PM^3: Power Modeling and Power Management for Processing-in-Memory

Chao Zhang, Tong Meng, Guangyu Sun
Center for Energy-efficient Computing and Applications, Peking University, Beijing, 100871, China
{zhang.chao, mengtong, gsun}@pku.edu.cn

ABSTRACT

Processing-in-Memory (PIM) has been proposed as a solution to accelerate data-intensive applications, such as real-time Big Data processing and neural networks. The acceleration of data processing using a PIM relies on its high internal memory bandwidth, which always comes with the cost of high power consumption. Consequently, it is important to have a comprehensive quantitative study of the power modeling and power management for such PIM architectures.

In this work, we first model the relationship between the power consumption and the internal bandwidth of PIM. This model not only provides a guidance for PIM designs but also demonstrates the potential of power management via bandwidth throttling. Based on bandwidth throttling, we propose three techniques, *Power-Aware Subtask Throttling* (PAST), *Processing Unit Boost* (PUB), and *Power Sprinting* (PS), to improve the energy efficiency and performance.

In order to demonstrate the universality of the proposed methods, we applied them to two kinds of popular PIM designs. Evaluations show that the performance of PIM can be further improved if the power consumption is carefully controlled. Targeting at the same performance, the peak power consumption of HMC-based PIM can be reduced from 20W to 15W. The proposed power management schemes improve the speedup of prior RRAM-based PIM from 69× to 273×, after pushing the power usage from about 1W to 10W safely. The model also shows that emerging RRAM is more suitable for large processing-in-memory designs, due to its low power cost to store the data.

1. INTRODUCTION

Emerging data-centric applications, such as real-time analytics, graph computations, and neural network algorithms, have a demanding requirement for high speed/bandwidth data retrieval. However, traditional von Neumann computing architecture has its intrinsic limitation on data access due to the latency inherent in hierarchical memory and interconnect architecture [33, 43]. This challenge motivates the practice to offload part of the data-intensive computation into memory, to fully utilize the bandwidth provided by the memory arrays [13, 19, 25, 34, 46, 55].

The method of offloading computation into memory dates back to 1970's [25, 55, 66]. However, those practices were not successfully applied due to two reasons. First, it is difficult to integrate computation logic with DRAM due to incompatible fabrication processes. Second, the lack of data-centric killer applications also prevented a wide adoption of such techniques. Recently, the first problem has been overcome by the state-of-the-art memory stacking techniques [7, 57]. With the bloom of data-centric applications, the idea of offloading computation to memory is reviving with a new name: processing-in-memory (PIM). Recently, PIM designs have been widely studied to accelerate data intensive applications [3, 4, 13, 20, 34, 59, 77, 78]. Computation tasks (e.g. Word Count, Range Find etc.) of PIM are normally simple but involve huge amounts or even all of the data in memory. Consequently, a PIM task can be split and offloaded into multiple memory-side processing units (e.g. HMC vaults and RRAM crossbar arrays) and accomplished locally in parallel.

It can utilize the bank-level or even cell-level bandwidth more efficiently so that data processing is accelerated significantly. The increased bandwidth utilization comes with the cost of the increased power consumption. Prior research has pointed out the concerns about high power consumption in PIM designs [18, 38, 40, 42, 64]. For example, Eckert *et al.* [18] has shown that cooling systems used by memory should be redesigned to dissipate increased power consumption. It induces extra cost of heat sink and design complexity.

However, the relationship between power consumption and data processing throughput in PIM is not yet sufficiently investigated. Without the guidance of a proper power model, a PIM architecture may be designed with mismatch between the internal memory bandwidth and power supply.

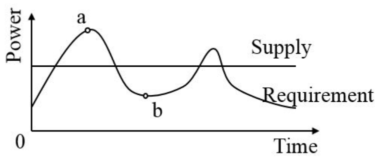

Figure 1: The mismatches between power requirement and supply in PIM

In fact, the power consumption at the ideal peak throughput in previous PIM designs [3, 59] may exceed its power supply. For example, our analysis shows that the HMC-based PIM in previous work [3] can consume up to 26W power, which is more than 2X of the released number (12W) in the real HMC chip [57]. And its power density can be as large as $177 mW/mm^2$, which is beyond the DRAM thermal tolerance capability with passive heat sink [18]. It means that a PIM working at its peak performance has potential power supply failure and memory reliability problems. This is illustrated with point (a) in Figure 1. Point (b) in Figure 1 demonstrates another common case in current PIM design, when data processing and corresponding memory accesses are not intensive. Obviously, the power supply is not fully utilized in this case. The problems in these two cases become more severe when the power supply of memory is dynamically

This work is supported by NSFC No.61572045.

adapted by modern power management techniques [64].

Considering power issues in these two cases, two challenges should be addressed so that the PIM designs can be practically feasible. First, a model of the relationship between power consumption and data processing throughput is necessary to help the early stage design of a PIM power system. Second, proper power management techniques are needed to achieve better power utilization in a PIM design. Both of them are addressed in this work, the contributions of which are summarized as follows:

- We model the relationship between bandwidth and power consumption, to facilitate early stage PIM power system designs.
- We propose the *Power-Aware Subtask Throttling* (PAST) technique to reduce PIM power requirement.
- We propose the *Processing-Unit Boost* (PUB) technique based on a greedy algorithm, to improve the performance of processing units.
- Furthermore, we propose the *Power Sprinting* (PS) technique to improve the energy efficiency when the power supply is dynamically adapted.
- In order to show the universality of the proposed techniques, we evaluate both HMC-based and RRAM-based PIM designs for their targeted applications.
- Our BP model and evaluations demonstrate that NVMs and 3D DRAM have their own advantages in different bandwidth usage cases to be selected as PIM memories.

The paper is organized as follows. Section 2 introduces the background for following discussion. We model the relationship between data processing throughput and power consumption in Section 3. The power management techniques are proposed in Section 4. Our evaluation and sensitivity analysis are included in Section 5. Related work and conclusions are presented at the end.

2. PRELIMINARIES

Thanks to the recent revolutionary progress in memory technologies, processing-in-memory has revived. However, existing power management techniques are mainly focusing on conventional processors and main memories, which cannot be straightforwardly applied to the memory used in processing-in-memory designs. This section gives a brief introduction to these related topics.

2.1 3D Stacked Memory and Logic

Combing DRAM dies together with standard CMOS dies via 3D stacking and through silicon via (TSV) has become a current trend in the memory industry. Representative examples include hybrid memory cube (HMC) [14] and high bandwidth memory (HBM) [31]. These representatives are capable of combining up to eight DRAM dies with one logic layer to construct a cube of stacked memory and logic. The logic die is implemented in standard CMOS technology, which is different from the SDRAM optimized dies. TSVs are used to connect and transmit data through these layers. These techniques enable higher memory bandwidth and more flexible design of control logic, compared with conventional DDRx DRAM. Thus, they become perfect platforms for processing-in-memory designs.

HMC is designed to provide data links mainly for CPUs. It uses high-speed links as connections. Serial links and high speed SerDes circuits are used to transmit data to and from the HMC cube. The HMC has been used by the Fujitsu SPARC64 XIfx processor [74]. Specified by the HMC 1.0 specification [14], each HMC cube can contain 4-8 DRAM dies, and 1 logic layer. A cube is connected by 4-8 high speed links, which has 16 lanes to form a full-duplex differential serial link. Each lane can provide up to 10Gb/s data rate, which enables up to 320GB/s bandwidth (send + receive) for each cube. A typical cube package with 4 links has 896 BGA pins and a 960 mm^2 package footprint (227mm^2 die footprint). According to the data released [57], a 4-link HMC cube consumes 11W power under full utilization of its 128GB/s IO bandwidth.

The HBM connects to the CPU or GPU via the silicon interposer. Several stacks of HBM are plugged onto the interposer besides the CPU or GPU, and the assembled module connects to the circuit board [7]. The first chip utilizing HBM is AMD Fiji which was released in June 2015, powering the AMD Radeon R9 Fury X [65]. According to its specification, HBM1.0 stacks 4 DRAM dies with two 128-bit channels per die on a base logic die. Each channel supports 1Gb capacity, features 8 banks and can operate at 1Gb/s data rate, enabling 1GB capacity and 128GB/s IO bandwidth of one HBM stack. The access energy of a stack is 6-7pJ/bit, and the power consumption for 128GB/s is 3.7W [69].

Delivering a large amount of power to a such a highly compact structure is challenging: On one hand, since more chips are folded together, the number of power pins is limited by a smaller footprint, compared with DDRx SDRAM chips. The limitation to the power supply becomes increasingly severe if more chips are stacked. Since the power pins have already dominated the total number of pins, increasing the number of power pins will significantly increase the cost of the cube [64]. On the other hand, stacking multiple dies causes thermal challenge. Since the thermal tolerance of the SDRAM is much lower than CMOS, the temperature roof is lower than that of processors, which leads to more harsh thermal control. Eckert et al. [18] have shown that if a passive heat sink is used, the maximum power of an HMC cube should be as low as 10W; otherwise, the DRAM has to be refreshed in double rate and the error rate may increase.

There are also other forms of stacked memory. In order to simplify our discussion, we take the HMC as a representative of stacked memory in following sections.

