The Bitlet Model

A Deluge of Processing-in-Memory Frameworks and How Analytical Modeling Could Help!

Kunal Korgaonkar[$,*], Ronny Ronen[$], Anupam Chattopadhyay[#], Shahar Kvatinsky[$]
[$]Technion [*]UC San Diego [#]NTU

1 Introduction and Motivation

Processing huge amounts of data on traditional von Neumann architectures involves many data transfers between the CPU and the memory. These transfers degrade performance and consume energy [9]. Enabled by emerging memory technologies, recent processing-in-memory (PIM) solutions show great potential in reducing costly data transfers by performing computations using individual memory cells [1], [7].

Despite the recent resurgence of PIM, it is still very challenging to analyze and quantify the advantages or disadvantages of PIM solutions over other computing paradigms. We believe a useful analytical modeling tool for PIM can play a crucial role. An analytical tool in this context has many potential uses, such as in (i) evaluation of applications mapped to PIM, (ii) comparison of PIM versus traditional architectures, and (iii) analysis of the implications of new memory technologies.

This paper describes an analytical modeling tool called Bitlet that can be used, in a parameterized fashion, to understand the affinity of workloads to processing-in-memory (PIM) as opposed to traditional computing. The tool uncovers interesting trade-offs between operation complexity (cycles required to perform an operation through PIM) and other key parameters, such as system memory bandwidth, data transfer size, the extent of data alignment, and effective memory capacity involved in PIM computations. The name Bitlet reflects PIM's unique bit-by-bit data element processing approach. The model is inspired by past successful analytical models for computing [2], [4] and provides a simple operational view of PIM computations.

In spite of its simplicity, the Bitlet model has already proven useful (publications related to this work [5], [6]). In the future, we intend to extend and refine Bitlet to further increase its utility.

2 The Bitlet Model

We derive a parameterized throughput metric for PIM followed by one for the CPU. The throughput focus is in alignment with the parallelism that the PIM approach offers. We first describe the model and then proceed to explain how to apply it. Throughout the text, 'PIM' is referred to as a framework for processing inside memories.

2.1 Deriving PIM Throughput

We based the PIM side of the Bitlet model on the principle of performing computations using memristive memory arrays, wherein processing occurs inside the memory arrays using a stateful in-memory logic family (*e.g.*, IMPLY [1] and MAGIC [7]).

We derive PIM throughput by considering operation complexity (more factors are discussed in the report [5]).

Operation Complexity. In the Bitlet model, the PIM computations are carried out as a series of NOR operations, applied on the memory cells of a row inside a memristive memory array. Each row of the memory array stores the input data required for processing. A two-input bit NOR gate processes two data bits within the row and stores the output bit in the same row. Any intermediate data are processed similarly. Processing proceeds sequentially in this fashion to produce the final output, which is also stored within the same row. Data processing as per the Bitlet model is best viewed as *row-wise* and *bit-by-bit* within the row of a memory array. We use a default two-input bit NOR gate as the basic logic operation [7], permitting a maximum of two input bits to be processed per memory cycle.

While each row is processed bit-by-bit, the effective throughput of PIM is increased by the inherent parallelism achieved by simultaneous processing of multiple rows inside a memory array and of multiple memory arrays in the system memory. We assume the same computations (*i.e.*, individual operations) applied to a row are also applied in parallel in every cycle across all the rows (ROW) of a memory array. This parallelism is made possible by the 2D structure of the memory arrays and by reuse of the voltage signals used to operate an individual row for all the rows. Although the choice to only process row-wise may seem restrictive, it naturally maximizes the data-level parallelism and hence PIM throughput. Moreover, the multiple memory arrays (MAT) further maximize this parallelism. Finally, the cycle time, CT, of a single basic PIM operation also impacts overall PIM performance. The shorter it is, the faster the processing.

