Low-power Near-data Instruction Execution Leveraging Opcode-based Timing Analysis

TZIOUVARAS ATHANASIOS, University of Thessaly, Department of Computer Engineering, Greece

DIMITRIOU GEORGIOS, University of Thessaly, Department of Computer Science, Greece

STAMOULIS GEORGIOS, University of Thessaly, Department of Computer Engineering, Greece

Traditional processor architectures utilize an external DRAM for data storage, while they also operate under worst-case timing constraints. Such designs are heavily constrained by the delay costs of the data transfer between the core pipeline and the DRAM, and they are incapable of exploiting the timing variations of their pipeline stages. In this work, we focus on a near-data processing methodology combined with a novel timing analysis technique that enables the adaptive frequency scaling of the core clock and boosts the performance of low-power designs. We propose a near-data processing and better-than-worst-case co-design methodology to efficiently move the instruction execution to the DRAM side and, at the same time, to allow the pipeline to operate at higher clock frequencies compared to the worst-case approach. To this end, we develop a timing analysis technique, which evaluates the timing requirements of individual instructions and we dynamically scale the clock frequency, according to the instructions types that currently occupy the pipeline. We evaluate the proposed methodology on six different RISC-V post-layout implementations using an HMC DRAM to enable the processing-in-memory (PIM) process. Results indicate an average speedup factor of 1.96× with a 1.6× reduction in energy consumption compared to a standard RISC-V PIM baseline implementation.

CCS Concepts: • **Hardware** → **Timing analysis**; *Logic synthesis*; **Hardware accelerators**; 3D integrated circuits; • **Computer systems organization** → *System on a chip*;

Additional Key Words and Phrases: Near-data processing, hybrid memory cube, timing analysis, better-than-worst case design, adaptive clock scaling

ACM Reference format:

Tziouvaras Athanasios, Dimitriou Georgios, and Stamoulis Georgios. 2022. Low-power Near-data Instruction Execution Leveraging Opcode-based Timing Analysis. *ACM Trans. Arch. Code Optim.* 19, 2, Article 19 (January 2022), 26 pages.

<https://doi.org/10.1145/3504005>

1 INTRODUCTION

Modern processor architectures face a scaling wall as the bottleneck is shifting from the core pipeline to the communication between the CPU and DRAM dies. Such a bottleneck influences

Authors' addresses: T. Athanasios and S. Georgios, University of Thessaly, Department of Computer Engineering, Volos, Greece, 38221; emails: attziouv@inf.uth.gr, georges@uth.gr; D. Georgios, University of Thessaly, Department of Computer Science, Lamia, Greece, 35100; email: dimitriu@uth.gr.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Association for Computing Machinery.

1544-3566/2022/01-ART19 \$15.00

<https://doi.org/10.1145/3504005>

the performance and energy consumption of the integrated circuits, as the CPU-DRAM communication imposes major penalties on the designs. To address such challenges, researchers utilize low-power and high-performance optimization techniques, some of which are gaining popularity in the past decade. Such techniques often include the **near-data processing (NDP)** and **better-than-worst-case (BTWC)** design paradigms.

The *NDP approach*, also referred as **processing-in-memory (PIM)**, increases the system throughput by migrating the instruction execution closer to the data, i.e., to the DRAM. NDP has been proposed in the past, but due to technology limitations the applicability of such an approach was limited [4]. Nowadays there is a resurgence in NDP research due to the emergence of the **through-silicon vias (TSV)** interconnections and 3D stacked memories, such as the **hybrid memory cube (HMC)** [31], both of which are key enablers for the NDP concept [4]. According to the NDP premise, hardware accelerators are deployed on the DRAM die [38], which are used for accelerating code execution, while the energy costs of data transfer operations are reduced due to their spatial proximity with the DRAM [5]. NDP approaches focus on either application specific designs, such as References [41, 46], or on general-purpose code execution techniques that are targeted for high-performance computer architectures, as in References [2, 13, 21]. Despite the fact that such designs achieve significant performance improvements, they fail to address the needs of low-power general-purpose computing that poses power and energy limitations on both architectural and circuit levels [15].

The *BTWC paradigm* includes a wide variety of techniques, such as the **timing speculation (TS)**, that treat the processor's critical path more flexibly than traditional approaches. In particular, TS aggressively violates critical path restrictions allowing and then correcting any resulting timing errors. This enables researchers to tamper with energy-to-performance tradeoffs and promises to efficiently increase the performance or lower the power consumption of a circuit [47]. BTWC also considers voltage regulation techniques, as in References [3, 26, 34], to adjust the clock frequency or the supply voltage of the circuits to optimize the performance-power tradeoffs [19, 28]. The resulting voltage variability may cause timing errors to incur, and, thus, error-correction mechanisms are employed to detect and correct such violations [11, 35]. Despite the fact that BTWC premise is well aligned with low-power techniques, its potential has not been properly explored within the NDP domain, since existing works focus on either general-purpose [11] or IoT processors [6, 12].

1.1 Contributions

In this work, we present a novel co-design approach that applies the BTWC paradigm to NDP architectures. We consider low-power processor pipelines ideal for PIM due to the area and power limitations of the DRAMs that support NDP. To this end, we build upon our previous work in Reference [44] and we present an opcode-based adaptive clock-scaling technique, that improves performance by executing opcodes in varying speeds, depending on the instruction's opcode. More specifically, we extend the timing analysis technique of Reference [44] for low-power, simple pipelines, and we expand our methodology to fully analyze each timing path of the processor. Further, we develop a methodology that exploits such information to scale the clock frequency, according to the timing needs of the instructions currently occupying the pipeline. In the sequel, we design a PIM core based on NDP principles and we utilize the proposed methodology to perform timing analysis on the design. We base several design choices on the results of such timing analysis methodology and we design our system according to the NDP principles. We proceed in post-layout simulations of the designs to evaluate our technique. Our main contributions can be summarized as follows:

- (1) We propose a novel timing analysis methodology that considers instruction opcodes for performance increase. Our work is aligned with the BTWC approach but it diverges from the existing solutions. Contrary to prior works, our methodology accurately identifies the timing requirements of incoming instructions on a cycle-to-cycle basis. Since we are a priori aware of such timing constraints we do not deploy any error detection or error correction mechanisms, and, thus, the hardware implementation costs are significantly reduced.
- (2) We co-design an low-power general-purpose NDP system with the proposed BTWC timing analysis approach. We implement an NDP architecture of a low-power, simple processor pipeline from the ground up, capable of facilitating our methodology. We design the processor pipeline to utilize the timing information of the executing instructions and we lower the power and area requirements of the system by avoiding complex optimizations such as deep pipelining or out-of-order execution. To the best of our knowledge, there is not any similar work in the existing literature that applies BTWC techniques on the NDP paradigm.
- (3) We propose a low-power, general-purpose design approach within the NDP domain. To align our work with the NDP principles we focus our efforts on the exploration of an architecture-oriented methodology that is suitable for simple, low-power systems. To this end, we utilize the **instruction set architecture (ISA)** of the processor to extract slack path information and we conduct timing analysis in a circuit-agnostic way. Further, we perform a detailed design space exploration to provide insights regarding the applicability of low-power computing in the NDP paradigm.

The rest of this article is organized as follows. Section 2 surveys prior work in this specific topic area, and Section 3 contains a short background on the HMC, whereas Section 4 presents the proposed NDP architecture. In Section 5, we present in detail our opcode-based clock-scaling technique. In Section 6, we elaborate on the BTWC-NDP co-design approach we employ, and in Section 7 we discuss the design implementation process we follow. Finally, Section 8 provides the evaluation of our methodology, and Section 9 gives the conclusion of our work.

2 RELATED WORK

NDP. There are two main sources of NDP relevant prior work organized as *application specific* architectures and *high-performance* computing techniques. Regarding the *application specific* designs, authors in References [24, 37, 41] and [8] propose an NDP framework for **deep neural network (DNN)** and **machine learning (ML)** application acceleration, in which they deploy specialized accelerators on the DRAM-side and they exploit the dataflow parallelism of the ML/DNN applications. In References [1, 23] and [32] researchers explore the data dependencies, computation complexities and design requirements of graph processing applications and they develop graph accelerators near the DRAM. Further, in Reference [30] authors focus on a big data processing NDP architecture and they design a system to accelerate commonly-used database search operations. NDP has also been proposed for MapReduce workloads, which consist of memory intensive tasks that require a significant amount of DRAM accesses [33]. In this work authors employ a multi-core approach where each core operates in a separate 3D-stacked die and executes Map operations without hitting the bandwidth wall. Previous research in Reference [46] integrates compute-logic on the DRAM die to accelerate sparse matrix multiplication operations by exploring high memory bandwidth and efficient data mapping to account for irregular memory access patterns. Another work in Reference [36] considers NDP for automata and regular expression execution by deploying dedicated hardware to accelerate in-memory pattern-matching. Regarding the *high-performance* designs, authors explore an NDP architecture for a wider range of applications such as MapReduce, graph processing and DNN workloads in Reference [16]. Their approach is

based on high-performance techniques and it employs multi-core processing, multi-threading and large-sized cache memories. In Reference [17] researchers explore a general-purpose approach to the NDP domain by employing flexible interconnects within a grid of processing elements that are deployed on an HMC. Their approach is based on a novel routing network that can be reconfigured in real time to deliver high performance for a wide range of workloads. In Reference [21] authors employ a novel data routing technique that enables computation en-route on a large-scale NDP system. Their approach focuses on an efficient instruction offloading technique and their framework introduces vault-level parallelism to improve computation throughput. In Reference [2] authors utilize a number of lightweight cores in conjunction with commodity two-dimensional (2D) DRAMs to explore a general-purpose NDP designs. They manage to implement an NDP execution framework that utilizes the same ISA with the host processor, and, thus, it is fully compatible with existing commercial processors. Previous work in Reference [13] employs a many-core approach combined with commodity 2D DRAM dies to achieve NDP. Authors utilize a large network of functional units capable of executing instructions, and they explore different interconnection patterns to optimize the execution latency of general-purpose workloads. Overall, existing NDP approaches explore either application specific accelerators or high-performance designs, whereas low-power architectures have received no attention so far. We believe a combination of low-power and general-purpose techniques is essential to meet the low-energy constraints of modern computational workloads.