2.2 Processing-in-Memory

Nowadays, processing-in-memory (PIM) acceleration solutions have been proposed for neural networks [13, 62], graph processing [3], Big Data processing [59], real-time analytics [28], sparse matrix multiplication [78], and in-memory databases [47]. Instead of unnecessarily moving all data into the processors, PIM improves the performance by better utilizing the memory internal bandwidth. The memory bandwidth information are summarized in Table 1. The internal bandwidth refers to the bank level bandwidth which can be leveraged by these PIM designs. The CPU-MEM interface

Table 1: The memory configurations used by recent processing-in-memory proposals

Design Name	Size	Memory Type	Configuration	CPU-MEM BW	Internal BW	Used Internal BW
Tesseract [3]	128GB	HMC	dragonfly [39], 16 HMC cubes, 8 4Gb-DRAM/cube	640.0GB/s	8.0TB/s	3.7TB/s
NDC [59]	256GB	HMC	daisy-chain, 64 HMC cubes, 8 4Gb-DRAM/cube	320.0GB/s	8.0TB/s	3.4TB/s
PRIME [13]	16GB	RRAM	533MHz IO bus, 8 chips/rank, 8 banks/chip	8.5GB/s	275.2GB/s	No show

bandwidth refers to the memory bus bandwidth used by the host processors.

Early PIM research [25] in the 90's integrated lightweight processors into the memory die, as illustrated in Figure 2(c). The idea was straightforward and effective, but it was criticized due to the fact that logic and DRAM process technologies are not compatible. Building processing elements in DRAM die results in significant area overhead, which is unlikely to be adopted by the cost-sensitive memory industry. However, with the development of technology and the increase in bandwidth demand from emerging applications, the PIM architecture is being studied again, and there are two approaches to address the previous logic-memory integration problem. These two approaches rely on the 3D die stacking technology and the emerging NVM technology, respectively. The **3D-based PIM** [3,13,62] takes advantage of the already existing architecture of decoupled memory and logic die (stacked with TSV) in HBM and HMC [63]. It designs processing elements in the logic-optimized dies and the memory in the memory-optimized dies, and combines them together via TSV, as illustrated in Figure 2(b). Besides 3D stacking, emerging NVM [71] provides an alternative for the capacity scaling. Emerging NVM technologies also provide another approaches to support PIM. The **NVM-based PIM** relies on the memory cell themselves to carry out the computing tasks. The NVM cell has some special features such as resistive cell and multi-level cell. Those features can be leveraged for logic operations, such as dot-product [13], bitwise operations [46] and TCAM [27], whose architectures are illustrated in Figure 2(a).

For either 3D-based PIM or NVM-based PIM, it has been pointed out that the performance improvement of PIM relies on the high internal bandwidth usage. However, the increased memory bandwidth inevitably leads to more power consumption, which must be managed properly.

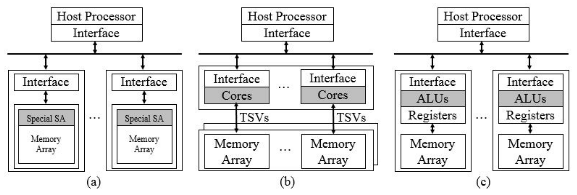

Figure 2: Typical PIM designs (a) Customized sense amplifiers and arrays are used for computation; (b) A die of cores is stacked with DRAM dies in the HMC-based PIM; (c) Integrating cores with DRAM in the same chip

2.3 Power Management Techniques

To manage power consumption of modern processors (CPU, GPU), dynamic voltage and frequency scaling (DVFS) and similar techniques have been used widely [24,29,51]. The basic idea is to reduce clock frequency or supply voltage of active cores if a power shortage is predicted to happen. Power consumption can be minimized at the cost of performance degradation [41,48,49,67]. These strategies can also help to achieve the maximal performance under a fixed power budget.

Besides controlling the power of cores, on-chip memory power can be managed by the activity control, via partially or completely turning off on-chip memories. To keep memory working at low power modes, Gated-V_{DD} [58] and Drowsy Cache [21] were proposed. Other methods turn off useless cache ways, sets, or their sub groups [5,9,12,35,53,68]. Cache replacement strategy can also be optimized to reduce power consumption contributed by on-chip caches [2,23,36,37,76].

However, there is few work discussing the memory power management of PIM, especially within the 3D-stacking memory scenario.

3. BP: BANDWIDTH PER POWER MODEL

The purpose of this model is to simplify the analysis for power-proportional performance, which is crucial for the scalability of processing in memory applications. The model estimates the PIM power consumption under various bandwidth, capacity and memory types. The term "Bandwidth per Power (BP)" is used to reflect the close-to-linear relationship between bandwidth and power consumption. In this section, we present the model derivation and parameters estimation for different memories. Further analysis of the BP model is shown in section 5.4.

3.1 Model Derivation

The bandwidth per power (BP) is calculated by B/P. The bandwidth usage of the memory is represented by B, and the denominator P represents the total power of the PIM component, including dynamic power (DP) and leakage power (LP). Without losing generality, a PIM system is considered as a set of processing units which is a pair of a core (computing logic) and a portion of memory. To facilitate the derivation, the symbols used are explained in Table 2.

Table 2: Parameters used in BP model

Symbol	Definition	Unit
BP	Bandwidth per Power (or watt)	Gb/s/W
DE	Dynamic energy per bit	J/b
C	Chip capacity	b
B	Used bandwidth of the chip	Gb/s
P	Total power of the chip	W
P_l	Leakage power per bit	W/b
P_c	Leakage power of core and memory controller	W
e_r	Dynamic energy used by routing for a bit	J/b
e_s	Dynamic energy to switch a bit	J/b
e_c	Dynamic energy for computing per bit	J/b
r_w	The ratio of write access within the bandwidth	-

The BP model is deducted in Equation (1). The bandwidth is measured by how many bits are read or written within

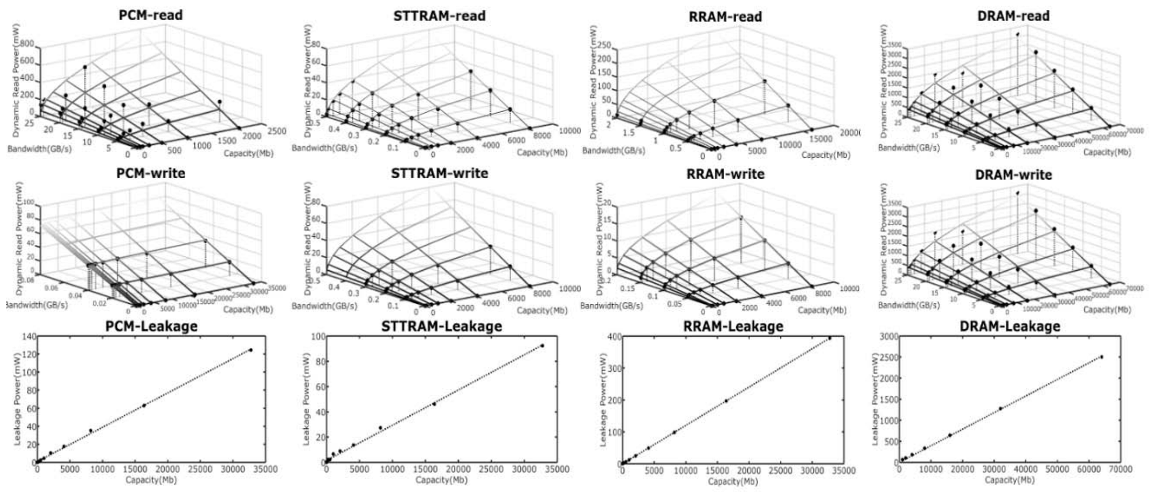

Figure 3: Validating the BP model with collected data points and the predicted dynamic power (DP) and leakage power(LP) by the BP model. (a)-(d) represent the data of PCM, STT-RAM, RRAM, and 3D DRAM, respectively.

a period of time. Since the core can collocate with a portion of memory in many PIM designs, the bandwidth used by the core can be higher than the package I/O bandwidth, but smaller than the aggregated cell-level bandwidth. The aggregated cell-level bandwidth is the theoretical maximum bandwidth, where all cells are accessed continuously in parallel. The power includes dynamic and leakage power of both memory arrays and cores. The leakage power includes the leakage power consumed to hold the data, including the power to refresh cells, the power to keep decoders and computing logic active, and the power leaked through parasitic paths. The dynamic power refers to the power used to conduct computation on fetched data and to access (read and write) the data, including cell activating, word-line driving, and sense amplifying etc.

$$BP = \frac{B}{DP+LP} = \frac{1}{\sqrt{C}e_r + r_w e_s + e_c + \frac{1}{B}(CP_l + P_c)} \quad (1)$$

The dynamic energy used by the PIM normalized to one bit is estimated in Equation (1) by $\sqrt{C}e_r + r_w e_s + e_c$. The write ratio ($r_w$) is 0 (or 1) if all accesses are read (or write). It is generally between 0 and 1. The $\sqrt{C}e_r$ term represents the energy used along the routing path to target cell, and thus is related to the capacity. The e_s is used to conduct resistance switch or state change in several memories, and thus is irrelevant to the capacity. The energy used to compute is represented by e_c.

