CoolPIM: Thermal-Aware Source Throttling for Efficient PIM Instruction Offloading

Lifeng Nai*, Ramyad Hadidi†, He Xiao†, Hyojong Kim†, Jaewoong Sim‡, Hyesoon Kim†
*Google, ‡Intel Labs, †Georgia Institute of Technology
*lnai@google.com, ‡jaewoong.sim@intel.com, †{rhadidi,hxiao,hyojong.kim,hyesoon.kim}@gatech.edu

***Abstract*—Processing-in-memory (PIM) is regaining attention as a promising technology for improving energy efficiency of computing systems. As such, many recent studies on 3D stacking-based PIM have investigated techniques for effectively offloading computation from the host to the PIM. However, the thermal impacts of such offloading have not been fully explored. This paper provides an understanding of thermal constraints of PIM in 3D-stacked designs and techniques to effectively utilize PIM. In our experiments with a real Hybrid Memory Cube (HMC) prototype, we observe that compared to conventional DRAM, HMC reaches a significantly higher operating temperature, which causes thermal shutdowns with a passive cooling solution. In addition, we find that even with a commodity-server cooling solution, when in-memory processing is highly utilized, HMC fails to maintain the temperature of the memory dies within the normal operating range, which results in higher energy consumption and performance overhead. Thus, we propose CoolPIM, a collection of thermal-aware software- and hardware-based source throttling mechanisms that effectively utilize PIM by controlling the intensity of PIM offloading in runtime. Our evaluation results demonstrate that CoolPIM achieves up to 1.4× and 1.37× speedups compared to non-offloading and naïve offloading scenarios.**

I. Introduction

Recent advances in 3D-stacking technology and the increasing need for energy efficient computing urged both industry and academia to revisit processing-in-memory (PIM) concept. Memory vendors such as Micron and Samsung have started productizing or publicly discussing PIM [26], [31]. Recent studies from academia have also shown promising performance and energy improvements *again* [2], [23] as previously demonstrated in the PIM research conducted decades ago [9]. With modern and emerging workloads that process massive amounts of data with irregular memory access behaviors, such as graph analytics, PIM is expected to play an important role in providing better energy efficiency for computing environments.

One of the key challenges of enabling 3D stacking-based PIM in conventional systems is maintaining a suitable thermal conditions for its operations. This is because, (1) while conventional memories offer tens of gigabytes of memory bandwidth, PIM, implemented via 3D-stacking techniques often provides hundreds of gigabytes of that. Thus, this increased bandwidth introduces new thermal constraints when it is highly utilized, as we will illustrate in Section III with measured data. (2) In typical 3D-stacked designs, memory dies are vertically stacked between a heat sink and a logic die, so their heat transfer capability is not as effective as conventional memories; therefore, memory dies in a 3D-stacked design operate in a higher temperature than conventional memories do. In addition, the execution of PIM instructions further introduces a non-negligible amount of heat, which exacerbates this thermal constraint. In fact, for dual-inline memory modules (DIMMs), the memory temperature rarely exceeds $85\,^{\circ}\mathrm{C}$ [20], which is the upper bound of the normal operating temperature of DRAM $85\,^{\circ}\mathrm{C}$.[1] Although JEDEC specifies the extended temperature range of $85\,^{\circ}\mathrm{C}$-$95\,^{\circ}\mathrm{C}$ with doubled DRAM refresh rate [15], operating DRAM in the extended temperature range incurs higher energy consumption and performance overhead than in the normal temperature range [20]. As such, DIMMs are mostly used with a passive heat sink (or even without any cooling solutions) without great concern about thermal constraints or performance implications. However, for the best performance of PIM, we need to carefully consider the mentioned thermal constraints.

In this paper, we explore managing such thermal constraints of 3D-stacked designs so that new 3D-stacked, PIM-enabled systems can effectively utilize hundreds of gigabytes of memory bandwidth and PIM capability of 3D-stacked designs. To understand the thermal challenges, we perform an analysis on a *real* Hybrid Memory Cube (HMC) 1.1 prototype while varying bandwidth utilization and cooling solutions. Our analysis shows that HMC cannot even operate at the peak bandwidth with a *passive* heat sink, which indicates that HMC systems need a strong cooling solution even without utilizing in-memory processing. Also, we model the next generation of HMC, HMC 2.0, with the PIM functionality of HMC 2.0. Our evaluations show that even a commodity-server cooling fails to maintain the memory temperature below the normal operating temperature range; thus forcing the device needs to shut down or increase/decrease the DRAM refresh rate/frequency before serving requests again.

Based on our observations, we propose CoolPIM, which provides a collection of thermal-aware software- and hardware-based techniques that effectively utilize PIM capability of 3D-stacked designs by dynamically controlling the intensity of PIM offloading. The proposed technique maintains the temperature of memory dies within the normal operating temperature, which leads to higher performance compared to naïve offloading. We evaluate CoolPIM with a GPU system

[1] This is the case-surface temperature of DRAM.

with a wide range of graph workloads, and our results show that CoolPIM improves performance up to 1.4× and 1.37× compared to non-offloading and naïve PIM offloading without thermal considerations, respectively. In summary, this paper makes the following contributions.

- We provide a thermal analysis of an HMC 1.1 prototype across various bandwidth utilization and cooling solutions to understand the thermal constraints of a real-world 3D memory. To our knowledge, this is the first work that evaluates a real HMC platform for thermal analysis.
- We provide a thermal analysis of HMC 2.0 with PIM offloading using thermal simulation. Our results show that the DRAM layers can exceed the normal operating temperature even with a commodity-cooling solution when PIM offloading is used.
- We propose CoolPIM, a collection of thermal-aware software- and hardware-based source throttling techniques that dynamically controls the intensity of PIM offloading and provides trade-offs between the two design options.

II. Background

This section provides background on HMC and its processing-in-memory (PIM) capability.

A. HMC Architecture

HMC integrates multiple DRAM dies and one logic die within a single package using die-stacking technology. HMC is organized into multiple vaults that are functionally and operationally independent. The memory partitions within a vault are connected via through-silicon vias (TSVs), each of which may have multiple memory banks. Each vault incorporates a vault controller in the logic layer that manages the memory partitions stacked on top of it, similar to a memory controller. A crossbar switch connects all vault controllers and external I/O links. Each I/O link consists of 16 input and 16 output serial lanes. The I/O links follow a packet-based protocol, in which the packets consist of 128-bit flow units named as FLIT [4]. Each response packet contains a tail field, which includes a 7-bit error status (`ERRSTAT[6:0]`). When exceeding the operational temperature limit, HMC sends back an error warning by setting the error bits to `0x01`.

B. PIM Instruction Offloading

Compared to conventional DRAM DIMMs, HMC not only provides a dramatic improvement in memory bandwidth, but also enables the possibility of supporting a variety of PIM functionalities starting from the HMC 2.0 specification [4]. PIM instructions of HMC enable limited arithmetic and logic functions by performing the following three steps: (1) Reading data from a DRAM address, (2) performing computation on the data in the logic layer, and (3) writing back the result to the same DRAM address. According to HMC 2.0, PIM units perform the read-modify-write (RMW) operation atomically. That is, the corresponding DRAM bank is locked during an RMW operation, so any other memory requests to the same bank cannot be serviced. In addition, all PIM operations include only one memory operand – the operations are performed on an immediate value and a memory operand. HMC 2.0 supports several types of PIM instructions (i.e., arithmetic, bitwise, boolean, comparison). Prior work also proposed instruction extensions for floating-point arithmetic [23]. Depending on the definition of specific commands, a response may or may not be returned. If the response is returned, it includes an atomic flag that indicates whether the atomic operation was successful. Depending on the commands, the original data may also be returned along with the response.