The number of computing cycles required for PIM-based processing number is affected by both the data sizes, as well as operation types (different operations follow a different curve on the graph). With this model, for example, n-bit AND requires $3n$ cycles (*e.g.*, for n=16 bits AND takes 16x3 = 48 cycles), ADD requires $9n$ cycles[1], and multiply (MPY) requires $13n^2-14n$ cycles [3]. We define the **operation complexity** parameter (OC) for a given operation type and data size, as the number of cycles required to process the corresponding data. The throughput of PIM is captured by four parameters: OC, MAT, ROW and CT (see Table 1). The throughput of the system in operations per second can be expressed as:

$$\textbf{Perf-PIM(Op)} = \frac{\textbf{ROW}\times\textbf{MAT}}{\textbf{OC}\times\textbf{CT}}. \quad (1)$$

2.2 Deriving CPU Throughput

For the workload phases we consider, PIM-based computations occur inside the memory arrays, without any data transfers occurring outside the memory arrays. That is, they are limited by operation complexity (and by the data placement and alignment costs, etc. [5]). On the other hand, we assume that the CPU throughput is primarily limited by its usage of external memory-bandwidth (i.e., by the cost of data transfers between the CPU and memory) ignoring the cost of computations and data movements performed within the CPU itself.

Data Transfer. The Bitlet model, therefore, derives the CPU throughput assuming both memory bandwidth between the CPU and the memories, and the amount of data transfer needed to perform an operation, as the primary limiting factors. Large amounts of data being transferred between the CPU and

- *Part of the work done while the author was a student at UCSD[(*)].*

1. ADD can be improved to $7n$ cycles using an algorithmic optimization that uses four-input NOR instead of two-input NOR.

Parameter name	Notation	Value(s)	Type
PIM operation complexity	OC	1 - 32k cycles	Algo.
PIM cycle time	CT	10 ns [8]	Tech.
PIM array dimensions	$ROW \times COL$	1024 x 1024	Tech.
PIM array count	MAT	1k - 16k	Arch.
CPU memory bandwidth	BW	1 to 16 Tbps	Arch.
CPU data in-out bits	DIO	24, 48	Algo.

TABLE 1. *Bitlet model parameters.*

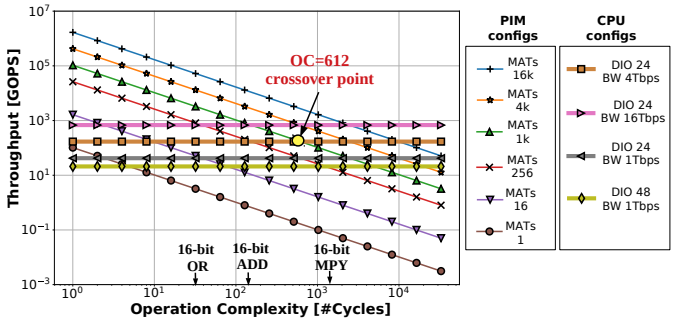

Fig. 1: Throughput comparison of CPU vs. PIM. A crossover point is where the CPU starts performing better than PIM.

the memory result in lower CPU throughput, while smaller volumes produce the opposite effect. The extent of data transfer between the CPU and the memory is captured by the **data in-out** (DIO) model parameter. The DIO is the average amount of data transferred per operation and must account for all the data transfers (in bits) between the CPU and the memory resulting from inputs, outputs, as well as any temporary results. Along with DIO, the external memory bandwidth (denoted as BW) between the CPU and the memory determines the final throughput. The CPU throughput, in operations per second, is defined as:

$$\textbf{Perf-CPU(Op)} = \frac{\textbf{BW}}{\textbf{DIO}}. \quad (2)$$

3 Applying the Bitlet Model

In this section, we apply the Bitlet model to compare PIM to CPU under a parameter design space which includes operation complexity, memory bandwidth, etc.

PIM's throughput is sensitive to various Bitlet model parameters. In this section, a sensitivity study performed to assess these model parameters, highlights some of the strengths and weaknesses of the new PIM paradigm. Fig. 1 shows the throughput of PIM versus that of the CPU and assumes PAC = 0. Diagonal lines represent PIM with varying numbers of $MATs$ (set to 1/16/256/1024/4096/16384 $MATs$). A single 1024×1024 memory array has a 128 KB capacity. Horizontal lines are for CPUs with varying DIO bits (set to 24/48) along with BW = 1Tbps/4Tbps/16Tbps.

