A Survey on Processing-in-Memory

Aashutosh Taikar[1] Ganesh Vhatkar[2] and Warit Paweenbampen[3]
School of Electrical and Computer Engineering
Oregon State University
{taikara,vhatkarg,paweenbw}@oregonstate.edu

***Abstract*— Processing-In-Memory (PIM) is a growing technology that aims at reducing the overheads associated with data movement within the processor and the memory. The traditional architecture has been using separate memory and processor that produces a latency during the migration of the data. The data propagation involves heat dissipation and thus more energy consumption. PIM focuses on moving the computation closer to the memory which will aid in minimizing the dormancy of the data migration that results in the reduction of power dissipation. In this survey paper, we have shown results that highlights the better performance of the PIM in comparison to the traditional architectures. Different topologies of PIM architectures with their challenges have been discussed. Further, applications of the PIMs for an advanced technological operation like DNA mapping, Artificial Neural Networks, etc., have been examined. PIM for Solid State Drives is also proposed in the paper. The survey has the objective to promote the use of PIM in the future computers.**

Keywords–Processing In-Memory, PIM interconnects, Artificial Neural Networks, Processing In-Flash

I. INTRODUCTION

This survey paper highlights the potential and capabilities of Processing In-Memory (PIM). PIM aims at performing the computation of data near memory or stacked memories. The reason is to reduce the data access and propagation time between the processor and the memory. The data propagates through interconnects which contributes to the complexity as well as latency. Every instruction that demands computation operations like arithmetic, Boolean, comparison, etc., requires the data to be fetched from the main memory (DRAM) and travel to the processor through the interconnects. The processor spends more time waiting for the data to be fetched from the memory. It is observed that reading a 64-bit integer from a DRAM consumes around 16000pJ of energy [1]. This not only leads to the transit delay as mentioned but also result in tremendous power dissipation. This kind of separate processor and the memory architecture can be found in the traditional computers which are built on Von Neuman architecture [15] as shown in Fig. 1.

PIM focuses on solving this problem by keeping the memory and the processor near to each other so that the data propagation time can be reduced and so does the power and energy dissipation. The memory and the processor can be integrated into the same die rather being into the separate dies [5]. Earlier PIM implementations namely FlexRAM [4] & Data Intensive Architecture (DIVA)[5] have tried this designed and shown some positive results in the reduction

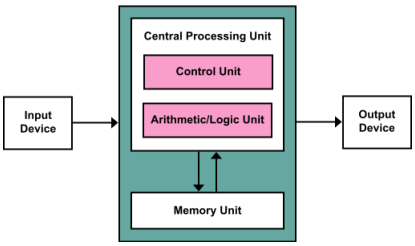

Fig. 1. Von Neumann Architecture. [9]

of the latency & power dissipation. Also, modern PIM implementations like the 3D die stacking using HMC [1] not only decrease the latency of the data access and energy consumption but also increases the memory bandwidth by 3D stacking of the DRAMs. Moreover, using multiple PIMs can also fasten the processing speed than just a single processor. Validated and promising results have been highlighted later in the paper. All these efforts by the researchers have shown hope for the implementation of the PIMs in the future computers.

The remainder of this survey paper is organized as follows. Section 2 presents a brief background about the PIM overview followed by the types of PIM implementations in Section 3. Section 4 describes the interconnection of these PIMs. Section 5 presents the concept of PIM to be implemented in SSDs.

II. BRIEF OVERVIEW OF PIM

Processing-In-Memory is the integration of a processor with a RAM on a single chip. PIM architecture is an approach to overcome the limitations of von Neuman architecture as mentioned above. PIM can also be used as one or many miniature computers controlled by a host CPU. In the fabrication of the PIM, the CMOS logic devices and memory cells are tightly coupled. The processor logic is directly connected to the memory stack. This results in an increment in the integration of processor and memory by increasing the processing speed and memory transfer rate and decreasing the latency and power usage. The modern-day computer uses 64-bit processors which have resulted in the storage of a tremendous amount of data in the memory. This has made the prices of the memory cheap and thus, processing in memory has become practical for increasing number of applications [9].

III. Traditional PIM implementations

PIM attracted significant attention in the research community around the beginning of this century. There were many attempts and efforts concentrated on embedded DRAM in the logic processes. These involved PIM designs like Intelligent RAM (IRAM) [13]. However, IRAM did not prove to be a cost-effective solution as it failed to accommodate enough capacity of memory to high-performance systems due to the reduced density of the embedded memory. Other efforts like the Data Intensive Architecture (DIVA)[5], Active Pages and the FlexRAM [6] tried to integrate logic on the memory dies. Sadly, it did not work as expected due to the decline in the performance of the logic implemented in the DRAM processes. They also reduced the performance of the in-memory processors. This led to the limited applicability and programmability of such PIMs.

A. FLEXRAM

The FlexRAM [6] was proposed in 1999 to power the general purpose computing requirements. The Merged Logic DRAM (MLD) technology is being used to implement FlexRAM. It addressed the general purpose computing so as to be compatible with the existing programs which use DRAM. Taking the compatibility into consideration the FlexRAM is implemented with the standard (Rambus) [6] that includes additional power and ground signals in order to enable on-chip processing.