Benefits of PIM: By offloading the operations that process irregular data, PIM instructions bring performance benefits due to multiple factors. First, instruction offloading saves bandwidth consumption between host and memory. The packet size of regular memory requests and PIM operations are summarized in table I. A 64-byte READ/WRITE request consumes 6 FLITs in total, while a PIM operation needs only 3 or 4 FLITs. Therefore, PIM offloading potentially can save up to 50% memory bandwidth. The bandwidth savings improve the performance of bandwidth-sensitive applications, such as GPU applications with intensive memory requests. Second, PIM offloading enable better cache utilization because when the offloaded instructions bypass cache, they reduce cache pollution and increase effective cache size. Third, PIM helps avoiding the overhead of host atomic instructions in CPUs [23]. However, because GPU applications usually exhibits smaller atomic overhead and less sensitivity of cache behavior, two last factors are less effective for them.

Architecture Support for PIM: In an HMC 2.0, a PIM instruction is similar to a regular memory request except that it contains a command field. Enabling PIM instruction offloading, however, requires a few architectural changes for identifying PIM instructions and maintaining data coherence. Prior works have proposed two major methods for utilizing PIM instructions. PEI [2] proposes new host instructions that correspond to each of PIM instructions. Hence, programmers or compilers use the new host instructions to perform PIM offloading. Instead of modifying host ISA, GraphPIM [23] specifies a PIM memory region and identifies host atomic instructions that access the memory region as offloading targets. While PEI maintains coherence between data copy in cache and memory by invalidating/writing-back the cache blocks accessed by PIM instructions, GraphPIM bypasses the cache for the data of offloading targets by exploiting the existing uncacheable region feature of x86 processors. As demonstrated in [23], the cache-bypassing policy can bring an additional performance benefit because of avoiding the

TABLE I
HMC memory transaction bandwidth requirement in FLITs
(FLIT size: 128-bit)

Type	Request	Response
64-byte READ	1 FLITs	5 FLITs
64-byte WRITE	5 FLITs	1 FLITs
PIM inst. without return	2 FLITs	1 FLITs
PIM inst. with return	2 FLITs	2 FLITs

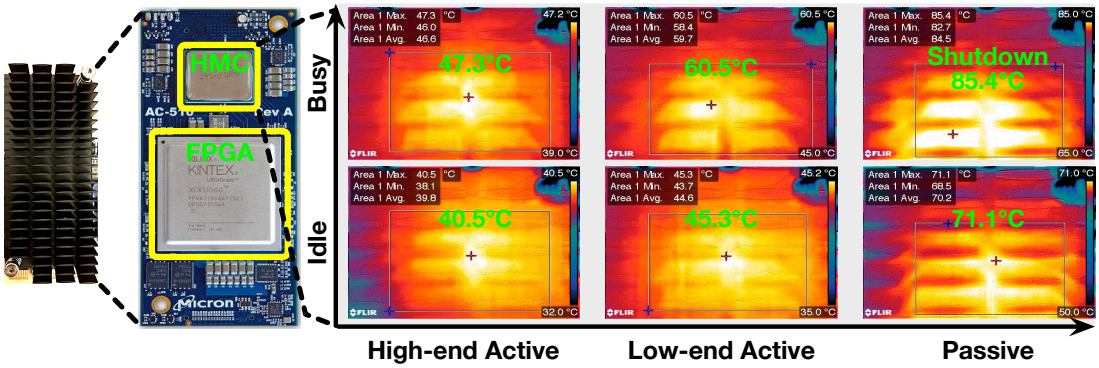


Fig. 1. Thermal evaluation of a real HMC prototype

unnecessary cache-checking overhead. In this paper, because we focus on the PIM instruction offloading for GPU platforms, in which the overhead of atomic operations usually is not a major bottleneck, we follow the ISA-based method, and incorporate new host-ISA instructions for each PIM instruction. In addition, to receive the benefit of cache-bypassing as demonstrated in GraphPIM, we also allocate the offloading target data in an uncacheable region.

III. HMC Thermal Challenges

The 3D stacking of DRAM and logic dies in HMC offers high memory bandwidth and enables PIM functionality. However, HMC exhibits high power density because of the die-stacking architecture and non-trivial power consumption of the logic layer, especially when PIM functionality is intensively utilized. Such high power density increases the temperature of the stacked DRAM dies and raises the thermal challenges for the entire HMC module. In this section, to understand the thermal challenges in HMC, we first evaluate an HMC 1.1 prototype by measuring the surface temperature of HMC across various bandwidth consumption and cooling methods. Then, we further evaluate/model an HMC 2.0 system with the energy information released in prior literature. After that, we analyze the thermal impact of PIM offloading and discuss its performance tradeoffs.

A. HMC 1.1 Prototype Evaluation

Experiment Platform: To analyze the thermal characteristics of HMC, we evaluate a real HMC prototype. The experiment platform (Pico SC-6 Mini [29]) has a PCIe backplane (EX-700 [28]) that can accommodate up to six compute modules (AC-510 [27]). Each compute module contains a Kintex Xilinx FPGA and an 4 GB HMC 1.1 [13]. The HMC has two half-width (x8) links that provide up to 60 GB/s bandwidth. An active heat sink is attached on the compute module for the cooling of both FPGA and HMC. To evaluate the thermal impact of HMC, we use a thermal camera and measure the surface temperature of the HMC. The thermal resistance of a typical transistor chip is insignificant compared with an external heat sink (i.e., *plate-fin heat sink*) [6], and the in-package junction temperature should be around 5 to 10 degree higher than its surface temperature, given a 20 Watt power to dissipate.

Observations: By measuring the surface temperature of HMC with a thermal camera, we evaluate its thermal impact with regard to bandwidth utilization and cooling methods. Figure 1 illustrates the results of our evaluation. We further elaborate our observations as follows.

1) *Surface Temperature:* Unlike conventional memories, HMC operates at a higher temperature. The runtime thermal images of the HMC under three types of heat sinks are shown in Figure 1. We observe that the surface temperature of highly utilized HMC exceeds 80 °C with a passive heat sink, and the junction temperature reaches to or exceeds 90 °C (with a typical thermal resistance from the package surface to the internal chip). Even with a high-end active heat sink, the surface temperature still reaches around 50 °C in our experiments. Since HMC 2.0 has a higher bandwidth than that of HMC 1.1 (60 GB/s), it will experience even worse thermal issues. Another study on HMC 1.1 has observed similar trends and behaviors [12].

2) *Overheated Behavior:* In our evaluation, with a passive heat sink, HMC cannot operate at the full bandwidth and shuts down when the surface temperature reaches around 85 °C (in-package DRAM temperature is close to 95 °C). Higher temperature makes DRAM cells weaker and charge leaking faster. Although by varying DRAM frequency and refresh interval, DRAM can operate at higher temperatures, our evaluation shows that the HMC prototype incorporates a more conservative policy, in which HMC stops completely when the DRAM stack is overheated. Although HMC can be re-enabled after the chip is cool again, data is lost and recovery delay is tens of seconds in our evaluation, which is much longer than the processing time of typical GPU kernels.