BTWC. We classify the main sources of BTWC literature according to their error handling approach. Thus, we consider *error-resilient* designs that employ error-correction mechanisms to identify and correct timing violations, and *error-free* techniques that do not allow any timing errors to occur. Regarding the *error-resilient* designs, razor [11] is a TS processor that improves the performance-to-power ratio, while also utilizes shadow latch circuits, to detect and correct the errors incurred by voltage alterations. A previous study demonstrates that through a careful application of timing optimizations designers may also obtain significant energy-performance trade-offs at higher architecture levels [3]. In Reference [25], authors employ a dynamic timing slack technique that evaluates the timing requirements of the executing instructions and allows certain timing errors to occur by scaling up the clock frequency. Although such errors are left uncorrected, authors argue that they impose low penalties on specific workload types, such as machine learning applications. In Reference [26] authors explore the voltage-reliability tradeoffs on micro-architectural level by employing a power-aware slack redistribution technique in near-critical timing paths. Data slack exploitation is also proposed in Reference [35] where authors argue that the underutilized portions of the clock periods, can often be as high as half of the clock period. Under this premise, they develop an error-resilient slack scheduling technique for the instructions queued for execution, combined with a bypass network, and they manage to opportunistically exploit the available timing margins. Reference [34] considers an application-based guardbanding approach that regulates the voltage and mitigates the timing errors that result from the clock frequency scaling. Regarding the *error-free* approaches, researchers employ deterministic timing prediction techniques that identify the requirements of processor paths, and, thus, they do not allow any timing errors to occur. Prior works often utilize adaptive clocking mechanisms that dynamically lower the clock frequency in real-time [20] to improve the power efficiency of the system [19]. In Reference [6] authors propose a technique that adapts the clock frequency in run-time to mitigate the effects of high-frequency supply voltage droops and to prevent timing errors. Also in Reference [10] researchers propose an adaptive clocking methodology that relies on **dynamic timing analysis (DTA)** to exploit the timing requirements of the executing instructions. As a result they manage to dynamically adapt the clock frequency of the processor, so that to adjust the corresponding timing margins. Reference [10] employs a dynamic timing slack technique,

Fig. 1. The HMC architecture that consists of vertically organized memory layers that are interconnected using TSV connections that provide high bandwidth data transfer capabilities. The lowest layer is reserved for logic implementation while the rest of the layers facilitate memory banks that store data.

which scales down the supply voltage of the circuit until the following clock edge, or after all the signals have propagated through logic paths to ensure error-free operation [12]. Also in Reference [28] authors employ a low-voltage adaptive clocking scheme that provides a safety-net for circuit operation under minimal voltage margins, and, thus, it reduces the energy consumption of IoT processors. Evidently, existing *error-free* approaches often utilize either DTA or dynamic slack recycling techniques, whereas other timing methodologies are not yet sufficiently explored for the BTWC paradigm.

3 BACKGROUND ON THE HMC

The HMC architecture is widely adopted by NDP designs as previous work in Reference [18] demonstrates, mainly due to the 3D-stacked DRAM layers it employs. Figure 1 depicts the architecture of the HMC DRAM according to the HMC specifications in Reference [31]. HMC is organized in vertically structured memory vaults that consist of DRAM partitions. Each partition facilitates a number of memory banks, and each bank is responsible for storing the DRAM data. The lower DRAM layer does not contain any partitions, instead it is reserved for logic implementation and facilitates vault controllers that manage data operations process between the memory vaults. An HMC DRAM consists of multiple DRAM layers and achieves high internal bandwidth by employing TSVs interconnects, which are vertical links that connect the 3D DRAM stacked layers. In this work, we design an NDP architecture for general-purpose processors, and, thus, we employ the HMC architecture to implement our design.

4 NDP SYSTEM ARCHITECTURE

The proposed NDP architecture consists of a host system and of an HMC DRAM. For the host system we implement a RISC-V processor core that facilitates the core pipeline, whereas the HMC provides DRAM functionalities for the host system and also facilitates the PIM core responsible for the NDP instruction execution. The host processor-HMC accelerator cooperation is frequently employed in NDP systems for general-purpose code execution as in References [4, 17].

4.1 Host System Architecture

The host system architecture is depicted in Figure 2. We employ a high-performance RISC-V BOOM [9] out of order core that supports 4-wide instruction issue width. The RISC-V BOOM architecture is a synthesizable and parameterizable open source RISC-V 9-stage pipeline that includes the following stages: Instruction fetch, **branch prediction (BP)**, instruction decode, reorder buffer

Fig. 2. The host system architecture that consists of the host core and the PIM pre-processing pipelines. The host pipeline is responsible for executing regular instructions while the PIM pre-processing pipeline is charged with fetching and decoding the instructions that are headed to the HMC DRAM for near data execution.

update, instruction dispatch, instruction issue, register file read, execute pipeline, and memory access. We extend the RISC-V ISA to include the necessary functionalities to support the NDP paradigm. Processor ISA extension for NDP is also considered in Reference [2] where authors argue that such an approach provides compatibility with existing processing platforms. To this end, we implement **jump-and-link-PIM (JalPim)**, an instruction that behaves as the original **jump-and-link (Jal)** instruction, and, thus, it triggers a function call. A key difference between Jal and JalPim is that the former initiates a function call that executes on the RISC-V BOOM pipeline, while the latter triggers a function call that executes on the PIM core. When the JalPim is evoked, the ID stage of the BOOM core pipeline issues a stall signal to the rest of the RISC-V pipeline and disables further instruction execution. Then, the PIM pre-processing pipeline is enabled, which is responsible for pre-processing the instructions of the JalPim function before dispatching them to the PIM core. After the JalPim instructions are offloaded to the PIM core, the RISC-V BOOM register file values are fetched in a serialized fashion and they are moved to the PIM core. Similarly, when the PIM core execution finishes, the PIM pipeline collects the register file values of the PIM core and dispatches them back to the BOOM pipeline. Finally, after the register file values are offloaded to the PIM core, the PIM pre-processing pipeline generates a *PIM_start* signal that propagates to the PIM pipeline and triggers the near-data instruction execution. Under this premise, we design the PIM pre-processing pipeline to include the following stages:

PIM-Decode (PD): The PD decodes the instructions of JalPim function and generates the necessary ALU/FPU signals for their PIM execution. We perform the PIM instruction decoding process on the host pipeline instead of the PIM core to reduce the power and area requirements of the PIM core due to the power and area limitations of the HMC logic layer [17, 31].

PIM-Transfer (PT): The PT stage offloads the decoded instructions and the register file values to the PIM core. The PIM core is located at the logic layer of the DRAM, and, thus, the PT utilizes the processor bus to transfer the corresponding values to the PIM core. To speed up the instruction transfer operation, the processor bus is dedicated to the PT transfer once the PT stage commences, and, thus, no other DRAM access operations are allowed.

We employ the PIM pre-processing pipeline and the JalPim instruction to maintain interoperability between the RISC-V BOOM core and the PIM core. Thus, the PIM pre-processing pipeline acts as an abstraction layer that hides the underlying complexity of the PIM core from the user. Also, the PIM core supports the RISC-V ISA without any additional modifications and users are capable of using the RISC-V instructions for NDP processing without requiring any further ISA extension.

Fig. 3. The PIM core architecture that is deployed on the logic layer of the HMC DRAM. The PIM processor consists of a simple, low-power pipeline that supports instruction execution under low-power and small area requirements.

4.2 PIM Core Architecture

Figure 3 depicts the architecture of the PIM core that is implemented on the logic layer of the HMC DRAM and is responsible for executing the instructions of the JalPIM function. We deploy a simple, low-power RISC-V superscalar pipeline capable of satisfying the power and area requirements of the HMC [31]. We opt for a lightweight RISC-V pipeline to stay in the low-power domain and to ensure that our design is well within the power and area budget of the logic layer of the HMC, similarly with previous works in Reference [16, 38]. To balance the power-performance tradeoffs we employ data forwarding functionalities and a 2 issue-width depth while we omit complex performance optimizations. Below we discuss the architectural details of the PIM core.

PIM Instruction Buffer (PIB): PIB is a simple buffer that stores the instructions that are headed for execution at the PIM core. Such instructions are already decoded by the PD stage of the PIM pre-processing pipeline of the host system and they are dispatched to the PIM core via the PT stage through the processor bus. We opt for a simple PIB design to avoid complex structures such as an I-cache, that require more power and area to operate.

PIM Instruction fetch (PIF): The PIM instruction fetch stage utilizes a PIM program counter (PIM-PC) to fetch the next two instructions from the PIB. The instructions are then headed to the PID stage. The PIF is also charged with updating the PIM-PC value accordingly.

PIM Register file (PRF): The PIM core employs a register file module of 32 registers. Although the PRF increases the power consumption on the PIM core, we consider such module integral to the PIM architecture for preserving the interoperability with the RISC-V core pipeline. Thus, registers used by the RISC-V workloads are mapped on the PRF without requiring any further modifications.

PIM Instruction Dispatch (PID): The PID stage reads the required registers from the PRF, performs sign extension and dispatches the instructions to the PEX or PMEM stages along with their corresponding operands. Furthermore, the PID stage implements the branch prediction functionality for the PIM pipeline. The branch predictor circuit functions independently from the BOOM core BP and it is used for PIM instructions only. Thus, the branch prediction of the BOOM core and the PIM pipeline are two independent operations that execute without exchanging any information.

PIM Execute (PEX): The PEX stage is responsible for the instruction execution process, according to the operands received by the PID stage. PEX supports the logical, shift, branch and arithmetic operations of the RISC-V ISA.