The leakage power is represented by $CP_l + P_c$. The leakage power of memory is related to the capacity, which is represented by CP_l. The P_c represents the leakage power of the core and controllers of memory.

The model can be simplified when the B or C is relative large or small. If B is relative small, the BP is mainly determined by $CP_l + P_c$. If B is small and C is large, the BP is mainly determined by P_l. If the B is relative large, the power of a memory is roughly $(\sqrt{C}e_r + r_w e_s + e_c)B$, which is linear to the bandwidth.

The maximum bandwidth that can be achieved is determined by the position of the computation logic within the memory. If the memory is organized in a bank-array-cell hierarchy, the maximum available bandwidth reduces from the cell level to the package I/O level. For example, the theoretical maximum cell level read bandwidth of a 512MB RRAM chip is 27.26PB/s, but it only provides 1.18GB/s read bandwidth at the package I/O.

3.2 Parameter Estimation for the BP Model

In order to estimate the parameters in BP model, we regress the parameters using collected data. The parameters used for different memories are estimated in Table 3. Although there is a little variation between the intrinsic value and the regressed value of these parameters, the model still provides a meaningful explanations for the parameters.

The P_c and e_c are related to the implementation of computing logic, and thus independent with memory types. Previous HMC-based PIM designs [3, 59] estimated the power as 80mW/core. According to their bandwidth usage, we assume the P_c as $2 \times 10^{-2}W$ and the e_c as $5.3 \times 10^{-11} J/b$. Note that the number can be fine tuned after more detailed power data provided by practical PIM designs.

Table 3: BP model parameters for different memories

Memory Type	PCM	STT-RAM	RRAM	3D DRAM
e_r(J/b)	8.15×10^{-17}	2.10×10^{-16}	1.17×10^{-16}	5.90×10^{-17}
e_s(J/b)	1.13×10^{-10}	5.27×10^{-13}	7.52×10^{-13}	2.03×10^{-14}
P_l(W/b)	3.80×10^{-12}	2.79×10^{-12}	1.20×10^{-11}	3.94×10^{-11}

The data are collected from previous validated simulation tools and literatures. The power used by memory is validated by its dynamic energy and leakage power. We verify their relationship under different settings of capacity and bandwidth usage. The data of spin torque transfer random access memory (STT-RAM), phase change memory (PCM), and resistance random access memory (RRAM) are collected from NVsim, and the data of 3D stacked dynamic random memory (3D DRAM) are collected from cacti-3DD. NVsim is calibrated with published literatures [54] [70] [72], and scaled into the 32*nm* processing node for comparison. The data points collected from the tools and predicted value for different memories are shown in Figure 3. The average prediction error of power per bandwidth is 6% for RRAM, 9% for 3D DRAM, 19% for STT-RAM, and 39% for PCM. The error comes from the function bias and the inaccuracy of memory simulators, which are limited by the available data to build the model.

4. PIM POWER MANAGEMENT

Three collaborative techniques are proposed to manage the power of PIM designs. The **goal** is to make full use of the provided power, while specified power constraints are met. To this end, we propose PAST and PUB: PAST ensures that the power constraint is never violated, by throttling the bandwidth usage; PUB, on the other hand, boosts the performance when current data level parallelism is not enough to make the power supply fully utilized. Besides these two methods with a fixed power cap, we also propose a third technique called PS to dynamically adjust the power cap, which further improves the energy efficiency.

4.1 PAST: Power Aware Subtask Throttling

PAST is a method proposed to solve the problem that the power requirement of a PIM task may exceed the power supply constraint.

The PAST architecture is shown in Figure 4. It can work with all types of PIMs in Figure 2. The subsection 4.1.1 introduces the PAST from the aspect of entire memory system. The subsection 4.1.2 and 4.1.3 explain the design details of the PAST mechanism. The implementation issues are covered in subsection 4.1.4. In section 4.1.5, we further extend this architecture to reuse the resource of prior HMC-based PIMs (Figure 2(b)), to reduce overhead.

4.1.1 Two-Level Power Arbitration

The purpose of two-level power arbitration is to reduce the control intensity of a centralized controller while providing flexible power control for different memory chips or banks. Figure 4(a) shows the two-level power arbitration system. Connected by networks, each memory chip contains a private PAST component (zoomed in Figure 4(b)). These chips are augmented by a shared secondary level arbitrator (L2). Within each chip, an L1 power arbitrator controls its own memory banks using PAST. The system can be further extended using more levels.

The two-level arbitration system works similar to a two-level cache system. The L2 arbitrator holds the total power budget of the memory (the sum of all L1's budget). The L1 only holds the power value for its own chip. The power budget of an L1 can be increased (decreased) by acquiring (releasing) a portion of power from the L2. Note that the L1 power budget should not violate the thermal design power (TDP) designed for that memory chip, thus only a limit portion of power can be reallocated to the L1. The PAST component in each chip first consults its local arbitrator (L1) for power. If the power is enough, the arbitrator replies the processing units (PUs) to start its execution. Otherwise, the L1 arbitrator then consults the L2 arbitrator. The granularity of power budget exchange between the L1 and L2 is configured carefully to reduce both the excessive supply of power budget and the high frequency of power budget exchange.

4.1.2 PAST within a Memory Chip

Figure 4 (b) shows the design and interaction in the PAST component within a chip. It takes a request from the network connection, divides the task into multiple subtasks, stores them in the subtask queue, and responds to the request sender. We define a *task* as a successive period of computation conducted in the memory-side PUs, which is initiated by host and responds to the host. A task consists of several subtasks. A *subtask* is conducted by only one memory-side PU. The PU is a pair of memory array and processing unit representing a simple core or some mathematical computing logic, represented by a "memory bank" in Figure 4(b). If the entire memory has n PUs, there are at most n subtasks executing at the same time.

Before an execution phase of any memory bank, the subtask queue acquires the power permission from the L1 power arbitrator using an *ACQUIRE* (①) signal, with the amount of required power (P). The queue *ISSUE*s a subtask to a memory bank, and the bank can also *NEW* a subtask at the tail of the queue. If there is enough power to execute a new subtask, a *START* (②) signal is sent to the bank and starts its execution. Otherwise, the calculation in the bank will be paused. Then the arbitrator puts the request into a queue. The bank is not activated until the power budget is enough. After the entire task is accomplished by the bank, a *RELEASE* (③) signal is sent to the arbitrator to release the power allocated for that bank.

4.1.3 Reorder Subtask Queue

If all the subtasks are independent, such as in the case of TCAM [27], the only limitation to initiate a subtask is the power. An FIFO queue is used to hold all subtasks and decide whether enough power is available to issue a subtask. Each entry of the queue holds the commands of the subtask, which contains the target PU index and the estimated power consumption of that unit to execute it. The FIFO queue is implemented as the white blocks in Figure 4(c). If a subtask from the head of the queue is issued, a portion of power is allocated for it until it finishes. If the remaining power is not sufficient for the subtask at the head of the queue, the queue is stalled until the remaining power is enough. If any new subtasks are initiated by the running subtasks, they are added to the tail of the queue. The PIM task ends when the queue is empty and receives finish signals from all issued subtasks.

To support more general scenarios where subtasks are dependent on each other, such as the case in PRIME [13], a reorder subtask queue is proposed. The reorder queue is an extension to the previous FIFO queue solution, as shown in Figure 4(c) (gray blocks are extended). It extends the FIFO by adding more head and tail pointers and adding more fields to each entry, to note all multiple ongoing subtasks.

Each entry of the queue then has 5 fields: index (ID), subtask command (Command), dependency mask (Mask), power specification (Power), and status (S). An index is attached to each entry. A subtask command contains chunked function calls to the corresponding PU (or memory banks). The power specification is the expected power it needs to conduct the computation and data access within that PU. The status indicates whether this subtask is pending (PD), issued (IS), or completed (CP). Once a subtask enters the queue, its initial

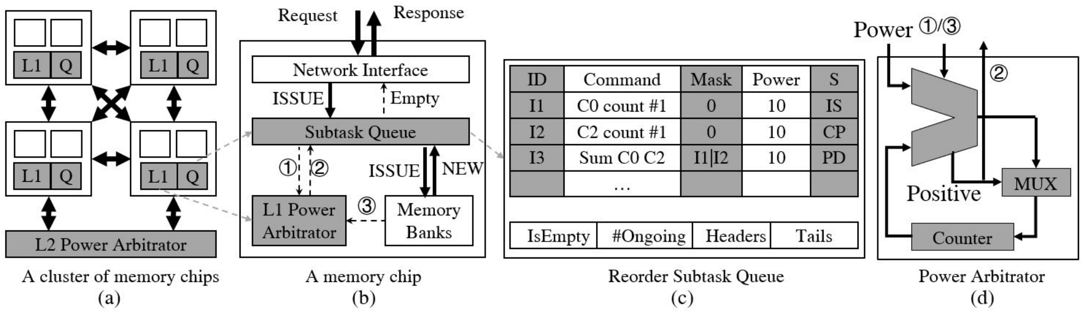

Figure 4: An overview of the *Power Aware Subtask Throttling* (PAST). (a) A 2-level power arbitration hierarchy; (b) Interaction within a chip; (c) Reorder subtask queue; (d) Implementation of a power arbitrator.

status is pending. It can only be issued when two conditions are both satisfied: (1) All its existing dependent entries have been completed. (2) The power requirement can be satisfied. Once the queue receives a finish signal from the corresponding PU, the status is changed to completed. The head entry with completed status will be retired and the space is freed for incoming subtasks. Same as the basic FIFO implementation, an extra counter is used to count the number of ongoing subtasks. If all heads are full with pending or issued subtasks, the queue stalls to ensure fairness.