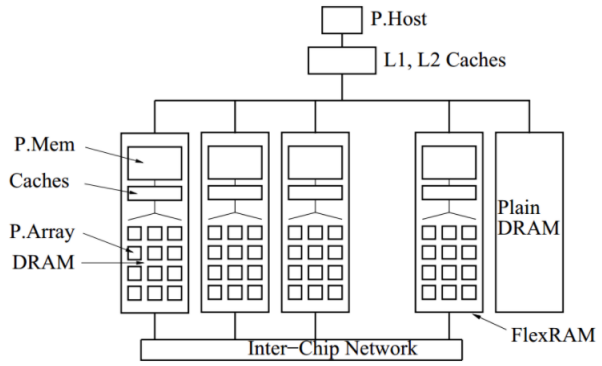

Fig. 2. Overall organization of the FlexRAM-based memory system. [6]

The FlexRAM comprises of P. Arrays with P. Array Engines and P. Mem Processor, as shown in Fig. 2. The P.Arrays are finely interleaved within the DRAM macrocells. They are implemented with a simple arithmetic RISC 32-bit compute engines. If required multipliers can be implemented to support applications like neural networks. This, however, needs to be shared by the P.Arrays to reduce the area. The P.Array engine works in Single Program Multiple Data (SPMD) mode vs the less efficient and restrictive Single Instruction Multiple Data (SIMD). Each P.Array needs a small instruction memory and the neighboring P.Array can share the memory. Each P.Array is associated with 1MB of DRAM memory. Each Flexram has 64 such P.Arrays which sums up to 64MB. P. Mem Processor has floating point execution units to process serial tasks which if absent have to be processed by the Host CPU. The P.Mems and the P.Arrays share some virtual data with the P.Host (the host CPU). The architecture used the Rambus model which allowed additional power and ground pins. Memory mapped I/O was used as the usage of Direct Memory Access (DMA) for bus mastering would have been cumbersome during that era. Also, each P.Array needed balanced clock signal which was challenging so Phase Locked Loop(PLL) was implemented [6]. The results of FlexRAM were appreciable but it was limited due to the challenges mentioned. Also, being the first implementation, scalability, complexity, and fabrication was an issue. Thus a new architecture was proposed which efficiently overcomes the complexity of FlexRAM.

B. Data Intensive Architecture (DIVA)

The DIVA architecture along with Double Data Rate(DDR) RAM is implemented shown in shown in [5]. The DIVA supports in order execution with 32-bit instructions and 32-bit addresses. Unlike FlexRAM which uses RAMBUS and memory mapped I/O standard, this architecture exploits the use of Direct Memory Access(DMA). It has two data paths executed by a single execution control unit: 32-bit scalar datapath and a 256-bit WideWord datapath.

The SRAM based chip from [5] implementation fabricated through MOSIS (a fabrication service provider) and TSMC (a semiconductor company) is shown in Fig. 3. The WideWord datapath includes eight single precision floating point units (FPU). The FPU itself being a 5 stage pipeline a complex pipeline controller was designed to handle instruction sequencing and to avoid data hazards. An address translation unit was incorporated to provide a virtual to physical address mapping for both instructions and data. The DIVA architecture still was implemented on a single layer, i.e., the processor and the memory were fabricated on a single plane. This introduced complexity considering the memory density and the processor interconnects. Also, routing the interconnects was cumbersome as it had to be implemented in a single plane.

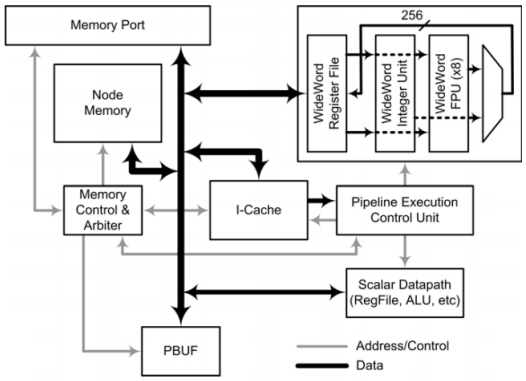

Fig. 3. DIVA PIM node organization. [5]

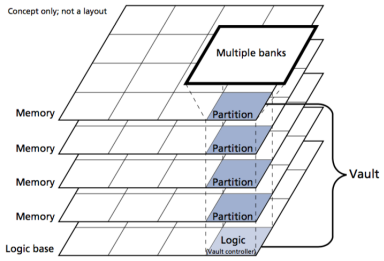

Fig. 4. Hybrid Memory Cube. [5]

IV. Modern PIM Implementation

Moving computation closer to data has the potential to improve both the performance and the energy efficiency of the memory accesses. One approach to achieve this is to integrate processing-in-memory (PIM) capabilities with memory dies using memory die Stacking.

A. 3D Die Stacking (using Hybrid Memory Cube)

Another way is using the approach of PIM implemented using the 3D die stacking so that we can tightly coupled processors implemented in a logic process, with memories implemented in memory process using through-silicon via (TSV). This circumvents the problems of logic performance and memory density. Recent evaluations of 3D-stacked PIM have shown tremendous promise with an order of magnitude or more improvements in energy efficiency and/or performance for memory-intensive workloads.