PIM Memory access (PMEM): Our design utilizes a two-layered memory hierarchy that consists of a small data cache (D-Cache) and of a main DRAM. PMEM is a two-stage operation that coordinates the memory access operations i.e., the load and store instructions. To this end, PMEM

utilizes a **load/store buffer (LSB)** that temporarily queues the memory read and write requests. Such requests are stored to the buffer until they are handled by either the D-cache or by the HMC DRAM. In the sequel the buffer dispatches the memory requests to the **address translation (ATR)** stage that translates the requests to physical memory addresses. To accelerate the physical address translation we also employ a small **translation look-aside buffer (TLB)**. The memory request requests are sent to the D-Cache and consequently to the HMC DRAM, in case of a D-Cache miss. On the DRAM side, the HMC vault controllers are charged with handling the read/write operations. To avoid complex cache coherence mechanisms, we consider the DRAM regions accessed by the PIM core uncachable as suggested in References [1] and [40].

PIM Write back (PWB): The PWB stage writes back the PEX or PMEM outputs to the corresponding registers in the PRF.

5 TIMING ANALYSIS IN PROCESSOR DATAPATHS

Timing analysis refers to a set of techniques used to analyze timing behavior of a digital circuit and to establish the optimal clock period for the designs. Regular **static timing analysis (STA)** calculates the worst-case delay of the analyzed circuit and reports whether setup or hold violations occur under certain design constraints. However, STA is overly pessimistic, since it considers worst-case delay for each timing path, and, thus, it is quite inefficient for the BTWC paradigm. However, DTA may be used to acquire very accurate timing information about a circuit design. An exhaustive DTA will display every possible timing path of a given circuit, at a very high time completion cost. Although DTA would be more appropriate for BTWC design timing analysis, its excessive time cost renders its usage impossible on large designs.

5.1 The Instruction Path Exhaustive Static Timing Analysis Concept

In this work we propose the **instruction path exhaustive static timing analysis (IPE-STA)** technique, which is inspired by the BTWC approach. We develop a timing analysis model to analyze each individual instruction of the processor with respect to its unique timing requirements. Each supported instruction is characterized by a unique opcode that can be used to identify it from the rest of the instruction pool. To this end, we isolate every datapath an opcode may excite, and we declare the rest of the paths as false i.e., not having any impact on the timing of the design. We repeat this process until we analyze every available instructions. Afterwards we perform STA on each separate datapath group. The results we obtain illustrate the worst-case timing requirements of each opcode, instead of depicting the worst-case timing delay of the circuit. Thus, IPE-STA is a hybrid between STA and DTA. It performs STA on each instruction datapath of the processor ISA, but it also analyzes all possible instructions, giving a DTA flavor to the result. Also, the IPE-STA is not as pessimistic as the regular STA, and it can produce designs that achieve significantly better performance when compared with designs produced by STA.

5.2 IPE-STA Methodology

Algorithm 1 depicts the proposed IPE-STA technique for the timing analysis of a single instruction. Initially we analyze the instruction (I) and identify its opcode (**inst_op**), while ignoring any register or data fields facilitated within the instruction word. In the sequel we set the necessary circuit inputs to fixed voltage values to represent the corresponding instruction opcode field. Then we perform STA, given the designated circuit inputs by propagating any generated control or data signals through the processor pipeline, and, thus, we analyze the timing requirements of each pipeline stage separately. As a result, we obtain timing results that display the timing requirements of each pipeline stage for the affixed instruction opcode. Such analysis utilizes STA to calculate the worst-case delay of the analyzed instruction. During this process, we keep record of the slowest pipeline

ALGORITHM 1: The IPE-STA pseudo-code for a single instruction word. IPE-STA extracts timing information on instruction level by utilizing the unique opcode field to perform STA. The analysis is performed by affixing the opcode bits to fixed values that represent the corresponding instruction.

```

inst_op = I[opcode]
Set fixed voltage values as inputs to form the inst_op
stage_delays = STA[inst_op]
worst_case = max[stage_delays]

```

stage of *inst_op* that characterizes the maximum clock frequency under which the instruction may execute without errors.

In real-time systems, instruction opcodes dynamically change as new values are stored into the pipeline registers at the rising edge of the clock signal. As a result, as new instructions are fetched for execution, inputs that represent the opcode field of each instruction are not constrained to fixed voltage values, but they change at each clock cycle, resulting in an unpredictable transient timing behavior in all pipeline stages after the fetch stage. More specifically, such behavior is observed at the decode stage due to the opcode bits per se, as well as at every subsequent stage due to the generated control signals. For this reason, the behavior discrepancy between our initial IPE-STA concept and that of a real-time system results in timing deviations that should be addressed.

To compensate for the dynamic voltage change in the processor's opcode inputs, we develop an extension of the aforementioned timing analysis methodology. Since our focus is now set on the timing deviations that are created by the dynamic behavior of the instruction opcode field, we treat such field as a bit vector whose length is defined by the ISA. Consequently, we have to take into account every possible bit value transition that leads to the target bit sequence. Normally the amount of every probable bit combinations grows exponentially with the length of the sequence. We consider this approach unsuitable for our needs as its high time cost makes it impossible for practical applications. Furthermore, we aim to develop a methodology that can be used to analyze any ISA, without requiring small instruction opcode fields to operate efficiently. The solution we propose to resolve such issue is based on the observation that, in processor architectures, not every possible bit transition leads to valid bit sequence of the opcode field. More specifically, the number of valid opcode bit combinations is correlated with the number of the instructions supported by the ISA. Thus, instead of analyzing the timing delay of each possible bit sequence transition, we opt to focus on the analysis of each actually probable instruction succession.

Algorithm 2 depicts the proposed solution, which consists of the original methodology as described above, augmented with the dynamic bit change compensation approach we discussed. In this solution, we analyze each instruction's (I_j) timing requirements individually as before, but instead of using fixed voltage values to describe the instruction under examination, we analyze each probable opcode transition that leads to the opcode field of the examined instruction. We denote such transitions as k , and they represent any rising or falling voltage values that may result in that particular bit sequence ($I_k[\text{opcode}] \rightarrow \text{inst_op}_j$). The timing analysis of such cases is studied individually, and we propagate every generated signals through the processor pipeline, as described above. After we complete the timing analysis of an instruction, we store its worst-case delay and we proceed with the analysis of the next instruction, until all supported instructions are exhausted. The method we proposed in this section again uses STA to find the worst-case delay path of each individual instruction. But to achieve an accurate timing result we utilize an exhaustive iterative analysis resembling more that of a DTA method. However, a DTA approach would require significantly more iterations to effectively analyze the timing requirements of each instruction. We discuss the overhead of the IPE-STA and compare it with the DTA overhead in Section 7.

ALGORITHM 2: The IPE-STA pseudo-code for every ISA-supported instruction. The IPE-STA methodology considers the continuous flow of instructions in a processor pipeline, and, thus, it accounts for the dynamic changes of the opcode field.

```

for Each supported instruction  $I_j$  do
  inst_op $_j$  =  $I_j$ [opcode]
  for Each  $k$  opcode transition:  $I_k$ [opcode]  $\rightarrow$  inst_op $_j$  do
    Perform STA as in algorithm-1
  end for
  worst_case[ $j$ ] = max_stage_delay( $k \rightarrow j$ )
end for

```

5.3 IPE-STA Technique versus Standard STA and DTA

It has already been mentioned that the IPE-STA technique lies somewhere between STA and DTA, closer to STA with respect to complexity, but closer to DTA with respect to output quality. We consider IPE-STA as more architectural oriented than regular STA. In regular STA, the analysis is performed at a low level, in an architecture independent fashion. The analyzer does not consider the microarchitecture details of the processor, and the methodology stays consistent within any given processor architecture. In contrast, with IPE-STA, we focus on the system architecture. We utilize the processor ISA to obtain the operation code of each instruction, and then we examine every possible datapath the instruction may excite. According to IPE-STA, different processor architectures require unique approaches, as the microarchitecture of such systems is not identical. Nevertheless, the need for adaptation of the technique to a new design is compensated by the high quality of the analysis results obtained. IPE-STA only considers variations in the opcode field of the instruction word. Apart from such variations, other dynamic changes that could be considered are related with data values obtained either from the instruction word, from the register file, or from the data cache. As we mentioned earlier, it is far too complicated an issue to account for such changes dynamically, and, thus, we do not address them in this work. Instead, we rely on static analysis to account for such changes in a worst-case manner.

6 IPE-STA AND PIM CO-DESIGN

6.1 PIM Core Adaptive Clock Scaling by Opcodes

Since IPE-STA can provide timing information for each instruction separately, we opt to use such information for a clock scaling methodology that is based on the instruction opcode field. To this end, we apply the IPE-STA methodology on the post-layout implementation of the PIM core architecture as described in Section 4. To evaluate our design in different supply voltage options, we implement three different power supply configurations for the PIM core, i.e., 0.72 V, 0.81 V, and 0.99 V. The timing results we obtain for each individual pipeline stage are depicted in Table 1. We observe that the ATR and PEX stages are the most demanding in terms of latency, with the latter also depicting the greater delay variance when compared with the rest of the pipeline stages. This behavior is expected as previous work in Reference [34] depicts, due to the fact that the execute stage contains the largest and most complex logic of the pipeline, and, thus, higher path slacks are expected. The latency variance of the execute stage is attributed to the timing requirements of the supported arithmetic and logic operations. For example, the *addition* operation requires significantly less time than *division*; hence the slack of the execute stage is heavily influenced by the executing instruction types. On the contrary, the same rule does not apply to the rest of the pipeline stages. In this sense, the *addition* and *division* instructions require similar of time to be fetched or decoded. As a result, the implementations depict a significant slack variance among

Table 1. The Timing Delay Distribution of Each Pipeline Stage for Three PIM Implementations (NPE)