The dependency mask is the OR'ed result of all its dependent entries' indexes. The dependent entry can be retrieved if the index of that entry equals the OR'ed result of the entry index and the dependency mask. This simple mechanism helps to reduce area overhead. The false detection of dependency can be mitigated using more bits for the index.

4.1.4 Hardware Implementations

Both L1 and L2 power arbitrators are implemented by a simple integer ALU, a register, and several MUXes in hardware. The diagram is shown in Figure 4(d). A counter is used to record the available power currently controlled by this power arbitrator. The value in the counter is subtracted by the given power value. If the result is positive, a *START* can be sent, and the power value in the counter is updated.

If the subtask queue is too short, the number of ongoing subtasks will be limited, but the overhead increases with the length of the queue. Thus, the length of the queue should be properly selected. Based on our evaluation, a 16-vault HMC cube needs a 32-entry queue, with 2 bytes for each entry to store power value and PU index. It has two 5-bit pointers for the head and tail. For the reorder queue, we double the number of entries and add extra 12 bits per entry (5 bits for the index, 5 bits for the mask, and 2 bits for the status).

The minimum latency to issue a subtask is the latency of the power arbitrator. However, if too much power is drained in a short period, the supply voltage suffers from a transit voltage drop, which may cause power failure. We avoid this by limiting the switch slope within $7.8mW/us$ as previous work [60].

We estimate the hardware cost via FPGA implementation. Using Vivado HLS v2015.4, we synthesized the FIFO, the reorder queue, and the power arbitrator on ZC702 FPGA board. They only occupy 80 LUT, 40 FF and 1 BRAM, which processes less then 0.5% of the chip. Both the power arbitrator and the queue achieve a short latency smaller than $0.5ns$. We take $1ns$ as a conservative estimation of the power arbitration system latency.

4.1.5 Extending PAST for HMC-based PIM

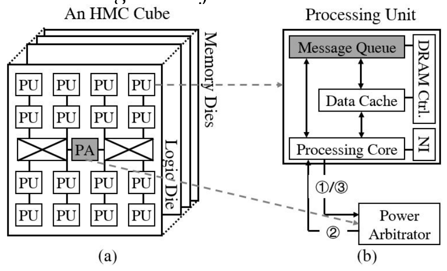

Figure 5: (a) An HMC cube with a power arbitrator, (b) Communication between power arbitrator and a PU used by Tesseract [3]

The basic PAST architecture in Figure 4 can be applied to different PIM design, such as PRIME [13], Pinatubo [46] and TCAM [27]. It can be applied without modifying the existing PIM designs. However, the PAST can also be tailored to leverage existing resources in HMC-based PIM designs, with reduced design overhead. Based on PIM designs [3, 4, 59] using lightweight cores as computing logic, the message queues existing in these cores can be taken to form a distributed reorder queue mentioned in the prior subsection.

The overview and interaction in this situation are shown in Figure 5. An HMC cube containing multiple PUs is shown in Figure 5(a), and the details of the PU are shown in (b). The crossbar network within the HMC cube is augmented by a power arbitrator. The subtasks are generated either by these PIM cores or by the host side controllers, residing in the message queues in the core. Different from the centralized reorder queue in subsection 4, the message queue in each PU only holds the subtasks allocated for the collocated PU. The generated inter-PU subtasks will be forwarded to targeted PUs' message queues, instead of stored in a centralized reorder queue. When the core is ready to execute a segment of PIM instructions inside the queue, it sends the *ACQUIRE* signal to the power arbitrator. A *START* response then starts the execution of the core. After a subtask is finished, the *RELEASE* signal is sent to the arbitrator.

In order to ensure the fairness among subtasks, the power arbitrator needs to record the failed power acquisition. Once the power arbitrator decides not to send a *START* to a PU, the index of the unit will be recorded to the tail of a pending subtask queue. Once a new *ACQUIRE* comes, the arbitrator first checks the head of the queue and decides whether to send a *START* to its corresponding unit. The item will be removed from the queue if a *START* is sent, otherwise the queue stalls, waiting for improved available power.

Compared with the situation in Figure 1, the power emergency is eliminated after PAST is applied, as shown in Figure 6.

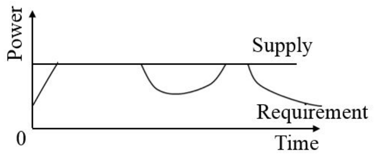

Figure 6: The relationship between power requirement and supply after PAST

4.2 PUB: Processing Unit Boost

Although the PAST has solved the power emergency problem, the problem of underutilized power supply is not well handled. The power is under utilized because no enough PUs can work together, limited by the dependency among subtasks. Based on PAST, processing unit boost (PUB) is designed to boost the performance of these subtasks in critical paths, via dynamically adjusting the power mode of PUs.

Our baseline PUs contain two power modes[1]: active mode and boost mode. The PUB can be taken as a kind of dynamic voltage and frequency scaling (DVFS) design, leveraging the characteristics of PIM designs.

4.2.1 Simple PUB

The goal of PUB is to assign the power modes for the PUs within PIM. Thus the key of the design is the scheduling algorithm. We start with a simple algorithm for PUB, which helps to understand the algorithm and also provides a simple way to implement it.

The simple algorithm issues only one subtask at a time: If the index of a PU is not in the queue, which means it will not be used, the unit will be put into the active mode. Once a queue entry is added, the power mode of the related PU will be upgraded. Then the power arbitrator evaluates the current remaining power with the required power. Scanning from the highest power mode to the lowest mode, if the free power is larger than the power for this mode, the PU is started with this power mode. If the PU cannot be started, the power arbitrator slowdowns current running PU from high power modes to lower modes. If the PU is still unable to start, the queue halts for enough free power.

However, this single-issue algorithm may cause frequent mode transition, which is inefficient. This problem will be addressed by the advanced PUB in next subsection.

4.2.2 Advanced PUB

The advanced PUB is a greedy algorithm for power arbitrator, based on the direct acyclic graph (DAG) of the subtasks. The algorithm works as a three-state finite state machine (FSM): *READY*, *UPDATE*, and *CHECK*. The initialization algorithm (Algorithm 1) puts the FSM into the *READY* state. If any subtask finishes, the *UPDATE* state is triggered and Algorithm 3 is executed to update the graph and the counter for current available power, and returns back to the *READY* state. If any update happens, the state is transferred to the *CHECK* state, and the Algorithm 2 is executed to determine the power modes of subtasks that will be issued. Note that if a subtask finishes within the *CHECK* state, the state transition to *UPDATE* will be triggered after the state changes back to the *READY* state.

[1]Without loss of generality, we take two modes as an example. More modes can be easily adopted and further improve the power efficiency.

Algorithm 1 Initiate the graph and free nodes

```
Require: $P_{remain}$, $P_{max}$ and $Nodes_{free}$.
1: for $node \in Nodes$ do
2:     if $node$.outEdges == 0 then
3:         $Nodes_{free}.add(node)$
4:     end if
5: end for
6: $P_{remain} \leftarrow P_{max}$
7: get $Edges$ from the $Nodes$
```

Algorithm 2 Adjust power modes for PUs

```
Require: $P_{remain}$, $Edges$, $Nodes_{free}$ and $PModes$.
1: Sort $Nodes_{free}$ by its $inEdge$ count from large to small
2: $PowerStates \leftarrow$ vector with length of $Nodes_{free}$
3: $P_{use} \leftarrow 0$
4: for $l \in PModes$ do
5:     for $node \in Nodes_{free}$ do
6:         if $node.PowerAt(l) - node.PowerAt(l-1) < P_{remain} - P_{use}$ then
7:             $PowerStates[node]$ upgrade by one step
8:             $P_{use} += node.PowerAt(l) - node.PowerAt(l-1)$
9:         else
10:            $P_{remain} -= P_{use}$
11:            return $PowerStates$
12:        end if
13:    end for
14: end for
15: $P_{remain} -= P_{use}$
16: return $PowerStates$
```

Algorithm 3 Update the graph after each subtask

```
Require: $Node_{end}$, $P_{remain}$ and $Nodes_{free}$.
1: for $node \in Node_{end}.inEdges$ do
2:     $node$.outEdge.remove($Node_{end}$)
3:     if $node$.outEdges == 0 then
4:         $Nodes_{free}.add(node)$
5:     end if
6: end for
7: $P_{remain} += Node_{end}.Power$
8: delete $Node_{end}$
```

A simplified example is given in Figure 7. The PIM task is divided into 7 subtasks (A-G) which form a DAG. Arrows represent the dependency: C points to A means A should be finished before C. Two power modes can be assigned to each subtask, including active and boost, indexed by (0, 1 in *PModes*). The voltage of a PU in boost mode is 1.5× of the active mode voltage. The power consumption of boost is roughly 2× of the active, and the latency is reduced. We estimate the latency in active mode is roughly 1.5× of the boost mode. In this example, the power cap (P_{max}) is 3. The time unit of each subtask is T. We normalize the PU power of active mode as 1.