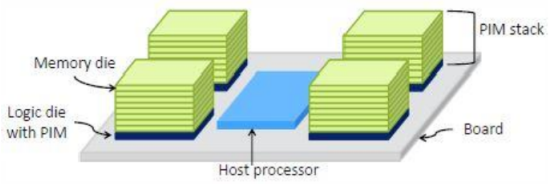

Fig. 5. 3D die stacking using HMC. [5]

Fig. 5 highlights the blocks of a 3D die stacked HMC System Diagram. It includes a Host Processor; an Auxiliary Processor called the Logic Die and DRAM Memory Dies. By implementing 3D die stacking, all the necessary memory-bound computations can be offloaded from the main host processor to these auxiliary, in-memory processors. Hybrid memory cube (HMC) as shown in Fig 4.[4] proposed by micron consists of DRAMs stacked and connected together. The DRAM layers connected to the same partition is controlled by the Vault Controller. The Vault controller also facilitates the Memory access requests from the host processors or the PIM processors. However, the logic die of the HMC had limited processing capabilities. Due to this advancement in the base logic die of the HMC was done which added more complex processing abilities like memory controllers to it and is called as Smart Memory Cubes(SMC). We use 25nm and 10nm nodes for the memory and logic die, respectively. The reason being, a 25nm node quadruples the DRAM density of a 50nm node. We assume the base logic die area matches the stacked memory die area because PIM is intended to supplement the host processor rather than implement full processing capability.

B. PIM logic die implementations

There are three architecture design options with a various power-density profile for the logic-die. The first one is the GPGPU Unit. A GPGPU consists of many lightweight SIMD units for data-parallel execution and can generate many concurrent memory accesses. The PIM logic dies consists of 16 Compute Units (CUs) and a 2MB shared L2 cache as shown in Fig. 6. The second design consists of four two-way multithreaded, out-of-order (OoO) core with a two-level cache hierarchy per core as shown in Fig. 6 and the third design combines half the resources from each of the first two design to form an Accelerated Processing Unit (APU). This design consists of two OoO cores, eight Cus, and a 2MB GPU L2 cache as shown in Fig. 6.

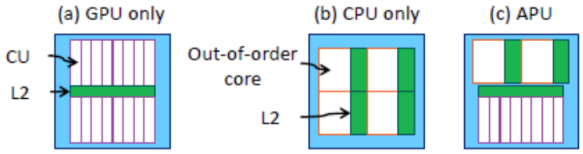

Fig. 6. PIM logic die design. [12]

C. Smart Memory Cube (SMC)

Smart Memory Cube, or SMC, is another approach that implements the logic base of PIM that uses HMC as a base-line, which is shown in Fig. 7 [10]. Unlike other implementations of PIM, SMC focuses on a scenario that a single PIM processor is used along with a single thread on a host and focuses more on virtual memory. Moreover, the main objectives of the proposal are (1) create an optimized memory virtualization scheme for zero-copy data sharing between host and PIM; (2) enhanced PIMs operations on the atomic processing in-memory operations; and (3) enhanced PIMs memory access from using Direct Memory Access (DMA) engine [10]. To model an SMC, SMCSim was used, which is a high-level simulation that is based on gem5, to model an SMC with the host SoC. This system uses the HMC concept as the main memory part which includes multiple vaults, a vault controller in each vault [10]. Vault controllers are augmented which will be used to handle the atomicity. According to the paper, with the SMC, any host platform will have an ability to communicate from the not-limited to ISA. Nevertheless, there is a PIM which includes many components inside [10].

Inside PIM, ARM-Cortex-A15 core with its own interconnection was implemented. The design also contains Scratchpad Memory (SPM), Direct Memory Access (DMA) engine, Translation Look-aside (TLB), and Memory Management Unit (MMU) [10]. The design of PIM includes TLB inside

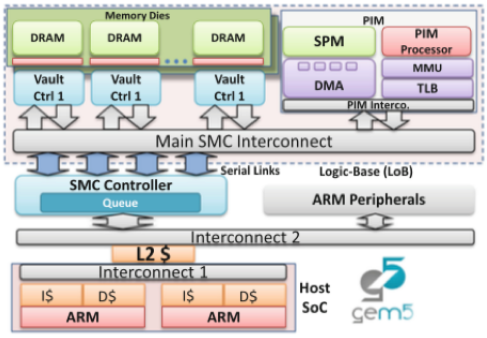

Fig. 7. SMCSim design. [10]

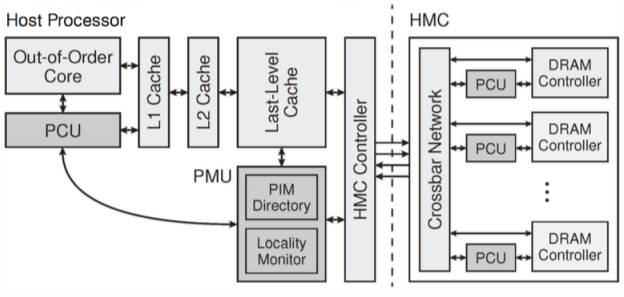

Fig. 8. Overview of the PEI architecture. [5]

it in order to be used for virtual memory access. TLB is included inside PIM in order to reduce the delay since PIM is far away from a host processor. Nevertheless, if there is a miss on TLB, this project uses Slices concept as a generalization to the memory page. Slice means "a region of memory composed by 1 or more memory-pages which are contiguous in both virtual and physical memory spaces" and has the ability to merge into a bigger slice, which similar to page-table concept that it will be consulted for OS to the translation of mapping [10]. After the mapping, the MMU is responsible to handle the granularity of the slices. Additionally, DMA engine is the component that will be used for transferring data between DRAM vaults and its SPM [10]. DMA engine and SPM are included in order to reduce the latency since PIM has high latency when accessing DRAM; consequently, with the smaller latency, it can also reduce the energy consumption. According to their result, the latency reduces up to 2 times when compare between PIM and host SoC, and 1.5 times when compare PIM to host-side accelerator from running with different benchmarks: average teen followers; breadth-first-search; page-rank; and bellman-ford-shortest path [10]. The energy is reduced about 70% and 55% when compare PIM with host side, and accelerator, respectively [10].