Pipeline stage	0.72 V	0.81 V	0.99 V
PIF	0.18–0.11 ns	0.16–0.09 ns	0.13–0.09 ns
PID	0.41–0.35 ns	0.37–0.32 ns	0.33–0.31 ns
PEX	4.7–0.6 ns	4.2–0.5 ns	3.3–0.4 ns
LSB	0.4–0.37 ns	0.37–0.35 ns	0.36–0.35 ns
ATR	2.2–1.1 ns	1.9–1 ns	1.5–0.8 ns
WB	0.25–0.13 ns	0.24–0.1 ns	0.22–0.1 ns

Each implementation operates under different supply voltage, i.e., 0.72 V, 0.81 V, 0.99 V. The PEX and the ATR stages consist of more complex logic than the rest of the pipeline stages, and, thus, they depict the largest slack.

their pipeline stages. Such a behavior is not uncommon, especially for low-power designs, and has also been demonstrated by previous works in References [10, 26, 34]. Under this premise, we focus our attention to the PEX and ATR stages as they can be considered as critical stages due to the timing limitations they impose on the pipeline. Further, we tighten the timing of the PIM core's functional units by deploying a modified PIM core that utilizes pipelined functional units. As a result, time-consuming operations will execute in a pipelined way, and they will require more clock cycles to complete, but with lower slack. As a result the distribution of PEX latencies will be more balanced for designs that employ pipelined functional units. By further analyzing the PEX and ATR stages, we obtain timing results that can be used to classify the instructions into 11 classes:

- The **Logical instruction class**, which includes logical operations such as *and*, *ori* and *xor*.
- The **Shift instruction class**, which includes shift operations such as *shift left logical* or *shift arithmetic*.
- The **Comparison instruction class**, which includes *bit comparison operations*.
- The **Jump instruction class**, which includes jump operations such as *jump register* or *jump*.
- The **Multiplication instruction class**, which includes *integer multiplication operations*.
- The **Division instruction class**, which includes any *integer division operations*.
- The **Other arithmetic instruction class**, which includes all other integer arithmetic operations except for multiplication and division, such as *addition* or *subtraction*.
- The **Memory access instruction class**, which includes any memory access operation such as *load word* or *store byte*.
- The **FP Multiplication instruction class**, which includes *floating point multiplication operations*.
- The **FP Division instruction class**, which includes *floating point division operations*.
- The **Other FP arithmetic instruction class**, which includes all other floating point arithmetic operations except for FP multiplication and division, such as *FP addition* or *subtraction*.

Each PIM instruction class contains a group of instructions that exhibit similar timing requirements. Table 2 depicts the results of the IPE-STA analysis on different combinations of functional unit types and supply voltages. In particular, we track down the slowest pipeline stage of each class in both **non-pipelined execution (NPE)** and **pipelined execution (PE)** PIM core designs. Then, we assign a *worst-case delay* value to each class, which corresponds to the highest instruction delay in the corresponding group. By studying the instruction classes we observe that each pipeline stage presents unique timing requirements depending on the instruction under execution. Further, some pipeline stages may produce error free results, even by utilizing higher clock frequencies than the rest of the stages. Such error-free instruction execution is guaranteed if we manage to

Table 2. The Instruction Classes which Formulate by the Application of the IPE-STA Technique on PIM Implementations

Instruction class	Slowest pipeline stage	0.72 V NPE - PE	0.81 V NPE - PE	0.99 V NPE - PE
Logical	PEX	1.6–0.8 ns	1.4–0.7 ns	1–0.5 ns
Shift	PEX	1.8–0.9 ns	1.5–0.8 ns	1.2–0.6 ns
Comparison	PEX	1.5–0.8 ns	1.3–0.7 ns	1.1–0.6 ns
Jump	PEX	1.2–0.6 ns	1–0.5 ns	0.8–0.4 ns
Multiplication	PEX	3.8–1.9 ns	3.2–1.6 ns	2.7–1.4 ns
Division	PEX	5–2.5 ns	4.2–2.1 ns	3.3–1.7 ns
Other arithmetic	PEX	1.8–0.9 ns	1.5–0.8 ns	1.1–0.6 ns
Memory access	ATR	2.2–1.1 ns	1.9–1 ns	1.5–0.8 ns
FP Division	PEX	4.7–2.4 ns	3.7–1.3 ns	3–1.5 ns
FP multiplication	PEX	3.7–1.9 ns	3.2–1.6 ns	2.2–1.1 ns
Other FP arithmetic	PEX	2.5–1.3 ns	2–1 ns	1.3–0.7 ns

We consider both PE and NPE designs to explore the impact of balanced and non-balanced timing requirements to the IPE-STA methodology.

Fig. 4. An instruction execution instance of the PIM core that illustrates the minimum operational clock period for each pipeline stage. The clock frequency adapts during each cycle to meet the timing requirements of the executing instructions. Thus, the clock period is designated by the slack of the slowest pipeline stage.

satisfy the timing requirements of each individual pipeline stage in respect with the executing instructions.

We deduce that we can dynamically increase the clock frequency of the PIM core during the runtime to achieve higher throughput, while also preserving the error-free instruction execution. To this end, we track every instruction class of the PIM core and we assign a minimum operational clock period to each one, with respect to their timing requirements. In this sense, our approach treats the instruction execution sequence as consecutive opcode alterations, and, thus, we adapt the clock frequency to meet the timing requirements of each instruction. When an instruction with large path delay is identified, we scale down the clock frequency, when the instruction enters the pipeline stage, which would otherwise cause a timing error. Figure 4 shows an execution instance of five instructions on the PIM core 0.99 V NPE implementation. There, we illustrate the maximum delay of each pipeline stage, and we mark the stages that contribute to frequency down-scaling. In this instance, the pipeline under examination may function at lower clock periods during specific time slots according to the worst-case delay of each active pipeline stage. In this sense, the operational clock period is 1.1 ns for the third cycle, 2.7 ns for the fourth and 1.5 ns for the fifth. We mark with black color the PIM core stages with the highest stage delays that force the clock to adapt to meet to their timing requirements.

We should note that the application of the adaptive clock scaling methodology is more efficient to designs with large timing variations within their pipeline stages. In this sense, our approach

Fig. 5. The CCU, which is implemented on the PIM core in the HMC logic layer. The CCU monitors the PIM pipeline and utilizes lookup tables to designate the optimal clock frequency according to the timing requirements of the executing instruction class.

cannot be applied on an well optimized and balanced pipeline that depicts zero timing variations. In modern processors though, designers try to balance out the pipeline timing delays by employing complex or energy costly optimizations. Such optimizations cannot be used at a large scale on an in-memory processing system due to the power and area constrains that are imposed by the logic layer specifications of the 3D RAM. As a result, we opt to employ the proposed adaptive clock scaling technique in such pipelines, i.e., pipelines that are designed under strict constrains that do not allow space for advanced optimizations.

6.2 PIM Core Microarchitecture with Adaptive Clock Scaling

In order for the PIM core to support the adaptive clock scaling mechanism, we design a **clock control unit (CCU)** that manages the dynamic changes in the clock frequency. The CCU utilizes the information that is extracted from the IPE-STA, to decide whether the clock frequency should be changed. Figure 5 depicts the CCU, which is deployed on the HMC logic layer. It consists of an instruction class snooping module, a decision logic circuit, a lookup table and a number of PLLs.

Instruction class snooping: The instruction class snooping circuit monitors the PIM pipeline and tracks down each pipeline stage separately. To boost the circuit's ability to identify the executing instruction classes, we also modify the PD stage of the PIM pre-processing pipeline of the host system. More specifically, we enable the PD stage not only to decode the fetched instructions but also to classify them into the corresponding instruction class they belong to, as mentioned in Section 6. Our designs contain 11 instruction classes, and, thus, 4 additional bits of information are required for each instruction to properly represent its class. Such information is forwarded through the PT stage of the PIM pre-processing pipeline to the PIM core. There, it is stored along with the instruction operands at the PIB and is forwarded on each pipeline register during the instruction execution process. Under this premise, the instruction class snooping circuit may quickly identify the instruction class that currently exists within each PIM pipeline stage.

Decision logic: This module utilizes the information of the instruction classes, which is provided by the snooping module, and designates the clock period of the PIM core for the next clock cycle. To facilitate such functionality, the decision logic uses lookup tables that store the timing requirements of each instruction class according to the IPE-STA analysis. It utilizes such information and generates a control signal to a multiplexer that selects the appropriate PLL for the current clock cycle. Each PLL is running at independent frequencies, and a multiplexer quickly switches between them in a single cycle, resulting in ultra-fast frequency changes as in Reference [34].

6.3 Resolving Architectural Considerations Using a PIM-IPE-STA Co-design Approach

To co-design the NDP system using the IPE-STA methodology and the PIM core, we utilize the IPE-STA timing reports to resolve critical dilemmas that arise within the PIM design process. Such dilemmas concern the decisions for the CCU and PIM core pipeline, as described below:

CCU. A consideration that arises concerns the identification of the instruction classes by the CCU. The CCU requires to quickly identify the instruction timing requirements (instruction class), so that to reduce the delay of the PLL switching mechanism. To tackle such a challenge, we opt to extend the functionalities of the pre-processing pipeline. To this end, we design the pre-processing pipeline to identify and detect the timing requirements of each instruction (during the PD stage) according to the IPE-STA methodology. As a result, each instruction is assigned a 4-bit value that represents the instruction class it belongs to, according to Table 2. Consequently we design the PIM core to be able to read the 4-bit value that represents each instruction's class, and, thus, the CCU may operate using 4-bit information only. Thus we co-design the CCU using both the timing requirements of the supported instructions, as they are extracted by the IPE-STA, and PIM pre-processing pipeline that generates the additional bits of information.

PIM core. The PEX stage for NPE designs depicts large timing variations for different operations. For example the division operation requires considerable more latency, compared with the addition operation. Thus, a question is arises whether we should employ additional hardware optimizations to balance out the timing of the supported PEX operations. In this work, we rely on IPE-STA to identify the corresponding PEX operation and then, we signal the CCU to scale the clock frequency. We opt for this solution instead of applying a fully pipelined PEX approach, which may reduce the latency of the operations but it will also increase the power and area requirements of the design.