3) *Cooling Methods:* Conventional DRAMs often use only ambient air cooling without even a passive heat-sink, but HMC chips require much better heat transfer capability. As shown in Figure 1, with a passive heat sink, the surface temperature exceeds 71 °C at an idle state, and HMC shuts down before the full bandwidth is achieved. Even if we include a low-end active heat sink, HMC temperature still reaches 60 °C at a busy state. Thus, an HMC system requires a strong cooling solution [12]. In addition, because newer generations of HMC provides substantially higher bandwidth, more efficient cooling methods are desirable.

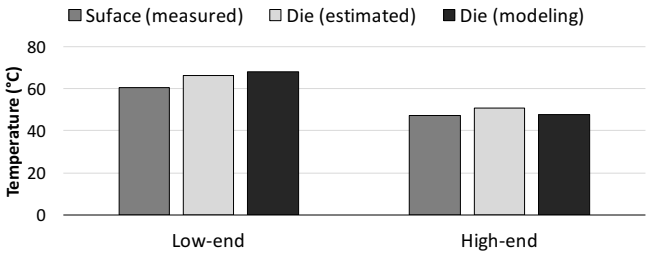


Fig. 2. Thermal model validation – Surface (measured): measured surface temperature of the real HMC 1.1. Die (estimated): estimated according to the surface temperature. Die (modeling): modeled die temperature

B. HMC 2.0 Thermal Modeling

In the previous section, we analyze the thermal issues of HMC by evaluating a real HMC 1.1 platform. However, HMC integrates PIM functionality starting from the HMC 2.0 specification [4], which has different off-chip bandwidth and architectural configurations. In this section, we conduct a thermal modeling of HMC 2.0 without PIM instructions based on the energy data reported by Micron and results from our gate-level synthesis.

Evaluation Methodology: To estimate the temperature of each layer, we perform thermal simulation using KitFox [32] and 3D-ICE [33]. We evaluate multiple cooling solutions from passive to high-end active heat sink as summarized in Table II. We follow the HMC 2.0 architectural configuration of an 8 GB cube, which consists of eight DRAM dies and one logic die as explained in Section II. (see Section V-A for more details about our evaluation methodology).

Model Validation: Before estimating the temperature of an HMC 2.0 cube, we first validate our thermal evaluation environment by modeling an HMC 1.1 system with the same cooling and bandwidth configuration and comparing the result with the real system measurements. Figure 2 shows the validation result. Our thermal modeling tool models the temperature of DRAM dies, which is usually higher than the surface temperature we measured using a thermal camera. Thus, we also estimate the die temperature based on the surface temperature using a typical thermal resistance model. The results show that our thermal model has a reasonable error compared to the real system measurements.

Observations: We summarize the thermal results in Figure 3 and Figure 4. With a commodity-server active heat sink, the temperature of HMC reaches 81 °C at a full off-chip bandwidth utilization (320 GB/s). Because of the physical stacking structure of HMC, the lowest DRAM die and logic layer reach the highest temperature. From the thermal map, we

TABLE II
TYPICAL COOLING TYPES

Type	Thermal Resistance	Cooling Power[b]
Passive heat sink	4.0 °C/W	0
Low-end active heat sink	2.0 °C/W	1x
Commodity-server active heat sink	0.5 °C/W	104x
High-end active heat sink	0.2 °C/W	380x[c]

[b]We follow the same plate-fin heat sink model for all configurations.
[c]The fan has 2x wheel diameter in this configuration.

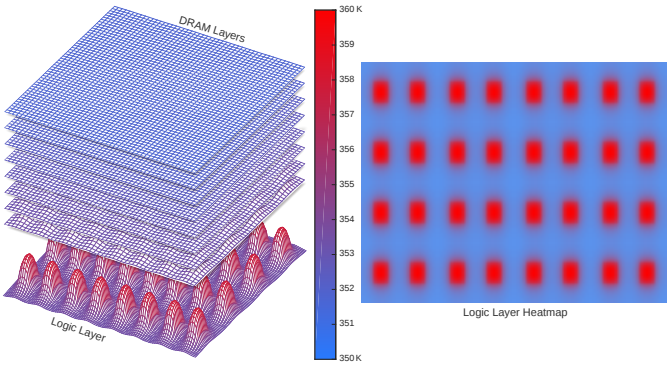


Fig. 3. Heat map with a full bandwidth utilization and a commodity-server active heat sink (left: 3D heat map of all layers. right: 2D heat map of logic layer)

can see that the hot spot appears at the center of each vault because of the high power density of the logic component. We also observe the thermal dependence of bandwidth utilization and cooling types summarized as follows.

1) Bandwidth Impact: The power consumption of the logic layer and DRAM dies is proportional to the bandwidth utilization. As shown in Figure 4, the peak DRAM temperature increases with higher bandwidth utilization. Because of the bottleneck of the off-chip link bandwidth, without PIM instructions, the maximum data bandwidth of HMC 2.0 is 320 GB/s (because of packet header overhead, aggregated link bandwidth is 480 GB/s). Accordingly, with a commodity-server active heat sink, the peak DRAM temperature reaches 81 °C at maximum bandwidth, and 33 °C at the idle state.

2) Cooling Impact: As demonstrated in our experiments of both the HMC prototype and the modeling, HMC temperature heavily relies on the cooling method of the HMC package. To suppress the temperature below 85 °C for a full-loaded PIM, we require the thermal resistance of the cooling structure less than 0.27 °C/W, which falls within the realm of high-end heat sinks. However, a strong cooling method is not free. From Table II, the fan power, calculated using the fan curve methodology [34], exaggerates from low-end heat sinks (1x) to commodity (104x) and high-end (380x) heat sinks. Specifically, the fan in a high-end plate-fin heat sink [36] of 0.2 °C/W consumes around 13 Watt (almost half as much as the power of a fully-utilized HMC 2.0 cube) in our extrapolated model of the fan power. Thus, for the sake of system power usage effectiveness, we have a limited thermal headroom for the HMC package given a restricted fan power, and the remainder of the paper assumes commodity-server cooling for the overall system.

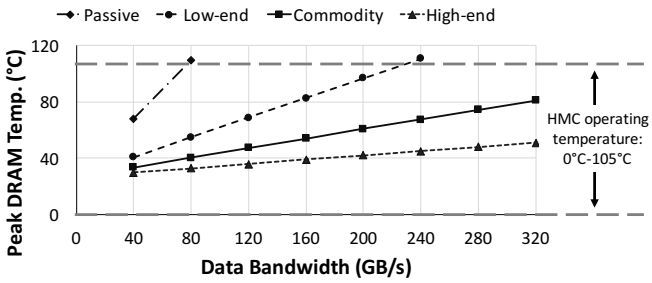


Fig. 4. Peak DRAM temperature with various data bandwidth and cooling methods

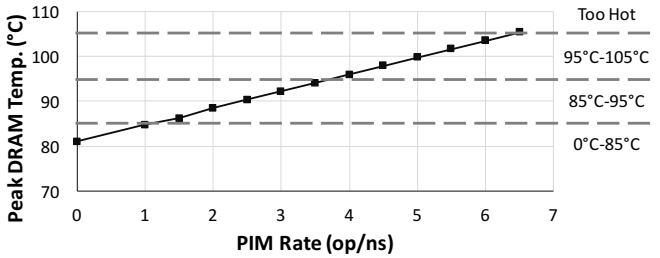


Fig. 5. Thermal impact of PIM offloading

C. Thermal Trade-off of PIM Offloading

As discussed in the previous section, HMC maintains a relatively reasonable operating temperature with a commodity-server cooling. When PIM offloading is used, however, the HMC would experience a non-trivial temperature increment because of the additional power consumption of both logic and DRAM layers. In this section, we discuss the thermal impact of PIM offloading and its impact on performance.