Using Eq. 1, we observe that PIM throughput increases with maximum MAT availability, peaking when maximum available memory arrays are used ($MATs$ = 16k), and the operation complexity is the lowest possible (OC = 1). In parallel, PIM throughput decreases with increasing operation complexity. Using Eq. 2, we see that the CPU throughput decreases with higher DIO. For instance, consider the lines shown for DIO = 24 and DIO = 48 for the same BW = 1Tbps. The CPU's performance for DIO = 48 is lower than for DIO = 24.

For a configuration of MAT = 1024, DIO = 24 and BW = 4Tbps, the CPU performs better than PIM at OC = **612** or higher. This marks the crossover point and sets the boundaries of a favorable region for PIM for this configuration. Note the placement of the OR, AND and MPY operations shown in Fig. 1 along the x-axis. Clearly, OR (OC = 32) and ADD (OC = 144) are located to the left of the crossover point and MPY (OC = 3104) is to the right. The left region is where PIM is superior, and the right region is where CPU is superior.

The crossover point shifts to the right for different DIO values. For instance, for MAT = 1024 and BW = 1024, the crossover point shifts roughly from OC = 2500 to OC = 5000 for DIO = 24 to DIO = 48, respectively. Thus, it is the algorithmic interplay of OC and DIO (along with other technological and architectural factors) that determines the throughput of PIM relative to that of CPU computing.

4 Current Limitations and Future Work

The Bitlet model includes several parameters for exploration of emerging systems that will employ PIM. The most important of these are operational complexity and data in-out. These along with the other supporting parameters allow various trade-off and limitation studies on PIM. Due to space constraints we focused more on studying the effects of algorithmic and architecture parameters, leaving more thorough exploration concerning the discussed and other parameters for future work.

Architectural and technological factors will likely undergo changes. For instance, emerging memory interfaces such as high bandwidth memories (HBM) will provide higher bandwidths. Similarly, the memory cycle time for new memories are seeing continuous improvement (sub 10 ns levels). The model analysis still applies to different bandwidth and memory cycle time values, but the trade-offs and break-even point will vary.

We also expect innovations that reduce the complexity of operations when executing on PIM. Such innovations include the usage of n-input NOR gates and new logic gates that reduce computation complexity. The Bitlet model is useful to understand the extent of overall PIM throughput improvements that are anticipated on these fronts.

CPU arithmetic complexity and data reuse can be taken into account by either integrating or using the Bitlet model with existing models (such as the Roofline model [10]). This opportunity will be useful for laying out the role of PIM in heterogeneous computing setups. We leave this exploration for future work.

5 Acknowledgement

This work was supported by the European Research Council through the European Union's Horizon 2020 Research and Innovation Programme under Grant 757259 and by the Israel Science Foundation under Grant 1514/17.

References

[1] J. Borghetti et al., Memristive switches enable 'stateful' logic operations via material implication," Nature, vol. 464, pp. 873–876, 2010.
[2] J. L. Gustafson, Reevaluating Amdahl's law, Commun. ACM, vol. 31, no. 5, 1988.
[3] A. Haj-Ali et al., Imaging: In-mem. algo. for image proc., IEEE TCAS I, vol. 65-12, 2018.
[4] M. Hill et al., Amdahl's law in the multicore era, IEEE Computer, vol. 41, no. 7, 2008.
[5] K. Korgaonkar et al. The Bitlet Model, 2019, arXiv preprint arXiv:1910.10234 (2019), https://arxiv.org/abs/1910.10234.
[6] K. Korgaonkar et al. The Bitlet Model, 2019, Compiler, Architecture, And Tools Conference (CATC), 2019.
[7] S. Kvatinsky et al., Magic: memristor-aided logic, IEEE TCAS II 2014.
[8] M. Lanza et al., Recom. methods to study resistive switching dev., Adv. Elec. Materials, vol. 5, no. 1, 2019.
[9] P. Ranganathan, From micro. to nanostores: Rethinking data-centric sys., IEEE Comp., vol. 44, no. 1, 2011.
[10] S. Williams et al., Roofline: An insightful vis. perf. model for multicore arch., CACM, vol. 52, no. 4, 2009.