The algorithm is executed as follows: It finds 2 free nodes (no predecessor) initially. The node with more decedents (B) is upgraded (from active to boost) in the *PowerStates*. Since the power cap is 3, A can only be upgraded to active. Thus, A and B are issued with *PowerStates* set as [active, boost].

After B finishes, the P_{remain} is updated to 2, and another 2 nodes (D and E) are set free. No free power lefts, after the $PowerStates$ of these two nodes are set to active. Thus, they are issued as [active, active]. After the execution of D and E, only one node F is free. Thus the $PowerStates$ is set to [boost] and F is issued. When C finishes, no free node can be found, thus it waits to the end of F. Then the G is issued with the highest $PModes$: boost. the whole PIM execution finishes after G is done.

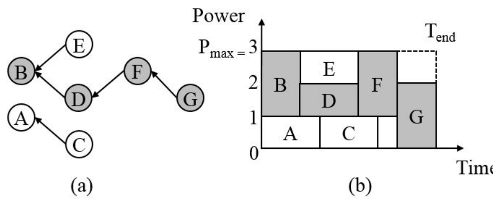

Figure 7: An example of advanced PUB. (a) The DAG of the subtasks (gray nodes are in the critical path), (b) The power consumption during the execution

4.2.3 Power Modes and Controller Cost

The PU to boost can be either memory arrays or cores. However, the exact speedups after boosting power for core and memory are different, according to different memory types and the implementations of the computation logic. The latency of the logic cores can be controlled by scaling the frequency. For the DRAM, the memory latency can be adjusted by the frequency of data bus; but for the RRAM, the latency can also be adjusted by slightly changing the supply voltage to the memory cells. Prior work [73] has shown the trade-off between the supply voltage and the response latency with RRAM. An RRAM cell's switch time has an inversely exponentially relationship with the voltage applied on the cell [26, 44, 75]. It has been demonstrated that, for a HfOx-based ReRAM, a 0.4V reduction in RESET voltage may increase RESET latency by 10× [26].

The voltage and frequency configurations of boost power modes are based on their amplification to the lowest mode. The voltage is $\sqrt{N}V_{low}$, when the mode power is targeting at N times of the lowest power mode, where V_{low} is the voltage of the lowest power mode. If the PU of a PIM is a core, the frequency of the core is set to $\sqrt{N}F_{low}$. If the PU is a memory array, the increase of voltage leads to a reduction of memory response cycles. Further evaluation of the latency and power consumption is conducted in section 5.

The Algorithm 2 and 3 are implemented in the PIM side via hardware. Using Vivado HLS v2015.4, we synthesized the control logic on ZC702 FPGA board. The resource cost only includes 208 LUT, 194 FF and 1 BRAM, which is trivial design overhead, possessing less than 1% of the resources in this small chip. Running at 100MHz frequency, the computation latency is much smaller than the time involved in memory access. Thus the area, latency, and power overhead can be safely ignored.

Compared with the situation using PAST in Figure 6, the power usage is improved by PUB, as shown in Figure 8.

4.3 Power Sprinting

Although the internal memory bandwidth and power supply are matched by PAST and PUB, there is still a conflict between the fixed power supply and the changing power requirement to achieve an optimal energy efficiency. In order to achieve better energy efficiency while supporting flexible memory power allocations [64], power sprinting is proposed.

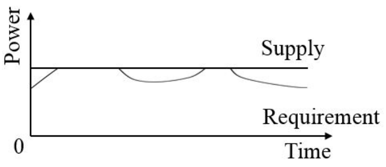

Figure 8: The relationship between power requirement and supply after PAST and PUB

The basic idea of power sprinting (PS) is to provide overloaded power for a short period of time, and return back to under-loaded power status to recover. Note that **only the power cap (P_{max}) is changed via providing more current**, previous PAST and PUB designs can collaborate with PS smoothly. Leveraging the PAST and PUB designs, PS enlarges the power cap in the power arbitrator within the sprinting period. When the sprinting period ends, the power arbitrator sends an extra *PAUSE* command to the queue and ongoing PUs, and reduces power consumption to its previous power cap.

The process can be categorized into 3 major stages: normal, sprint, and recover, as shown in Figure 9. The t_N, t_S, t_R are used to represent the time for the majority stages. The minimum recover time (t_R) is the maximum of time to recharge the extra power source for sprinting and the time to dissipate extra heat. After the recover stage, the memory returns to the normal stage, where it is ready for the next sprint. Compared with the situation in Figure 8, the power requirement is better fulfilled by the limited power supply.

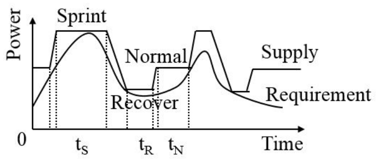

Figure 9: The relationship between power requirement and supply, labeled by the major stages in memory sprinting

The key factor that limits the capability of power sprinting is the thermal capacitance of the package. Previous work [10, 30, 60, 61] uses bulk metal or phase change material to store heat, and uses super capacitors to store energy. The heat is stored by these material and finally dissipated by the heat sink. For a 4Gb HMC, we attach a piece of metal inside the package and a standalone super capacitor beside the package. A piece of $1mm$-thick copper ($3.45J/cm^3K$) is spread for $227mm^2$, and a $1F$ super capacitor is used. The recharge latency is set same to the heat dissipation time. We assume the efficiencies of both boost and recover are 90%. For an extra 4W sprinting power with $1s$ sprint duration (t_S) and $10s$ recover duration (t_R), the increased temperature within the boost period is 5.1°C, and 0.49W power has to be allocated to recharge the super capacitor at the recover stage.

5. EVALUATION

We evaluate the performance and power of proposed techniques based on prior PIM designs via architecture-level simulation. The speedup against non-PIM systems is used as a metric of performance. Various memories are analyzed under the BP model to show their fitness for PIM designs.

5.1 Evaluation Methodology

We build our evaluation system on SMCSim [8]. Power consumption of caches is evaluated by McPat [45]. HMC power is collected from Micron SDRAM power calculator [50] and CACTI-3DD [11], scaled with released HMC data [57]. The platform is extended to support multiple processing units in memory side, such as one core per vault in an HMC cube. The data collected from the simulation is used to calculate the performance, bandwidth, and power consumption.

Three systems are evaluated, including a baseline system with HMC and two PIM designs using HMC and RRAM. Since our three techniques are independent with specific PIM designs, both HMC-based and RRAM-based PIMs can leverage the benefits. The baseline system uses 16 OoO cores and 32 x 4GB HMC-based memory. The HMC-based PIM design is adopted from previous work [3,59], which has the same host CPU and HMC memory configurations but extends each HMC cube by 16 simple cores. The RRAM-based PIM design uses the same configuration as PRIME [13]. More highlighted architecture configurations and timing/power parameters are shown in Table 4. The rows of "Memory" and "HMC Cores" are used for HMC-based systems, and the "RRAM" row is used for RRAM based PIM system. The baseline system only uses the "Memory" row without HMC cores attached.

Various benchmarks are used for the comprehensive evaluations. We use similar benchmarks in previous work [3,8,13,59]. To evaluate HMC-based PIM designs, we choose benchmarks from Big Data analysis and graph computing domains, including matrix addition (MA), tree search (TS), array walk (AW), average teenage follower (TF), page rank (PR), and bellman-ford (BF). To evaluate RRAM-based designs, we select both general applications and neural network designs. The general benchmarks selected from Axbench [1] include 3D gaming "jmeint"(JM), image compression "jpeg" (JP), and image edge detection "sobel" (SO). And the neural network benchmarks include a CNN design and an MLP design for MNIST dataset, and VGG-D known for ImageNet. The codes are manually modified, split into driver and kernel parts. Once an application is launched, the driver on host side calls the kernels running on the PIM side. The NN benchmarks are executed only for inference. Their dot-production operations and related load and store are executed on the PIM side.

5.2 PIM Designs Analysis

We first evaluate the speedup and energy of existing PIM systems, where the energy refers to the portion used by memory, not including the host processor. The results are shown in Figure 10. Compared with the baseline system, the HMC-based PIM reduces energy by about 62% and delivers 5.9× speedup. The RRAM-based PIM reduces energy by 2200× and delivers about 70× speedup. This agrees with prior studies: Large energy reduction and speedup can be achieved by PIM designs.

Table 4: System configurations

Component	Configuration
Host CPU	4 CMPs, 4 out-of-order cores/socket, 4GHz, 4-issue, 128-ROB, 32 KB private I/D cache, 2MB shared L2 per socket
Memory	32 cubes, 8 4Gb DRAM layers, 16 vault/cube, 4 links/cube DDR3-1600 10-10-10, 11pJ/b, 2.6W standby/cube, 1W/link
HMC Cores	16 1-issue in-order cores/cube, 1GHz, 32KB L1/core (60mW dynamic + 20mW static)/core
RRAM	16GB, 8 ranks, 8 4Gb-die/rank, 256 × 256 cells/array DDR3-1066, 7-7-7, R:0.293pJ/b, W:0.865pJ/b, L:92.9mW/die

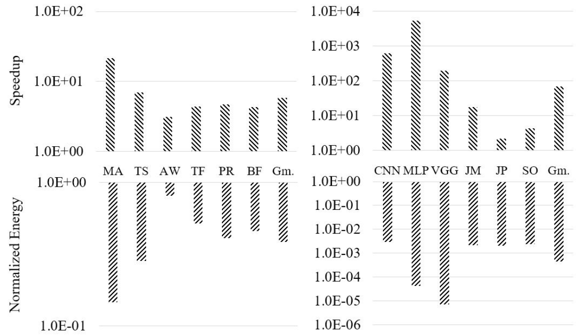

Figure 10: The energy of the memory subsystem and the speedup of applications for various applications

The benefits actually rely on the significantly improved bandwidth. Figure 11 shows the bandwidth used by the PIM design and the baseline CMP based system. The bandwidth has been improved by 2 orders on average, from 0.57GB/s to 178GB/s.