D. Thermal challenges of 3D stacked PIM

A potentially significant challenge for PIM is thermal management. Because the memory lies between the heat sink and the logic die, heat generated from the logic die raises the temperature of the memory. The typical operating temperature range for DRAM is under 85°C. After the temperature exceeds that threshold, the refresh rate must be double for every 10°C increase. Higher refresh rates not only consume more power but also reduce the availability of the DRAM, resulting in lower memory performance. The cooling techniques involved methods like using passive heat-sinks, Low-end active heat-sinks (fans), Commodity-server active heat sink, and High-end-server active heat sink.

However, researchers have carried out simulations using HotSpot Thermal Simulation tool and calibrated it against thermal models [12]. It was found that the die farthest away from the heat sink is the logic die in an eight-die stacked DRAM. They concluded that even the low-cost passive heat sink can sustain up to 8.5W of logic power while keeping the nominal DRAM refresh rate and therefore maintaining the DRAM bank availability. This suggests that PIM capability can be made comparable to those mobile processors, opening the possibility of integrating a fairly intelligent in-memory processor into a memory stack to combat the increasing off-chip-memory-access inefficiency. For a 1W increase in the stacked-memory power, the logic-die power budget needs to decrease by approximately 1W to keep the memory temperature below 85°C.

All these results have shown that PIM is feasible even with low-end, fanless cooling solutions. Thus, it has kept the researchers interested in this technology for further improvements and exploration.

E. PIM- Enabled Instructions

PIM-Enabled Instructions (PEI) was a proposed-on PIM architecture that supports PIM operation and identifies the locality and execution of data in the processor. According to the proposal, this architecture supports current ISA and has an ability to choose what to execute which will be done at a hardware level [3]. The architecture combines the Von Neumann architecture and PIM concept together as the overall concept of PEI, showing that there are two sides: host-side, and memory side. In host side, there are out-order core and multiple levels of cache, e.g., L1, L2, L3; also, two new components are included which are PEI Computation Unit (PCU), and PEI Management Unit (PMU). In memory side (PIM) use HMC as a base-line, has multiple stacked DRAMs and small computation unit in the logic base, which also includes multiple PCUs.

Computational Unit

The overview of the architecture, the processor communicates with 3D-stacked DRAM within a die; however, as shown in Fig. 8, some additional components are implemented which are PCUs and PMU [3].

1. PEI computation Unit (PCU)

The PEI Computation Unit (PCU) is the component that executes PEIs. Multiple PCU are used in (1) PIM section and (2) between host processor core and PMU. Each PCU has the same logic as others and includes the operand buffer [3]. The operand buffer is an SRAM buffer which will take advantage of memory level parallelism from the

communication between cores, sending a read request to the target cache block one there is a free buffer entry, then fetched data until it is available [3].

2. PEI management Unit (PMU)

The PEI Management Unit (PMU) component contains 2 parts which totally has three main tasks: maintain atomicity property of PEIs, cache coherence management, and locality of data [3]. The first part, called PIM directory, is to organize the atomicity of PEIs by tracking all PEIs to assure that there is only one writer PEI; in other words, it is a locking mechanism. For example, PIM directory will block the read PEI arrive and mark as non-writable to block incoming write PEIs. When write PEI arrives, it has to assure and allow only single write for each cache block, then make sure the starvation cannot be occurred by marking it as non-readable [3]. In addition, for the cache coherence management, it is the situation when the PMU get a PEI, it will request back-invalidation or back-write for the target cache block [3]. This will assure that on-chip caches or main memory will have a copy of data before/after they execute PIM operation. Nevertheless, the mechanism that decides whether the PEI should be executed in the host-side or memory-side (PIM) is called locality monitor. The locality monitor allows the data locality of PEI to be known by checking whether target cache block hit or not. For example, if the target cache block has high locality, it will be more likely presented in the monitor; however, if the target cache block does not have high locality, it means that the target is more likely can be found in memory-side. Nevertheless, if the target cache block is either hit or miss, the locality will be updated in any cases.

PEI execution

There are two main parts of PEI execution: host-side and memory-side (PIM).

1. Host Side

For host side, the overall steps are shown in Fig 9. First, the host processor will send the input PEI operand to PCU to issue it. Then, the host-side PCU will access the PMU to get a read-write lock and decide the best location to execute by using locality monitor. Next, the host-side will allocate a new operand buffer entry and copy the input operand, which is located in a memory-mapped register, and load the cache block from L1. Then, PCU will execute the PEI and start a store request to L1 if needed. After that, the host side will do the notification of the completion of PMU to release the PIM directory. Lastly, the host processor will read output in PCU and deallocate the buffer.

2. Memory Side

For memory-side, the steps are shown in Fig. 10. The first two steps are the same as the host side. The next step is that, the PMU will send a write back signal to the last level of cache to clean the copy in the cache block, and at the same time, transfers the input operand to the main memory and then release the corresponding PIM directory entry and send output back to PCU on the host side.

