory modules by minimizing data cell size. This growing performance gap between processing and memory is exemplified in Figure 1.1.

This phenomenon, known as ”The Memory Wall” [3], severely limits the effective speed of modern computing systems and prevents the rapid performance improvements that are seen in the processors. The slow effective speed of the memory is limited by

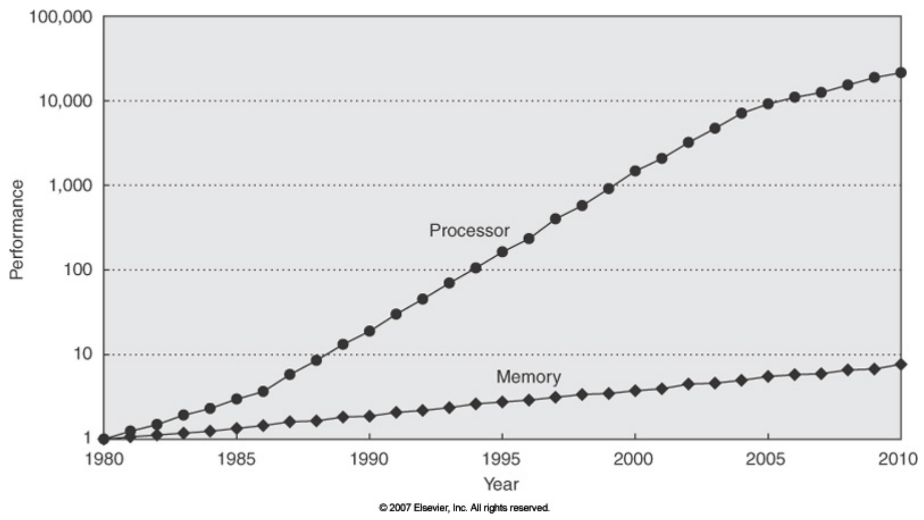


Figure 1.1: Growing performance gap of processors and DRAM memory compared to technology in 1980 [2].

two main factors: long access times to read data out of DRAM and a large delay penalty caused by transferring the data over I/O channels. To help alleviate this large delay, many modern designs make use of high speed Static Random Access Memory (SRAM) that are physically close to the processor, known as cache. In modern DDR3 memory, the time to read data out of DRAM can take upwards of 10ns [4], not including time to transfer the data to the processor. In comparison, reading data out of SRAM takes only 0.3ns [4]. Since SRAM is constructed using CMOS technology, it also benefits from improved speeds from technology node scaling. As a result, SRAM speeds typically scale with processor performance.

In addition to the long delay penalty, accessing data from main memory also incurs significantly more power than SRAM. A DRAM access can use 1-2nJ of energy while embedded cache memory uses only 10pJ [5]. The energy required to transfer the data to the processor can also be upwards of 20pJ, not including the static power required to keep the I/O channel functional [5]. With repeated accesses to main memory, a significant amount of power is wasted due to the interconnects.

The impact of the large main memory delay can be seen easily seen through the basic equation for calculating the average latency of a memory access shown in Equation 1.1 [3].

$$t_{avg} = p \times t_c + (1 - p) \times t_m \tag{1.1}$$

In Equation 1.1, t_c is the access latency of cache memory, t_m is the access latency of main memory, and p is the probability of finding the data in cache. Since, p is effectively the probability of a cache hit, $(p - 1)$ is therefore the probability of a cache miss, or the probability of needing to fetch the data from main memory. In an ideal cache system, only cache misses should access main memory to retrieve new data that has not been previously accessed. In this case, the probability of finding the data in cache, p approaches one. The chance of retrieving data from main memory, $(p - 1)$, is therefore very small but nonzero. Based on Equation 1.1, as processor speeds increase and cache latency decreases, the average memory latency will be increasingly dominated by the main memory's $(1 - p) \times t_m$ term.

In order to see increased performance benefits from modern computing systems, new innovations are required to reduce the impact of the Memory Wall. One such innovation is the idea of Processing-in-Memory (PIM). In a PIM based architecture, the number of data transfers between processing and memory can be reduced by performing some of the operations within the memory itself. In doing this, the computation to communication ratio can be improved and less time will be wasted by data transfers.

Chapter 2

Background

2.1 Processing-in-Memory

Processing-in-Memory (PIM) is one proposed solution to the ”Memory Wall” problem that modern computing systems are facing. Speed improvements are limited by the large latency of accessing data from DRAM and transferring the data to the processor. Some of this latency can be reduced by using high speed cache memory hierarchies to store data closer to the processor, however these memory blocks have a much larger area compared to DRAM, limiting the amount of storage that can be put onto a single die [4]. Even with these cache hierarchies, large capacity DRAM is still required for managing larger datasets, which will incur a large latency penalty when accessed. PIM solves this issue by processing the data closer to or within the memory itself, reducing or eliminating the latency of memory accesses. An architecture comparison between PIM and traditional architectures is shown in Figure 2.1.

Processing-in-Memory has been considered since at least 1970 [7], however it has not been a feasible option until more recent years. Previously, incorporating large scale DRAM technology with standard CMOS logic on the same die results in fabrication complexities that make it an unfeasible solution. Modern technology has allowed potential PIM architectures to flourish. Three-dimensional stacked RAM such as Micron’s Hybrid Memory Cube (HMC) [8] and AMD’s High-Bandwidth Memory (HBM) [9] allow traditional CMOS logic to be constructed on a logic layer with quick access to a stack of DRAM dies which

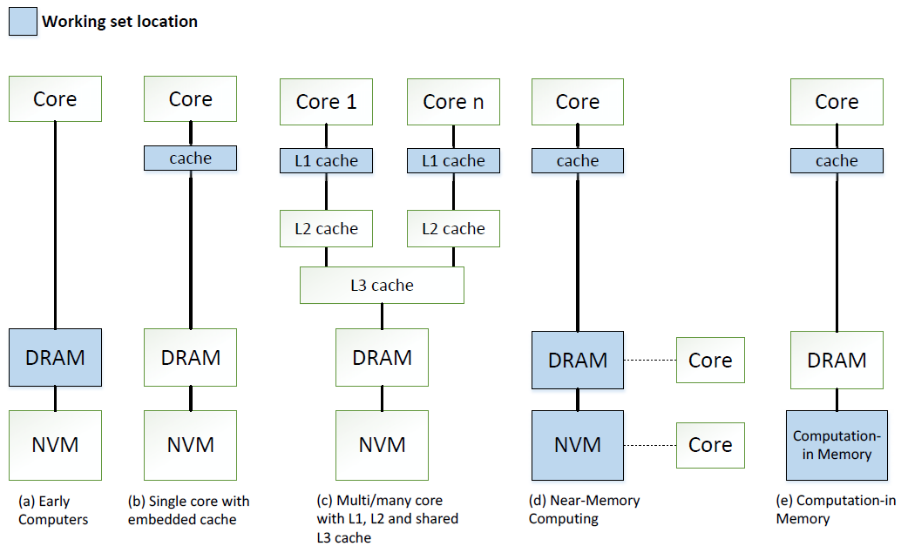

Working set location
Core
cache
Core 1
Core n
L1 cache
L2 cache
L3 cache
DRAM
NVM
Core
Computation-in Memory
(a) Early Computers
(b) Single core with embedded cache
(c) Multi/many core with L1, L2 and shared L3 cache
(d) Near-Memory Computing
(e) Computation-in Memory

Figure 2.1: Architecture overview of traditional PIM paradigms based on the location of the working set of data. Figures (a), (b), and (c) show typical Von Neumann architectures with a separation of processing circuits and memory. Data is passed from DRAM or cache memory to be operated on by the processing core. The Near-Memory Computing paradigm is shown in (d), where processing cores are placed very close to the DRAM and Non-Volatile Memory. In doing so, the delay of transmitting data to a processing unit can be reduced due to the shorter travel distance. The Processing-in-Memory architecture, also called Computation in Memory, is shown in (e). Processing units are embedded directly within the DRAM memory, reducing delay by eliminating the need for costly data transfers. [6].

incurs minimal latency. Further development of HMC, however, will not be supported by Micron. Other approaches include using specialized circuits within the RAM blocks [10] or using novel Non-Volatile Memory devices such as Resistive RAM (ReRAM), which is constructed using memristors, to perform logic within the memory cells [11]. Using these advancements, small and efficient PIM architectures can be designed for a wide range of applications. Typically, these architectures are designed to be application specific and offer minimal flexibility. Some examples of application specific PIM architectures are a Multiply-Accumulate engine [12], neural network engines [13, 14], and image recognition [15].

2.2 Matrix Multiplication

The Multiply-Accumulate (MAC) Operation is a powerful function that is used in a wide range of applications. It is the primary operation when performing matrix multiplication, which is used frequently by convolutional neural networks (CNNs). CNNs are very popular in modern technology trends due to their ability to break down and classify data, commonly for image recognition and other machine learning applications. The equation for a MAC operation is given in Equation 2.1.

$$A \leftarrow A + B \times C \tag{2.1}$$

In Equation 2.1, B and C are the primary operands to multiply together. The result of the operation is then added to the accumulator, A. After repeated MAC operations, the accumulator will contain the final summation of each multiplication product. In a hardware implementation with a fixed bit length where B and C have n bits, the size of the accumulator must be at least $2n$ bits long to successfully store the result of one MAC operation. To prevent overflow in repeated MAC operations, one extra bit is needed for every two successive operations to store the carry out of the addition.

A large issue with the MAC operation, however, is its high memory access frequency. During each MAC operation, the B and C values must be retrieved from memory. These values are not guaranteed to be stored close to each other in memory, making efficient caching difficult. This can cause a large memory access penalty by frequently retrieving values from main memory, leading to a low computation to communication ratio and degrading the overall performance. One potential solution to this problem is to use two separate memories to hold B and C in order to maximize spatial locality of successive operands.

Multiplication of two matrices can be performed using repeated MAC operations on matrix elements. Multiplication of matrices A_{mxp} and B_{pxn} to produce a new matrix, C_{mxn}

is shown below where each element of the matrix C_{m_xn} is given by Equation 2.2.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mp} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{p1} & b_{p2} & \dots & b_{pn} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=0}^{p} a_{ik} b_{kj} \tag{2.2}$$

During each iteration of the summation in Equation 2.2, elements of the A_{m_xp} and B_{p_xn} matrix are multiplied together and the product is accumulated for all iterations. Using this, the element of the resulting matrix c_{ij} can be calculated through a series of MAC operations. In addition, each element of the C_{m_xn} matrix is independent from the other elements and can therefore each element can be computed in parallel.

2.3 Networks-on-Chip

Advances in fabrication technology have allowed integrated circuit designs to be exponentially larger and support extremely complex designs. Managing the flow of data across these large, complicated designs has steered circuit designers towards the use of Systems-on-Chips (SoCs) in order to ease the design. Systems-on-Chip design consists of many independent modules being connected together, typically through a shared bus. This eases the design process by allowing components to be reused between designs. Instead of designing commonly used components from scratch, Intellectual Property (IP) can be purchased or reused from previous designs. The design process then shifts to the integration and connection of these parts rather than low level module design.

A large issue with this paradigm, however, is the limited scalability for large designs with many components. Interconnect strategies such as shared buses cannot effectively support a large number of components on the bus due to contention over the shared medium. SoCs also face issues such as global synchronization due to the difficulties of creating a high speed, robust clock across the chip [16]. In response to this, recent design trends have shifted towards the Network-on-Chip (NoC) paradigm.

In a Network-on-Chip, the components are linked together through the use of switches and routers, similar to traditional computer networks. This allows multiple independent links to be used to pass data between components rather than a single shared medium. Efficient design of the network topology connecting the components can allow them to communicate with one another quickly and with a minimal number of hops between routers. Topologies such as Small World Networks allow for a high scalability by minimizing the typical distance between any two nodes in the network through the use of both short and long distance links [17]. Designing for scalability allows new nodes to be added to or removed from the network with minimal impact on the number of hops required to reach any other node, which is ideal for networks with a large number of nodes.

2.4 Wireless Networks-on-Chip

Improvements in the fabrication process and technology node shrinks in continuation with Moore's Law have allowed for the production of increasingly complex circuits. Connecting the components in these circuits becomes an issue, however, as metal interconnects are not scaling in performance at the same rate as the transistors. The NoC paradigm helped to alleviate this issue by offering a more efficient communication framework for components on the chip, however its effectiveness is limited by the physical distance separating the components. The performance of long metal wires is limited by the RC delay incurred by the wires, which becomes more apparent as the technology node continues to shrink. Unique solutions have been proposed to address this issue such as 3-dimensional integrated circuits [18], photonic interconnects [19], and wireless interconnects [17].

Wireless interconnects are designed to overcome the large RC delay of long metal wires by using electromagnetic waves to send and receive data between modules. Metal wires are still used for short routing, however the integrated wireless transceivers working at 16Gb/s [17] allow data to be sent across long distances in significantly less time and requires less power. Control of the shared wireless medium requires accessing a Multiple Access Control (MAC) protocol to be implemented, such as Code Division Multiple Access (CDMA), Time Division Multiple Access (TDMA), or Carrier-Sense Multiple Access-(CSMA) [17].

Some drawbacks of using wireless interconnects include the overhead of the MAC protocol, increased area due to the wireless transceiver circuits, and increased fabrication difficulty. They make up for these drawbacks, however, by offering high speed data transfer across large on-chip distances. In addition, multi-casting and broadcasting of data to multiple wireless transceivers is inherently supported by the architecture as multiple receivers can read the same transmission simultaneously.

2.5 Supporting Work

Due to the manufacturing complexity of integrating dense logic and DRAM on the same die, processing in memory was a relatively unexplored field for many years. A new rush of PIM designs was recently enabled by Micron's Hybrid Memory Cube (HMC) architecture [8], which uses 3D stacked memory to enable logic and memory to be closely integrated. HMC is constructed by stacking 4-8 dies of DRAM memory onto a base logic layer. All of the layers in the stack are connected using Through-Silicon Vias (TSVs), which act as high speed buses to transmit data between the layers. Many PIM architectures use HMC as a basis for their designs due to the ability to integrate computational logic within the logic layer of the package, which enables computations to be performed very close to memory. Micron has since stopped their support of HMC. An alternative to HMC comes in the form of High-Bandwidth Memory (HBM) created by AMD [9]. Similar to HMC, HBM uses a 3D stack of DRAM dies connected by TSVs. Unlike HMC, however, HBM is designed to connect directly to a CPU/GPU through a silicon interposer which offers high speed communication between the components. Future PIM designs will likely be designed around the use of HBM memory.

Typically, PIM architectures are designed to perform a single task such as Multiply-Accumulate (MAC) operations [12], Convolutional Neural Network (CNN) operations [14], or image processing [15]. Other PIM architectures have been designed for database searches, data analysis, and graph processing [6]. These architectures are able to offer a significant speedup over conventional computing for their designed tasks but lack the flexibility to be used in other situations. In doing so, these designs limit their viability outside of application specific hardware. In contrast to this, the proposed architecture will be reconfigurable to allow for arbitrary functions to be implemented.

