Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories

Shuangchen Li[1]*, Cong Xu[2], Qiaosha Zou[1,5], Jishen Zhao[3], Yu Lu[4], and Yuan Xie[1]

University of California, Santa Barbara[1], Hewlett Packard Labs[2]
University of California, Santa Cruz[3], Qualcomm Inc.[4], Huawei Technologies Inc.[5]
{shuangchenli, yuanxie}ece.ucsb.edu[1]

ABSTRACT

Processing-in-memory (PIM) provides high bandwidth, massive parallelism, and high energy efficiency by implementing computations in main memory, therefore eliminating the overhead of data movement between CPU and memory. While most of the recent work focused on PIM in DRAM memory with 3D die-stacking technology, we propose to leverage the unique features of emerging non-volatile memory (NVM), such as resistance-based storage and current sensing, to enable efficient PIM design in NVM. We propose **Pinatubo**[1], a Processing In Non-volatile memory ArchiTecture for bUlk Bitwise Operations. Instead of integrating complex logic inside the cost-sensitive memory, Pinatubo redesigns the read circuitry so that it can compute the bitwise logic of two or more memory rows very efficiently, and support one-step multi-row operations. The experimental results on data intensive graph processing and database applications show that Pinatubo achieves a ∼500× speedup, ∼28000× energy saving on bitwise operations, and 1.12× overall speedup, 1.11× overall energy saving over the conventional processor.

1. INTRODUCTION

In the big data era, the "memory wall" is becoming the toughest challenge as we are moving towards exascale computing. Moving data is much more expensive than computing itself: a DRAM access consumes 200 times more energy than a floating-point operation [14]. Memory-centric processing-in-memory (PIM) architecture and Near-data-computing (NDC) appear as promising approaches to address such challenges. By designing processing units inside/near main memory, PIM/NDC dramatically reduces the overhead of data movements. It also takes advantage of the large memory internal bandwidth, and hence the massive parallelism. For example, the internal bandwidth in the hybrid memory cube (HMC) [20] is 128 times larger than its SerDes interface [4].

Early PIM efforts [19] are unsuccessful due to practical concerns. They integrate processing units and memory on the same die. Unfortunately, designing and manufacturing the performance-optimized logic and the density-optimized memory together is not cost-effective. For instance, complex logic designs require extra metal layers, which is not desirable for the cost-sensitive memory vendors. Recent achievements in 3D-stacking memory revive the PIM research [18], by decoupling logic and memory circuits in different dies. For example, in HMC stacked memory structure, an extra logic die is stacked with multi-layer of DRAM dies using massive number of through-silicon-vias [18].

Meanwhile, the emerging non-volatile memories (NVMs), i.e., phase changing memory (PCM) [10], spin-transfer torque magnetic random access memory (STT-MRAM) [24], and resistive random access memory (ReRAM) [8] provide promising features such as high density, ultra-low standby power, promising scalability, and non-volatility. They have showed great potential as candidates of next-generation main memory [15, 28, 27].

*The **goal** of this paper is to show NVM's potential on enabling PIM architecture, while almost all existing efforts focus on DRAM systems and heavily depend on 3D integration.* NVM's unique features, such as resistance-based storage (in contrast to charge-based in DRAM) and current-sensing scheme (in contrast to the voltage-sense scheme used in DRAM), are able to provide inherent computing capabilities [13, 16]. Therefore, NVM can enable PIM without the requirement of 3D integration. In addition, it only requires insignificant modifications to the peripheral circuitry, resulting in a cost-efficient solution. Furthermore, NVM-enabled PIM computation is based on in-memory analog signals, which is much more energy efficient than other work that uses digital circuits.