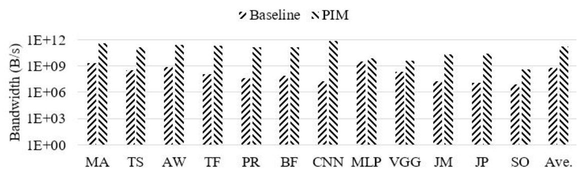

Figure 11: The bandwidth usage for the HMC-based PIM system and the baseline system

We analyze the average power of PIM kernels for further insight. The power only includes the PIM-side power for memory access and processing cores. We breakdown the power for HMC and RRAM arrays respectively, as shown in Figure 12. The "DRAM static" refers to the refresh power and leakage power of the 8 DRAM dies within a cube. The "HMC links' power" is consumed to keep HMC high speed links active. Since the links have to be used for data transmission among cubes, they can not be arbitrarily powered off to save power. The "PIM cores" refers to the power consumed by cores or computing logic resident in the HMC cube or RRAM array, including their interface, buffers, and ALU pipelines. The memory dynamic power includes the activation, precharge, reading, writing, and data transmission power caused by the memory access. As shown by the results, the power of some benchmark (MA) can reach 22W, which is about 2× of the power expectation of an HMC cube.

The memory dynamic, memory static, HMC links and PIM cores consume 50.1%, 17.1%, 26.3%, and 6.6% of the HMC power budget. The overall RRAM power is as low as 0.8W. This is caused by the ultra low memory power usage. On average, RRAM-based PIM consumes about 58% power on the logic and the reset on memory operations. Thus, without carefully handling of power, the PIM design is impractical or inefficient.

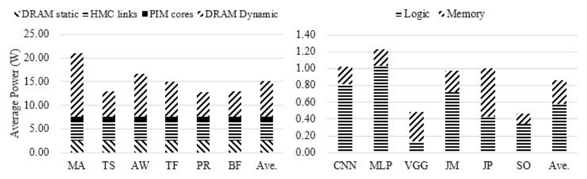

Figure 12: Breakdown of the power usage in an HMC cube or RRAM for different applications

5.3 Evaluation of the PAST, PUB and PS

This section analyzes the performance and power consumption using the proposed techniques. The first two subsections evaluate the standalone effects of PAST and PUB. The following two sections evaluate the combined effect of these three techniques. The last one analyzes the sensitivity of input parameters.

5.3.1 Evaluation of Standalone PAST

The purpose of PAST is to constrain the PIM power consumption, but the performance may also be hurt. As shown in Figure 13, the speedup of HMC-based PIM designs is reduced after applying PAST. The most significant performance slowdown of HMC-based PIM comes from MA, which uses very large bandwidth. The average slowdown is 93%. The most significant slowdown of RRAM-based PIM comes from MLP. The average slowdown is 1%. The power cap is set to 11W for the HMC and 0.9W for the RRAM. And the result reflects a conversion from the performance to the power.

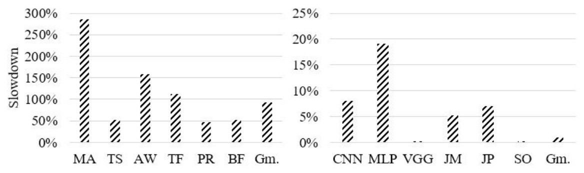

Figure 13: The slowdown against PIM baseline designs after using PAST

5.3.2 Evaluation of Standalone PUB

The power and speedup of PIM designs after using the processing unit boost (PUB) are shown in Figure 14. We compare the power of a memory chip with 3 various implementations of the boost mode in PUB: The X in "PUB-X" represents the ratio of power usage between the boost mode and active mode. For example, PUB-2 means that the power of a unit in boost mode is 2× of the active mode. The label "None" stands for the original PIM designs. The speedup refers to the time ratio between these modes and normal power mode without boosting. Since the speedup of boosting RRAM PIM leverages more on the accelerated memory programming, the performance improvement in RRAM applications is more significant. The average speedup of using PUB with 2× boost mode is 1.82×.

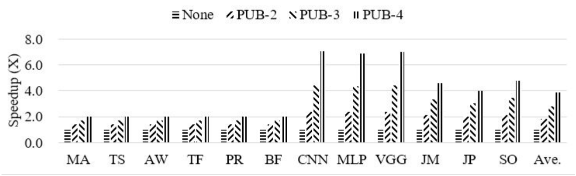

Figure 14: The speedup after using PUB with different power configuration of the boost mode

5.3.3 Evaluation of combined PAST and PUB

Combining PAST and PUB, the power-proportional performance is achieved. Figure 15 shows the speedup after applying combined PAST and PUB onto the two PIM designs, using several different power caps. For the HMC-based PIM, when the power cap is set to 10W, the average speedup cannot reach the 5.88× indicated by the original PIM design. Only when the cap is set at 15W, the speedup can be almost similar to the original design. For the RRAM-based PIM, if the cap is set at 2W, the speedup is improved from 69× to 84×. When the power cap reaches 10W, the speedup could be up to 273×. Enabled by the PAST and PUB, the performance of existing PIM designs can be improved without violating fine controlled power limitations.

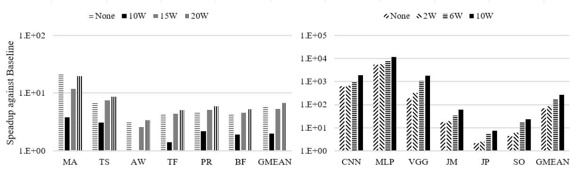

Figure 15: The speedup of PIM designs using PAST and PUB together

5.3.4 Put It All Together

Combining the PAST, PUB, and PS together provides a more energy efficient system. The results are shown in Figure 16. An extra 4W or 8W power aid from sprinting is provided to the HMC PIM designs using PAST + PUB. The "None" represents the system speedup targeting at the original PIM system, which has no power supply capability guarantee. The following bars represent the speedup achieved with different configurations of the power management: For example, "10+PS4" represents a 10W basic power cap with an additional 4W power sprinting capability. The results show that although the performance may be hurt when the power cap is low, the performance improvement can be retrieved after applying power sprinting. On average, 10W with 8W PS can achieve 4.09× speedup (higher than the original 3.78× speedup). The performance can be further improved to 5.81× with a 20W power cap and 8W sprinting power. In summary, the performance can be further improved for existing PIM designs if our power management techniques are properly configured.

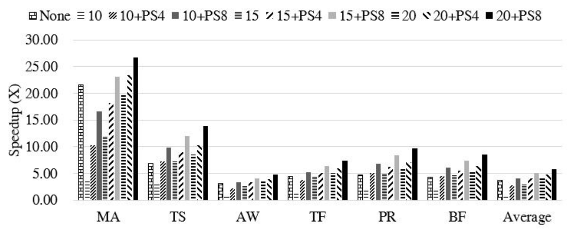

Figure 16: The normalized speedup achieved by 4W and 8W power sprinting for typical power cap set as 10W, 15W and 20W for each HMC cube

5.4 Memory Selection based on BP model

Given power limitation and throughput requirement, selecting appropriate memory configurations is a critical stage in PIM system design. As revealed by the BP model, the power consumption is a function of bandwidth and capacity. This section compares the power-capacity and power-bandwidth relationship for different memories using the BP model, trying to easy the memory selection for different PIM tasks.

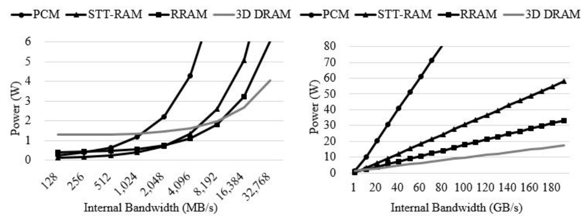

Figure 17: The power consumption of a 4GB memory to support various bandwidth

We applied the BP model to PCM, STT-RAM, RRAM and 3D-stacked DRAM. All memories use single-die 4GB memory chip under 32*nm* processing node, and the bandwidth refers to the read bandwidth. The results are plotted in Figure 17. Due to the large leakage power, DRAM shows an almost constant bias against other memories, when the bandwidth is smaller than 512MB/s. However, this trend changes when the bandwidth increases. Due to the larger read/write energy per bit of emerging NVMs, their power cost exceeds the power of DRAM as the bandwidth increases. Crossing with DRAM, the crosspoint for PCM, STT-RAM and RRAM is 1GB/s, 8GB/s, and 16GB/s, respectively. This means with higher than 16GB/s bandwidth requirement, 3D DRAM shows the best bandwidth scalability: The power increase rate is slower than others, due to its smaller e_r. Meanwhile, the NVMs show benefits in smaller-than-8GB/s bandwidth usage region. In a word, to optimize the power consumption, DRAM is preferred when the bandwidth is high, while NVMs are preferred when the bandwidth is low.