Impact on Power & Temperature: For PIM functionality, HMC incorporates functional units (FUs) that consume energy when executing offloaded PIM instructions. Assuming that a single FU operation consumes E Joule/bit, the additional power consumption is computed as $Power(FU) = E \times FU_{\text{width}} \times PIM_{\text{rate}}$, in which FU_{width} is the bit width of each functional unit, which is 128bit, and PIM_{rate} is the number of PIM operations per second. As explained in Section II, each PIM instruction in HMC 2.0 requires two DRAM accesses (read and write) internally. Thus, PIM offloading increases internal DRAM bandwidth utilization, which can exceed 320 GB/s. This increase in internal DRAM bandwidth utilization leads to additional DRAM power consumption that also increases the temperature. Figure 5 shows the relationship between PIM offloading rates and HMC temperature. In this evaluation, we assume that a full-bandwidth utilization is achieved by the PIM operations and regular memory requests. In our modeling, because of the thermal limitation of $105\,^{\circ}\text{C}$, the maximum PIM offloading rate is 6.5 PIM op/ns. The result show a clear positive correlation between the offloading rate and temperature. As shown in Figure 5, for maintaining the DRAM temperature below $85\,^{\circ}\text{C}$, the PIM offloading rate should be lower than 1.3 op/ns.

Performance Trade-off: By offloading operations that process irregular data, a PIM system reduces external memory bandwidth consumption, which can be translated into performance benefits for bandwidth-intensive workloads. A high offloading rate provides more bandwidth savings, and thus more performance benefits. However, as demonstrated in the prior section, a higher offloading rate introduces thermal problems in HMC, which brings a negative impact on the performance of the memory system.

A conservative operation policy for cooling down an overheated HMC is to completely shut down the HMC. This policy leads to an extremely long stall time as we observed in the HMC prototype evaluation. Another policy is to follow a dynamic DRAM management method as suggested in prior literature [10], [18], which is changing the DRAM frequency/refresh interval at high temperatures, but this one also brings a non-trivial performance degradation because of slowing down not only PIM instructions but regular memory requests. To strike a balance between thermal limitations and PIM offloading, we propose CoolPIM, a collection of thermal-aware software- and hardware-based source throttling techniques

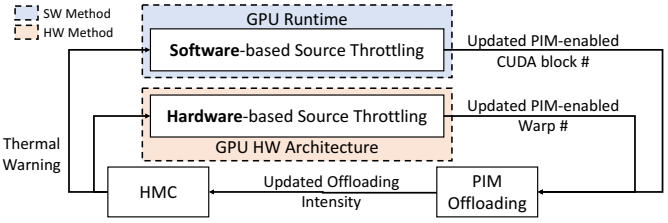


Fig. 6. Illustration of CoolPIM feedback control

IV. CoolPIM: Thermal-Aware Source Throttling

In this section, we propose CoolPIM, which utilizes a simple and practical method for controlling the intensity of PIM offloading with thermal considerations. We first provide an overview of CoolPIM and then discuss our software- and hardware-based source throttling mechanisms respectively.

A. Overview

As shown in Figure 6, CoolPIM performs dynamic source throttling using the *thermal warning message* from HMC. At a high level, our throttling methods use a closed-loop feedback mechanism that control *PIM intensity*. That is, a thermal warning message leads to a reduction of the number of PIM instructions that are executed within the HMC, thereby decreasing the internal temperature of HMC. We present both software- and hardware-based mechanisms, which offer different throttling granularities. The software-based mechanism controls the number of *PIM-enabled CUDA blocks* launched on a GPU using a specialized thermal interrupt handler and software components in the GPU runtime; and therefore works without additional hardware support. In contrast, the hardware-based mechanism controls the number of *PIM-enabled warps* and thus enables a more fine-grained control; but at the cost of an extra hardware unit in each GPU core. We present our mechanisms in the following sections.

B. Software-based Dynamic Throttling

Figure 7 summarizes the overview of our proposed software-based dynamic throttling technique (*SW-DynT*), which controls PIM offloading intensity at a *CUDA block granularity*. The GPU runtime implements an offloading controller that maintains a *PIM token pool (PTP)*. The PTP value represents the number of maximum thread blocks that are allowed to use PIM functionality. Before launching a thread block, the

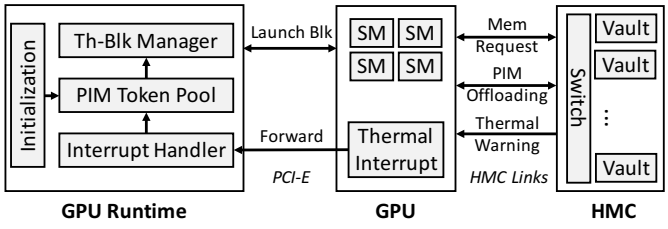


Fig. 7. Overview of software-based dynamic throttling

thread block manager first needs to request a PIM token from PTP. On a success, the runtime launches the original code that contains PIM instructions, otherwise it launches a pre-generated *shadow non-PIM code* (i.e., when no token is available in the token pool). During execution, if HMC gets overheated it issues a thermal warning message which will then cause SW-DynT to reduce the PTP size. Note that HMC 2.0 provides thermal feedback in its specification via error messages; hence, no hardware modification is required.

PTP Initialization: To estimate the initial PTP size, the software-based technique performs a static analysis at compile time. As discussed in Section III, the HMC temperature (HMC_{Temp}) is a function of the bandwidth utilization (BW_U) and PIM offloading rate (PIM_{Rate}) (i.e., $HMC_{Temp} = F(BW_{\text{U}},\ PIM_{\text{Rate}})$). As both PIM and non-PIM code consume similar link bandwidth, the PIM offloading rate essentially dictates the difference in HMC temperatures between PIM and non-PIM code.

$$PIM_{\text{Rate}} = PIM_{\text{PeakRate}} \times PIM_{\text{Intensity}} \times (^{PTP\_Size}/_{\text{MaxBlk\#}}) \times (1 - Ratio_{\text{DivergentWarp}}) \quad (1)$$

Equation 1 shows that we estimate the PIM offloading rate based on PIM peak rate ($PIM_{PeakRate}$), intensity of PIM instructions ($PIM_{Intensity}$), percentage of PIM-enabled CUDA blocks, and the ratio of divergent warps ($Ratio_{\text{DivergentWarp}}$). Among the parameters, both PIM_{PeakRate} and MaxBlk# are hardware dependent features that can be measured by performing a simple trial run on the target platform or estimated from the hardware specification. Also, we can compute the PIM instruction intensity in the compilation stage. Although the ratio of divergent warps typically is around zero in typical GPU programs, it can be large in some GPU applications, such as graph analytics. In such cases, we can estimate its range with the help of the algorithm knowledge. For example, topological-driven graph algorithms have a high ratio, while warp-centric ones have a low ratio. With all the parameters estimated and measured, we first estimate the required PIM offloading rate threshold from the hardware platform as Equation 1. Then, we compute the PTP size with the given PIM offloading rate. Because the feedback control only down-tunes the pool size, we add a small margin to the computed value in order to be not conservative (i.e., $PTP_{\text{Initial Size}} = PTP_{\text{Calculated}} + margin$); we use a margin of 4 thread blocks for our evaluation.