7 IMPLEMENTATION

7.1 Parameter Considerations

Table 3 depicts the RISC-V host system and the RISC-V PIM baseline parameters we have used for the implementation process. For the host system we opt for a 64-bit RISC-V BOOM core that consists of 9 pipeline stages and 4-wide instruction issue. The host system employs a branch prediction mechanism and a TLB for fast physical address translation. We utilize the host system as described in Section 4, and, thus, we implement it on a separate die from the HMC DRAM. To properly analyze the performance of the IPE-STA methodology we implement a baseline PIM pipeline on the logic layer of an HMC. More specifically, we utilize a simple in-order core with 6 pipeline stages, 2 issue width, branch prediction capabilities and small I-Cache and D-Caches so that to meet the timing and area constraints of the HMC logic layer. We should note that the PIM baseline does not employ the IPE-STA approach, and, thus, its clock frequency is statically set to 400 MHz to meet the timing requirements of the design and to maintain the power consumption at low levels.

Table 4 includes the design parameters for the PIM core implementations, as well as the parameters of the HMC DRAM. For the PIM cores we employ a super-scalar architecture with an issue width of 2 and we apply the IPE-STA methodology on each of the corresponding design. We also implement 6 different PIM cores that operate in various supply voltages (0.72 V, 0.81 V and 0.99 V) and they facilitate both pipelined (PE) and non-pipelined (NPE) functional units, as described in Section 6.1. We use the following notation for referencing the PIM designs: PIM-1 (0.72 V NPE), PIM-2 (0.72 V PE), PIM-3 (0.81 V NPE), PIM-4 (0.81 V PE), PIM-5 (0.99 V NPE), and PIM-6 (0.99 V PE). The PIM cores operate under an adaptive clock frequency range of 200 MHz–1.5 GHz, depending on the executing instruction timing requirements and on the supply voltage of each PIM implementation. As a result, the PIM designs require an amount of clocks equal to the number of the instruction classes, i.e., 11.

Table 3. The Design Parameters for the Host System and PIM Baseline

Parameters	RISC-V BOOM host pipeline	RISC-V PIM baseline
ISA	RISC-V	RISC-V
Number of cores	1	1
Instruction execution	Out-of-order	In-order
Pipeline width	4	2
Pipeline depth	9	6
Instruction width	64-bits	64-bits
I-cache	4-way, 8 KB, 2cc latency	direct mapped, 1 KB, 1cc latency
D-cache	4-way, 16 KB, 5cc latency	2-way, 1 KB, 1cc latency
Branch prediction	g-share	g-share
BTB size	256 entries	128
TLB size	512 entries	256 entries
Number of Clocks	1	1
Clock frequency	800 MHz	400 MHz
Supply voltage	0.81 V	0.72 V

The RISC-V PIM baseline is implemented on the logic layer of and HMC DRAM, while the host system is deployed on a separate silicon die.

Table 4. The HMC DRAM and the PIM Core Design Parameters

HMC DRAM		IPE-STA PIM core	
Organization	2 GB, 4 layer	ISA	RISC-V
HMC-BOOM Bus channels	8 channels, bidirectional	Number of cores	1
Bus channel	128 bits, 6 cycle delay, pipelined	Branch prediction	g-share
DRAM timing	tCK=1.5 ns, tRAS=12 ns	Pipeline width/depth	2/6
	tRCD=8 ns tCAS=4 ns	Instruction width	64-bits
	tWR=6 ns tRP=8 ns	D-cache	2-way, 1 KB, 1cc latency
HMC data latency	7–22 ns	PIB/LSB/PRF sizes	512/32/32
HMC bandwidth	160 GBps	Clocks	11
		Clock frequency	200 MHz–1.5GHz
		Supply voltage	0.72 V, 0.81 V, 0.99 V

The PIM cores are designed using the IPE-STA methodology and they are implemented on the logic layer of the HMC DRAM.

7.2 Design Toolflow

The design toolflow that is used for the IPE-STA application on the PIM cores is illustrated in Figure 6. We use Verilog HDL to design the PIM core, the Synopsys Design Compiler with NanGate 15-nm Open Cell Library [29] for the PIM core logic synthesis, and the Synopsys IC Compiler for the place and route operations. In the sequel, we apply the IPE-STA technique on the post-layout netlist of each PIM core, as described in Section 6. To this end, we evoke the *case_analysis* option of the Synopsys PrimeTime tool, that can be used to set any circuit inputs at fixed voltage values. Then we perform STA, given the affixed inputs, as we discussed in Section 5.2. The PIM core architecture supports 228 instructions, and, thus, we require 228^2 IPE-STA iterations to properly cover every possible opcode transition as discussed in Section 5. As each IPE-STA iteration requires 1 ms to complete we consider the overall overhead of the IPE-STA technique trivial, especially when compared with the DTA technique that would require $2^{64} - 1$ iterations. After the IPE-STA completes, we extract the instruction classes' timing information to design the CCU as described in Section 6. We then integrate the CCU into the PIM core and we implement the PIM core pipeline on the logic layer of the HMC DRAM. We repeat the sign-off process again to obtain the post-layout netlist of the NDP design and we use the back annotated netlist to evaluate our methodology. The evaluation is conducted by performing cycle-to-cycle gate-level simulations using the back annotated netlist on the ModelSim tool.

Fig. 6. The integration process of the IPE-STA approach to the CAD Toolflow. The IPE-STA methodology takes place during the timing analysis stage and substitutes the standard STA technique.

7.3 Adaptive Clock Scaling with Multiple Clocks

In order for the IPE-STA to function seamlessly, a large amount of clocks is required, and, thus, the synthesis and layout operations should be conducted accordingly. Nonetheless, we identify three major problems that may cause catastrophic consequences for the PIM timing, if left unresolved: The clock skew, the clock jitter and the PLL synchronization phenomena. We address the clock skew and clock jitter issues through an efficient *clock tree synthesis* operation and we manage to resolve the clock synchronization problem via a proper design of the *clock selector module*.

Clock tree synthesis. Regarding the clock skew of each PLL, we implement the clock control unit at the base of the clock tree, and, thus, the design requires only one globally synthesized clock tree for the whole system as depicted in Figure 7(a). More specifically, the total delay of the clock signal for each pipeline stage can be calculated using the following formula: $T_{total} = T_{pll} + T_{cntrl} + T_{prop}$. T_{pll} is the clock skew between the PLLs and the CCU, T_{cntrl} is the amount of time the CCU requires to designate the clock frequency for the following clock cycle and T_{prop} is the clock propagation delay of the clock distribution network of the chip. Our PIM design utilizes one global clock distribution network (H-tree clock synthesis), and, thus, the T_{prop} delay remains the same for each pipeline stage. Moreover, the T_{cntrl} is constant as it represents the delay of the clock control unit, including the PLL multiplexing overhead as described below. In this sense, we have to make sure that the T_{pll} is the same for each available PLL to ensure a uniform clock skew along the integrated circuit. To this end we synthesize and place the PLLs at an equal distance in terms of *nm* from the CCU, so that the T_{pll} remains the same for each clock. As a result, we ensure that every PLL shares the same clock distribution network, and, thus, we provide a uniform clock skew for each pipeline stage while also alleviating any additional energy and routing costs of synthesizing auxiliary clock distribution networks. We should also note that the T_{total} delay is very small as the clock control unit imposes a trivial skew to the clock signal i.e., in the order of magnitude of some picoseconds (*ps*). This behavior is expected if we consider the three simple operations performed by the CCU, namely (a) The read of the instruction class of each pipeline stage that is expressed in 4-bits as discussed in Section 6.2, (b) the utilization of a simple decision logic to infer the optimal clock period, and (c) the simple clock selection operation that utilizes multiplexers and the clock selector module. STA analysis shows that the T_{total} delay is 170 ps, and, thus, the CCU switching can be performed well within the available timing margins. To account

Fig. 7. The (a) clock tree synthesis operation and (b) the synthesis of clock selector module. The clock distribution network ensures a uniform clock skew along the chip area and addresses the clock jitter phenomenon. The clock selector manages to synchronize multiple input PLLs to generate the clock signal for the next cycle.

for the clock jitter phenomenon, we design the PIM implementations so that to be able to operate under a clock uncertainty of 10%. To achieve this we perform re-timing optimizations to relax the timing requirements of the execute stage by the corresponding amount of time.

Clock selector module synthesis. The clock selector is responsible for generating a synchronous clock output using individual PLLs that run at independent frequencies. Figure 7(b) depicts our approach, in which we deploy a delay-locked loop, which can be used to compare the PIM pipeline clock phase with the phase of the input PLL as in Reference [19]. Such comparison is performed by the phase selector that drives the output to a voltage control module. Voltage control generates the corresponding signals to adjust the phase selection process of a multi-phase **all-digital phase-locked loop (ADPLL)**. ADPLL utilizes a fixed-frequency input PLL to generate multiple equally-spaced phases using a digital controlled oscillator and supports real-time clock period modulation, similarly to Reference [12]. In this sense, the ADPLL can adjust the phase of the PLL, while preserving its frequency intact. We assign each PLL to a single clock selector unit, to support cycle-by-cycle clock period adjustment. As a result, we implement 11 clock selectors, since we require 11 PLLs running in different frequencies, as described in Section 6. In the sequel, we multiplex the outputs of each clock selector to designate the PIM pipeline clock signal for the following cycle. Previous works have proven that multiplexing multiple PLLs is possible through a fast adaptive clocking circuit; provided that the PLLs run at independent frequencies [14, 25, 34, 43]. For this process, we utilize the decision logic module that generates the necessary control signals regarding the instructions that currently occupy the pipeline, as described in Section 6.