Currently, a matrix multiplication algorithm is envisioned using the proposed architecture. This will allow rapid computations for use in CNNs, which are used frequently in artificial intelligence and image recognition applications. As a result, the proposed architecture can be compared to both [12] and [14]. The architecture proposed in [12] makes use of HMC to incorporate MAC processors near the data lines that access the 3D stacked memory. In each HMC-MAC instruction, a series of MAC operations are performed using data supplied by the host processor and the memory. An HMC controller is added to the host processor and is used to assemble the host supplied values for the MAC operation as well as manage communications with the HMC module using one of 6 created instructions. Similar to the proposed architecture, [12] is capable of performing multiple repeated MAC instructions in parallel across the available memory. By constructing the MAC unit close to the high bandwidth data line of the stacked DRAM, a high volume of data can be processed quickly with minimal communication time.

Another comparable architecture is shown in [13], which is designed to perform CNN functions using HMC. In their design, CNN Logic Units (CLUs) are placed near the high bandwidth data lines of the HMC module. The CLUs are designed to perform convolution operations which form the basis of CNNs. The design makes use of a floating point multiplier, floating point adder, and embedded SRAM to quickly apply filter values to a dataset. The floating point operations are done using IEEE-754 double precision format to maintain accuracy. An issue with this, however, is the large silicon area required to implement it due to the complex circuitry required. This is an issue in PIM applications where a small, low impact circuits are a priority.

Both [12] and [13] show significant improvement over conventional processing architectures due to their highly efficient specialized designs. The minimal data communication time between the processing element and the memory allowed both architectures to obtain effective processing rates that were far greater than a traditional processing and memory structure. Where both architectures falter, however, is for general use cases. In applications which do not directly conform to their intended domain, both architectures would offer no performance improvements while still taking up valuable silicon area on the die. The proposed architecture aims to improve upon this by allowing for full reconfigurability of the operation to perform. Since any arbitrary function can be implemented using the proposed architecture, any theoretical workload which requires frequent memory accesses can instead be performed in the memory with significantly reduced communication latency.

Many modern PIM designs also focus on the use of Resistive Random Access Memory (ReRAM) to perform arithmetic operations within the memory storage units themselves. ReRAM is a type of non-volatile memory built using memristors, which store data using programmable resistive states. Both a high and low resistive states can be stored into a memristor by controlling the voltage difference across it. Binary values of ”0” and ”1” can then be read as a function of the current through the memristor, where a high resistive states equate to logic ”0” and low resistive states are logic ”1”. State-of-the-art PIM designs such as [20], [21], and [22] use this ability to implement functions directly within the memory cells.

In [20], a ReRAM crossbar array is used to perform neural network operations using the resistive states of the memory cells. Inputs are sent to the array as analog voltages and the weights are represented as the resistive states. The current flowing through the outputs is based on the resistances of the cells and determines the output value. This design is capable of very fast and compact operations by using the memory cells as logic units, however it requires a wide range of dedicated peripheral hardware to function. To perform the neural network functions, the design requires digital-to-analog converters, analog-to-digital converters, sigmoid units, and subtraction circuits [20]. This array of peripheral circuitry tailored to perform neural network operations limits the overall flexibility of the design as supporting other logic functions would require additional peripherals.

The designs proposed in [21] and [22] are similar to [20] in that they perform neural network operations within the memory cell array. They differ, however, by implementing a pipelined based approach where layers of the neural network can be processed in a parallel manner. Both designs use pipeline stages to execute small portions of the layers at a time and use the results in future pipeline stages. The design in [22] improves upon [21] by avoiding pipeline bubbles and maintaining a constant series of operations through the pipeline such that the efficiency is optimized. Similar to the issues described for [20], these designs are highly specialized to perform neural network operations, reducing their flexibility to be used for a wide range of tasks. Implementation of other functions would require a redesign of the architecture, or additional peripherals to be constructed.

Chapter 3

Processing-in-Memory Architecture

The center of the proposed architecture is the PIM Core which functions as the primary logic unit. The PIM Core is capable of performing reconfigurable functions on two 4-bit inputs to produce an 8-bit output. Nine PIM cores were then grouped together to form a PIM Cluster to perform larger operations such as Multiply-Accumulate (MAC) using 8-bit operands. An array of PIM Clusters can then be used to perform large scale operations such as matrix multiplication in parallel.

3.1 PIM Core

The primary logic of the proposed architecture is the small, simple PIM Core which is capable of performing arbitrary 4-bit operations. A primary objective of this architecture is flexibility to support a wide range of potential applications. To achieve this, the PIM Core functions similarly to a large LookUp Table (LUT) with the function to implement being stored in memory. This allows the PIM Core to implement any arbitrary function based on the data that is stored in memory. A block diagram of the PIM core is shown in Figure 3.1.

The LUT functionality is achieved using an 8-bit, 256-to-1 multiplexer. The 8-bit multiplexer is required to support the multiplication of two 4-bit operands, which results in an 8-bit result. The multiplexer has eight select lines which are used to determine the 8-bit output given 256 8-bit options. The select lines are controlled by two 4-bit inputs which serve as the operands.

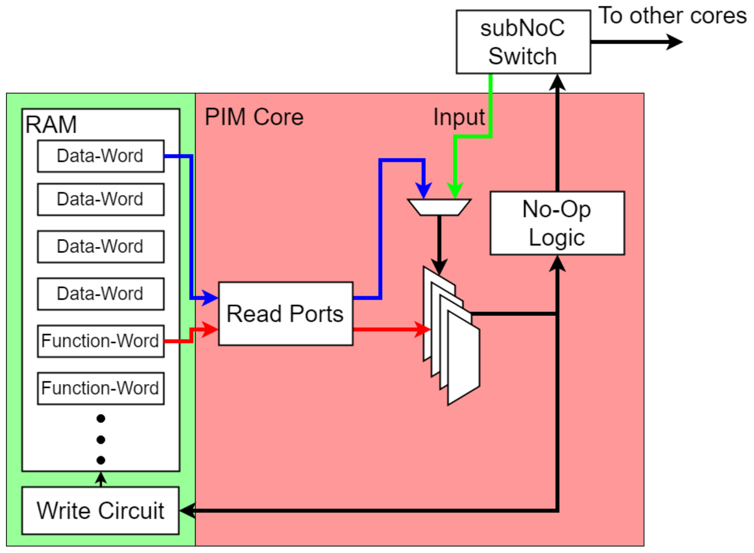


Figure 3.1: High level block diagram of the proposed PIM Core. The PIM Core logic is shown within the red boundary.

The inputs to the multiplexer representing the function to implement are read directly from the Random Access Memory (RAM) block and are referred to as *function-words*, shown in Figure 3.1 as the red arrow. New functions can therefore be implemented on the PIM Core by reading a new set of *function-words* from the RAM block using the read port. Many sets of *function-words* can be stored in the memory to allow for rapid reconfiguration of the PIM Core function to implement. This allows the PIM Core to implement addition, subtraction, multiplication, or other Boolean functions dynamically by reading new values from RAM, allowing the PIM Core to remain flexible to perform different tasks while still being constructed using simple logic.

The two 4-bit inputs which are connected to the select lines of the multiplexer and serve as the operators to the function are referred to as *data-words*. These values can either be obtained from RAM, or they can be given through a sub Network-on-Chip (subNoC) switch which is attached to the PIM Core. Values obtained from the RAM block, shown in Figure 3.1 using a blue arrow, can be used to process data stored in memory by a host device. In contrast, values obtained from the subNoC switch, shown as a green arrow, can be received from other PIM Cores. The subNoC is an interconnection fabric which is used to transmit data between locally adjacent cores contained within the same PIM Cluster. The subNoC connections are discussed further in Section 3.3. These connections between cores allows larger functions to be implemented using multiple PIM Cores, as results from other operations can be used as inputs to the PIM Core.

The included No-Op logic block will be used reduce total communication time for sparse data applications. When the result of a PIM Core operation is equal to zero, the No-Op detection block will prevent the data from being sent out of the core. A zero value will then be inferred by the lack of data from the core and no additional time or energy will be used to transmit the data.

3.2 Multiply-Accumulate Using PIM Cores

To complete matrix multiplication operations, an 8-bit Multiply-Accumulate (MAC) function was implemented using a PIM Cores, resulting in a 16-bit value. To achieve this, the 8-bit MAC instruction was decomposed into a series of 4-bit operations which can be performed by the PIM Cores. The Multiply-Accumulate (MAC) instruction contains two primary operations, an 8-bit multiplication and a 16-bit addition. The multiplication can be broken down into a series of 4-bit multiplications resulting in 8-bit partial products, which can then be added together to form the final result. The partial products V_x are defined as follows, where the subscripts H and L represent the upper and lower four bits of the operand respectively:

1. $V_0 = a_L \times b_L$
2. $V_1 = a_L \times b_H$
3. $V_2 = a_H \times b_L$
4. $V_3 = a_H \times b_H$

After obtaining the partial products V_x through multiplication, the final product Y can be obtained through cascaded addition. This process is shown in Figure 3.2. The multiplication result of two 8-bit numbers will always be contained within 16-bits, therefore there is no need for overflow protection.

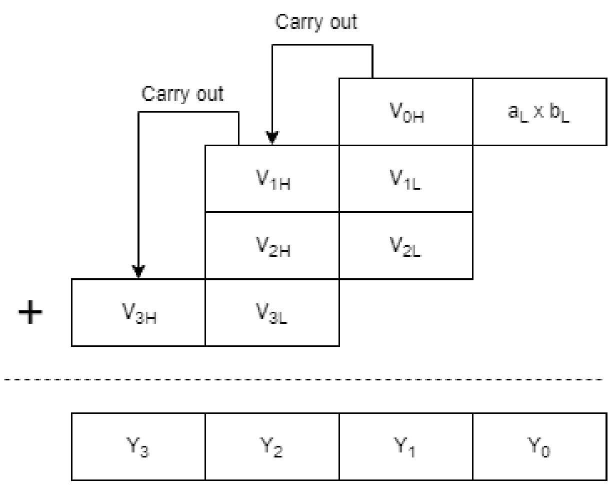


Figure 3.2: Decomposition of 8-bit multiplication into a series of 4-bit multiplications and additions.

The 16-bit addition operation can be decomposed into a series of 4-bit additions in a similar manner. Ripple-carry addition is used to compute the final 16-bit result. This process is shown in Figure 3.3 in the context of the MAC operation, where the 16-bit multiplication result Y is added to the 16-bit accumulator A and the result is stored into A.

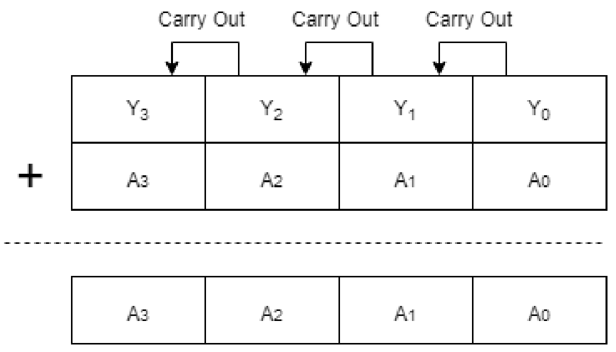


Figure 3.3: Decomposition of 16-bit addition into a series of 4-bit additions.

The size of the accumulator can be increased in order to account for additional overflow which is generated through repeated additions. In this work, the accumulator size was limited to 16 bits, however it can be increased to 20 bits by adding the carry out from the computation of A_3.

The multiplication and addition algorithms can then be translated to be performed using PIM Cores. A sequential model for performing the MAC operation was created where the output of each PIM core is sent directly to the input of another core. A diagram of the sequential model PIM MAC operation is shown in Figure 3.4, where each PIM Core is given a unique identifying label. Each core within the diagram represents a single 4-bit operation of the decomposed multiplication and addition operations. The left side of the diagram represents the operations required to perform the addition of partial products, while the right side of the diagram shows the 16-bit addition operation.

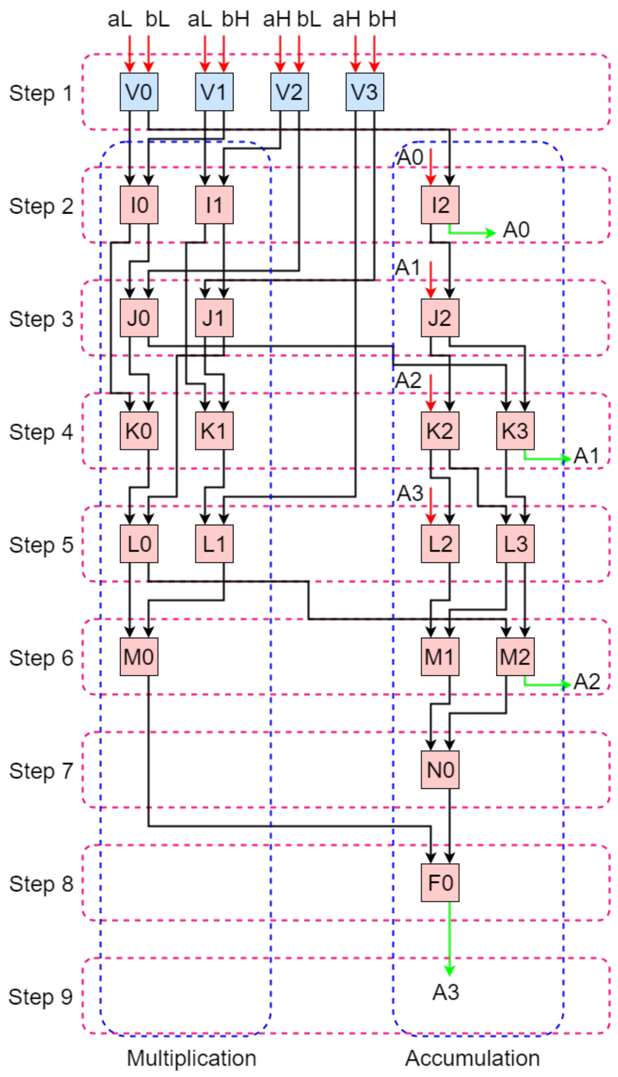


Figure 3.4: Sequential model of 8-bit MAC operation. In the figure, blue boxes represent 4-bit multiplication and red boxes represent 4-bit addition. The arrows coming out of each box represent the upper and lower 4-bit results of the core's operation. The upper four bits are denoted by the left arrow, while the right arrow represents the lower four bits. The red arrows designate inputs to the system in the form of the inputs a and b as well as the accumulator A. The final accumulator results are denoted with a green arrow.

The sequential model is designed such that each PIM Core will be utilized once during the MAC instruction. This configuration simplifies the flow of data, however it uses significantly more resources than is necessary to complete the instruction. In the sequential model, 23 PIM Cores are required to complete the operation. By reusing cores to compute multiple steps of the operation, the total number of cores needed can be reduced to 9, four for multiplication and five for addition. Instead of using fixed paths, this new model, designated as the compact model, uses data destinations that change based on which portion of the MAC operation is being computed.

The complete set of steps required to execute the compact model are shown in Figure 3.5. During each step in the figure, the cores are labeled to match the corresponding operation from the sequential model in Figure 3.4. Similar to the sequential model, black arrows show communications between the cores, the red arrows designate inputs to the system, and the green arrows denote completed accumulator outputs. In Figure 3.5, the blue arrows represent the core internally holding a result from its operation for use in the next calculation, reducing the amount of data that needs to be transmitted. This MAC operation can be repeated multiple times using a shared accumulator value in order to execute dot product operations, which will be used to perform matrix multiplication.