Source Throttling: PTP maintains the information on the maximum possible number of PIM-enabled blocks. Each new PIM-enabled CUDA block requests a token from the pool when being launched, and returns the token to the PTP when completed the execution. PTP processes the token requests based on the first-come-first-serve policy, which issues a token to the requester until the number of on-going PIM-enabled blocks reaches the PTP size. With a successfully fetched token, the block manager launches the CUDA block using the original PIM code by configuring the corresponding code entry pointer. If the block fails to get a token, the block manager launches the block using the generated non-PIM code. The dynamic controlling of PIM and non-PIM CUDA blocks determines the PIM offloading intensity and the temperature of the HMC.

As explained in Section II, when the temperature reaches a warning threshold, the HMC sets the error status bits in the response packets.[4] When receiving the thermal warning messages from the HMC side, the host GPU will trigger a thermal interrupt and forward it to the GPU runtime. The interrupt handler then updates the PTP size to reduce the offloading intensity of PIM instructions. We then calculate the new PTP size as $PTP_{\text{Size}} = Min(\ PTP_{\text{Size}} - CF,\ \#issuedToken\ )$ by comparing the current number of issued tokens and the PTP size after reduction. The reduction granularity is controlled by *ControlFactor (CF)*. A larger CF value allows for a fast cooldown of HMC; however, it also increases the chance of under-tuning the PTP size. A small control factor leads to a longer time for reducing the pool size down to the proper number and for cooling down the HMC.

Code Generation for Non-PIM Code: SW-DynT, depending on the thermal condition, launches thread blocks with a PIM-enabled or non-PIM code. For example, if there is no token left in the PTP, the runtime launches a new block with an entry point of `cuda_kernel_np`. The GPU compiler generates PIM-enabled and non-PIM kernels at compile time by mapping CUDA atomic functions to PIM instructions or vise versa. All PIM instructions, including the ones defined in HMC 2.0 and the extended ones proposed in [23], can be directly mapped to CUDA atomic functions. For instance, a `PIM_Add` atomic instruction can be mapped to the CUDA atomicAdd function. Note that because these mappings are simple source-to-source translations at the abstract syntax tree (AST) level (or at the IR level, which is also similarly simple), changes to GPU compiler is trivial.

C. Hardware-based Dynamic Throttling

In addition to the software-based technique, we also present a hardware-based dynamic throttling (HW-DynT) method. By dynamically controlling PIM offloading in the GPU hardware architecture, HW-DynT enables fast thermal-feedback reaction and achieves fine-grained PIM intensity control.

Hardware PIM Offloading Control: Similar to SW-DynT, HW-DynT performs offloading control based on the thermal feedback from the HMC. However, when receiving a thermal warning message, instead of forwarding the thermal interrupt to the GPU runtime, HW-DynT directly performs source throttling in the GPU hardware. To do so, each GPU core includes an extra hardware component called *PIM Control Unit (PCU)*. When thermal warning is triggered, PCU collects the thermal feedback from the HMC controller and reduces the number of PIM-enabled warps by *control factor (CF) in warps*. The PIM-disabled warps will then execute the GPU kernel code while translating PIM instructions into non-PIM ones. In HW-DynT, because of the hardware support, we control the intensity of PIM offloading at the warp granularity,

[4]The current HMC 2.0 specification defines a single thermal error state, but it can trivially define multiple error states as multiple unused error status bits are available in the field.

TABLE III
EXAMPLES OF PIM INSTRUCTION MAPPING

Type	PIM instruction	Non-PIM
Arithmetic	signed add	atomicAdd
Bitwise	swap, bit write	atomicExch
Boolean	AND/OR	atomicAND/atomicOR
Comparison	CAS-equal/greater	atomicCAS/atomicMax

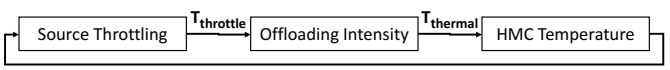
which is more fine-grained than the thread-block granularity in SW-DynT. Also, unlike SW-DynT, which waits for the execution of ongoing thread-blocks to finish, HW-DynT takes effect immediately because PIM-enabled warps is disabled in hardware. Because of the fast reaction of the hardware-based mechanism, HW-DynT does not require a precise initial configuration. Thus, we set the initial number of PIM-enabled warps to the maximum.

Delayed Control Updates: HW-DynT allows a faster reaction to the thermal feedback than SW-DynT does. Although PIM offloading intensity can be changed immediately by updating the PCU in HW-DynT, the actual HMC temperature change requires a longer response time, which is usually an order of milliseconds. If HW-DynT updates the PCU too frequently in the course of the temperature change, we could over-reduce the offloading intensity. Therefore, in our mechanism, we intentionally delay the PCU updates so that the number of PIM-enabled warps will be updated only after the HMC temperature has been settled accordingly. In this way, we address the over-reduction issue and also avoid the performance overhead caused by frequent PCU updates.

Dynamic PIM Instruction Translation: Because all PIM instructions in HMC 2.0 and the extended instructions proposed in [23] have the corresponding CUDA instructions, each PIM instruction is dynamically translated to a regular CUDA instruction according to the PIM-enable/disable condition. As shown in Table III, all PIM instructions have a corresponding CUDA instruction, and thus can be interpreted as regular non-PIM instructions during the decoding process in the frontend.

D. Discussion

Feedback-control Granularity: Our proposed methods follow a closed-loop feedback control mechanism to keep the HMC temperature within a proper range. Thermal warning messages will trigger source throttling in the corresponding software or hardware components. SW-DynT reduces the number of PIM-enabled CUDA blocks, whereas HW-DynT reduces the number of PIM-enabled warps. However, source throttling does not lead to an immediate reduction of PIM offloading intensity. To model this behavior, as shown in Figure 8, we introduced a delay of $T_{throttle}$ (different values for SW-DynT and HW-DynT). Similarly, HMC temperature is sensed with an extra delay of $T_{thermal}$ after offloading intensity is changed. Therefore, the granularity of our feedback control cannot exceed the total delay of this loop, which is $T_{throttle} + T_{thermal}$. For instance, if an application requires N control steps for achieving its optimal HMC temperature, its overall control delay would be $N \times (T_{throttle} + T_{thermal})$.