From the simulations and experiments, this architecture uses the positive side of conventional architecture and simple PIM operation to improve the performance and energy con-

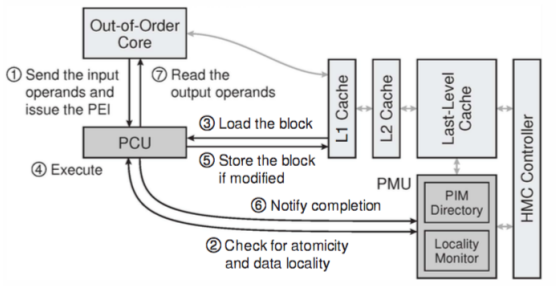

Fig. 9. Host side execution. [3]

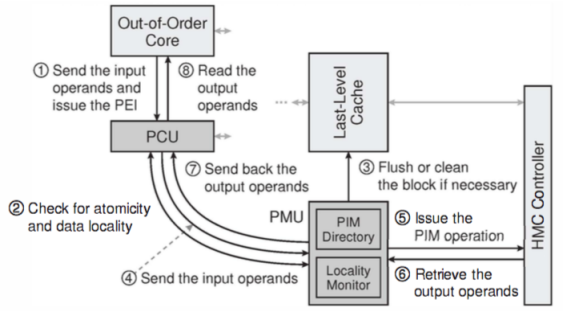

Fig. 10. Memory side execution. [3]

sumption [3]. The specification of the simulator is described in this paragraph. It uses an in-house x86-64 simulator, which is based on Pin, used with 16 out-of-order cores, 4 GHz, 4-issue [3]. There are 3 caches, the L1 is Private, 32 KB, 4/8-way, 64 B blocks, 16 MSHRs while L2 is Private, 256 KB, 8-way, 64 B blocks, 16 MSHRs, and Shared, 16 MB, 16-way, 64 B blocks, 64 MSHRs for L3 [3]. For the interconnection, Crossbar with 2 GHz, 144-bit links is used as their On-Chip network. The main memory is 32 GB with 8 HMCs; when HMC is 4 GB, 16 vaults, 256 DRAM banks, including DRAM and Vertical Links [3]. They evaluate the result by distinguishing into 3 criteria: host-only, PIM-only, and Locality-Aware (PEI) from different benchmarks: average teenage follower, breadth-first-search, page rank, single source-shortest path, and weakly connected components. According to their result, they first evaluate the bandwidth consumption performance. For large inputs, PIM not only significantly decreases the off-chip bandwidth consumption but also greatly increases the off-chip bandwidth consumption when they use small inputs [3]. Second, the speedup of PIM-Only for large input about 47% over host-only; nevertheless, host-only has better speedup over PIM-only about 32% when it is small inputs [3]. However, by using PEI (local-aware) has about 12% over host-only and PIM-only [3]. Moreover, the result from the experiment on Multiprogrammed Workloads is that using PEI has better performance over host-only and PIM-only because PEIs are executed even if applications with very different locality behavior are mixed [3]. The researchers also state that this architecture improves the performance on balanced dispatch up to 25% and has less energy consumption for small, medium, and large input [3].

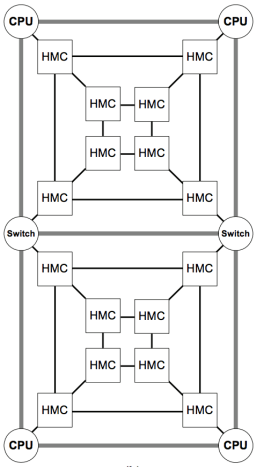

Fig. 11. Memgrid network with 16 HMC modules. [2]

V. MemGrid and MemGrid HMC switch (PIM-PIM interconnection network)

A. Connection topologies of Memgrid network

With growing data-intensive applications like social media and video streaming, there is a necessity to use more than one PIM. When multiple PIMs need to communicate with each other it results in bottlenecks and complexities due to a large amount of data. Raw data on the individual memory chips can be preprocessed to enhance the performance of the memory subsystem which also increases the actual bandwidth. [2] In the HMC memory modules, the connectivity is good as it has symmetric connections in X-axis and Y-axis which results in the low diameter of the data packets. The HMC network can be easily extended by adding switches between the interface as shown in Fig. 11. The Memgrid topology has dedicated paths which form a mesh network. By using Memgrid, there is a freedom for the CPU when trying to access an HMC module. The CPU can either choose the HMC cubic paths or the inter-CPU mesh paths. Thus depending on the distance and congestion, a suitable path is selected. [2]

B. Network switch architecture

The overview of the architecture is shown in Fig. 12. The HMC module is interfaced via Memgrid HMC switch to other HMC modules and CPUs. The data packets generated by the CPU are forwarded via the network switch further to their corresponding vault controller which control the DRAM-layer banks. To facilitate efficient routing the Memgrid HMC Switch is equipped multiple channels to connect to other HMC modules and CPUs [2]. Each individual channels consists of an Input queue, output queue and Switch Routing Unit. Between the queues, the Internal network is implemented to arbitrate and forward packets to the output queue. The Switch Routing Unit by a proper routing algorithm determines the target output queue after accepting requests from an input queue.