3.3 PIM Cluster

In order to facilitate larger operations such as the proposed 8-bit MAC scheme, groups of nine PIM Cores connected together to form a PIM Cluster. The nine PIM Cores within a cluster are connected using the sub Network-on-Chip (subNoC), an all-to-all network such that any core can directly send data to and receive data from any other core within the cluster. This interconnection fabric is shown in Figure 3.6. Communication between the PIM Cores is implemented through the transmission of single 32-bit packets, or Flow Control Units (flits).

This interconnection fabric was chosen due to the low data output size, close physical proximity, and high speed requirements of the PIM Cores. By directly linking cores together, the high overhead of traditional Network-on-Chip (NoC) architectures such as ring and mesh networks can be eliminated. The overhead required in these networks includes area and power used by the switches and routers in addition to increased latency due to multi-hop routing. Using the All-to-All network, delay penalties are limited to the RC delay of the wires connecting the cores.

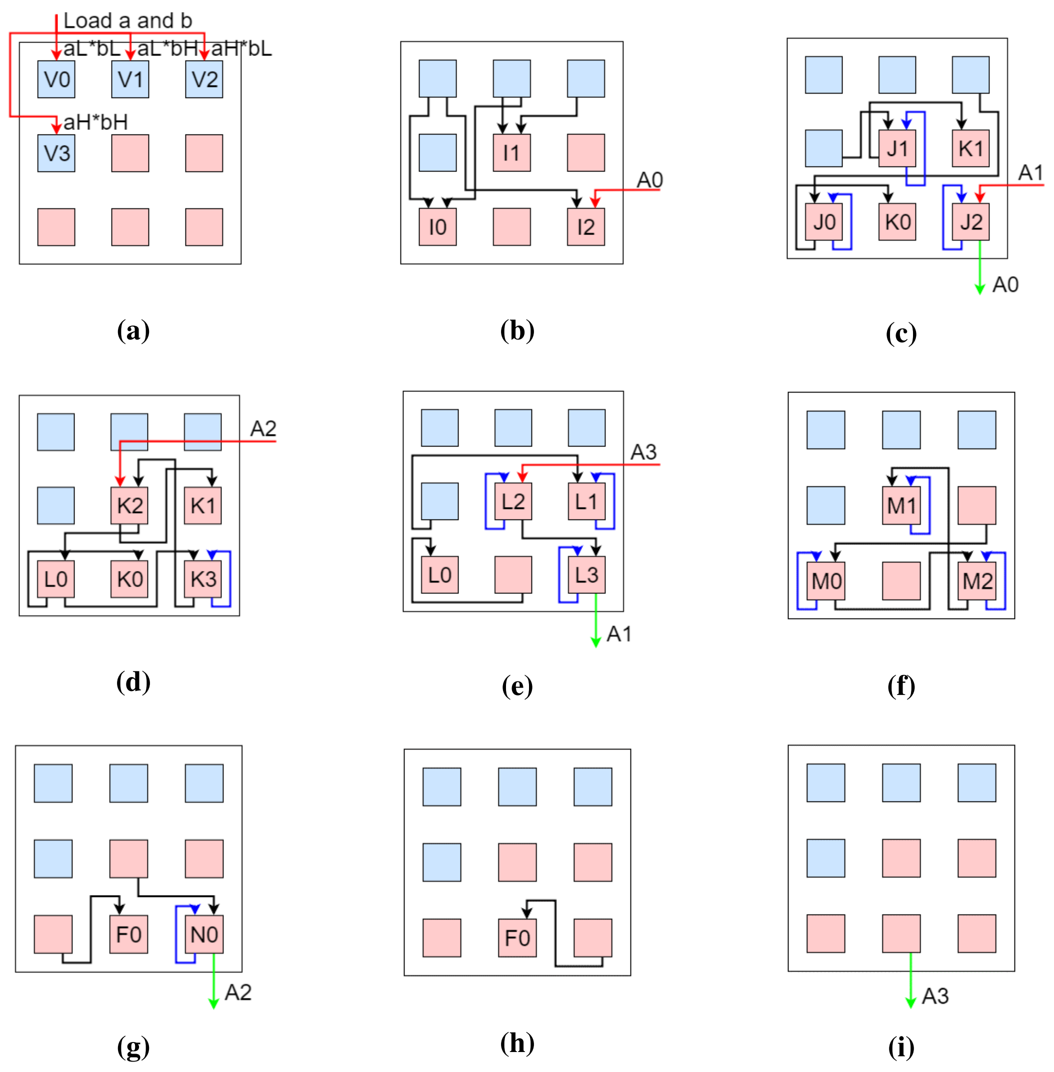


Figure 3.5: Nine Steps of the MAC operation using a 3x3 array of PIM cores.

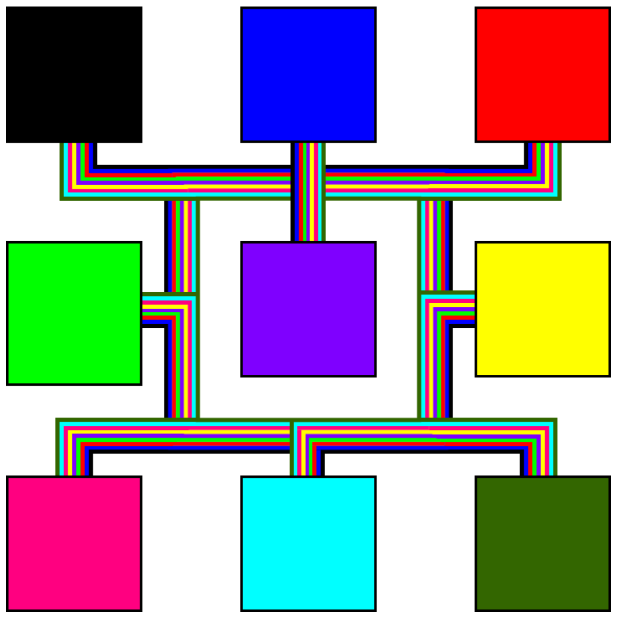


Figure 3.6: All-to-All Network connecting the nine PIM Cores within a PIM Cluster. Each colored wire represents a bidirectional communication path between two PIM Cores.

The All-to-All network is not scalable for larger sized networks due to the large number of connections needed. In this architecture, however, where the size of the network is limited to nine nodes, it is sufficient. If the number of PIM Cores contained in a cluster were to be increased, the network topology connecting them would likely need to be reevaluated and modified to support the larger number of nodes. For larger cluster sizes, a mesh network topology would be more beneficial due to the reduced number of connections needed, however this would require additional area, power, and logic to support routing data between the cores.

A key benefit of the All-to-All network is its flexibility to enable arbitrary communication between any two PIM Cores rather than relying on fixed paths. In this section, an 8-bit MAC operation was implemented using PIM Core logic, however other functions can be implemented by changing the *function-words* of each core and defining different routing behavior. By doing this, other large functions such as 32-bit addition or multiplication can be achieved using the PIM Cluster.

Communication between the PIM Cluster and outside sources will be handled through the use of routing logic located in the center of the cluster. This router will be integrated into a 2-D mesh NoC architecture to facilitate communication with other PIM Clusters and Memory Controllers (MCs), forming a two-tiered hierarchical mesh. This router can be implemented using traditional wired interconnects or using high performance wireless interconnects. Due to the 16-bit results generated by PIM Cluster operations, the communication can occur through single 32-bit flit transmissions using the same flit structure as the subNoC. This enables larger scale functions to be implemented using the PIM architecture, such as Matrix Multiplication.

3.4 Matrix Multiplication Using PIM Clusters

A scheme for performing matrix multiplication operations using the outlined PIM architecture is proposed. This PIM architecture is suited for computing matrix multiplication as PIM Clusters can compute an element in the result matrix through repeated MAC operations. By using a 2D array of PIM Clusters, each element of the result matrix can be calculated in parallel.

In order to perform the matrix multiplication operations, the relevant row and column data from the input matrices must be loaded into each cluster. Data inputs to the PIM clusters is handled through the use of multicasting, which allows data to be sent to multiple destinations within the 2D array of PIM Clusters. As a result, rows/columns of data from the input matrices can be sent to all clusters which require the data at the same time. Multicasting data for matrix multiplication of two matrices A and B to produce a 3x3 matrix C is exemplified in Figure 3.7.

After a PIM Cluster receives the input data, it can begin performing MAC operations. As shown in Equation 2.2, the number of MAC operations to perform in each cluster is equal to the size of the shared dimension of the two input matrices, P. Using the input data distribution method shown in Figure 3.7, the time when a PIM Cluster finishes all P MAC operations is dependent on its location within the 2D array of clusters because clusters

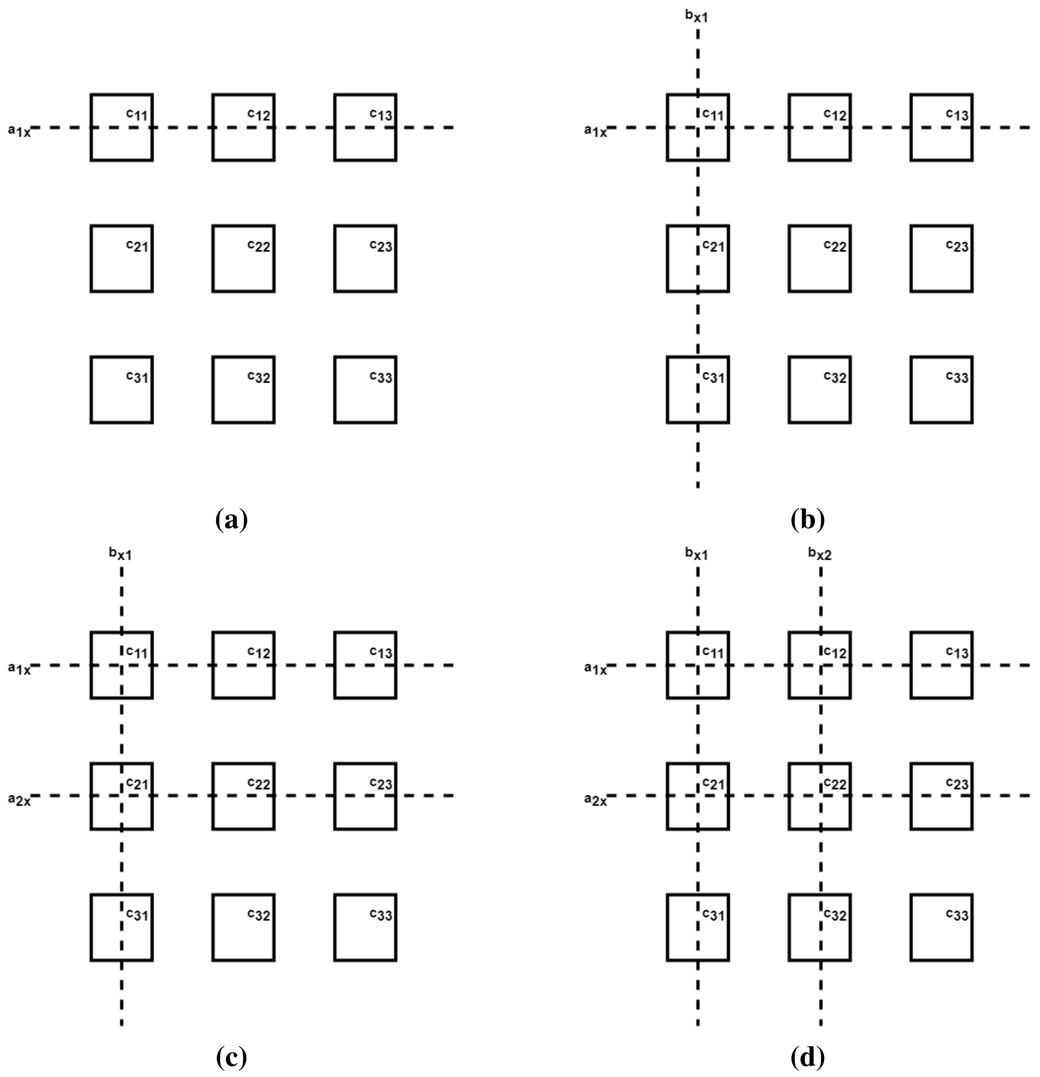


Figure 3.7: Multicasting scheme to transmit data for matrix multiplication. (a) First row of column A is sent to the first row of clusters. (b) First column of B is sent to the first column of clusters. (c) Second row of A is sent to the second row of clusters. (d) Second column of B is sent to the second column of clusters. The process is continued until all rows and columns from the input matrices are sent to the clusters.

which receive both sets of input data can begin computation while the rest of the inputs are being transmitted. After each cluster finishes its computations and produces a final result, it can transmit the result to a MC to be stored into memory.

Chapter 4

Analytical Modeling of PIM Cluster

4.1 PIM Cluster Area

Each PIM Cluster contains nine PIM Cores which are arranged as a 3x3 array. In this analysis, the cores are assumed to be located directly next to each other in a uniform grid such that no gaps exist between the cores. In addition, each cluster is assumed to be surrounded by a block of RAM which can be used to hold local *data-words* and *function-words*. A memory access port is assumed to be located on the outer boundary of the PIM cores. A diagram of the PIM Cluster is shown in Figure 4.1.

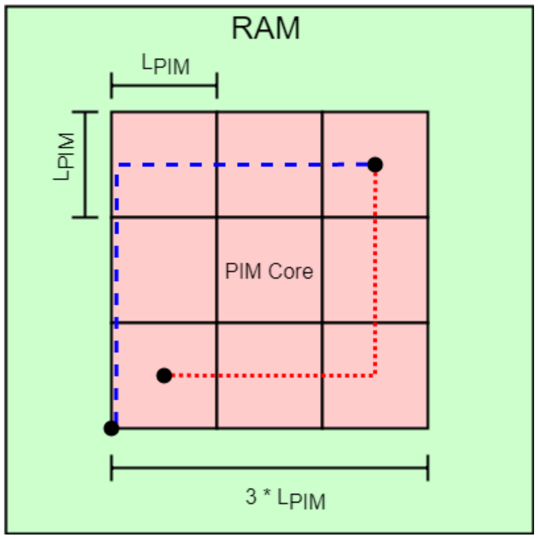


Figure 4.1: Block diagram of PIM Cluster. The worst case core-to-core communication path is shown in red. The worst case core-to-memory path is shown in blue.

Communication between the cores is handled through the subNoC, where each of the interconnects used to transmit data between the cores is assumed to originate at the center of the core. To ensure the validity of the design, it must be possible to retrieve data from any other core or the memory and perform a computation within a single clock cycle. The worst case core-to-core communication path occurs when data is sent across the diagonal of the 3x3 array, as shown by the red line in Figure 4.1. This Manhattan distance can be obtained in terms of the side length of a PIM Core, L_{PIM}. The data must travel through the lengths of three PIM Cores as well as two times the distance from the center of the core to a perpendicular edge. The total length is therefore equal to $4L_{PIM}$.