6. RELATED WORK

The PIM power management is different from conventional memory power management techniques. Prior work mainly relies on the temporal and spatial locality of the access sequence [15, 16, 17, 52, 56]. The power of memory is reduced via cutting the leakage power, by concentrating accesses in a few banks or time intervals [52]. Diniz et al. [17] reduced the memory power, assuming untouched memory devices can be turned to low power modes, via greedy algorithms. Based on host-side memory usage information, the frequency scaling is also used to save dynamic power. Deng et al. [16] applied DVFS on memory controller, suggesting a power reduction if the memory utilization is low. David et al. [15] proved that applying DVFS into main memory is feasible. Reducing DRAM bus frequency contributes to 10% of power reduction, when the bandwidth usage is low.

It is not straightforward to apply these techniques to the PIM designs. The increased bandwidth significantly enlarges the portion of the dynamic power from memory, which offsets the benefits of the effort to reduce the leakage power of memory. The activity of different memory arrays is controlled by the massive memory-side processing units, so the host-side controller will have difficulty predicting this activity. The complex write mechanism used by non-volatile memories increases the memory power management complexity. These changes leave the PIM power management techniques seldom explored.

Prior work also explored the methods to leverage the temporary mismatch between the heat dissipation and the power consumption. Current CMPs have employed power boosting, such as the turbo techniques proposed by Intel and AMD [6, 32]. Heat sinks are designed to achieve thermal feasible DRAM stacking based PIM designs [3, 22, 38, 40, 42, 59]. Computational sprinting (CS) [60] was proposed to provide instantaneous computing power by exceeding thermal budget temporarily. Previous evaluation shows a 16W extra power can be tolerated during one second, using a phase change material [60]. However, the CS designed for computing faces challenges when applied in PIM: The activation and the deactivation of the CS require the operating system to migrate halt threads to newly activated cores, and merge them back to a single core when the computation exceeds the sprint capability. This is infeasible in PIM. Since the computation of the PIM task is bounded with its data in memory, mitigating the task around the memory will lead to extra data copy tasks, which recursively leads to more power overheads.

As far as we know, this is the first work to provide a comprehensive power management solution for current processing-in-memory designs.

7. CONCLUSIONS

To facilitate early stage PIM power system designs, we study the relationship between bandwidth and power consumption in processing-in-memory designs. The bandwidth-power model is validated using data of PCM, STT-RAM, RRAM and 3D DRAM. The model demonstrates the trade-off between power and bandwidth in memory. Based on this model, we propose a set of power management solutions for PIM, including PAST, PUB, and PS. These solutions help to reduce the power failure and improve the energy efficiency of PIM. Our evaluation shows that performance can be further improved without significantly modifying existing PIM designs. From the perspective of power, the NVMs are more suitable for PIM designs in the case of large memory capacity and low bandwidth usage. The 3D stacked DRAM is more suitable for PIM designs in the case of small dataset and high bandwidth usage.

8. REFERENCES

[1] P. L.-K. H. E. A. Yazdanbakhsh, D. Mahajan, "Axbench: A multi-platform benchmark suite for approximate computing," *IEEE Design and Test, special issue on Computing in the Dark Silicon Era*, 2016.

[2] J. Abella, A. González, X. Vera, and M. F. P. O'Boyle, "Iatac: a smart predictor to turn-off l2 cache lines," *ACM Trans. Archit. Code Optim.*, vol. 2, no. 1, pp. 55–77, Mar. 2005.

[3] J. Ahn, S. Hong, S. Yoo, O. Mutlu, K. Choi, J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," vol. 43, no. 3, pp. 105–117, jun 2015.

[4] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015, pp. 336–348.

[5] D. H. Albonesi, "Selective cache ways: on-demand cache resource allocation," *MICRO32 Proceedings of the 32nd Annual ACMIEEE International Symposium on Microarchitecture*, vol. 2, pp. 248–259, 1999.

[6] AMD, "Amd turbo core technology," 2011.

[7] ——, "High bandwidth memory, reinventing memory technology," 2015.

[8] E. A. B, D. Rossi, I. Loi, and L. Benini, "Architecture of computing systems – arcs 2016," *Architecture of Computing Systems – ARCS 2016*, vol. 9637, pp. 19–31, 2016.

[9] A. Basu, D. R. Hower, M. D. Hill, and M. M. Swift, "Freshcache: Statically and dynamically exploiting dataless ways," *2013 IEEE 31st International Conference on Computer Design, ICCD 2013*, pp. 286–293, 2013.

[10] L. Cao, J. P. Krusius, M. a. Korhonen, and T. S. Fisher, "Transient thermal management of portable electronics using heat storage and dynamic power dissipation control," *114 Ieee Transactions on Components, Packaging, and Manufacturing Technology-Part a*, vol. 21, no. 1, pp. 113–123, 1998.

[11] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory," *Proceedings of the Design, Automation and Test in Europe*, pp. 33–38, 2012.

[12] H.-Y. Cheng, M. Poremba, N. Shahidi, I. Stalev, M. J. Irwin, M. Kandemir, J. Sampson, and Y. Xie, "Eecache: Exploiting design choices in energy-efficient last-level caches for chip multiprocessors," in *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, ser. ISLPED '14. New York, NY, USA: ACM, 2014, pp. 303–306.

[13] 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," pp. 27–39, June 2016.

[14] T. H. M. C. Consortium, "Hybrid memory cube specification 1.0," pp. 1–122, 2013.

[15] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," *Proceedings of the 8th ACM International Conference on Autonomic Computing*, pp. 31–40, 2011.

[16] Q. Deng, L. Ramos, R. Bianchini, D. Meisner, and T. Wenisch, "Active low-power modes for main memory with memscale," *IEEE Micro*, vol. 32, no. 3, pp. 60–69, 2012.

[17] B. Diniz, D. Guedes, W. Meira, and R. Bianchini, "Limiting the power consumption of main memory," *ACM SIGARCH Computer Architecture News*, vol. 35, p. 290, 2007.

[18] Y. Eckert, N. Jayasena, and G. H. Loh, "Thermal feasibility of die-stacked processing in memory," *Proceedings of the 2nd Workshop Near-Data Processing*, 2014.

[19] D. G. Elliott, W. M. Snelgrove, and M. Stumm, "Computational ram : A memory - simd hybrid and its application to dsp," 1992.

[20] Z. Fang, L. Zhang, J. B. Carter, S. A. Mckee, A. Ibrahim, M. A. Parker, X. Jiang, Z. Fang, M. A. Parker, L. Zhang, J. B. Carter, S. A. Mckee, and X. Jiang, "Active memory controller," *J Supercomput*, vol. 62, pp. 510–549, 2012.

[21] K. Flautner, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," in *Proceedings 29th Annual International Symposium on Computer Architecture*. IEEE Comput. Soc, 2002, pp. 148–157.

[22] M. Gao and C. Kozyrakis, "HRL: Efficient and flexible reconfigurable logic for near-data processing," *HPCA*, vol. 2016-April, pp. 126–137, 2016.

[23] J. Gaur, M. Chaudhuri, and S. Subramoney, "Bypass and insertion algorithms for exclusive last-level caches," in *Proceedings of the 38th annual international symposium on Computer architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 81–92.

[24] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong, "Effects of dynamic voltage and frequency scaling on a k20 gpu," pp. 826–833, 2013.

[25] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: the terasys massively parallel PIM array," *Computer*, vol. 28, no. 4, pp. 23–31, 1995.

[26] B. Govoreanu, G. S. Kar, Y. Y. Chen, V. Paraschiv, S. Kubicek, A. Fantini, I. P. Radu, L. Goux, S. Clima, R. Degraeve, N. Jossart, O. Richard, T. Vandeweyer, K. Seo, P. Hendrickx, G. Pourtois, H. Bender, L. Altimime, D. J. Wouters, J. A. Kittl, and M. Jurczak, "$10 \times 10 nm^2$ hf/hfo x crossbar resistive ram with excellent performance, reliability and low-energy operation," *Technical Digest - International Electron Devices Meeting, IEDM*, pp. 729–732, 2011.

[27] Q. Guo and X. Guo, "A Resistive TCAM Accelerator for Data-Intensive Computing Categories and Subject Descriptors," *Micro 2011*, pp. 339–350, 2011.

[28] Z. Guz, M. Awasthi, V. Balakrishnan, M. Ghosh, A. Shayesteh, T. Suri, and S. Semiconductor, "Real-Time Analytics as the Killer Application for Processing-In-Memory," *Near Data Processing (WoNDP)*, pp. 10–12, 2014.

[29] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, "An energy efficiency feature survey of the intel haswell processor," *Proceedings - 2015 IEEE 29th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2015*, pp. 896–904, 2015.

[30] M. Hodes, R. D. Weinstein, S. J. Pence, J. M. Piccini, L. Manzione, and C. Chen, "Transient thermal management of a handset using phase change material (pcm)," *Journal of Electronic Packaging*, vol. 124, no. 4, pp. 419–426, dec 2002.