7.4 Area and Power Budget

The NDP design constraints derive from the limited area and power budget of the HMC logic layer. Table 5 depicts the area and power requirements of the RISC-V BOOM host pipeline, the RISC-V PIM baseline and the PIM core implementations. We utilize a 1-mm² RISC-V BOOM and a 0.004 mm² RISC-V pre-processing pipeline as a host processor with a combined power consumption of 0.2 mW. We should note that the RISC-V PIM pre-processing pipeline imposes less than a 0.3% area and 1% power overhead to the host pipeline. Regarding the PIM core area and power limitations, the HMC memory consortium [31] specifies that the maximum power consumption of the HMC logic is 5 W while the maximum area budget is 7 mm², and, thus, our design operates well within the required budget. Further, the CCU of PIM implementations impose less than 1% power and 0.1% area overhead to the PIM cores. To properly analyze the power and area requirements of the CCU, we should first provide a breakdown of its components. The CCU is composed of an instruction snooping module, a decision logic, a lookup table and a series of multiplexers that manage

Table 5. The Area and Power Requirements of RISC-V Host, RISC-V PIM Baseline, and PIM Core Implementations

Implementation	Area	Power	Clock (MHz)
RISC-V BOOM host pipeline	1 mm ²	0.4 W	800
RISC-V pre-processing	0.0004 mm ²	10 μ W	800
RISC-V PIM baseline	0.05 mm ²	25 mW	400
Clock control unit	600 μ m ²	12 μ W	—
PIM-1	0.03 mm ²	2–8 mW	200–750
PIM-2	0.04 mm ²	4–11 mW	400–750
PIM-3	0.03 mm ²	2.5–12 mW	240–900
PIM-4	0.04 mm ²	5.5–14 mW	470–1 GHz
PIM-5	0.03 mm ²	3–18 mW	300–1.2 GHz
PIM-6	0.04 mm ²	7–23 mW	660–1.5 GHz

Reports are generated via the corresponding CAD tools, after the implementation process completes.

Table 6. The Workload Characterization of Each Kernel Used for the Evaluation Process

Benchmark	Kernel	Type	PIM execution	Instruction count
Leela [7]	K1	AI: Monte Carlo tree search	75%	$>10^{14}$
x264 [7]	K2	Video compression	80%	$>10^{17}$
xz [7]	K3	General data compression	72%	$>10^{15}$
Inception v3 [42]	K4	DNN training	78%	$>10^{23}$
XML Parsing [39]	K5	Parser	73%	$>10^{13}$
Azure Table Lookup [39]	K6	I/O	75%	$>10^{13}$
K-means [22]	K7	Machine learning	76%	$>10^{16}$
PageRank [22]	K8	Web search engine	74%	$>10^{12}$
FFT [45]	K9	Signal processing	77%	$>10^{14}$
Connected Component [45]	K10	Graph processing	73%	$>10^{13}$

the PLL selection. The decision logic and the multiplexers contribute trivially to the power and area requirements of the CCU due to their small size and simple logic. The instruction snooping module monitors the PIM pipeline and detects each instruction's class, which is represented using a 4-bit field, as discussed in Section 6.2. Using such an approach we omit the need to detect the instruction's opcode; instead we utilize the class information only as it is derived from the PIM pre-processing pipeline (in Section 4.1). To reduce the complexity and size of the lookup table, we use it to store only the instruction class timing requirements, instead of using it to store individual instruction timing requirements. The PIM designs we employ contain 11 instruction classes, and, thus, 11 lookup table entries are sufficient for our methodology to properly function. Consequently, to use the IPE-STA methodology on larger systems we only need to track down the timing requirements of the processor's instruction classes. Given the fact that the amount of such classes should be significantly lower than the total amount of supported instructions, we conclude that the scalability of the proposed approach to more complex systems is guaranteed. Furthermore, the small size of the PIM pipeline is attributed to the lack of complex control logic, to the absence of an I-Cache and to the small size of D-Cache. Table 5 also depicts the operating clock frequencies for each PIM implementation, which are extracted by the IPE-STA analysis.

8 EVALUATION

8.1 Workload Characterization

We evaluate our designs using several benchmarks that reflect on a wide range of applications. Table 6 depicts the benchmarks used for the evaluation process. Under this premise, we use the specCPU 2017 suite [7], the Google's Inception V3 deep neural network training model [42],

Fig. 8. The speedup of each kernel, normalized to host only-execution, for the PIM core and PIM baseline implementations. PIM cores that utilize the IPE-STA methodology manage to aggressively scale the clock frequency and outperform the PIM baseline by a large margin.

machine learning and I/O benchmarks [22, 39] and big data workloads [45]. We run each benchmark's kernel on the RISC-V BOOM pipeline (for host-only execution), on the RISC-V PIM baseline (for baseline comparison) and on the PIM cores we designed to compare our findings. We also present the percentage of the total instructions that are executed on the PIM cores. Further, we should note that the PIM designs are not suitable of executing software loops that contain a very high instruction count due to the small size of the instruction buffer (PIB), which stores up to 512 instructions. Thus, to run the aforementioned benchmarks on the PIM cores, we utilize the host pipeline to segmentize the kernel loops into smaller loops, by using loop fission techniques as in Reference [27], which can be properly mapped on our designs. To this end, we dispatch the smaller loops that compose the benchmark binaries to the PIM cores, in a serialized fashion, and we collect the results in the host pipeline after each loop execution completes.

8.2 Speedup

Figure 8 illustrates the speedup of each kernel execution on the PIM core and baseline implementations normalized to RISC-V core execution only ($\frac{Host_{execution\ time}}{PIM_{execution\ time}}$). To properly measure each kernel's execution time, we also include in our calculations the PIM pre-processing pipeline's time cost for each kernel. We observe that the kernel speedup factors, over the host-only execution, depend on the characteristics of each workload and on the corresponding PIM core implementation parameters. In this sense, the near-data execution seems to benefit more the applications with large memory overheads such as K2, K4, K7, and K10. This behavior is expected due to the fact that PIM execution does not utilize a data intensive host processor-DRAM communication, and, thus, the processor bus bottleneck is reduced. For memory intensive workloads, NDP achieves $21\times$ – $30\times$ faster execution over the host processor, depending on the requirements of each kernel. Further, PIM designs with greater variance in path slack such as PIM-1, PIM-3, and PIM-5 match the speedup values of the pipelined designs, as the CCU compensates for such variations in timing paths. Further, PIM cores with higher voltage supply such as PIM-5 and PIM-6 outperform the rest of the implementations as they provide more opportunities for aggressive clock scaling. The average speedup factors over the host-only execution range from $20.8\times$ to $26\times$ and they demonstrate the efficiency of the PIM designs in terms of execution latency.

Fig. 9. The performance improvement breakdown of PIM cores for the K3 kernel. The clock scaling technique is more efficient on designs that operate within higher voltage supplies, as they present more opportunities for aggressive clock scaling.

To measure the impact of the IPE-STA technique on each implementation separately, we also provide a RISC-V PIM baseline for comparison, as described in Section 7. We observe that the IPE-STA approach manages to improve the execution time by an average factor of $1.96\times$ over the RISC-V PIM baseline. Further, kernels that provide more opportunities for aggressive clock scaling such as K1, K2, K6, and K10 benefit more from the IPE-STA technique, while the PIM baseline fails to exploit such opportunities. Also, PIM cores that operate under with higher voltage thresholds such as PIM-5 and PIM-6 exploit the timing differences of the executing instructions more efficiently, and, thus, they widen the performance gap between the baseline and PIM core implementations.

8.3 Performance Improvement Breakdown

Figure 9 illustrates the breakdown of the performance improvement of an individual kernel (K3), regarding its execution on the PIM cores. We classify the speedup increase factors into the following categories: (a) The near data execution paradigm, (b) the PIM pre-processing pipeline, and (c) the adaptive clock scaling mechanism. The near data execution paradigm moves the instruction execution to the HMC logic layer (PIM core) instead of the standard CPU execution model, where instructions execute in the host pipeline. In this case, the CPU-DRAM traffic is significantly reduced, since the instructions execute on the PIM cores and the DRAM access requests are generated by the PIM core instead of the core pipeline. This decrease in the bus traffic results in a large performance increase, since the DRAM requests are served internally in the HMC die. More specifically, for the PIM-1 design 52% of the speedup is accounted for the NDP paradigm, while for the PIM-6 implementation the 41% of the speedup increase is attributed to the NDP execution. The PIM pre-processing pipeline is charged with pre-processing the PIM instructions (by decoding and identifying their timing class, on the host side), and, thus it contributes to the achieved speedup. The contribution of the PIM pre-processing pipeline to the performance increase of the PIM cores ranges from 8% (PIM-6) to 10% (PIM-10) of the total speedup. Finally, the adaptive clock scaling mechanism dynamically changes the PIM clock frequency according to the executing instructions' timing requirements. Thus, architectures with higher voltage supply such as PIM-6 present greater opportunities for aggressive clock scaling, whereas architectures that operate under lower voltage margins require a more conservative clock scaling approach. As a result, the proposed approach contributes to the PIM core speedup by a factor 38% for PIM-1, and by a factor of 51% for the PIM-6 implementation.

Fig. 10. The reduction of the energy consumption of the PIM core and PIM baseline implementations, normalized to the RISC-V host pipeline. PIM designs that operate under lower voltage supplies tend to significantly reduce the energy consumption of the system. On the contrary, implementations that operate under higher voltage thresholds achieve marginally better energy consumption than the PIM baseline.

8.4 Energy Consumption

Figure 10 depicts the reduction of energy consumption of each kernel on the PIM cores normalized to the host-only execution ($\frac{Host\ energy\ consumption}{PIM\ energy\ consumption}$). We observe that the PIM core manages to reduce the energy consumption of the workloads from $9\times$ up to $15.3\times$ on average over the host processor. Such a high reduction derives from both NDP and IPE-STA application on PIM-cores. Since the kernel execution is conducted on the HMC logical layer, the large energy overhead of data transmission between the host system and the DRAM is drastically reduced. Further, the IPE-STA manages to overclock the PIM cores, and, thus, reduces the execution time of each benchmark while increases the core's the power consumption due to the aggressive clock scaling mechanism. Since the benchmark execution is accelerated by an average factor of $23\times$ and the power consumption is increased by a factor of $3\times$ – $8\times$ (over the host system), the benefits of the execution speedup outweigh the power costs, and, thus, the energy consumption is reduced. Furthermore, the energy reduction of each kernel fluctuates due to the supply voltage variations, clock scaling opportunities and architectural parameters. More specifically, PIM cores with lower supply voltage such as PIM-1 and PIM-2 tend to reduce the energy consumption by a larger factor, when compared to PIM cores with higher supply voltage such as PIM-5 and PIM-6. Further, designs with pipelined execution units such as PIM-2, PIM-3, and PIM-6 tend to consume more energy than the NPE PIM cores due to the power overhead of the additional pipeline registers. Also, high frequency operating PIM cores such as PIM-5 and PIM-6 achieve larger speedups, and, thus, but they also consume more energy when compared with PIM cores with relatively slower clocks, such as PIM-1, PIM-2, PIM-3, and PIM-4.