The total delay incurred by this distance can then be obtained using the Elmore Delay model shown in Equation 4.1 in conjunction with a known interconnect RC delay measurement for a given process node. The known interconnect RC delay is given in ps/mm, which can be expressed as a reference delay T_{ref} divided by a reference distance L_{ref}. By keeping the values of r and c constant and using the reference RC delay, the Elmore Delay model can then be rearranged to calculate the interconnect delay for a given interconnect length, T_{int} as shown in Equation 4.2.

$$D = 0.4rcL^2 \tag{4.1}$$

$$T_{int} = T_{ref} \times \left(\frac{L_{int}}{L_{ref}}\right)^2 \tag{4.2}$$

The same procedure is repeated to obtain the delay of the longest core-to-memory path, which is shown by the blue line in Figure 4.1. In this longest path, the data must travel through the length of four PIM Cores in addition to two times the distance from the center of the core to the perpendicular edge. The total distance is therefore equal to $5L_{PIM}$. The RC delay of this path can then be calculated using Equation 4.2.

4.2 PIM Cluster MAC Energy

The total energy required to perform a MAC operation using a PIM Cluster is equal to the total energy used by the PIM Cores and wired interconnects during each of the nine steps of the MAC operation. The total power of a single PIM Core, P_{core} can be obtained through static power analysis using circuit analysis tools. The energy used by the core, E_{core}, can then be obtained by multiplying the core power by the delay of the PIM Core. The delay of the PIM Core can also be obtained using circuit analysis tools. During operation, each of the nine PIM Cores is assumed to be fully powered on, therefore the total energy used by the cores is equal to $9E_{core}$.

The total power of the interconnects was calculated by measuring the distance traveled by data during each step of the MAC operation. The distance of each data communication was obtained by using the Manhattan distance between the centers of the source and destination PIM Cores. This distance, $L_{int_{ij}}$, was determined for every data transmission during each of the nine steps of the MAC operation, where i is the numbered step of the operation and j is the number of the transmission during step i. Using the capacitance per unit length for a given technology node, c_{int}, the transmission energy of sending the 32-bit packet can be obtained using Equation 4.3, where α is the activity factor and V is the supply voltage.

$$E_{int_{ij}} = 32 \times \alpha \times (c_{int} \times L_{int_{ij}}) \times V^2 \tag{4.3}$$

The total energy used by the interconnects during a MAC operation can then obtained through a summation of the energy used by each of the interconnects, as shown in Equation 4.4, where n is the number of data transmissions present during step i of the MAC operation.

$$E_{int\_Total} = \sum_{i=1}^{9} \sum_{j=1}^{n} E_{int_{ij}} \tag{4.4}$$

The total energy used during an entire MAC operation using the PIM Cluster can then be expressed as the sum of the energy used by nine PIM Cores for nine steps and the interconnect energy, as shown in Equation 4.5.

$$E_{MAC} = 9 \times 9E_{core} + E_{int\_Total} \tag{4.5}$$

The static power of the PIM Cores and interconnects are not considered in any of the energy models. Instead, the models are entirely based on the dynamic energy required to perform the operations. This allows the models to more closely reflect the energy used by the algorithms themselves rather than intrinsic device characteristics. In addition, the static and dynamic energy required by the memory is not included because it is dependent on the type of memory used and the size of the memory, which have not been defined in this architecture.

4.3 PIM Cluster MAC Timing

The time required to complete a MAC operation within a cluster, T_{MAC}, can be expressed as the sum of the core processing time, T_{PIM}, and the delays caused by the interconnects, as shown in Equation 4.6. The PIM Core processing time can be obtained using static timing analysis with circuit analysis software.

$$T_{MAC} = \sum_{i=1}^{9} T_{PIM} + T_{int_i} \tag{4.6}$$

The delay caused by the wired interconnects during a given step i, T_{int_i}, represents the longest delay caused by transmission lines during each of the nine steps of the MAC operation. The delay changes during each of the steps of the MAC operation due to the different data communication patterns present during each step. The interconnect delay of each step can be obtained by finding the longest data communication distance. The longest distance will have the largest RC delay and will therefore dictate the longest delay during each step. The RC delay of each step can be obtained using Equation 4.2.

Chapter 5

Timing Analysis Models

5.1 Matrix Multiplication Using Wired Interconnects

Matrix multiplication is achieved by performing repeated MAC operations in each of the PIM Clusters. Each PIM Cluster computes a final element of the resulting matrix in parallel. This operation can be broken into three primary steps: sending input matrix data to the PIM Clusters, performing MAC operations using the data, and transmitting the final result to a Memory Controller (MC) to store it into memory. The time to send the input data and receive the final results are dependent on the interconnect architecture used to connect the PIM Clusters and the MC while the time to perform the MAC operations is invariant. In this section, the timing and power of completing a matrix multiplication operation using an array of PIM Clusters connected with wired interconnects is proposed.

The PIM Clusters are assumed to be arranged as a 2D grid and connected using a 2-D mesh network. The clusters are assumed to be laid out such that distance incurred by hops between has an RC delay of 1ns. In addition, the routers used to control the flow of data communications is assumed to cause a delay of 1ns. The combination of these two delays is denoted as T_{hop}. The MCs are assumed to be located along one edge of the array of PIM Clusters and are connected to the mesh network through an adjacent PIM Cluster. This link between the MCs and the PIM Clusters also has an RC delay of T_{hop}. This configuration is exemplified in Figure 5.1. The proposed model assumes a variable number of MCs can be present and are evenly distributed among the columns of the 2-D mesh network. Each

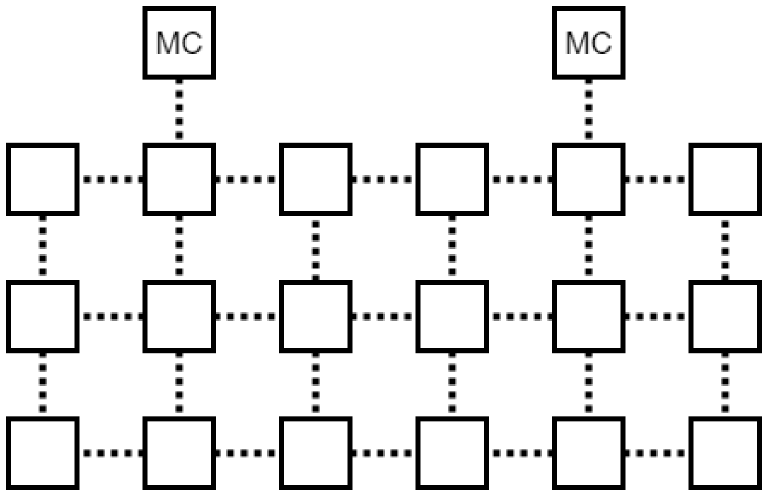


Figure 5.1: Example configuration of PIM Cluster array and 2-D mesh network. Two Memory Controllers are evenly distributed over the six columns of the array. The dashed lines represent the interconnects of the mesh network.

MC will be responsible for distributing data to the columns closest to it, which allows for increased parallelism. The number of columns controlled by each MC is assumed to be an equal portion of the maximum number of clusters. Cases with 1, 4, and 8 MCs are examined.

To perform timing analysis using the circuit setup outlined in Figure 5.1, a set of variables describing the setup of the PIM Cluster array was used. The array of clusters is assumed to have M rows and N columns. The size of the shared dimension between the two input matrices, which determines the number of MAC operations to perform, is P. The following variables are then used for analysis, where n_{mc} is the number of memory controllers in the system:

1. $c_{total} = \left\lceil \frac{N}{n_{mc}} \right\rceil$
2. $c_{mc} = \left\lceil \frac{N}{2 \times n_{mc}} \right\rceil$
3. $c_{left} = c_{mc} - 1$
4. $c_{right} = c_{total} - c_{mc}$

The variable c_{total} is the largest number of columns which is assigned to any MC in the system. For configurations in which the memory controllers cannot be evenly distributed, some MCs will be tasked with providing data to more clusters than others. The next variable, c_{mc}, represents the column index of the MC within its assigned columns. The value of c_{mc} is 1-indexed such that a value of '1' depicts that the memory controller is located in the leftmost column. Finally, the c_{left} and c_{right} variables denote the number of columns to the left and right of the MC, respectively. In unequally distributed systems, the number of columns on the left and right will be different. Using these variables, the timing and power of the system can be quantified.

5.1.1 Sending Input Data

Data from the input matrices can be sent to the clusters in a row-wise or column-wise fashion, which will be referred to as *row-casting* and *column-casting*, respectively. These transmission schemes allow an entire row/column of PIM Clusters to obtain the same input data while minimizing the total data communication time. The scheme used to perform row-casting is shown in Figure 5.2.

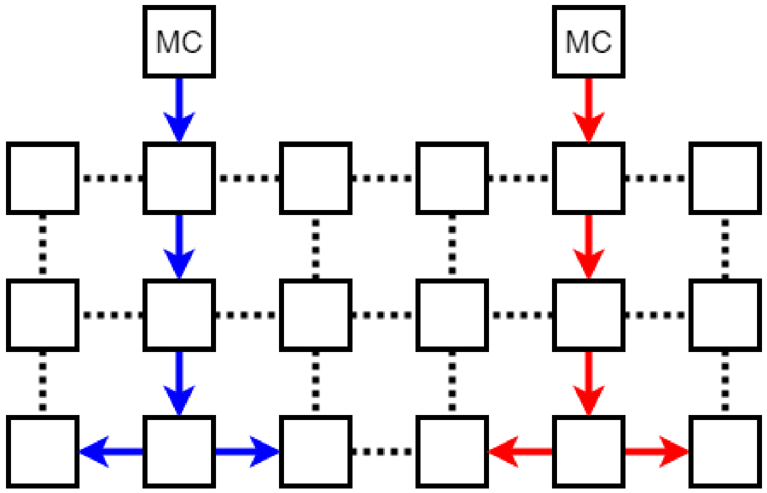


Figure 5.2: Example of sending data to PIM Clusters located in row 3 of the 2-D array. The two MCs work in conjunction to cast data across the row.

Using the scheme outlined in Figure 5.2, a formula for the number of hops required to row-cast to a given row, i, was derived. To reach the given row, i hops are needed to travel down the column containing the MC. Then, the data branches and transmits to the left and right columns simultaneously. The number of hops required to fully branch to the left and right columns is determined by the maximum of c_{left} and c_{right}. When performing matrix multiplication, an entire row/column of data with P elements is needed, therefore a total of P elements are required to be transmitted to all clusters in the row. These values can be transmitted in a lock-step fashion such that the next value will be one hop behind the current transmission. The total time to row-cast to a given row i is the sum of these times, as shown in Equation 5.1. Since the row-casting done under each MC is done in parallel, the total execution time is equal to the worst case of any given grouping of columns under an MC.

$$T_{row}(i) = (i + max(c_{left}, c_{right}) + (P - 1)) \times T_{hop} \tag{5.1}$$

During the course of a matrix multiplication, data is cast to every row in the PIM Cluster array. The total time to send all rows of data can be expressed as the sum to row-cast the data to every row of clusters, as shown in Equation 5.2.

$$T_{row\_total} = \sum_{i=1}^{M} T_{row}(i) \tag{5.2}$$

Column-casting can be used to send the data to all clusters in the same column of the PIM array. The scheme used to perform column-casting is shown in Figure 5.3. Data is first directed to the required column, then the data is propagated down the column such that it reaches every cluster in the column. The number of hops required to reach a given column j is equal to the number of columns between j and the closest MC. Similar to row-casting, P elements are required to be transmitted in lock-step fashion after the first data element. If the value of j is assumed to be a local index between 1 and c_{total}, then the time required to column-cast to a given column j can be expressed as shown in Equation 5.3.

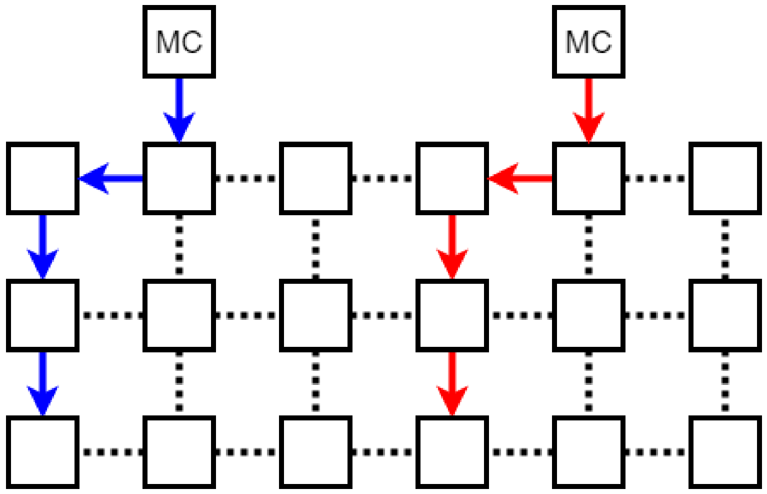


Figure 5.3: Example of sending data to PIM Clusters located in columns 1 and 4 of the 2-D array. The two MCs work in conjunction to cast two columns of data simultaneously.

$$T_{col}(j) = (|c_{mc} - j| + M + (P - 1)) \times T_{hop} \tag{5.3}$$

During a matrix multiplication operation, values will be column-cast to every column of the PIM cluster array. Each MC can perform column-cast operations on its assigned columns in parallel, therefore the total time to column-cast all the data is limited by the grouping with the most columns. The total time to column-cast the data can be separated into two primary stages; time to hop to the correct column and time to transmit the data down the correct column. The total number of hops required to reach every column across all required column-casts, n_{hops} is expressed in Equation 5.4.

$$n_{hops} = \sum_{k=1}^{c_{left}} k + \sum_{k=1}^{c_{right}} k \tag{5.4}$$

The number of hops required to transmit data down a column is equal to the number of rows in the column. Since each column within the MC grouping requires a column-cast to obtain data, this number of hops must be repeated c_{total} times. This, combined with the number of lateral hops calculated using 5.4, results in a total column casting time as shown in Equation 5.5.

$$T_{col\_total} = c_{total} \times (M + (P - 1)) \times T_{hop} + n_{hops} \times T_{hop} \tag{5.5}$$

5.1.2 PIM Computation

The number of MAC operations to perform on the input data is given by the size of the shared dimension between the two input matrices, P. Each cluster in the PIM Cluster array is required to perform P MAC operations to produce a final element in the result matrix. Each cluster's computations can be performed in parallel due to data independence. As a result, the total time required to perform all computations, $T_{compute}$, is equal to the time to perform P MAC operations, as expressed in Equation 5.6.

$$T_{compute} = P \times T_{MAC} \tag{5.6}$$

5.1.3 Data Retrieval

After each cluster completes its computations, it produces a single value in the result matrix which it must transmit to the memory controller to be stored into memory. This can be performed in a lock-step fashion where results are funneled to the memory controller as adjacent clusters send their results. The result of this is a result reaching the MC after every hop. Within each MC's grouping exist at most M rows and c_{total} columns of PIM Clusters. Therefore the maximum number of results to transmit is the equal to the product of M and c_{total}. Using this, the maximum total time to retrieve the data, $T_{R_{max}}$ is given in Equation 5.7.

$$T_{R_{max}} = M \times c_{total} \times T_{hop} \tag{5.7}$$

Due to the included No-Op logic where results with a value of zero are not transmitted, however, the time to acquire the results is a function of the sparsity of the result matrix,

β. The value of β expresses the number of non-zero elements in the final result matrix and their proximity to the MC. No formal function for β is proposed, as it is outside the scope of this investigation. In a worst case fully dense matrix with all elements of non-zero value, β will have a maximum value of '1'. A best case fully sparse matrix with all zero results will have a β value of zero. Sparsity conditions between the best and worst cases will have a value such that $0 < \beta < 1$. The total time to retrieve the data from the clusters with sparsity is expressed in Equation 5.8.

$$T_{R_{sparse}}(\beta) = \beta \times T_{R_{max}} \tag{5.8}$$

5.1.4 Final Timing Model

Due to the parallel nature of the cluster operations, asymmetric data distribution scheme, and fast computation time, the computation time of the MAC operations can be masked by the data transmission times. The first PIM Cluster can begin computations after the first row and column are multicast. During this computation time, input data will continue to be transmitted to the clusters. This trend continues until all input data has been transmitted. Once all input data has been transmitted, the results from the finished PIM Clusters can begin being sent to the MCs. After all results are retrieved from the PIM Clusters, the matrix multiplication operation is completed. A timing diagram outlining the steps of the matrix multiplication operation is shown in Figure 5.4.