[31] M. S. Insight, "High-bandwidth memory (hbm) reinventing memory technology industry standards on-die gpus," vol. 5, p. 8350, 2015.

[32] Intel, "Intel turbo boost technology 2.0," 2011.

[33] ——, "Intel core i7-6785r processor," 2016.

[34] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas, "FlexRAM: Toward an advanced Intelligent Memory system," in *2012 IEEE 30th International Conference on Computer Design (ICCD)*. IEEE, sep 2012, pp. 192–201.

[35] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay," in *Proceedings of the 28th annual international symposium on Computer architecture - ISCA '01*, vol. 29, no. 2. New York, New York, USA: ACM Press, jun 2001, pp. 240–251.

[36] S. M. Khan, Y. Tian, and D. A. Jimenez, "Sampling dead block prediction for last-level caches," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '43. Washington, DC, USA: IEEE Computer Society, 2010, pp. 175–186.

[37] M. Kharbutli and Y. Solihin, "Counter-based cache replacement and bypassing algorithms," *IEEE Trans. Comput.*, vol. 57, no. 4, pp. 433–447, Apr. 2008.

[38] M. J. Khurshid and M. Lipasti, "Data compression for thermal mitigation in the hybrid memory cube," *2013 IEEE 31st International Conference on Computer Design, ICCD 2013*, pp. 185–192, 2013.

[39] G. Kim, J. Kim, J. H. Ahn, and J. Kim, "Memory-centric system interconnect design with Hybrid Memory Cubes," *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, pp. 145–155, 2013.

[40] H. Kim, H. Kim, S. Yalamanchili, and A. F. Rodrigues, "Understanding Energy Aspects of Processing-near-Memory for HPC Workloads," *Proceedings of the 2015 International Symposium on Memory Systems - MEMSYS '15*, pp. 276–282, 2015.

[41] W. Kim, M. S. Gupta, G. Y. Wei, and D. Brooks, "System level

analysis of fast, per-core dvfs using on-chip switching regulators," *Proceedings - International Symposium on High-Performance Computer Architecture*, pp. 123–134, 2008.

[42] Y. Kim, "Analysis of thermal behavior for 3d integration of dram," pp. 5–6, 2014.

[43] V. Krishnaswamy, J. Brooks, G. Konstadinidis, C. McAllister, H. Pham, S. Turullols, J. L. Shin, Y. YangGong, and H. Zhang, "Fine-grained adaptive power management of the sparc m7 processor," in *International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, 2015, pp. 74–76.

[44] H. Y. Lee, Y. S. Chen, P. S. Chen, P. Y. Gu, Y. Y. Hsu, S. M. Wang, W. H. Liu, C. H. Tsai, S. S. Sheu, P. C. Chiang, W. P. Lin, C. H. Lin, W. S. Chen, F. T. Chen, C. H. Lien, and M. J. Tsai, "Evidence and solution of over-reset problem for hfox based resistive memory with sub-ns switching speed and high endurance," *Technical Digest - International Electron Devices Meeting, IEDM*, pp. 7–10, 2010.

[45] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE, 2009, pp. 469–480.

[46] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories," in *Proceedings of the 53rd Annual Design Automation Conference on - DAC '16*, 2016, pp. 1–6.

[47] K. Lim, D. Meisner, A. G. Saidi, P. Ranganathan, and T. F. Wenisch, "Thin servers with smart pipes: designing SoC accelerators for memcached," in *ISCA'13*, 2013, p. 36.

[48] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey, "A voltage reduction technique for digital systems," in *37th IEEE International Conference on Solid-State Circuits*. IEEE, 1990, pp. 238–239.

[49] M. Martonosi and D. Clark, "Voltage and frequency control with adaptive reaction time in multiple-clock-domain processors," in *HPCA*. IEEE, 2005, pp. 178–189.

[50] Micron, "Ddr3 sdram system-power calculator," July 2011.

[51] A. Mishra and N. Khare, "Analysis of dvfs techniques for improving the gpu energy efficiency," *Open Journal of Energy Efficiency*, vol. 04, no. 04, pp. 77–86, 2015.

[52] S. Mittal, "A survey of architectural techniques for dram power management," *International Journal of High Performance Systems Architecture*, vol. 4, no. 2, pp. 110–119, 2012.

[53] S. Mittal, Z. Zhang, and J. S. Vetter, "Flexiway: A cache energy saving technique using fine-grained cache reconfiguration," *2013 IEEE 31st International Conference on Computer Design, ICCD 2013*, pp. 100–107, 2013.

[54] H. Noguchi, K. Ikegami, K. Kushida, K. Abe, S. Itai, S. Takaya, N. Shimomura, J. Ito, A. Kawasumi, H. Hara, and S. Fujita, "A 3.3ns-access-time 71.2uw/mhz 1mb embedded stt-mram using physically eliminated read-disturb scheme and normally-off memory architecture," *ISSCC*, vol. 58, pp. 136–137, 2015.

[55] M. Oskin, F. T. Chong, T. Sherwood, M. Oskin, F. T. Chong, and T. Sherwood, "Active Pages: A Computation Model for Intelligent Memory," *ACM SIGARCH Computer Architecture News*, vol. 26, no. 3, pp. 192–203, 1998.

[56] H. Park, S. Yoo, and S. Lee, "Power management of hybrid dram/pram-based main memory," in *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*. San Diego, California: ACM, jun 2011, pp. 59–64.

[57] J. T. Pawlowski, "Hybrid memory cube (hmc)," *Hotchips*, pp. 1–24, 2011.

[58] M. Powell, S.-H. Y. S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-vdd: a circuit technique to reduce leakage in deep-submicron cache memories," *ISLPED*, pp. 90–95, 2000.

[59] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, "NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads," in *ISPASS 2014 - IEEE International Symposium on Performance Analysis of Systems and Software*, 2014, pp. 190–200.

[60] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. K. Martin, "Computational sprinting," *HPCA*, pp. 1–12, 2012.

[61] G. Setoh, F. L. Tan, and S. C. Fok, "Experimental studies on the use of a phase change material for cooling mobile phones," *International Communications in Heat and Mass Transfer*, vol. 37, no. 9, pp. 1403–1410, nov 2010.

[62] 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 *ISCA*, 2016, pp. 14–26.

[63] P. Siegl and R. Buchty, "Data-Centric Computing Frontiers : A Survey On Processing-In-Memory," *Memsys*, vol. 1000, 2016.

[64] D. Skarlatos, R. Thomas, A. Agrawal, S. Qin, R. Pilawa-Podgurski, U. R. Karpuzcu, R. Teodorescu, N. S. Kim, and J. Torrellas, "Snatch: Opportunistically Reassigning Power Allocation between Processor and Memory in 3D Stacks," *MICRO*, 2016.

[65] R. Smith, "The amd radeon r9 fury x review," 2015.

[66] H. S. Stone, "A logic-in-memory computer," *IEEE Trans. Comput.*, vol. 19, no. 1, pp. 73–78, Jan. 1970.

[67] B. Su, J. Gu, L. Shen, W. Huang, J. L. Greathouse, and Z. Wang, "Ppep: Online performance, power, and energy prediction framework and dvfs space exploration," *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 445–457, 2014.

[68] K. T. Sundararajan, V. Porpodas, T. M. Jones, N. P. Topham, and B. Franke, "Cooperative partitioning: Energy-efficient cache partitioning for high-performance cmps," *Proceedings - International Symposium on High-Performance Computer Architecture*, pp. 311–322, 2012.

[69] K. Tran and J. Ahn, "Hbm : Memory solution for high performance processors," *Memcon*, no. October, 2014.

[70] C. Villa, D. Mills, G. Barkley, H. Giduturi, S. Schippers, and D. Vimercati, "A 45nm 1gb 1.8v phase-change memory," pp. 270–271, Feb 2010.

[71] Y. Xie, Ed., *Emerging Memory Technologies*. New York, NY: Springer New York, 2014. [Online]. Available: http://link.springer.com/10.1007/978-1-4419-9551-3

[72] C. Xu, D. Niu, N. Muralimanohar, and R. Balasubramonian, "Overcoming the challenges of crossbar resistive memory architectures," pp. 476–488, 2015.

[73] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," *HPCA*, pp. 476–488, 2015.

[74] T. Yoshida, N. Generation, T. Computing, and F. Limited, "Fujitsu ' s next generation processor for hpc," in *HotChips*, 2014.

[75] S. Yu and H. S. P. Wong, "A phenomenological model for the reset mechanism of metal oxide rram," *IEEE Electron Device Letters*, vol. 31, no. 12, pp. 1455–1457, 2010.

[76] C. Zhang, G. Sun, P. Li, and T. Wang, "Sbac: a statistics based cache bypassing method for asymmetric-access caches," *ISLPED*, pp. 345–350, 2014.

[77] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "TOP-PIM," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing - HPDC '14*. New York, New York, USA: ACM Press, 2014, pp. 85–98.

[78] Q. Zhu, T. Graf, H. E. Sumbul, L. Pileggi, and F. Franchetti, "Accelerating sparse matrix-matrix multiplication with 3D-stacked logic-in-memory hardware," in *2013 IEEE High Performance Extreme Computing Conference, HPEC 2013*, 2013, pp. 1–6.