	Software-based	Hardware-based
$T_{throttle}$ (Source throttling delay)	~0.1ms	~0.1us
$T_{thermal}$ (Thermal delay)	~1ms	~1ms

Fig. 8. Delay time in our feedback control

Software- vs. Hardware-based: CoolPIM presents software- and hardware-based techniques for dynamic source throttling. Although both follow similar feedback control mechanisms, they have differences in a number of aspects and introduce interesting trade-offs. 1) *Control delay:* As explained previously, our mechanism takes a delay of $T_{throttle}+T_{thermal}$ for each control step. While for HW-DynT the source throttling delay, $T_{throttle}$, takes only tens of cycles, SW-DynT has a much longer $T_{throttle}$. This is because of the overhead of interrupt handling and the delay for waiting the execution of ongoing CUDA blocks. For GPU kernels with short execution time, despite of the longer $T_{throttle}$ in SW-DynT, the thermal response time, $T_{thermal}$, is still the major delay bottleneck. Thus, both HW-DynT and SW-DynT would have a similar control delay. However, for long-execution kernels, $T_{throttle}$ is comparable or even larger than $T_{thermal}$. In this case, the per-step control delay of HW-DynT would be significantly shorter than SW-DynT. 2) *Initialization:* PTP initialization plays an important role in defining the total number of control steps before reaching to the optimal temperature. Since SW-DynT has a long control delay, it relies on a proper initialization to reduce the number of overall control steps. For SW-DynT, we perform static analysis on target applications. Although SW-DynT is only a one-time effort, it still adds an extra step for software compiling. On the contrary, since HW-DynT has a fast control reaction, it still can reach to a proper PIM offloading intensity within a short amount of time without any special initialization. 3) *Complexity:* Our software-based technique utilizes the existing PIM hardware and requires only non-intrusive modifications of the software runtime. However, to achieve fine-grained and fast control of PIM offloading, the hardware-based technique requires an extra hardware component in each GPU core, which adds non-trivial modifications compared to the software-based technique.

V. EVALUATION

A. Evaluation Methodology

Figure 9 shows an overview of our evaluation infrastructure. First, we measured the temperature of a real HMC 1.0 using a thermal camera. The results are used for validating our thermal modeling environment. Then, we model an HMC 2.0 system based on the specification and the power/area numbers derived from the Synopsys tools. Finally, we estimate the system performance by performing timing simulations together with our thermal models.

Power Estimation: To measure power consumption of the functional units (FUs) in HMC 2.0, we design a fixed-point functional unit in Verilog and synthesize it with Synopsys tools using a 28 nm CMOS library [35]. For DRAM and logic dies, because no public information about the design details is released, we use the energy numbers reported in prior literature from Micron [14]. Average energy consumptions per

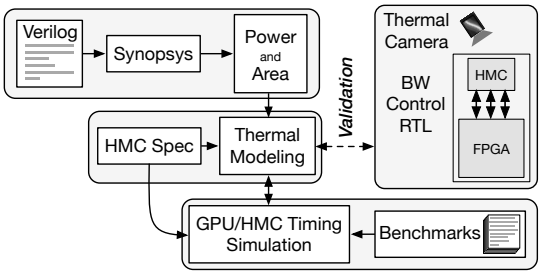


Fig. 9. Overview of evaluation infrastructure

bit are 3.7 pJ/bit for DRAM layers and 6.78 pJ/bit for the logic layer. We then estimate the power consumption according to the bandwidth utilization (i.e., $power = energy/bit \times bandwidth$).

Area Estimation: HMC 1.1 die size is reported to be 68 mm^2 [14]. Assuming the die is evenly partitioned into 16 vaults, each vault occupies 4.25 mm^2. Since no VLSI-level information of HMC 2.0 is released, we assume that HMC 2.0 occupies the same per-vault area as HMC 1.1. Similarly, we also use the same 28 nm CMOS process for the logic layer and 50 nm process for the DRAM layers. In each vault, we place a vault controller and a functional unit at the center. In our synthesis, the functional unit occupies an area of 0.003 mm^2. We also use the area of a vault controller synthesized in 28 nm CMOS from [3].

Thermal Modeling: To model the temperature of the HMC, we use KitFox [32], which is an integrated power, thermal, and reliability modeling framework. In KitFox, we use 3D-ICE [33] for a detailed thermal analysis of the 3D-stacked architecture. By following the specification, we model an 8 GB HMC 2.0 which consists of a bottom logic layer and eight DRAM dies stacked on top. We assume that a commodity-server cooling capability is applied. Also, we use the same process technology, floorplan, and power consumption as previously explained in this section.

Performance Evaluation: We evaluate the performance of our proposed technique by performing detailed timing simulation because HMC 2.0 hardware is not publicly available. We use the Structural Simulation Toolkit (SST) [30] as a simulation framework, and MacSim [1], a cycle-level architecture simulator, for host-side GPU simulation. We also use VaultSim, an in-house 3D-stacked memory simulator, to model an HMC architecture. As explained in Section III, for the thermal impact on HMC performance, we partition the HMC operating temperature into three phases, 0 °C-85 °C, 85 °C-95 °C, and 95 °C-105 °C, and assume a 20% DRAM frequency reduction when switching to a higher temperature phase. Table IV summarizes the configuration of our evaluations.

TABLE IV
PERFORMANCE EVALUATION CONFIGURATIONS

Component	Configuration
Host	GPU, 16 PTX SMs, 32 threads/warp, 1.4GHz 16KB private L1D and 1MB 16-way L2 cache
HMC	8 GB cube, 1 logic die, 8 DRAM dies 32 vaults, 512 DRAM banks [4] $t_{CL} = t_{RCD} = t_{RP} = 13.75\ ns,\ t_{RAS} = 27.5\ ns$ [17]
	4 links per package, 120 GB/s per link [4] 80 GB/s data bandwidth per link
	DRAM Temp. phase: 0-85 °C, 85-95 °C, 95-105 °C 20% DRAM freq reduction (high temp. phases)
Benchmark	GraphBIG benchmark suite [24] LDBC graph dataset [7]

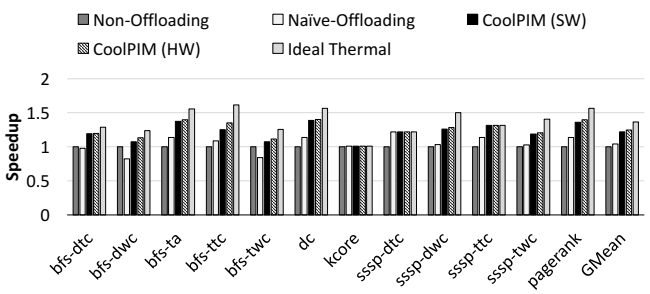


Fig. 10. Speedup over the baseline system without PIM offloading

B. Evaluation Results

In this section, we evaluate our proposed CoolPIM technique with four system configurations, which are explained as follows. In our evaluation, all speedup results are compared with the baseline system unless otherwise stated.

- *Non-Offloading (Baseline):* This is a conventional architecture using HMC as GPU memory and does not utilize PIM offloading functionality.
- *Naïve Offloading:* This configuration follows a PIM offloading mechanism similar to the previously proposed PEI [2]. PIM offloading is enabled for all thread blocks without any source control. The HMC uses a commodity-server active heat sink.
- *CoolPIM:* This is our proposed thermal-aware source throttling method, in which we keep the HMC in a cool temperature range by performing source throttling. We compare both CoolPIM (SW), which is a software-based dynamic throttling method (SW-DynT), and CoolPIM (HW), which is a hardware-based dynamic throttling method (HW-DynT). Similarly, a commodity-sever active heat sink is applied on the HMC.
- *Ideal Thermal:* This is the scenario, in which the HMC has an unlimited cooling capability and remains at a low temperature regardless of PIM offloading intensity.