An equation for the total execution time is given in Equation 5.9, which accounts for the masked execution time.

$$T_{wired\_mult} = T_{row\_total} + T_{col\_total} + max(T_{compute}, T_{R_{sparse}}(\beta)) \tag{5.9}$$

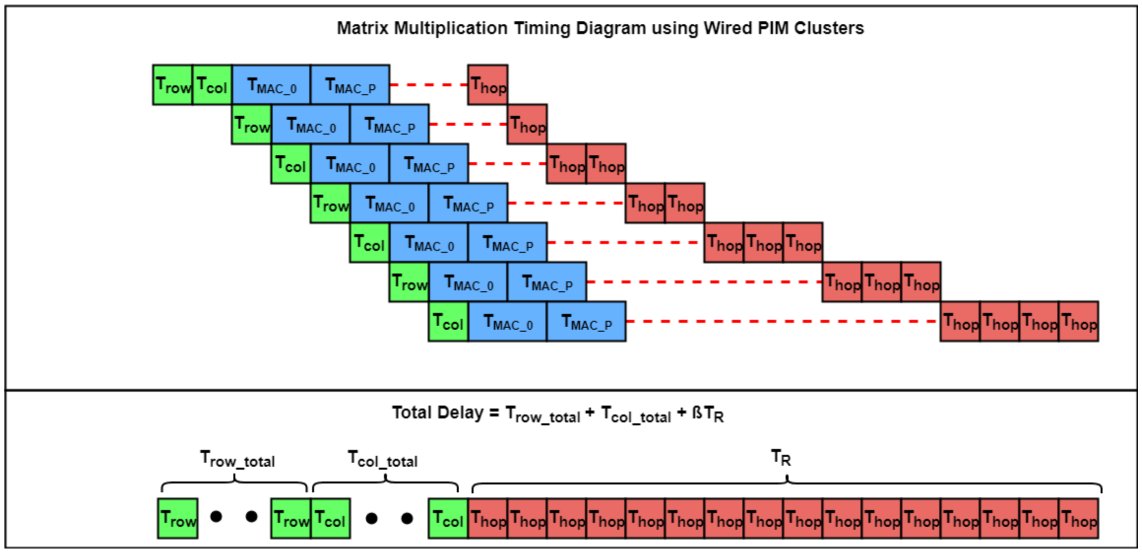

Matrix Multiplication Timing Diagram using Wired PIM Clusters
T_{row}
T_{col}
T_{MAC_0}
T_{MAC_P}
T_{hop}
Total Delay = $T_{row\_total} + T_{col\_total} + ßT_R$
T_{row_total}
T_{col_total}
T_R

Figure 5.4: Timing diagram of a matrix multiplication operation using wired mesh network.

5.2 Matrix Multiplication Using Wireless Interconnects

The proposed PIM architecture can also be implemented using a wireless interconnection fabric. In this setup, the wired switch and router present in each cluster are replaced with a wireless transceiver operating at 60GHz, based on the architecture proposed in [23] and [24]. The throughput of the wireless network is given to be 16Gb/s. The wireless transceiver is also used to facilitate communication with the memory controller. The use of wireless interconnects allows a minimal transmission delay and simultaneous multicasting and broadcasting to any clusters in the network. As a result, row-casting and column-casting can be achieved without the need to transmit data using multiple hops. Similar to the wired interconnect model, the total execution time of a matrix multiplication operation can be broken into three primary stages: transmission of input data, computations using PIM Clusters, and receiving final results. The computation time of the PIM clusters is identical to the wired interconnect model described in Equation 5.6.

5.2.1 Sending Input Data

Similar to the wired interconnect model, rows and columns of input data can be multicast to multiple clusters in the PIM array. Unlike the wired, model, however, one MC is responsible for wirelessly sending the data, which eliminates the parallelism gained by using multiple MCs. The loss of parallelism is made up for by the rapid multicasting which is possible using wireless interconnects. Input data can be sent into the array of PIM Clusters using the same scheme as outlined in Figure 3.7. When a row-cast or column-cast is made, all clusters within the destination row/column receive the data at the same time. The time to perform a row-cast or column-cast is therefore equivalent to the time required to transmit a series of 32-bit flits containing the P elements of the input row/column using the wireless interconnects. The equation for calculating the time required to multicast P elements using wireless interconnects, $T_{M/C}$, is shown in Equation 5.10.

$$T_{M/C} = P \times \frac{32bits}{16Gb/s} \tag{5.10}$$

To perform a matrix multiplication operation, each of the M rows and N columns of input data must be sent to the clusters. Each of these transmissions requires a multicast to send data to a given row/cluster, which requires $T_{M/C}$ time to perform. Using this, the equation for the total time required to send input data, T_{in}, is shown in Equation 5.11.

$$T_{in} = (M + N) \times T_{M/C} \tag{5.11}$$

5.2.2 Data Retrieval

Once a PIM Cluster has completed all of its MAC operations, it can transmit its final result to the MC using the wireless transceiver. The cluster must wait for the wireless medium to be available first, however, meaning that sending results cannot occur at the same time that input data is being transmitted. Once the medium is free, the results can be transmitted one at a time to the MC using 32-bit flits. In a fully dense matrix with all non-zero results, each of the $M \times N$ elements must be sent to the MC, one at a time. In a sparse matrix with S non-zero elements, only S results must be sent to the MC where $0 <= S <= M \times N$. Therefore the time to send all result data to the MC can be expressed as shown in Equation 5.12.

$$T_{results}(S) = S \times \frac{32bits}{16Gb/s} \tag{5.12}$$

5.2.3 Final Timing Model

The total time to complete a matrix multiplication operation using wireless interconnects is equal to the sum of the time to send data inputs, perform computations, and gather the final results. As in the wired interconnect model presented in Section 5.1.4, the computation time of the PIM Clusters can be masked by the data transmissions. This is exemplified in the timing diagram shown in Equation 5.5, which shows the timing of a fully dense matrix.

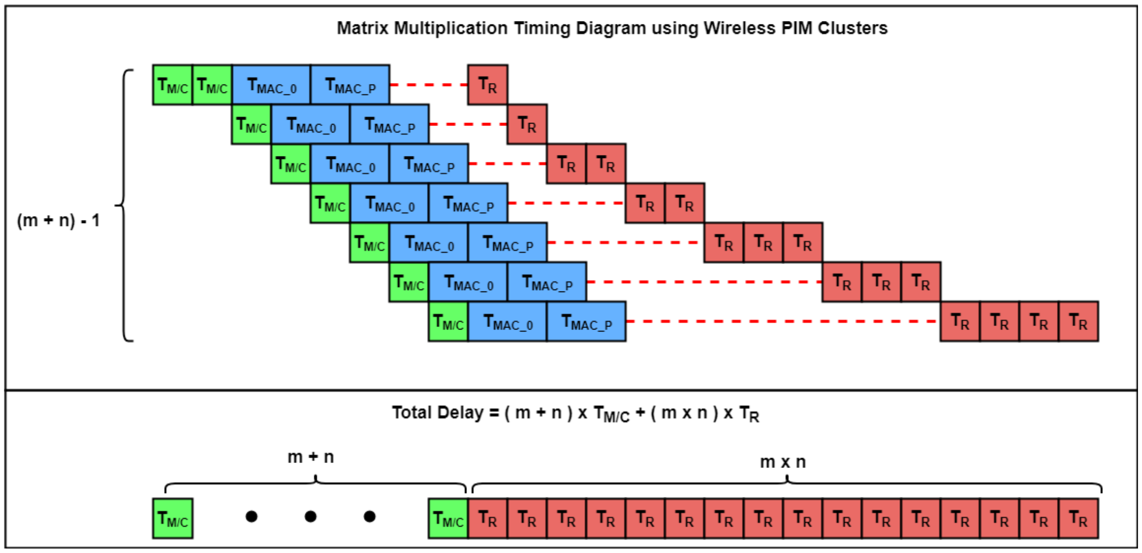

Matrix Multiplication Timing Diagram using Wireless PIM Clusters
(m + n) - 1
$T_{M/C}$
T_{MAC_0}
T_{MAC_P}
T_R
Total Delay = (m + n) x $T_{M/C}$ + (m x n) x T_R
m + n
m x n

Figure 5.5: Timing diagram of a matrix multiplication operation using wireless interconnects.

For sparse matrices, the computation time will be masked only if the time to send the non-zero elements is greater than the computation time of a single cluster. This is because the final PIM Cluster computation must finish before the matrix multiplication operation can complete. The final timing equation for the wireless interconnect model is given in Equation 5.13.

$$T_{wireless\_mult} = T_{in} + max(T_{compute}, T_{results}) \tag{5.13}$$

Chapter 6

Energy Analysis Models

Like the timing model, the energy used to perform a matrix multiplication operation can be divided into three primary sections: sending input data to the clusters, performing the MAC operations within each cluster, and sending the results to a memory controller.

6.1 Matrix Multiplication Using Wired Interconnects

In the wired interconnect model, each of the PIM Clusters and MCs are connected in a 2-D mesh network using switches and routers in each node. The distance between each node is assumed to take 1ns to traverse. Using this delay and a modified version of Equation 4.2, the physical distance can be estimated as shown in Equation 6.1. Using the estimated distance and known parasitic capacitance per unit length for a given technology node, c_{int}, the energy to traverse the distance, E_{hop_int}, can be obtained using Equation 4.3. Each router was assumed to use some amount of energy, E_{router}, during its use. The total energy per hop, E_{hop}, is therefore the sum of E_{hop_int} and E_{router}.

$$L_{hop} = L_{ref} \times \sqrt{\frac{1ns}{T_{ref}}} \tag{6.1}$$

6.1.1 Sending Input Data

Row-casting and column-casting are used to distribute data from the MCs to the desired row or column in the array of PIM Clusters. The total energy used during each of these operations is equal to the number of hops required for all the data to reach its final destinations multiplied by the energy per hop. To send P elements of data to row i of the array using the scheme outlined in Figure 5.2, the data must travel i hops to reach the correct row. Then, $c_{total} - 1$ hops are required to propagate the data throughout the row. This process is repeated for each of the P data packets sent. The power used to row-cast to a given row i is expressed in Equation 6.2.

$$E_{row}(i) = (c_{total} + i - 1) \times P \times E_{hop} \tag{6.2}$$

During a column-cast, data must travel laterally from the column containing the MC to the correct column, then travel down the M rows of the column. This data flow is repeated for each of the P packets to send. The energy to perform a column cast to a given column j can therefore be expressed as shown in Equation 6.3.

$$E_{col}(j) = (|c_{mc} - j| + M) \times P \times E_{hop} \tag{6.3}$$

The total amount of energy used to transmit all input data is equal to the energy required to perform all row-cast and column-cast operations. In this process, M row-casts and c_{total} column-casts are required for each MC. The total energy to send all input data is expressed in Equation 6.4.

$$E_{input} = (\sum_{i=1}^{M} E_{row}(i) + \sum_{j=1}^{c_{total}} E_{col}(j)) \times n_{mc} \tag{6.4}$$

6.1.2 PIM Computation

Each PIM Cluster in the MxN array must perform P MAC operations during its computations. Each of these MAC operations requires E_{MAC} energy to perform as derived in Equation 4.5, which includes the energy of the PIM Cores and the subNoC interconnects. The total energy used during the PIM Clusters during computations is expressed in Equation 6.5.

$$E_{compute} = M \times N \times P \times E_{MAC} \tag{6.5}$$

6.1.3 Data Retrieval

Data retrieval from the PIM Clusters in the wired mesh network functions such that the results are funneled one at a time to the MC. The flow of data from the PIM Clusters to the MC follows a tree-like structure as shown in Figure 6.1.

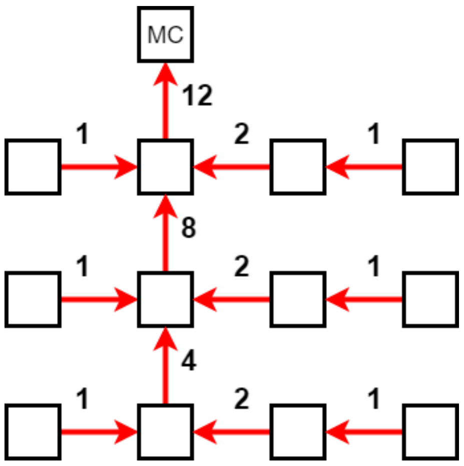


Figure 6.1: Example of retrieving results from a 3x4 array of PIM Clusters using wired interconnects. Data is passed to the MC in a lock-step fashion. The numbers next to each interconnect represent the number of packets sent through the link during the data retrieval process.

The energy used during the data retrieval process can be calculated by counting the total number of hops performed by each link to move all data to the MC. The number of times each vertical link in the column containing the MC is used is equal to the number of PIM

Clusters in the rows below it. As shown in Figure 6.1, the vertical link connecting row 3 to row 2 is used 4 times because the 4 elements contained in row 4 must send their data through the link. Similarly, the vertical link connecting row 2 and row 1 is used 8 times because all the data contained in rows 2 and 3 must pass through the link. The total number of times the horizontal links are used in each row is equal to n_{hops}. In a dense matrix, the total energy used can be obtained by multiplying the total number of hops by the energy required per hop. For sparse matrices, however, where the number of non-zero results is less than the number of PIM Clusters, the total energy usage depends on the number of results and their location in the array. This sparsity characteristic is described through the parameter β. The total energy usage required to retrieve data from the array of PIM Clusters is shown in Equation 6.6.

$$E_{results}(\beta) = \beta \times (\sum_{i=1}^{M}(c_{total} \times i + n_{hops})) \times E_{hop} \tag{6.6}$$

6.1.4 Total energy

The total energy required to complete a matrix operation using the array of PIM Clusters with wired interconnects is equal to the sum of the energy used to send the input data, perform the MAC computations, and send the final results to the MCs. Of these values, only the energy used to send final results is affected by the sparsity of the matrix. The total energy is therefore expressed as shown in Equation 6.7.

$$E_{wired\_mult} = E_{input} + E_{compute} + E_{results}(\beta) \tag{6.7}$$

6.2 Matrix Multiplication Using Wireless Interconnects

6.2.1 Sending Input Data

Input data is sent to the array of PIM Clusters through the use of wireless multicasting. Each multicast can be used to send P values to an entire row/column of the PIM array at the same time using 32-bit packets. The energy required to perform this operation is equal to the energy per bit required to send and receive data using the wireless interconnects, $E_{wireless}$, multiplied by the number of bits sent. Each of the M rows and N columns of input data must be multicast to the clusters, therefore the total energy used to transmit the input data is given by Equation 6.8.

$$E_{input} = (M + N) \times (32 \times P \times E_{wireless}) \tag{6.8}$$

6.2.2 PIM Computation

The amount of energy required to perform the MAC computations is invariant of the interconnection architecture used to link the PIM Clusters. As a result, the energy is equivalent to the model given in Equation 6.5.