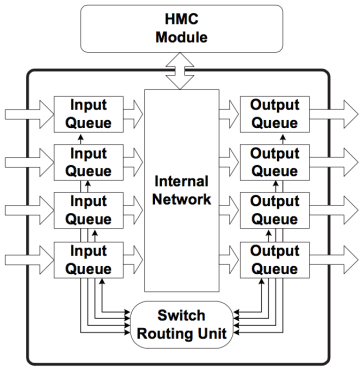

Fig. 12. The architecture of Memgrid HMC switch. [2]

VI. PIM accelerators

A. Data Structure Rearrangement

Irregular data-intensive applications need not require all the data stored in the memory. Instead, they only require accessing certain memory locations. The time required to iterate over the unnecessary elements is can induce unnecessary delay. To reduce this type of delay a new hardware implementation Data Rearrangement Engine (DRE) can be included in PIM [7]. However, this type of hardware can be exploited only if the application needs to access random data patterns. This is suitable for server databases. Applications that manipulate complex, linked data structure benefit less with this technique but can benefit the data-intensive applications with a little spatial/ temporal access locality. A good example of such application is extracting a reduced resolution image from full resolution. In such type of a memory access, the application has to access every 4 pixels which waste the memory bandwidth.

To save the memory bandwidth the data can be rearranged to match the processing needs and can be accomplished by the DRE. The application is interfaced to the hardware in the base logic layer of a PIM to traverse and reorder data structures by using DRE [7]. To perform the operation a strided (long) DMA units are used to gather the scattered data over different memory locations. The aim is to provide data to all the cores to keep them occupied to maximize the data accesses. The DRE consists of three elements as shown in Fig. 13. [7]

View buffer: SRAM/DRAM buffer is used to temporarily store the data on the DRE.
Data Mover: It moves the data from the DRAM banks to the View buffer.
Control Processor: It accepts the commands from the CPU.

When an application sends a request to the CPU the CPU commands the Control Processor. The Control Processor then issues commands to Data Mover. Thereafter, the Data Mover moves/ gathers the data into the View Buffers. Both the CPU and DRE can read/write the view buffer. When the Control Processor receives a command from the CPU it in

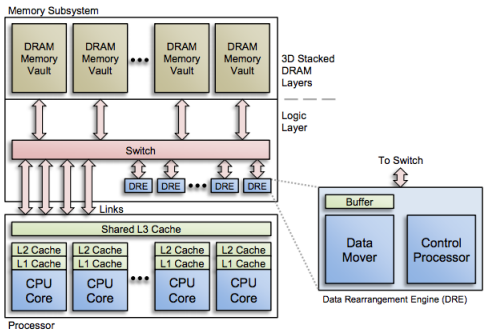

Fig. 13. Data Rearrangement Engine overview. [7]

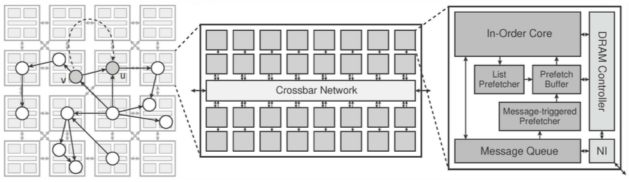

Fig. 14. Tesseract Architecture. [11]

turn issues commands to the DMA engine to initiate the data transfer and command the Data Mover. The application has to provide a Control Processor program to be written in the Control Processor instruction memory which resides in the logic layer. After loading the program the application can communicate to the program through Memory mapped address range. The commands sent by an application are:
Setup: Initializes the base addresses to access the random memory patterns.
Fill: Copy contents from the DRAM to View Buffers
Drain: Store the contents from View buffer back to DRAM after processing on them.

B. Tesseract

Nowadays, graph data structure, as an algorithm, is applied for many applications. While it may be true that graph processing algorithms can be parallelized, there are some problems that affect the efficiency which is challenging due to the large number of random memory accesses. Adding more cores along with HMC can improve the speed up but at the same time, it leads to more bandwidth usage. One way to overcome the need for high bandwidth problem is to move the computation unit inside the memory, or processing-in-memory is considered which is called Tesseract. Tesseract was the design that uses PIM concept for large-scale graph processing which was a new hardware that fully uses all available memory bandwidth and improves the efficiency of the communication between different memory partitions [11]. Fig. 14 shows the overall system that uses Tesseract architecture which used HMC as a base-line.

Each HMC has eight 8 Gb DRAM layers of memory and 32 vaults in total with 40 GB/s serial link and a crossbar network [11]. For each vaults memory layer, there are multiple DRAMs stacked, 16-bank DRAM partition and

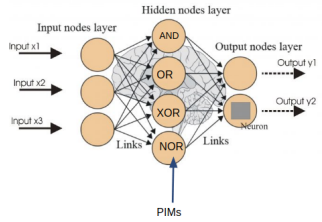

Fig. 15. Artificial Neural Network. [16]

memory controller [11]. Also, for each vaults logic base layer, there is a core which used Tesseract architecture [11]. The Tesseract architecture has its own own main memory. The difference in terms of the logic base between the conventional HMC and Tesseract is that the Tesseract has in-order core, and a prefetch buffer. The prefetcher buffer is the component that stores the block that already prefetched instead of store in L1, in order to lower the accessing time [11]. The 2 types of prefetcher are List-Prefetcher, and Message-triggered Prefetcher [11], which can handle sequential accesses pattern and random accesses pattern, respectively. List-Prefetcher (LP) is a modified stride prefetcher which the multiple cache blocks can be prefetched ahead to exploit the high memory bandwidth [11]. According to the interface of this approach program, there is a for-loop which provide some information [11]. The prefetcher will get the information of the start address, the size, and the stride of the application software. Those will be recorded in the four-entry list table at the beginning of a program loop and will be removed when the loop ends [11]. The prefetcher will keep track on only the memory region and do prefetching. Message-triggered Prefetcher (MTP), is to prefetch data that is needed before the execution [11]. In the random memory access, the LP cannot be applied since it could be the situation with different strides and difficult to predict. This is the reason that MTP is needed which can do the prediction based on the hints of the program [11]. Consequently, one of the special features of MTP is that the processor stalls from memory accesses can be eliminated which can lead to the improvement of performance.