1) Performance Evaluation: Figure 10 shows the performance results of our proposed technique. Compared with the non-offloading scenario, both software-based and hardware-based technique achieves over a 1.3× speedup for `dc`, `bfs-ta`, and `pagerank`. On average, CoolPIM improves performance by 21% for software-based method and 25% for hardware-based method over the baseline. On the contrary, instead of providing performance benefits, naïve offloading leads to performance degradation for `bfs-dwc` and `bfs-twc` by 18% and 16% respectively. Except for `sssp-dtc`, all benchmarks show only negligible or even negative performance improvements over the baseline for naïve offloading. Both `kcore` and `sssp-dtc` show the same speedup for native-offloading and CoolPIM scenarios. This is because the PIM offloading intensity is low for those workloads, which does not trigger the thermal issue of the HMC. In addition, we can see that in the ideal thermal scenario, PIM offloading

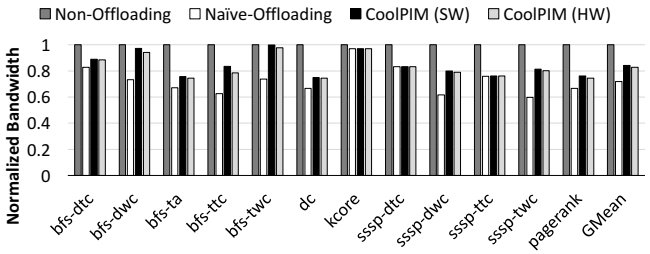


Fig. 11. Bandwidth consumption normalized to the non-offloading baseline

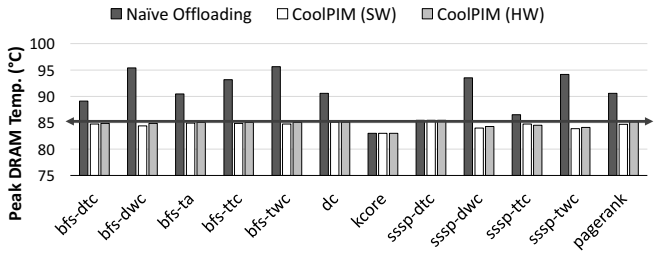


Fig. 13. Peak DRAM temperature

has a great performance potential, providing a performance improvement up to 61% and 36% on average. However, such performance benefits can only be achieved with an unrealistic cooling capability, which is not applicable in a realistic system because of the non-trivial power and space overhead.

Our performance evaluation demonstrates that although PIM offloading shows a large performance benefit in the ideal thermal scenario, such performance benefits will be completely offset because of the memory slowdown triggered by the thermal issue. By controlling PIM offloading intensity from the source side, CoolPIM balances the trade-off between performance and thermal awareness.

2) Bandwidth-Savings Analysis: Most of the GPU applications are bandwidth sensitive; therefore, the major source of benefits from PIM offloading is from bandwidth savings. Bandwidth savings improves not only energy efficiency of data movement, but also performance. Thus, it is usually an intuitive assumption that a larger bandwidth savings will lead to better performance. However, because of the thermal issue of PIM offloading, we observe quite different behaviors.

Figure 11 shows the bandwidth consumption of each workload normalized to the non-offloading baseline system. Naïve offloading reduces bandwidth by 39% for `sssp-dwc`, while CoolPIM (HW) only reduces bandwidth by 21% for the same benchmark. However, as shown in Figure 10, naïve offloading receives only a negligible performance improvement over the baseline, while CoolPIM achieves a 1.28× speedup. We can also observe the same behavior for all other benchmarks except `kcore` and `sssp-dtc`. This is because these benchmarks have a low PIM offloading intensity; therefore, offloading does not trigger a thermal issue. Figure 11 demonstrates that although naïve offloading enables high bandwidth savings, it instead bring performance degradation because of the thermal issues caused by the offloading.

3) Thermal Analysis: As explained in Section III, the temperature of an HMC depends on the utilization of its bandwidth and intensity of PIM offloading. Because the graph computing benchmarks used in our evaluation are bandwidth saturated, the PIM offloading rate between the scenarios is the main deciding factor in determining the peak DRAM temperature for the HMC. Figure 12 shows the average PIM offloading rate with a naïve-offloading method and with our proposed CoolPIM mechanisms. Without any source throttling, naïve offloading reaches close to 4 op/ns (PIM operations per nanosecond) for `bfs-dwc` and `bfs-twc`, and more than 3 op/ns for other benchmarks, such as `bfs-ttc`, `sssp-dwc`, and `sssp-twc`. Such a high PIM offloading intensity would lead to a high temperature of the HMC and trigger the thermal issue. However, source throttling of CoolPIM keeps the PIM offloading rate below 1.3 op/ns for all benchmarks. The results demonstrate the effectiveness of our proposed method, which successfully keeps the PIM offloading intensity within a desirable range. Figure 13 shows the thermal evaluation results. In the results, with the naïve-offloading method, the peak DRAM temperature exceeds 90 °C for most benchmarks. Some of them, such as `bfs-dwc` and `bfs-twc`, even reach 95 °C. However, with our proposed CoolPIM, all benchmarks maintain below 85 °C, keeping the HMC at a cool state.

4) Software- vs. Hardware-based Analysis: As explained in Section IV-D, the software-based dynamic throttling usually introduces much longer throttling delay than the hardware-based method. However, because of the long thermal response time, such difference in throttling delay may not be significant in the overall control delay. To evaluate its impact, we also analyze the PIM rate over time during our experiments. We sample the target benchmarks at the granularity of one millisecond and compare the PIM rate variations of software- and hardware-based methods. In our results, we observe only sub-millisecond difference in the overall control delay. To better illustrate the observation, Figure 14 shows the PIM rate variations over time for bfs-ta benchmark. We select this benchmark because of its relatively larger delay difference and longer execution time. As shown in Figure 14, naïve offloading maintains extremely high PIM offloading rate with only small variations. However, both software- and hardware-based CoolPIM methods successfully control the PIM rate and keep it within a proper range. Although software-based method consumes close-to one more millisecond than hardware-based

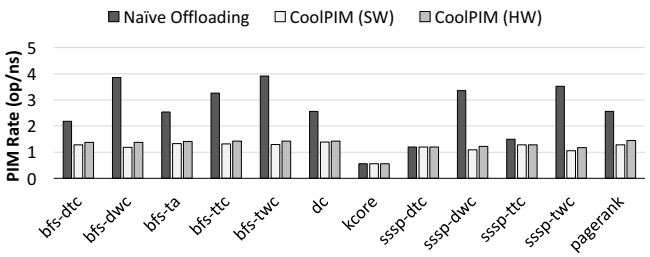


Fig. 12. Comparison of average PIM offloading rate

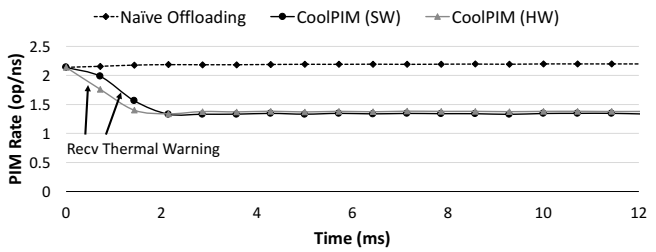


Fig. 14. PIM rate variation over time with software and hardware controls

one, it still takes only trivial time compared to the millisecond-level thermal response time and the long total execution time.