6.2.3 Data Retrieval

After each PIM Cluster is finished with its computations and the wireless medium is open, it is able to send a 32-bit packet of data containing its final result to the MC. In a dense matrix, every cluster in the MxN array must send its results to the MC, one at a time. In a sparse matrix with S non-zero elements, only S results must be sent to the MC where $0 <= S <= M \times N$. Therefore the energy to send all result data to the MC can be expressed as shown in Equation 6.9.

$$E_{result}(S) = S \times 32 \times E_{wireless} \tag{6.9}$$

6.2.4 Total Energy

Similar to the wired interconnect model, the total energy used during a matrix multiplication operation is equal to the sum of the energies used to transmit the input values, perform MAC operations, and send the final results to the MC, as expressed in Equation 6.10.

$$E_{wireless\_mult} = E_{input} + E_{compute} + E_{result}(S) \tag{6.10}$$

Chapter 7

Execution of Large Datasets

The previously discussed timing and energy models assume that there exists a dedicated PIM Cluster to compute the results for every element in the MxN result matrix. For larger matrix sizes, this approach is not feasible due to the limited area available on the silicon die for logic. As a result, a new set of models is required to perform matrix multiplication operations using large matrix sizes with a limited number of PIM Clusters. In this analysis, a 2D array of PIM Clusters with X rows and Y columns is proposed. The exact size of the cluster array will be determined by the size of the final PIM Cluster architecture and the available die area.

Large size matrices can be multiplied using the proposed PIM architecture by dividing the matrix into blocks of size XxY and performing the matrix multiplication on each of the blocks independently. This segmentation of the result matrix is shown in Figure 7.1, where we assume a 5x5 result matrix and a 2x2 array of PIM clusters.

The result matrix is first divided into blocks equal to the size of the PIM Cluster array. The number of rows and columns of blocks is calculated using Equation 7.1 and Equation 7.2, respectively. In cases where a complete block that fills the entire PIM array cannot be allocated, an incomplete block is constructed and executed as shown in Figure 7.1 (c), (f), (g), (h), and (i).

$$n_{row\_blocks} = \left\lceil \frac{N}{Y} \right\rceil \tag{7.1}$$

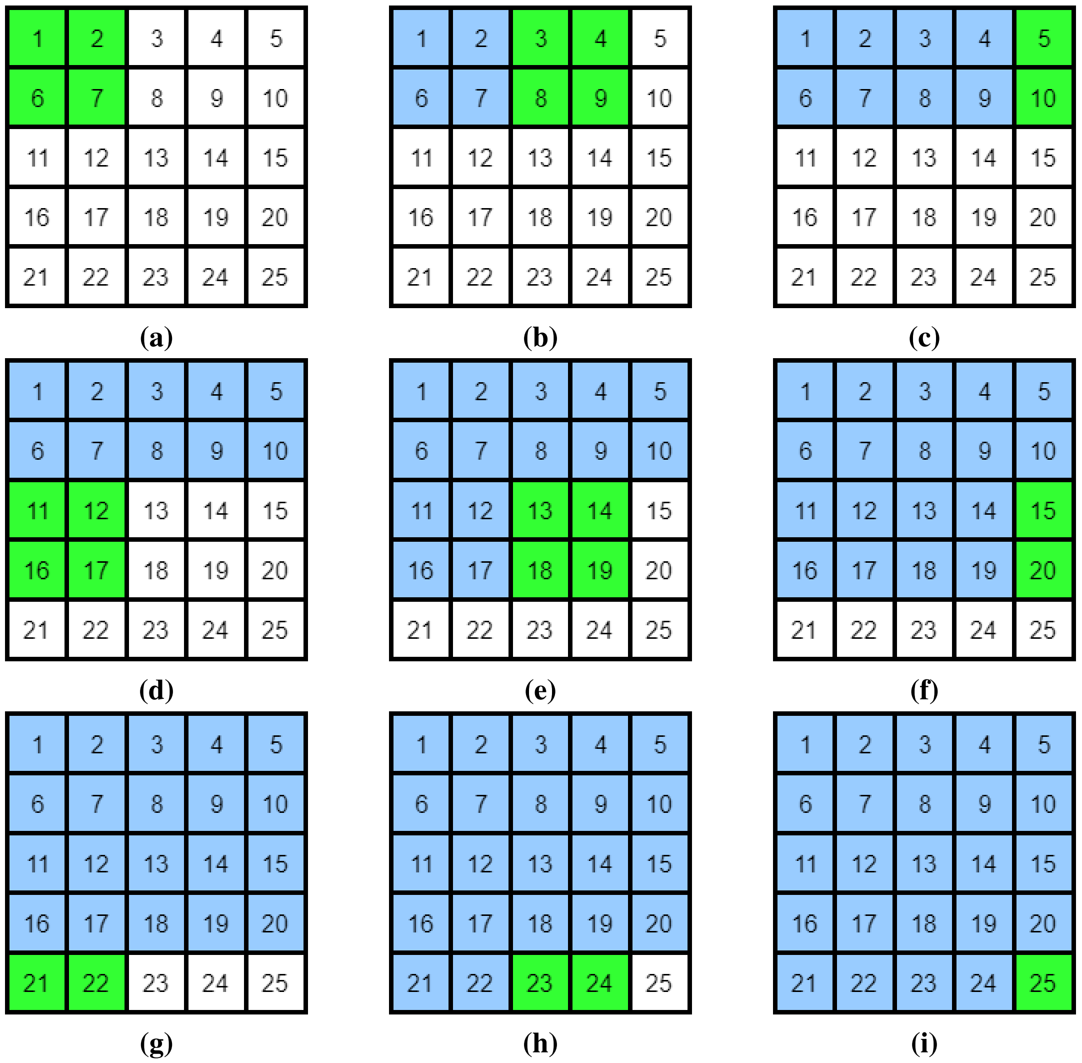


Figure 7.1: Diagram outlining execution order of folded matrix multiplication model. Result matrix is of size 5x5 and PIM Cluster array is 2x2. During each step (a)-(g), green squares represent the block to perform matrix multiplication on using the cluster array while blue squares show finished elements.

$$n_{col\_blocks} = \left\lceil \frac{M}{X} \right\rceil \tag{7.2}$$

Matrix multiplication operations can then begin on each block, going in a row-wise order. The first block of each row will behave as described in Section 8.2. The row data used by each block in the row will be the same, therefore only new column data must be sent to the clusters to execute a new block of matrix multiplications. Any incomplete blocks which may exist at the end of the rows will be treated in the same manner as the complete blocks. This process is repeated for each row of blocks until all elements of the result matrix have been calculated.

7.1 Timing Analysis

The first block in each row will require both row and column input data to be sent to the array, therefore the total execution time of the block is equal to the normal execution time of a matrix multiplication as described in Section 8.2. Each subsequent block in this row will reuse the same row data inputs, therefore only new column data must be sent to the clusters. Using the wired architecture, this can be modeled using Equation 7.3, where we assume $M = X$ and $N = Y$ when performing the timing equations.

$$T_{wired} = \sum_{i=1}^{n_{col\_blocks}} \left[T_{wired\_mult} + \sum_{j=1}^{n_{row\_blocks}-1} \left(T_{wired\_mult} - T_{row\_total} \right) \right] \tag{7.3}$$

The wireless model operates in the same manner as the wired. Blocks are processed in a row-wise order and after the first block in a row, row data can be reused for the following block calculations. Therefore the execution time of the wired model can be expressed as shown in Equation 7.4.

$$T_{wireless} = \sum_{i=1}^{n_{col\_blocks}} \Big[T_{wireless\_mult} + \sum_{j=1}^{n_{row\_blocks-1}} (T_{wireless\_mult} - M \times T_{M/C}) \Big] \tag{7.4}$$

7.2 Energy Analysis

The total energy required to perform the segmented matrix multiplication model is equal to the sum of the energy required to compute each of the block multiplications. This can be obtained using the same scheme as the timing model. The power required to transmit row data from to each block after the first in a row can be eliminated. This is reflected in the wired energy model in Equation 7.5 and the wireless model in Equation 7.6.

$$E_{wired} = \sum_{i=1}^{n_{col\_blocks}} \Big[E_{wired\_mult} + \sum_{j=1}^{n_{row\_blocks-1}} (E_{wired\_mult} - \sum_{k=1}^{M} E_{row}(k)) \Big] \tag{7.5}$$

$$E_{wireless} = \sum_{i=1}^{n_{col\_blocks}} \Big[E_{wireless\_mult} + \sum_{j=1}^{n_{row\_blocks-1}} (E_{wireless\_mult} - M \times (32 \times P \times E_{wireless})) \Big] \tag{7.6}$$

Chapter 8

Results

8.1 Cluster Characteristics

The Proposed PIM Core was synthesized and analyzed using Synopsys Design Compiler. The design was synthesized using TSMC's 65nm low power standard cell library. Synopsys Design Compiler, PrimeTime, and PrimeRail were used to calculate the PIM Core's silicon area, critical path delay, and power usage. The results are shown in Table 8.1. The synthesized core design includes the multiplexers used for logic and 256 8-bit registers to hold function-words.

	Speed (ns)	**Dynamic Power (μW)**	**Static Power (μW)**	**Area (μm^2)**
PIM Core	0.66	751.9282	4.9686	14351.58
SRAM [25]	0.3	0.45/cell	0.3/cell	0.5915/cell
Embedded DRAM [25]	0.7	0.10/cell	1.00E-5/cell	0.0554/cell

Table 8.1: Characteristics of PIM components in 65nm node

The characteristics of both SRAM and embedded DRAM in the 68nm process node are included in Table 8.1 due to their required inclusion in the overall PIM architecture. The decision to use SRAM vs DRAM in the final design has not been made, as each device has their own advantages and disadvantages. SRAM is significantly faster, however the area required per cell is substantially higher and it requires additional power. This would limit the total amount of memory which can be stored on the die. Using DRAM in the design would greatly increase the potential memory capacity, however the peripheral circuitry required to access and manage the memory is substantially larger than SRAM. This would limit the number of memory controllers (MCs) which could be integrated into the design.

8.1.1 Area Analysis

Using the Synopsys tools, an area of 14351.58µm^2 was obtained. To simplify analysis, the layout of the PIM Core is assumed to be a uniform square. The side length of the PIM Core, L_{PIM}, is there therefore equal to the square root of the total area, or 119.798µm. In the 65nm node, the interconnect RC delay of intermediate wires is equal to 741ps/mm as reported by [4], which accounts for width-dependent scattering in the wire. The RC delay of the worst case core-to-core communication path was then calculated to be 0.1702ns using Equation 4.2 where T_{int} was equal to $4L_{PIM}$. The worst case core-to-memory delay can be obtained using the same method, where T_{int} was equal to $5L_{PIM}$. The worst case core-to-memory delay was calculated to be 0.2659ns.

The PIM Cluster is proposed to operate at a 1GHz frequency. As a result, it must be possible to complete a data transmission and perform a computation within a single clock period of 1ns. From the Synopsys tools, a computation time of 0.66ns was obtained, as shown in Table 8.1, leaving 0.34ns for communication time. Both the worst case core-to-core and core-to-memory delays are less than the allotted communication time, therefore operation at 1GHz is feasible.

The number of PIM Clusters available to perform computations is limited by the potential size of a silicon die. Given the calculated PIM Core length of 119.798µm and each PIM Cluster containing three cores per side, the minimum length of a PIM Cluster would be 359.39µm. On a silicon die with a size 20mm by 20mm, a 55x55 grid of PIM Clusters could be placed. To account for additional area required for routing and additional logic, we assume a maximum PIM array size of 40x40.

8.1.2 MAC Timing

Given a computation time of 0.66ns and the known core-to-core distances, the total time to complete a MAC instruction in a PIM Cluster can be obtained. Each of the nine steps of the MAC operation shown in Figure 3.5 were analyzed to determine the worst case path, which in term creates the longest delay. Each of these distances were then used in Equation 4.2 to get the interconnect delay of each stage, T_{int_i}. This was then used with Equation 4.6 to obtain a final MAC operation time of 10.7ns.

8.1.3 MAC Energy

The power used by a PIM Core was obtained using the Synopsys tools and was found to be 751.9282µW. The energy of the PIM Core can then be obtained by multiplying this power by the total computation time of 0.66ns, resulting in 0.49627pJ of energy used per computation. The distance of every data communication was then used to calculate the total interconnect energy required using Equation 4.4, where the interconnect capacitance, c_{int} was 0.18fF/µm [4]. The total interconnect energy used during a MAC operation was found to be 42.402pJ. This can then be used in conjunction with Equation 4.5 to calculate the total energy required to perform a MAC operation, which was found to be 82.6pJ.

8.2 Matrix Multiplication Timing

Using the calculated MAC execution time, the time to complete a matrix multiplication operation can be obtained for both the wired and wireless interconnect architectures using Equations 5.9 and 5.13, respectively. Each architecture's total timing results were calculated for result matrices of size 1x1 up to 40x40. For the wired architecture, the multiplication time was calculated using 1, 4, and 8 memory controllers. The results are shown in Table 8.2, which includes a comparison to the matrix multiplication performed using a CPU and GPU. The results are also displayed graphically in Figure 8.1. These results assume a dense matrix result with no non-zero elements.