VII. Applications of PIM

A. Application in Artificial Neural Networks (ANNs)

Some data-intensive application like ANN require days of offline training to understand the network behavior. The bandwidths supported in current GPGPUs cannot satisfy the requirement of ANNs as a huge amount of data is continuously traversing back and forth from the memory to the CPU. By using SIMD along with VLIW alone increases the speedup by 10x. This is usually done in many GPPs. But the algorithms implemented in ANNs are constantly learning and changing for improvements especially in the nonsupervised type ANN. These algorithms are implemented usually over a bulk data like audio or video processing. Thus using PIM in this application will provide a great advantage of raw processing of the data.

The Bottlenecks in the parallel execution make the loops non-parallelizable due to which loops the functions cannot interact with the SIMD engine well. This usually happens because of the functions having per iteration divergence, inter-iteration dependencies. The issues like the inter-iteration loop dependencies can be resolved by adding special function units (SFUs) within the PEs by PIMs as shown in Fig. 15.

If the SIMD width is extended beyond an optimal point, the speedup and energy efficiency will be adversely affected. This occurs due to the overhead of large vectors held in non-contiguous addresses in memory. If there happens to be divergence or dependencies in the chain on large vectors then this overhead grows. If there are certain redundant chains of large vectors but the energy efficiency of a useful operation is low then by using the only SIMD will have a lot of overhead. Thus, implementation of PIM would enable the operations over 100s to 1000s of operations per call giving the best possible energy efficiency.

By using PIMs a lot of memory bandwidth limitations and data movement latencies are resolved. Thus by introducing PIMs large matrices can be arbitrarily computed and the raw bandwidth in the individual memory bank can be exploited. Also, the amount of energy consumption is reduced as the data movement is reduced over the bus but the CPUs will consume some amount of energy required to process the data. The bus bandwidth can be used for the useful operation like IO requests or processing over the results obtained from PIMs.

B. Processing in-flash (PIF)

The HDDs are rapidly being replaced by SSDs due to no seek time, reads data very fast. Unlike the flash memories the hardware inside the SSD is very complex. If flash memories are used alone without a special processor used in SSDs the performance of the memory may degrade. However, SSDs have complex hardware along with the flash memory. It can be thought of as a brain to the flash memory. The logic inside the existing SSD architecture include a microprocessor which has Flash Translation Layer(FTL) and the flash memory banks.

SSD architecture with the Processor In-Flash (PIF)

As shown in the Fig. 16, the existing SSD architecture can be modified by adding one PIMF (Processing In-Memory-Flash) and Processing In-Flash(s) (PIFs). In the PIMF, the Microprocessor can be implemented in the logic layer of HMC and the HMC memory banks can be modified to accommodate both flash and DRAM (for data buffers). The processor will be responsible for performing the FTL tasks. Adding flash to the processor is necessary to save the states/offsets of other PIMs and flash.

The PIFs consist of the processor in the logic layer of HMC. The processor acts as a flash controller along with simple computational units. The HMC memory banks can be modified to accommodate flash memory instead of DRAM memory. The major advantage of using PIM in SSD is that a significant amount of the processor load is distributed towards the SSD due to the simple computational units

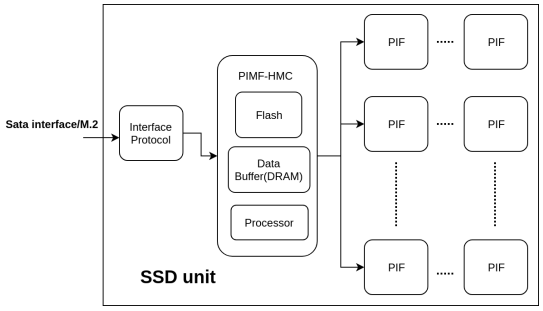

Fig. 16. SSD and PIM.

in PIFs. The implementations like PEI along with an API are required to modify the instruction set specific design to compute inside SSD. Additionally, to save the result of the computation extra memory locations needs to be specified for the CPU to fetch the results only. The PIF-PIF interconnect can be implemented by using MemGrid interconnection network topology to get maximum performance. Thus, a significant amount of CPU load can be balanced by implementing a PIM like architecture in modern SSDs.

VIII. Evaluation

Table I and II show the evaluation results of the researchers. The benchmarking tools used were NAS (Numerical Aerodynamic Simulation) Parallel Benchmarks developed at NASA Ames Research Center to study the performance of parallel computers and Open source Gem5 simulator with added logic to the RAMs to act as PIM respectively. The results show tremendous improvement in the performance with the use of PIMs. Various parameters like frequency, DRAM bandwidth, energy, power, etc., show improved results using PIM than just a single host processor. PIM implemented in different technology nodes like the 22nm and 16nm and their performance is highlighted. It can be seen that the PIM has outplayed the single host processor in every aspect. These results are promising and validated and thus gives a boost to the researchers for implementation of the PIMs in the modern and future computers.