We also observe that the PIM cores reduce the energy consumption of the system by an average of $1.2\times$ to $2\times$ over the PIM baseline. This decrease is smaller compared to the reduction obtained over the host system as the PIM baseline is also implemented on the HMC logic layer, and, thus, it also minimizes the energy consumption of the processor-DRAM traffic. Further, in some kernels such as K5 and K8 the PIM baseline consumes almost the same energy with the PIM-6 design. Such behavior is expected due to the high operating voltages (0.99 V) and aggressive clock scaling capabilities of the PIM-6, since the PIM baseline operates in lower voltage conditions. Nonetheless,

Fig. 11. The energy efficiency (a) and area efficiency (b) of the PIM core and PIM baseline implementations, normalized to the RISC-V host pipeline. NPE designs achieve greater area and energy efficiency, since their functional units require less area and more power to operate.

the PIM cores manage to reduce the overall energy requirements of the system due to the faster execution times they achieve when compared to the PIM baseline. Additionally, PIM designs that operate under lower voltage thresholds such as PIM-1 and PIM-2 achieve greater results in terms of energy consumption, and, thus, they fit better in the low-power computing domain.

8.5 Energy Efficiency

Figure 11(a) depicts the energy efficiency in terms of Gops/watt for the RISC-V host, PIM core and PIM baseline implementations. We observe that PIM designs achieve an average 202 Gops/W and are 10×–14× times more energy efficient, compared to the RISC-V host pipeline, while they manage to outperform the PIM baseline by a factor of 2.3× in terms of energy efficiency. Also the performance of PIM designs drops as the supply voltage increases, but even under high voltage thresholds such as PIM-6 they manage to maintain a 160 Gops/W energy efficiency, which we consider significant. Further, PE PIM designs tend to be less energy efficient when compared with the corresponding NPE implementations, since the extra pipeline registers tighten the circuit timing, and, thus, the IPE-STA technique has less opportunities for aggressive clock scaling. Further, the average contribution of the IPE-STA technique to the overall energy efficiency of the NDP designs varies depending on the timing opportunities of each PIM implementation. As a result, designs with larger timing variations tend to benefit more from the adaptive clock scaling technique, while designs with tighter timing achieve a lower but noticeable energy efficiency improvement.

8.6 Area Efficiency

Figure 11(b) depicts the area efficiency in terms of Gops/mm² for the RISC-V host, PIM core and PIM baseline implementations correspondingly. The PIM designs manage to outperform the RISC-V host pipeline by 24.5× times, achieving an average area efficiency of 54 Gops/mm². Also the PIM cores achieve 3× times greater area efficiency on average when compared with the PIM baseline pipeline. Such an area efficiency escalates as the voltage supply increases, since higher supply voltages do not pose additional area requirements; hence they only contribute to the throughput improvement. As a result, the IPE-STA manages to aggressive over-scale the clock frequency while

the area requirements of the designs remain the same. Further, NPE PIM cores such as PIM-1, PIM-3 and PIM-5 are more area efficient compared to the corresponding PE PIM implementations. Such behavior is attributed to the fact that PE implementations require more area to operate while their performance increase, due to the pipelined execution, does not compensate for such increase in area requirements. In conclusion we deduce that both the adaptive clock scaling mechanism and the NDP execution are critical to the area efficiency of the PIM cores.

9 CONCLUSION

In this work we present an NDP design methodology for low-power, simple processor pipelines. Our approach consists of an NDP architecture combined with a novel timing analysis technique inspired by the BTWC paradigm, called IPE-STA. We employ an HMC DRAM that is a 3D stacked DRAM that enables NDP by facilitating logic cells in the lower DRAM layer. We design and implement six different PIM core pipelines based on the RISC-V BOOM processor, by taking into account the HMC requirements while also respecting the low-power design constraints. In the sequel, we analyze each PIM core's timing requirements via the IPE-STA technique. IPE-STA allows us to obtain timing information for each instruction's worst case delay, instead of analyzing the worst case delay of the circuit's critical path. We utilize use such information to design and implement a clock control unit capable of identifying the timing requirements of any upcoming PIM instruction and selecting the appropriate clock frequency so that no timing violations to occur. Such a mechanism is implemented on the PIM cores and is supported by a PIM pre-processing pipeline that is deployed on the host system. We evaluate our methodology in post-layout simulations using general-purpose workloads from a wide range of application fields. Results indicate an execution speedup of $1.96\times$ on average, and an energy reduction factor of $1.6\times$ over a PIM baseline implementation.

REFERENCES

- [1] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA'15)*. 105–117. <https://doi.org/10.1145/2749469.2750386>
- [2] Hadi Asghari-Moghaddam, Amin Farmahini-Farahani, Katherine Morrow, Jung Ho Ahn, and Nam Sung Kim. 2016. Near-DRAM acceleration with single-ISA heterogeneous processing in standard memory modules. *IEEE Micro* 36, 1 (2016), 24–34. <https://doi.org/10.1109/MM.2016.8>
- [3] Omid Azizi, Aqeel Mahesri, Benjamin C. Lee, Sanjay J. Patel, and Mark Horowitz. 2010. Energy-performance trade-offs in processor architecture and circuit design: A marginal cost analysis. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*. Association for Computing Machinery, New York, NY, 26–36. <https://doi.org/10.1145/1815961.1815967>
- [4] Rajeev Balasubramonian, Jichuan Chang, Troy Manning, Jaime H. Moreno, Richard Murphy, Ravi Nair, and Steven Swanson. 2014. Near-data processing: Insights from a MICRO-46 workshop. *IEEE Micro* 34, 4 (2014), 36–42. <https://doi.org/10.1109/MM.2014.55>
- [5] Shekhar Borkar. 2013. Role of interconnects in the future of computing. *J. Lightw. Technol.* 31, 24 (2013), 3927–3933. <https://doi.org/10.1109/JLT.2013.2283277>
- [6] Keith A. Bowman, Sarthak Raina, J. Todd Bridges, Daniel J. Yingling, Hoan H. Nguyen, Brad R. Appel, Yesh N. Kolla, Jihoon Jeong, Francois I. Atallah, and David W. Hansquaine. 2016. A 16 nm all-digital auto-calibrating adaptive clock distribution for supply voltage droop tolerance across a wide operating range. *IEEE J. Solid-State Circ.* 51, 1 (2016), 8–17. <https://doi.org/10.1109/JSSC.2015.2473655>
- [7] James Bucek, Klaus-Dieter Lange, and Jakim v. Kistowski. 2018. SPEC CPU2017: Next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE'18)*. Association for Computing Machinery, New York, NY, 41–42. <https://doi.org/10.1145/3185768.3185771>
- [8] Weidong Cao, Yilong Zhao, Adith Bolor, Yinhe Han, Xuan Zhang, and Li Jiang. 2021. Neural-PIM: Efficient processing-in-memory with neural approximation of peripherals. *IEEE Trans. Comput.* (2021), 1–1. <https://doi.org/10.1109/TC.2021.3122905>