Table 8.2: Matrix multiplication time

Matrix Size	CPU Time (ms)	GPU Time (ms)	Wired Time (1 MC) (ms)	Wired Time (4 MC) (ms)	Wired Time (8 MC) (ms)	Wireless Time (ms)
1x1	0.0001	0.0756	0.000006	0.000006	0.000006	0.000006
2x2	0.0001	0.0769	0.000022	0.000036	0.000020	0.000020
3x3	0.0002	0.0752	0.00005	0.000082	0.000040	0.000040
4x4	0.0003	0.0834	0.000089	0.000156	0.000066	0.000066
5x5	0.0005	0.0881	0.000140	0.000242	0.000138	0.000098
10x10	0.0032	0.0869	0.000559	0.00102	0.000488	0.000428
15x15	0.0104	0.0890	0.001257	0.002302	0.00108	0.000868
20x20	0.024	0.1022	0.002235	0.00414	0.001862	0001578
25x25	0.0467	0.1206	0.003492	0.006462	0.00306	0.00255
30x30	0.0798	0.1264	0.005029	0.00936	0.004366	0.00351
35x35	0.1356	0.1341	0.006845	0.012722	0.005832	0.004832
40x40	0.2005	0.1439	0.008941	0.01668	0.00759	0.006122

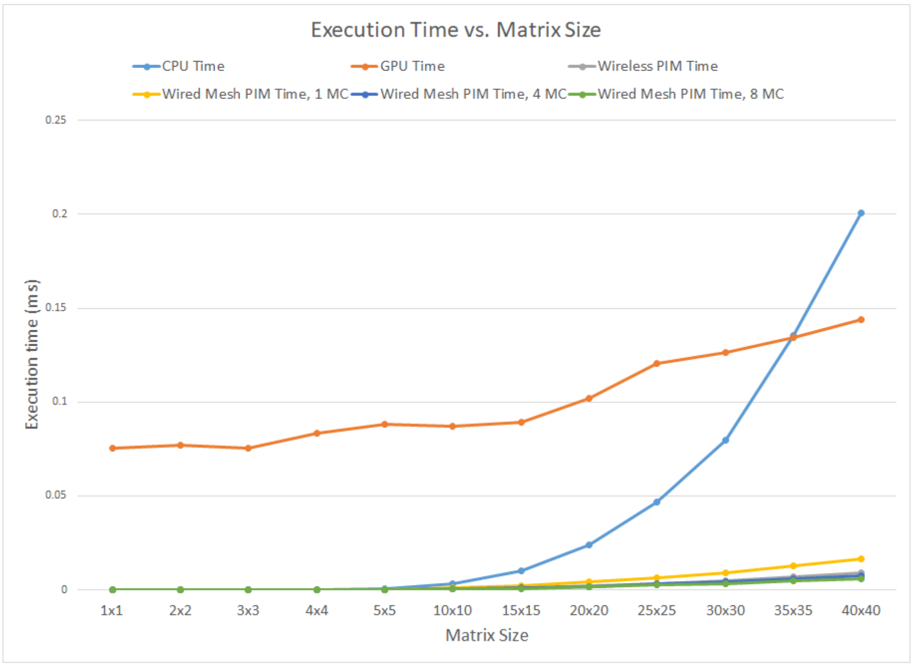


Figure 8.1: Execution time of Matrices sized between 1x1 and 40x40 for CPU, GPU, and PIM architectures.

The CPU time was obtained using a Core i5-2500 at 3.10GHz and includes all the time to access memory and manage the operating system in addition to the time required to perform the computation time. The GPU time was obtained in a similar manner using a GTX 1080 and does not include the time to initially transfer data from the host to the GPU. Both the CPU and GPU results were performed with 8GB of DRAM and VRAM, respectively.

Due to the single threaded implementation of the matrix multiplication algorithm, the execution time of the CPU increases exponentially with the size of the matrix. The GPU time, however, is highly parallelized and completed using hundreds of threads, resulting a more linear increase in execution time. Both implementations have substantially higher execution times than the wired and wireless PIM architectures. This was expected because the memory data transfer time was substantially reduced with the PIM architectures. The performance difference between the wired and wireless PIM architectures is highlighted in Figure 8.2.

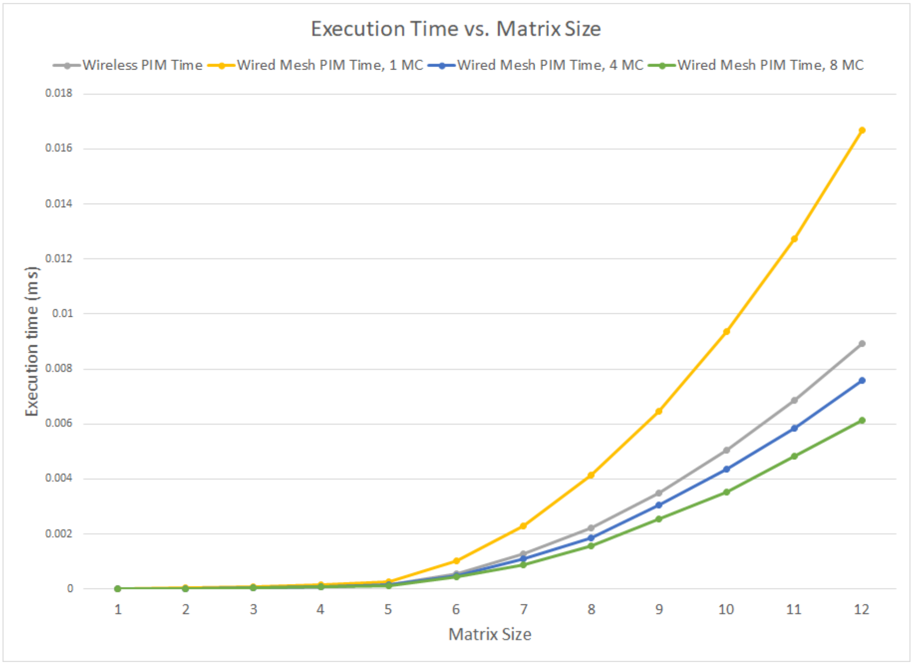
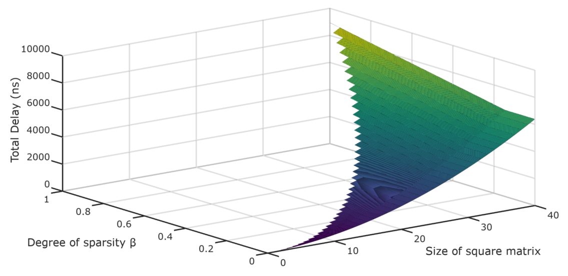


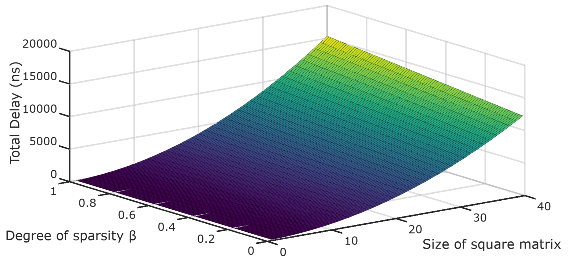
Figure 8.2: Execution time of Matrices sized between 1x1 and 40x40 for wired and wireless PIM architectures.

The graph in Figure 8.2 shows the performance of the PIM architecture when implemented using wireless interconnects and wired interconnects with 1, 4, and 8 memory controllers. The wired architecture with a single MC performed significantly worse than the other models, which was expected due to the additional latency of the wired interconnect and no parallelism gained by having multiple MCs. The wireless interconnects performed better, however the total execution time was longer than both the 4 MC and 8 MC wired models. This was due to the system being limited to a single transmission using the wireless medium at any given time, reducing potential parallelism. The 4 MC and 8 MC models had the lowest execution time due to each MC being able to act in parallel.

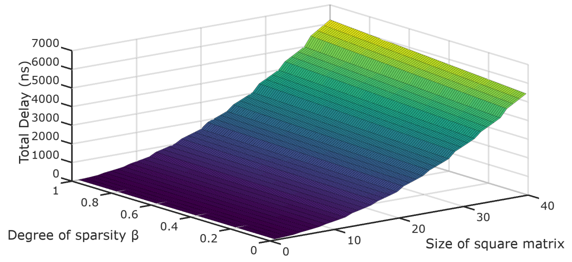
Based on the sparsity of the final result matrix, the total execution time will differ from the dense case results shown in Table 8.2. The sparsity was varied for each of the matrix sizes and used to calculate a new execution time. The calculated execution times were used to construct a 3-D surface plot for both the wired and wireless architectures as shown in Figure 8.3.



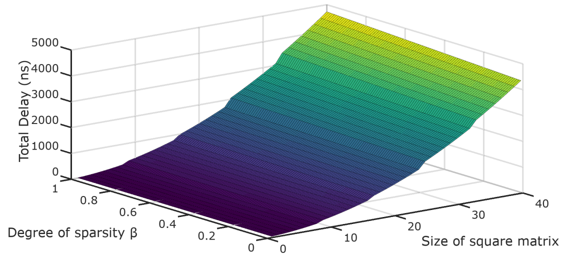
(a) Wireless interconnects



(b) Wired interconnects with 1 MC



(c) Wired interconnects with 4 MCs



(d) Wired interconnects with 8 MCs

Figure 8.3: Matrix multiplication execution time using wired and wireless interconnects with varying sparsity.

As shown in 8.3, the sparsity of the result matrix has a large impact on the total execution time. As a matrix has a larger number of non-zero elements, the time required to transmit the results increases. In the wireless plot, a linear region exists in the surface plot in which the computation time is longer than the time to retrieve the results. This region does not exist for the wired results, however, as the communication time is always longer than the computation time. In the wired plots for the 4 MC and 8 MC models, sharp increases in execution time can be observed in some portions of the surface. These sharp increases occur when the size of the square matrix is a multiple of the number of MCs in the system. This occurs because when the matrix size is a multiple of the number of MCs, the matrix can be evenly distributed among the MCs. When the matrix size is one larger, however, the unevenly distributed results causes additional time to process compared to the evenly distributed results. This manifests as a sharp increase in execution time between the two points.

8.3 Matrix Multiplication Energy

The total energy required to perform the complete the matrix multiplication operation for different sized matrices was calculated for both the wired and wireless interconnect architectures. For the wired architecture, the hop distance required to have a transmission delay of 1ns was calculated to be 1.16mm using Equation 6.1 with a known interconnect delay of 741ps/mm [4]. For the wired interconnect model, the energy required to transmit a 32-bit packet was then obtained using the interconnect distance using Equation 4.3 and found to be 6.69pJ where we assume a worst case α of 1 and a capacitance per unit length of 1.8pF/cm. In addition, each transmission using the router is assumed to use 2.5pJ of energy per packet based on post-synthesis RTL models of the NoC switch. For the wireless model, the wireless interconnects are assumed to use 1.45pJ of energy per bit transmitted, based on the work in [23] and [24].

The energy required to perform multiplication of matrices with sizes between 1x1 and 40x40 were calculated using Equation 6.7 for the wired interconnects and 6.10 for the wireless interconnects. In the calculations, the results were assumed to be dense matrices. The wired architecture power was obtained for configurations using 1 MC, 4 MCs, and 8

MCs. The results are listed in Table 8.3 and shown graphically in Figure 8.4.

Table 8.3: Matrix multiplication energy

Matrix Size	**Wireless Energy (nJ)**	**Wired Energy (1 MC) (nJ)**	**Wired Energy (4 MC) (nJ)**	**Wired Energy (8 MC) (nJ)**
1x1	0.222	0.110	0.110	0.110
2x2	1.218	0.918	0.900	0.900
3x3	3.483	3.085	3.140	3.140
4x4	7.514	7.345	7.713	7.713
5x5	13.805	14.323	17.309	15.609
10x10	96.520	114.765	128.734	154.650
15x15	310.095	387.263	424.896	480.587
20x20	716.480	918.120	994.948	1168.823
25x25	1377.625	1793.088	1993.660	2319.905
30x30	2355.480	3098.655	3411.299	3873.372
35x35	3711.995	4920.389	5377.453	6235.615
40x40	5509.120	7344.960	7984.584	9093.266

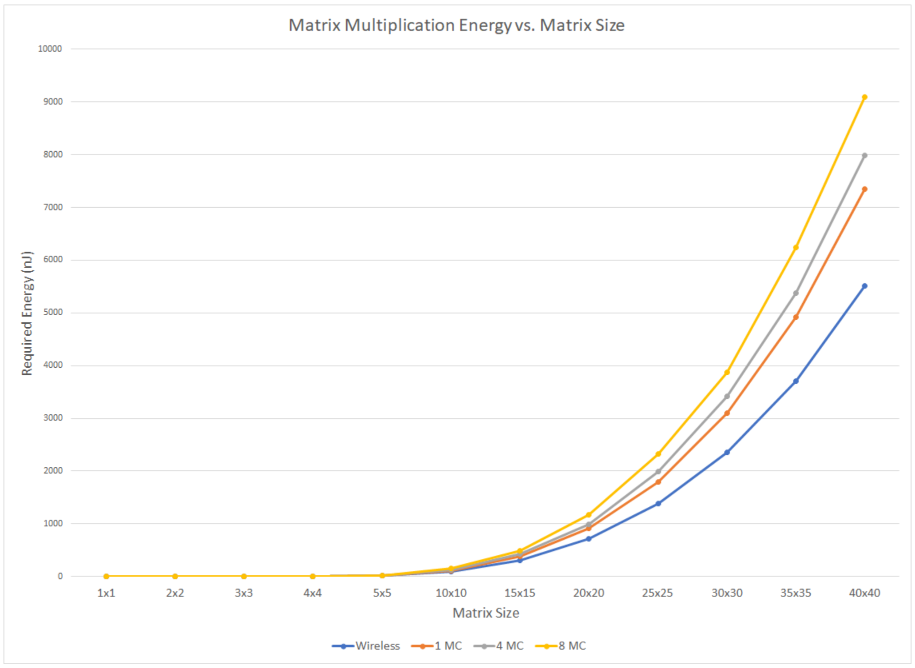
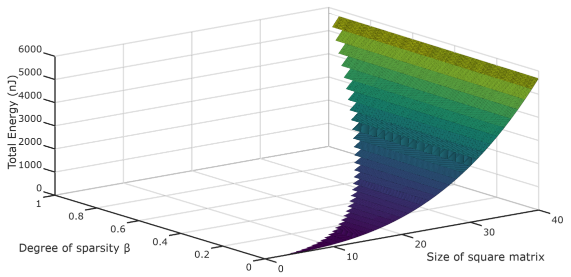


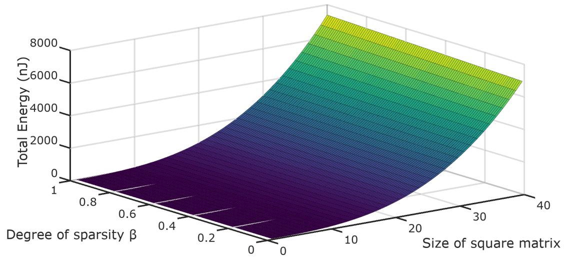
Figure 8.4: Total energy required for Matrices sized between 1x1 and 40x40 using wired and wireless PIM architectures.

For matrices larger than 4x4, the wireless interconnects used less energy than the wired interconnect model. This was due to the simultaneous multicasting which can be achieved using the wireless interconnects such that multiple PIM Clusters can receive data from the same transmission, reducing the number of wireless transmissions required. The increased parallelism gained by using more than one MC has an energy trade off, as using more memory controllers requires additional energy to perform the computations.

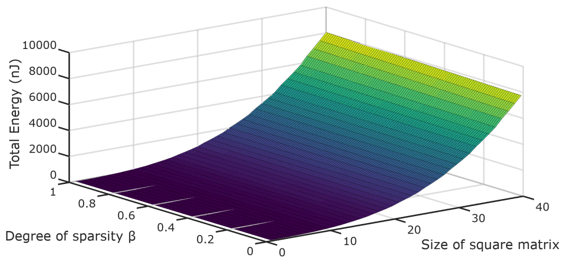
As discussed in Section 6, the total energy used during a matrix multiplication varies depending on the sparsity of the result matrix for both the wired and wireless interconnect architectures. For each matrix size, the sparsity was varied to represent result matrices containing many non-zero elements versus few non-zero elements. The results were used to generate surface plots as shown in Figure 8.5, which shows how the total energy required to complete the operation changes based on the number of non-zero elements in the result matrix.