VI. Related Work

Processing-in-Memory: Recent advances in 3D-stacking technology reinitiated the PIM study. Industry has proposed several PIM designs, such as HMC [14], [26], High Bandwidth Memory (HBM) [19], and Active Memory Cube (AMC) [25]. Academic researchers have proposed multiple fully-programmable PIMs for various applications and platforms [8], [16]. In addition, PIM taxonomy also includes fixed-function PIMs [11], among which HMC is one of the examples of industrial proposals. Examples also include PIM-enabled instructions [2] and GraphPIM [23]

Thermal Analysis: 3D-stacking exposes more severe thermal challenges. Dragomir et al. conducted a thermal analysis on a 3D server-on-chip architecture, and observed a 175-200 °C chip hotspot within a 20 W power consumption [22]. Yasuko et al. examined the thermal feasibility of die-stacked PIM [5]. with various cooling methods. Moreover, Hadidi et al., in an HMC characterization study, observed HMC shutdown around 85 °C [12].

Temperature Impact on DRAM: Temperature affects the refresh interval and latency of DRAM cells. Guan et al. analyze the temperature impact on DRAM refresh rate and propose a temperature aware refresh mechanism for 3D-ICs [10], which doubles the refresh rate for every 10 °C increment after 85 °C. DDR4 DRAMs also incorporate a similar doubled refresh rate at 85-95 °C [21]. Lee et al. exploit the temperature impact on DRAM latency by profiling real DRAM DIMMs and propose an adaptive-latency DRAM [18], which can reduce the most critical timing parameters by a minimum/maximum of 17.3%/54.8% at 55 °C.

VII. Conclusion

Processing-in-memory (PIM) offers promising performance and energy gains, but its thermal constraints can prevent applications from benefiting its full potential. To understand the thermal impact of PIM, this paper performs an analysis on a prototype of HMC across a wide range of bandwidth utilization and cooling solutions. Our results show that naïvely using PIM offloading functionality causes a thermal bottleneck and degrades system performance even compared to the non-offloading case depending on the workloads. Based on our findings, we propose CoolPIM, a source throttling technique that controls PIM offloading intensity to keep the operating temperature in check. CoolPIM presents both hardware and software mechanisms to overcome the thermal bottleneck in HMC. The hardware method provides fast feedback reaction and fine-grained control, while the software method requires only non-intrusive changes in software runtime and compiler. Compared to the non-offloading and naïve offloading systems, CoolPIM improves perfomance up to 40% and 37% for a set of graph workloads by effectively managing thermal constraints.

References

[1] "MacSim," http://code.google.com/p/macsim/.
[2] J. Ahn *et al.*, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," ser. ISCA'15.
[3] E. Azarkhish *et al.*, "High Performance AXI-4.0 Based Interconnect for Extensible Smart Memory Cubes," ser. DATE'15.
[4] H. M. C. Consortium, "Hybrid Memory Cube Specification 2.0," *Retrieved from hybridmemorycube.org*.
[5] Y. Eckert *et al.*, "Thermal Feasibility of Die-Stacked Processing in Memory," ser. WoNDP'14.
[6] L. P. Eric Bogatin, Dick Potter, *Roadmaps of Packaging Technology*. Integrated Circuit Engineering Corporation, 1997.
[7] O. Erling *et al.*, "The LDBC Social Network Benchmark: Interactive Workload," ser. SIGMOD'15.
[8] M. Gao *et al.*, "Practical Near-Data Processing for In-Memory Analytics Frameworks," ser. PACT'15.
[9] M. Gokhale *et al.*, "Processing in Memory: the Terasys Massively Parallel PIM Array," ser. IEEE Computer.
[10] M. Guan and L. Wang, *VLSI'16*.
[11] G. H. *et al.*, "A Processing-in-Memory Taxonomy and a Case for Studying Fixed-Function PIM," ser. WoNDP'13.
[12] R. Hadidi *et al.*, "Demystifying the Characteristics of 3D-Stacked Memories: A Case Study for Hybrid Memory Cube."
[13] HMC Consortium, "Hybrid Memory Cube Specification 1.1," *Retrieved from hybridmemorycube.org*.
[14] J. Jeddeloh and B. Keeth, "Hybrid Memory Cube New DRAM Architecture Increases Density and Performance," ser. VLSIT'12.
[15] JEDEC, "DDR4 SDRAM Specification," 2013.
[16] D. Kim *et al.*, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," ser. ISCA'16.
[17] G. Kim *et al.*, "Memory-Centric System Interconnect Design with Hybrid Memory Cubes," ser. PACT '13.
[18] D. Lee *et al.*, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," ser. HPCA'15.
[19] D. U. Lee *et al.*, "25.2 A 1.2V 8Gb 8-Channel 128GB/s High-Bandwidth Memory (HBM) Stacked DRAM with Effective Microbump I/O Test Methods Using 29nm Process and TSV," ser. ISSCC'14.
[20] J. Liu *et al.*, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ser. ISCA'12.
[21] Micron, "4Gb: x4, x8, x16 DDR4 SDRAM Features," https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr4/4gb_ddr4_sdram.pdf.
[22] D. Milojevic *et al.*, "Thermal Characterization of Cloud Workloads on a Power-Efficient Server-on-Chip," ser. ICCD'12.
[23] L. Nai *et al.*, "GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks."
[24] L. Nai *et al.*, "GraphBIG: Understanding Graph Computing in the Context of Industrial Solutions," ser. SC'15.
[25] R. Nair *et al.*, "Active Memory Cube: A Processing-in-Memory Architecture for Exascale Systems," *IBM Journal of Research and Development*.
[26] J. T. Pawlowski, "Hybrid Memory Cube (HMC)," ser. Hot Chips 23.
[27] Picocomputing, "AC-510 HPC Module," http://picocomputing.com/ac-510-superprocessor-module/.
[28] Picocomputing, "EX700 Backplane," http://picocomputing.com/products/backplanes/ex-700/.
[29] Picocomputing, "SC6-Mini," http://picocomputing.com/products/picocube/picomini/.
[30] A. F. Rodrigues *et al.*, "The Structural Simulation Toolkit."
[31] Samsung, "The Future of Graphic and Mobile Memory for New Applications," Hot Chips 28 Tutorial.
[32] W. J. Song *et al.*, "KitFox: Multi-Physics Libraries for Integrated Power, Thermal, and Reliability Simulations of Multicore Microarchitecture," *IEEE Trans. on Component, Packaging, and Manufacturing Technology*.
[33] A. Sridhar *et al.*, "3D-ICE: Fast Compact Transient Thermal Modeling for 3D ICs with Inter-Tier Liquid Cooling."
[34] J. Stein and M. M. Hydeman, "Development and Testing of the Characteristic Curve Fan Model," *ASHRAE Transactions*.
[35] synopsys, "Synopsys 32/28nm Generic Library," https://www.synopsys.com/COMMUNITY/UNIVERSITYPROGRAM/Pages/32-28nm-generic-library.aspx, 2016.
[36] P. Teertstra *et al.*, "Analytical Forced Convection Modeling of Plate FIN Heat Sinks."