TABLE I
Performance comparison Host vs 8-PIMs [1]

	Host alone	**8-PIMs**
Power	130 watts	1 watt
Frequency	900MHz	140MHz
Time	∼860us	∼90us
Speed-up	1x	8x

IX. Future Work

The concept of PIF and PIMF in SSD can be very useful but at the same time can be very complex considering the FTL tasks to be performed in the PIMF. The PIMF is a combination of processor, DRAM, and flash. There is a need to validate and determine the interconnection between processors, DRAM, and flash. The offsets to be stored in the PIMF need to be predefined to perform FTL tasks especially

TABLE II

PERFORMANCE COMPARISON HOST VS PIM FOR DIFFERENT TECHNOLOGIES [1]

Technology	**22nm**		**16nm**	
Host/PIM	**Host**	**PIM**	**Host**	**PIM**
Frequency	1GHz	650MHz	1GHz	650MHz
No. of CUs	32	8	64	12
No. of Memory Stacks	2		4	
DRAM BW(GB/s)	160	640	160	640
Dynamic Power Scaling	0.61	0.25	0.41	0.17
Memory Energy(pJ/64b)	522	159	520	155

for wear leveling. Also, the processing elements in the memory side are required to be associated with a particular chunk of memory accoding to the Garbage Collection algorithm. The other challenges include the fabrication of this device due to its complex construction. To validate the performance of PIMF and PIF in SSD full system simulation and the system call simulations are essential. To accomplish this a SimObject for PIMF and PIF can be created as per the specifications in gem5 (a computer architecture simulator).

X. CONCLUSION

Thus, PIM is a powerful technology which can revolutionize the future of the computers. It has shown encouraging results by credible researchers and companies. PIM implementation will aid to reduce the access latency, decrease the energy consumption, increase the memory bandwidth, and fasten the processing speed. However, there is a need to standardize the PIM architecture so that it can be developed and used on a large scale while offering compatibility to all types of computers as well as servers. Instead of each business unit coming up with new PIM implementation, they should collaborate and work jointly on this project and develop the PIM technology at a standard level. To conclude, it can be said that PIM is going to be the future of modern day computing.

REFERENCES

[1] Dong Ping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph Greathouse, Mitesh Meswani, Mark Nutter, Mike Ignatowski.(June 2013) A New Perspective on Processing-in-memory Architecture Design, Research Group, AMD, California, USA, ACM 978-1-4503-1219-6/12/06.

[2] S. L. Chu, W. C. Ho, C. F. Chen, K. W. Ceng and M. H. Liu, "Design a Novel Memory Network for Processor-in-Memory Architectures," 2017 13th International Conference on Semantics, Knowledge and Grids (SKG), Beijing, China, 2017, pp. 56-61. doi: 10.1109/SKG.2017.00018

[3] J. Ahn, S. Yoo, O. Mutlu and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, OR, 2015, pp. 336-348. doi: 10.1145/2749469.2750385

[4] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," 2012 Symposium on VLSI Technology (VLSIT), Honolulu, HI, 2012, pp. 87-88. doi: 10.1109/VLSIT.2012.6242474

[5] T. Barrett, S. Mediratta, T.-J. Kwon, R. Singh, S. Chandra, J. Sondeen, and J. Draper, A double-data rate (DDR) processing-in-memory (PIM) device with wideword floating-point capability, 2006 IEEE International Symposium on Circuits and Systems.

[6] J. Torrellas, FlexRAM: Toward an advanced Intelligent Memory system: A retrospective paper, 2012 IEEE 30th International Conference on Computer Design (ICCD), 2012.

[7] M. Gokhale, S. Lloyd, and C. Hajas, Near memory data structure rearrangement, Proceedings of the 2015 International Symposium on Memory Systems - MEMSYS 15, 2015.

[8] J. Schabel, L. Baker, S. Dey, W. Li, and P. D. Franzon, Processor-in-memory support for artificial neural networks, 2016 IEEE International Conference on Rebooting Computing (ICRC), 2016.

[9] SearchBusinessAnalytics. (2018). What is processing in memory (PIM)? - Definition from WhatIs.com. [online] Available at: http://searchbusinessanalytics.techtarget.com/definition/processing-in-memory-PIM [Accessed 11 Mar. 2018].

[10] E. Azarkhish, D. Rossi, I. Loi, L. Benini, Design and Evaluation of a Processing-in-Memory Architecture for the Smart Memory Cube, Cham, Switzerland:Springer, pp. 19-31, 2016.

[11] J. Ahn, S. Hong, S. Yoo, O. Mutlu and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, OR, 2015, pp. 105-117.

[12] Eckert, Yasuko ,Nuwan Jayasena, and Gabriel H. Loh Thermal Feasibility of Die-Stacked Processing in Memory. AMD Research; Advanced Micro Devices, Inc, 2014.

[13] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, Intelligent RAM (IRAM): chips that remember and compute, 1997 IEEE International Solids-State Circuits Conference. Digest of Technical Papers.

[14] J. Luo, L. Fan, Z. Chen, and Z. Li, A solid state drive architecture with memory card modules, IEEE Transactions on Consumer Electronics, vol. 62, no. 1, pp. 1722, 2016.

[15] Wikimedia Commons contributors, "File:Von Neumann Architecture.svg," Wikimedia Commons, the free media repository, $https://commons.wikimedia.org/w/index.php?title=File:Von_Neumann_Architecture.svg\&oldid=262902966$ (accessed March 18, 2018).

[16] Syed, Ali Danish, and Rahul Ahuja. The Evolution and Core Concepts of Deep Learning & Neural Networks. Analytics Vidhya, 3 Aug. 2016, www.analyticsvidhya.com/blog/2016/08/evolution-core-concepts-deep-learning-neural-networks/.