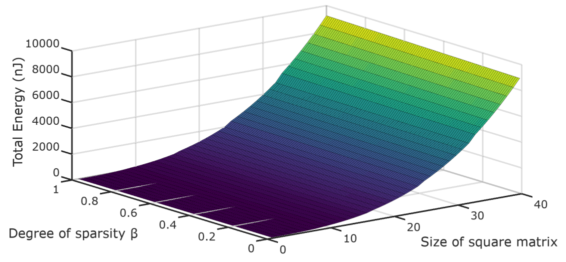
(a) Wireless interconnects



(b) Wired interconnects with 1 MC



(c) Wired interconnects with 4 MCs



(d) Wired interconnects with 8 MCs

Figure 8.5: Matrix multiplication total energy using wired and wireless interconnects with varying sparsity.

For both the wired and wireless architectures, the total energy is minimally impacted by the sparsity of the result matrix. This is because the majority of the energy is used in the initial sending of input data as well as the calculations within the PIM cores, rather than retrieving data from the clusters.

8.4 Large Dataset Timing

Larger data sets which exceed the 40x40 array of PIM Clusters must be handled through repeated batches of smaller matrix multiplications. The total execution time is therefore equal to the sum of the time required to complete each batch of matrix multiplication operations as shown in Equations 7.3 and 7.4 for the wired and wireless models, respectively. Using these models, the execution time for common video resolutions were used for the input matrix sizes. In these results, matrices of size 480x272, 720x480, 1280x720, 1440x1080, and 1920x1080 were examined to emulate a potential real world application. Each input matrix size was assumed to be multiplied by another matrix with size equal to its transverse, resulting in a square matrix. The results are listed in Table 8.4.

Table 8.4: Large dataset Timing

Matrix Size	**CPU Time (ms)**	**GPU Time (ms)**	**Wired Time (1 MC) (ms)**	**Wired Time (4 MC) (ms)**	**Wired Time (8 MC) (ms)**	**Wireless Time (ms)**
480x272	183.7	6	7.744	4.431	3.885	6.263
720x480	767.1	21.5	28.201	16.703	14.802	24.129
1280x720	4089.6	94.4	128.431	77.345	68.879	112.862
1440x1080	7819.2	172	237.159	144.509	129.129	212.330
1920x1080	17094	321.2	421.614	256.904	229.561	377.473

Both the wired and wireless PIM architectures are significantly faster than the CPU implementation, which was expected due to the parallelism and reduced memory access time. The parallelism gained by using multiple MCs allows the wired architecture to outperform the wireless architecture. The wireless and 1 MC models have a longer execution time than the GPU test. This was likely caused by the extreme levels of parallelism which is available when using a GPU. The 4 MC and 8 MC wired models have a lower execution time than GPU.

8.5 Large Dataset Power

Similar to the large dataset timing models, the matrix multiplication can be broken into and executed in batches. The total energy of the matrix multiplication is therefore the sum of the energy used in each of the batch executions, as described in Equations 7.5 and 7.6 for the wired and wireless architectures, respectively. The total power was calculated for large input matrices of sizes 480x272, 720x480, 1280x720, 1440x1080, and 1920x1080. Each input matrix was assumed to be multiplied by a matrix with size equal to its transpose, resulting in a square matrix. The results are displayed in Table 8.5.

Table 8.5: Large dataset energy

Matrix Size	**Wireless Energy (μJ)**	**Wired Energy (1 MC) (μJ)**	**Wired Energy (4 MC) (μJ)**	**Wired Energy (8 MC) (μJ)**
480x272	5304.951	6032.322	7146.697	8314.274
720x480	21042.063	23746.262	28245.950	32894.596
1280x720	99703.194	111953.302	133539.245	155619.994
1440x1080	189228.442	212139.438	253244.058	295184.301
1920x1080	336392.847	376758.827	450024.271	524616.464

Similar to the results from Table 8.3, the wireless PIM architecture used the least energy to perform the matrix multiplication operation. Increasing the number of MCs in the system increases the total energy required to perform the operation. Using the total time from Table 8.4 and energy required from Table 8.5, the average power of the design can be estimated by dividing the energy by the execution time. The estimated power is given in Table 8.6.

Table 8.6: Large dataset estimated power

Matrix Size	**Wireless Power (W)**	**Wired Power (1 MC) (W)**	**Wired Power (4 MC) (W)**	**Wired Power (8 MC) (W)**
480x272	0.847	0.779	1.615	2.140
720x480	0.872	0.842	1.691	2.222
1280x720	0.883	0.872	1.727	2.259
1440x1080	0.891	0.895	1.752	2.286
1920x1080	0.891	0.894	1.7517	2.285

As shown by Table 8.6, the average power to perform the matrix multiplication is very low compared to the CPU and GPU used for testing which have rated Thermal Design Power (TDP) of 95W and 180W respectively. The TDP is a maximum rating for the chips which represent the thermal power that must be dissipated during a maximum intensive workload. As such, the actual power consumption may be higher than this listed amount. In addition, the TDP spec is a representation of the total power of the system and therefore includes both dynamic power and static power. The static power of the CPU and GPU are not publicly available metrics, however it is likely small due to the high concentration of logic in the processors. This power does not take the interconnect power required to access main memory or the power required by the main memory itself, however, which are considerable for data intensive workloads. The proposed PIM architecture had a maximum calculated dynamic power of 2.286 W when using the 8 MC configuration, which is significantly less than both the CPU and GPU implementations. This measure does not include the static or dynamic power of the memory as those are depend on size and configuration of the memory. Compared to the CPU and GPU, however, the memory would have a considerable amount of leakage power. For both SRAM and DRAM, the static power required per cell is fairly high compared to the dynamic power required. Over time, this can become dominant part of the total power consumption.

An estimation of the static and dynamic power required by the memory can obtained using the power per cell listed in Table 8.1. To obtain a minimum bound, the size of the memory is assumed to be large enough to fit a single frame of the standard video resolution matrices used in previous calculations. Each element in the matrix is assumed to be composed of three 8-bit values to represent the three color values of an RGB pixel. The memory must be large enough to hold two frames of input data, plus the resulting matrix. The static and dynamic power is then calculated by multiplying the power per cell by the number of cells as shown in Table 8.7.

In order to store the required number of elements for each matrix size, the required

Table 8.7: Memory Power Estimates

Matrix Size	Total Number of Elements to Store	Standard Memory Size Required	SRAM		DRAM	
			Dynamic Power (W)	Static Power (W)	Dynamic Power (W)	Static Power (W)
480x272	1474560	2 MB	0.944	0.629	0.210	2.10E-05
720x480	3628800	4 MB	1.887	1.258	0.419	4.19E-05
1280x720	10444800	16 MB	7.550	5.033	1.678	1.68E-04
1440x1080	15552000	16 MB	7.550	5.033	1.678	1.68E-04
1920x1080	23500800	32 MB	15.099	10.066	3.355	3.36E-04

size was rounded up to the closest standard memory size which are based on powers of two. This standard memory size was used in the calculations to find the static and dynamic power of both SRAM and DRAM implementations. Based on the results from Table 8.7, the power requirements of the memory are very high compared to the power of the PIM Clusters themselves. For the largest matrix size of 1920x1080, the SRAM dynamic power of 15.099W is substantially larger than the power to complete the matrix multiplication using both the wired and wireless interconnects. In addition, the static power is also very high for SRAM, being equal to 66% of the required dynamic power. The power of a DRAM block of the same size is substantially less, however the dynamic power is still more than the power of the wired and wireless PIM architectures. Additionally, the power metrics of DRAM does not account for refreshes to keep the data in memory.

Chapter 9

Conclusion

In order to overcome the growing processing and memory gap, unique and novel architecture solutions are required. Processing-in-Memory (PIM) architectures are designed to alleviate the issue by reducing the total number of data communications required between the processor and the memory. In this work, a novel Processing-in-Memory architecture is proposed which uses simple, reconfigurable logic to perform arbitrary functions. These small logic units, called PIM Cores, are capable of implementing any potential function using two 4-bit inputs and producing an 8-bit output by reading the function values from memory. By combining nine PIM Cores together using an all-to-all network, a PIM Cluster capable of performing larger functions such as 8-bit Multiply-Accumulate can be implemented. Many PIM Clusters can then be used in parallel to perform large scale operations such as matrix multiplication. The design makes use of novel wireless interconnects to aid in data communication. The flexible wireless interconnects provide a seamless mechanism to transition between dense and sparse matrix multiplication applications, where time and energy can be saved by transmitting only non-zero results.

Analytical models were proposed to evaluate the proposed architecture in terms of area, execution time, and energy using both wired and wireless interconnects. The execution time was compared against CPU and GPU matrix multiplication implementations to evaluate the architecture when compared to conventional architectures. The calculated power was also compared to obtain an estimate of the power efficiency when compared to the

CPU and GPU implementations. When applied to real-world matrix sizes, the proposed architecture using wired interconnects was found to offer a best case execution time speedup of 74.5x and 1.4x when compared to the CPU and GPU. Using the wireless interconnects, speed ups of 45.3x and 0.85x were obtained. The wireless interconnects excel in terms of power, however, as the wireless interconnect architecture offered a 260.5x reduction in power consumption compared to the wired model.

9.1 Future Work

The presented work serves as an initial proposal to the outlined PIM architecture off of which future work can expand upon and improve the design. The design can be more fully elaborated to include the construction of an Instruction Set Architecture (ISA) for programming the proposed architecture. An ISA is needed to establish a standard set of communication behavior between the PIM architecture and the host processor. This ISA would need to include functions for programming the PIM cores, assigning memory addresses to the PIM Clusters from which they will pull data, and establishing the required communication patterns between the cores of a PIM Cluster. These functions will allow a more integrated design to take shape such that the host processor can manage and control the PIM architecture.

In addition, the exact communications required within the PIM architecture can be further elaborated to establish the packet structures required to execute the proposed algorithms. In this work, 32-bit flits were assumed for all communications within the subNoC and within the array of PIM Clusters. With an in depth analysis, a defined packet length and packet structure can be created for each of the required communication types used. In doing so, the packet sizes can potentially be reduced, requiring less transmission energy to send the flits.

The proposed large dataset timing and power models can be further elaborated to account for varying workloads which exist when evaluating incomplete batches with fewer elements. Currently, the models treat these incomplete batches as if they were full, which leads to an overestimation of the required time and energy to complete the multiplication. By more accurately reflecting the number of PIM Clusters that must perform operations and send results, the performance of the models will likely improve. This would more accurately show the potential of the architecture and provide a better comparison against the CPU and GPU implementations.

The final step required in the elaboration of this design is full construction and verification. An initial design of the PIM Core was constructed for this work, however the full architecture was not designed and simulated. To gather more accurate timing, power, and area data, the full integrated system must be constructed and analyzed. A full system test can then be designed to execute the proposed architecture and verify functionality. If possible, the architecture can also be fabricated as an ASIC circuit, or implemented on a small scale using an FPGA.

Bibliography

[1] D. Efnusheva, A. Cholakoska, and A. Tentov, “A survey of different approaches for overcoming the processor - memory bottleneck,” *International Journal of Computer Science and Information Technology*, vol. 9, no. 2, pp. 151–163, apr 2017.

[2] J. Hruska, “MIT Develops 3D Chip That Integrates CPU, Memory,” July 2017. [Online]. Available: https://www.extremetech.com/computing/252007-mit-announces-breakthrough-3d-chips-integrate-memory-cpu

[3] W. A. Wulf and S. A. McKee, “Hitting the memory wall: Implications of the obvious,” *ACM SIGARCH Computer Architecture News*, vol. 23, no. 1, pp. 20–24, mar 1995.

[4] S. I. Association, “Emerging research devices,” *2007 International Technology Roadmap for Semiconductors*, 2007.

[5] M. Horowitz, “1.1 computing’s energy problem (and what we can do about it),” in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, feb 2014.

[6] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra, “A review of near-memory computing architectures: Opportunities and challenges,” in *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE, aug 2018.

[7] H. S. Stone, “A logic-in-memory computer,” *IEEE Transactions on Computers*, vol. C-19, no. 1, pp. 73–78, Jan 1970.

[8] J. T. Pawlowski, “Hybrid memory cube (HMC),” in *2011 IEEE Hot Chips 23 Symposium (HCS)*. IEEE, aug 2011.

[9] AMD, “High-Bandwidth Memory,” 2015. [Online]. Available: https://www.amd.com/en/technologies/hbm

[10] N. Jao, A. K. Ramanathan, S. Srinivasa, S. George, J. Sampson, and V. Narayanan, “Harnessing emerging technology for compute-in-memory support,” in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2018, pp. 447–452.

[11] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, “Magic—memristor-aided logic,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, Nov 2014.

[12] D.-I. Jeon, K.-B. Park, and K.-S. Chung, “HMC-MAC: Processing-in memory architecture for multiply-accumulate operations with hybrid memory cube,” *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 5–8, jan 2018.

[13] P. Das, S. Lakhotia, P. Shetty, and H. K. Kapoor, "Towards near data processing of convolutional neural networks," in *2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID)*. IEEE, jan 2018.

[14] Y. Wang, W. Chen, J. Yang, and T. Li, "Towards memory-efficient allocation of CNNs on processing-in-memory architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 6, pp. 1428–1441, jun 2018.

[15] D. Pala, G. Causapruno, M. Vacca, F. Riente, G. Turvani, M. Graziano, and M. Zamboni, "Logic-in-memory architecture made real," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, may 2015.

[16] L. Benini and G. D. Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan 2002.

[17] M. S. Shamim, N. Mansoor, R. S. Narde, V. Kothandapani, A. Ganguly, and J. Venkataraman, "A wireless interconnection framework for seamless inter and intra-chip communication in multichip systems," *IEEE Transactions on Computers*, vol. 66, no. 3, pp. 389–402, March 2017.

[18] A. W. Topol, D. C. L. Tulipe, L. Shi, D. J. Frank, K. Bernstein, S. E. Steen, A. Kumar, G. U. Singco, A. M. Young, K. W. Guarini, and M. Ieong, "Three-dimensional integrated circuits," *IBM Journal of Research and Development*, vol. 50, no. 4.5, pp. 491–506, July 2006.

[19] X. Wu, Y. Ye, W. Zhang, W. Liu, M. Nikdast, X. Wang, and J. Xu, "Union: A unified inter/intra-chip optical network for chip multiprocessors," in *2010 IEEE/ACM International Symposium on Nanoscale Architectures*, June 2010, pp. 35–40.

[20] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, jun 2016.

[21] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, jun 2016.

[22] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, feb 2017.

[23] X. Yu, S. P. Sah, H. Rashtian, S. Mirabbasi, P. P. Pande, and D. Heo, "A 1.2-pJ/bit 16-gb/s 60-GHz OOK transmitter in 65-nm CMOS for wireless network-on-chip," *IEEE Transactions on Microwave Theory and Techniques*, vol. 62, no. 10, pp. 2357–2369, oct 2014.

[24] X. Yu, H. Rashtian, S. Mirabbasi, P. P. Pande, and D. Heo, “An 18.7-gb/s 60-GHz OOK demodulator in 65-nm CMOS for wireless network-on-chip,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 3, pp. 799–806, mar 2015.

[25] S. I. Association, “System drivers,” *2007 International Technology Roadmap for Semiconductors*, 2007.