- [9] Christopher Celio, Pi-Feng Chiu, Borivoje Nikolic, David A. Patterson, and Krste Asanović. 2017. *BOOM v2: An Open-source Out-of-order RISC-V Core*. Technical Report UCB/EECS-2017-157. EECS Department, University of California, Berkeley.
- [10] Hari Cherupalli, Rakesh Kumar, and John Sartori. 2016. Exploiting dynamic timing slack for energy efficiency in ultra-low-power embedded systems. In *Proceedings of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA'16)*. 671–681. <https://doi.org/10.1109/ISCA.2016.64>
- [11] D. Ernst, Nam Sung Kim, S. Das, S. Pant, R. Rao, Toan Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. 2003. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*. 7–18. <https://doi.org/10.1109/MICRO.2003.1253179>
- [12] Yuanbo Fan, Simone Campanoni, and Russ Joseph. 2019. Time squeezing for tiny devices. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA'19)*. Association for Computing Machinery, New York, NY, 657–670. <https://doi.org/10.1145/3307650.3322268>
- [13] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. 2015. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In *Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. 283–295. <https://doi.org/10.1109/HPCA.2015.7056040>
- [14] Brian A. Floyd. 2008. Sub-integer frequency synthesis using phase-rotating frequency dividers. *IEEE Trans. Circ. Syst. I: Regul. Pap.* 55, 7 (2008), 1823–1833. <https://doi.org/10.1109/TCSL.2008.918077>
- [15] Pierre-Emmanuel Gaillardon, Edith Beigne, Suzanne Lesecq, and Giovanni De Micheli. 2015. A survey on low-power techniques with emerging technologies: From devices to systems. *J. Emerg. Technol. Comput. Syst.* 12, 2, Article 12 (September 2015), 26 pages. <https://doi.org/10.1145/2714566>
- [16] Mingyu Gao, Grant Ayers, and Christos Kozyrakis. 2015. Practical near-data processing for in-memory analytics frameworks. In *Proceedings of the International Conference on Parallel Architecture and Compilation (PACT'15)*. 113–124. <https://doi.org/10.1109/PACT.2015.22>
- [17] Mingyu Gao and Christos Kozyrakis. 2016. HRL: Efficient and flexible reconfigurable logic for near-data processing. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'16)*. 126–137. <https://doi.org/10.1109/HPCA.2016.7446059>
- [18] Saugata Ghose, Kevin Hsieh, Amirali Boroumand, Rachata Ausavarungnirun, and Onur Mutlu. 2018. Enabling the adoption of processing-in-memory: Challenges, mechanisms, future research directions. arXiv:cs.AR/1802.00320. Retrieved from <https://arxiv.org/abs/1802.00320>.
- [19] Aaron Grenat, Sanjay Pant, Ravinder Rachala, and Samuel Naffziger. 2014. 5.6 adaptive clocking system for improved power efficiency in a 28nm x86-64 microprocessor. In *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC'14)*. 106–107. <https://doi.org/10.1109/ISSCC.2014.6757358>
- [20] Tetsutaro Hashimoto, Yukihiro Kawabe, Michiharu Hara, Yasushi Kakimura, Kunihiko Tajiri, Shinichiro Shirota, Ryuichi Nishiyama, Hitoshi Sakurai, Hiroshi Okano, Yasumoto Tomita, Sugio Satoh, and Hideo Yamashita. 2018. An adaptive-clocking-control circuit with 7.5 frequency gain for SPARC processors. *IEEE J. Solid-State Circ.* 53, 4 (2018), 1028–1037. <https://doi.org/10.1109/JSSC.2017.2777101>
- [21] Jiayi Huang, Pritam Majumder, Sungkeun Kim, Troy Fulton, Ramprakash Reddy Puli, Ki Hwan Yum, and Eun Jung Kim. 2021. Computing en-route for near-data processing. *IEEE Trans. Comput.* 70, 6 (2021), 906–921. <https://doi.org/10.1109/TC.2021.3063378>
- [22] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. 2010. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *Proceedings of the IEEE 26th International Conference on Data Engineering Workshops (ICDEW'10)*. 41–51. <https://doi.org/10.1109/ICDEW.2010.5452747>
- [23] Yu Huang, Long Zheng, Pengcheng Yao, Jieshan Zhao, Xiaofei Liao, Hai Jin, and Jingling Xue. 2020. A heterogeneous PIM hardware-software co-design for energy-efficient graph processing. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'20)*. 684–695. <https://doi.org/10.1109/IPDPS47924.2020.00076>
- [24] Won Jeon, Jiwon Lee, Dongseok Kang, Hongju Kal, and Won Woo Ro. 2021. PIMCaffe: Functional evaluation of a machine learning framework for in-memory neural processing unit. *IEEE Access* 9 (2021), 96629–96640. <https://doi.org/10.1109/ACCESS.2021.3094043>
- [25] Tianyu Jia, Russ Joseph, and Jie Gu. 2019. 19.4 an adaptive clock management scheme exploiting instruction-based dynamic timing slack for a general-purpose graphics processor unit with deep pipeline and out-of-order execution. In *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC'19)*. 318–320. <https://doi.org/10.1109/ISSCC.2019.8662389>
- [26] Andrew B. Kahng, Seokhyeong Kang, Rakesh Kumar, and John Sartori. 2010. Designing a processor from the ground up to allow voltage/reliability tradeoffs. In *Proceedings of the 16th International Symposium on High-Performance Computer Architecture (HPCA'10)*. 1–11. <https://doi.org/10.1109/HPCA.2010.5416652>

- [27] M. Kaul, R. Vemuri, S. Govindarajan, and I. Ouais. 1999. An automated temporal partitioning and loop fission approach for FPGA based reconfigurable synthesis of DSP applications. In *Proceedings of the Design Automation Conference*. 616–622. <https://doi.org/10.1109/DAC.1999.782017>
- [28] Sae Kyu Lee, Paul N. Whatmough, David Brooks, and Gu-Yeon Wei. 2019. A 16-nm always-On DNN processor with adaptive clocking and multi-cycle banked SRAMs. *IEEE J. Solid-State Circ.* 54, 7 (2019), 1982–1992. <https://doi.org/10.1109/JSSC.2019.2913098>
- [29] Mayler Martins, Jody Maick Matos, Renato P. Ribas, André Reis, Guilherme Schlinker, Lucio Rech, and Jens Michelsen. 2015. Open cell library in 15nm FreePDK technology. In *Proceedings of the Symposium on International Symposium on Physical Design (ISPD'15)*. Association for Computing Machinery, New York, NY, 171–178. <https://doi.org/10.1145/2717764.2717783>
- [30] Puya Memarzia, Suprio Ray, and Virendra C. Bhavsar. 2020. The art of efficient in-memory query processing on NUMA systems: A systematic approach. In *Proceedings of the IEEE 36th International Conference on Data Engineering (ICDE'20)*. 781–792. <https://doi.org/10.1109/ICDE48307.2020.00073>
- [31] Hybrid memory cube consortium. 2018. *Hybrid Memory Cube Specification 2.1*. Technical Report. Retrieved from https://www.nuvation.com/sites/default/files/Nuvation-Engineering-Images/Articles/FPGAs-and-HMC/HMC-30G-VSR_HMCC_Specification.pdf.
- [32] Newton, Virendra Singh, and Trevor E. Carlson. 2020. PIM-GraphSCC: PIM-Based graph processing using graph's community structures. *IEEE Comput. Arch. Lett.* 19, 2 (2020), 151–154. <https://doi.org/10.1109/LCA.2020.3039498>
- [33] Seth H. Pugsley, Jeffrey Jestes, Huihui Zhang, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, Alper Buyukto-sunoglu, Al Davis, and Feifei Li. 2014. NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'14)*. 190–200. <https://doi.org/10.1109/ISPASS.2014.6844483>
- [34] Abbas Rahimi, Luca Benini, and Rajesh K. Gupta. 2014. Application-adaptive guardbanding to mitigate static and dynamic variability. *IEEE Trans. Comput.* 63, 9 (2014), 2160–2173. <https://doi.org/10.1109/TC.2013.72>
- [35] Gokul Subramanian Ravi and Mikko Lipasti. 2019. Recycling data slack in out-of-order cores. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'19)*. 545–557. <https://doi.org/10.1109/HPCA.2019.00065>
- [36] Elaheh Sadredini, Reza Rahimi, Mohsen Imani, and Kevin Skadron. 2021. Sunder: Enabling low-overhead and scalable near-data pattern matching acceleration. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'21)*. Association for Computing Machinery, New York, NY, 311–323. <https://doi.org/10.1145/3466752.3480934>
- [37] Fabian Schuike, Michael Schaffner, Frank K. Gürkaynak, and Luca Benini. 2019. A scalable near-memory architecture for training deep neural networks on large in-memory datasets. *IEEE Trans. Comput.* 68, 4 (2019), 484–497. <https://doi.org/10.1109/TC.2018.2876312>
- [38] Marko Scrbak, Mahzabeen Islam, Krishna M. Kavi, Mike Ignatowski, and Nuwan Jayasena. 2017. Exploring the processing-in-memory design space. *J. Syst. Archit.* 75, C (April 2017), 59–67. <https://doi.org/10.1016/j.sysarc.2016.08.001>
- [39] Anshu Shukla, Shilpa Chaturvedi, and Yogesh Simmhan. 2017. RIoT Bench: An IoT benchmark for distributed stream processing systems. *Concurr. Comput.: Pract. Exp.* 29, 21 (October 2017), e4257. <https://doi.org/10.1002/cpe.4257>
- [40] Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, and Albert-Jan Boonstra. 2018. A review of near-memory computing architectures: Opportunities and challenges. In *Proceedings of the 21st Euromicro Conference on Digital System Design (DSD'18)*. 608–617. <https://doi.org/10.1109/DSD.2018.00106>
- [41] Chirag Sudarshan, Taha Soliman, Cecilia De la Parra, Christian Weis, Leonardo Ecco, Matthias Jung, Norbert Wehn, and Andre Guntoro. 2021. A novel DRAM-based process-in-memory architecture and its implementation for CNNs. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASPDAC'21)*. Association for Computing Machinery, New York, NY, 35–42. <https://doi.org/10.1145/3394885.3431522>
- [42] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. 2818–2826. <https://doi.org/10.1109/CVPR.2016.308>
- [43] James Tschanz, Nam Sung Kim, Saurabh Dighe, Jason Howard, Gregory Ruhl, Sriram Vangal, Siva Narendra, Yatin Hoskote, Howard Wilson, Carol Lam, Matthew Shuman, Carlos Tokunaga, Dinesh Somasekhar, Stephen Tang, David Finan, Tanay Karnik, Nitin Borkar, Nasser Kurd, and Vivek De. 2007. Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging. In *Proceedings of the IEEE International Solid-State Circuits Conference*. 292–604. <https://doi.org/10.1109/ISSCC.2007.373409>
- [44] Athanassios Tziouvaras, Georgios Dimitriou, Michael Dossis, and Georgios Stamoulis. 2019. Instruction-flow-based timing analysis in pipelined processors. In *Proceedings of the Panhellenic Conference on Electronics Telecommunications (PACET'19)*. 1–6. <https://doi.org/10.1109/PACET48583.2019.8956266>

- [45] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, Chen Zheng, Gang Lu, Kent Zhan, Xiaona Li, and Bizhu Qiu. 2014. BigDataBench: A big data benchmark suite from internet services. In *Proceedings of the IEEE 20th International Symposium on High Performance Computer Architecture (HPCA'14)*. 488–499. <https://doi.org/10.1109/HPCA.2014.6835958>
- [46] Xinfeng Xie, Zheng Liang, Peng Gu, Abanti Basak, Lei Deng, Ling Liang, Xing Hu, and Yuan Xie. 2021. SpaceA: Sparse matrix vector multiplication on processing-in-memory accelerator. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA'21)*. 570–583. <https://doi.org/10.1109/HPCA51647.2021.00055>
- [47] Rong Ye, Feng Yuan, Jie Zhang, and Qiang Xu. 2015. On the premises and prospects of timing speculation. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'15)*. EDA Consortium, San Jose, CA, USA, 605–608.

Received June 2021; revised November 2021; accepted December 2021