In this paper, we propose Pinatubo, a Processing In Non-volatile memory ArchiTecture for bUlk Bitwise Operations, including OR, AND, XOR, and INV operations. When Pinatubo works, two or more rows are activated simultaneously, the memory will output the bitwise operations result of the open rows. Pinatubo works by activating two (or more) rows simultaneously, and then output of the memo-

* This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Award number DE-SC0013553 with disclaimer at http://seal.ece.ucsb.edu/doe/. It was also supported in part by NSF 1533933, 1461698 and 1500848, and a grant from Qualcomm. Zhao is supported by UCSC start-up funding.

[1] Mount Pinatubo is an active volcano that erupted in 1991. We envision our design to invigorate the future PIM research, similar to the rejuvenation of life after an volcanic eruption.

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.

DAC '16, June 05-09, 2016, Austin, TX, USA

© 2016 ACM. ISBN 978-1-4503-4236-0/16/06...$15.00

DOI: http://dx.doi.org/10.1145/2897937.2898064

ry is the bitwise operation result upon the open rows. The results can be sent to the I/O bus or written back to another memory row directly. The major modifications on the NVM-based memory are in the sense amplifier (SA) design. Different from a normal memory read operation, where the SA just differentiates the resistance on the bitline between R_{high} and R_{low}, Pinatubo adds more reference circuit to the SA, so that it is capable of distinguishing the resistance of $\{R_{\text{high}}/2$ (logic "0,0"), $R_{\text{high}}||R_{\text{low}}$ (logic "0,1"), $R_{\text{low}}/2$ (logic "1,1")$\}$ for 2-row AND/OR operations. It also potentially supports multi-row OR operations when high ON/OFF ratio memory cells are provided. Although we use 1T1R PCM as an example in this paper, Pinatubo does not rely on a certain NVM technology or cell structure, as long as the technology is based on resistive-cell.

Our contributions in this paper are listed as follows,

- We propose a low-cost processing-in-NVM architecture with insignificant circuit modification and no requirement on 3D integration.
- We design a software/hardware interface which is both visible to the programmer and the hardware.
- We evaluate our proposed architecture on data intensive graph processing and data-base applications, and compare our work with SIMD processor, accelerator-in-memory PIM, and the state-of-the-art in-DRAM computing approach.

2. NVM BACKGROUND

Although the working mechanism and the features vary, PCM, STT-MRAM, and ReRAM share common basics: all of them are based on resistive-cell. To represent logic "0" and "1", they rely on the difference of cell resistance (R_{high} or R_{low}). To switch between logic "0" and "1", certain polarity, magnitude, and duration voltage/current are required. The memory cells typically adopt 1T1R structure [10], where there are a wordline (WL) controlling the access transistor, a bitline (BL) for data sensing, and a source line (SL) to provide different polarized write currents.

Architecting NVM as main memory has been well studied [28, 15]. The SA design is the major difference between NVM and DRAM design. Different from the conventional charge-based DRAM, the resistance-based NVM requires a larger SA to convert resistance difference into voltage/current signal. Therefore, multiple adjacent columns share one SA by a multiplexer (MUX), and it results in a smaller row buffer size.

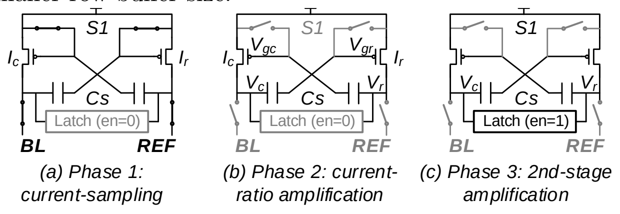

Figure 1: A current-based SA (CSA) [8].

Fig. 1 shows the mechanism of a state-of-the-art CSA [8]. There are three phases during sensing, i.e., current-sampling, current-ratio amplification, and 2^{nd}-stage amplification.

3. MOTIVATION AND OVERVIEW

Bitwise operations are very important and widely used by database [26], graph processing [5], bio-informatics [21], and image processing [6]. They are applied to replace expensive arithmetic operations. Actually, modern processors have already been aware of this strong demand, and developed accelerating solutions, such as Intel's SIMD solution SSE/AVX.

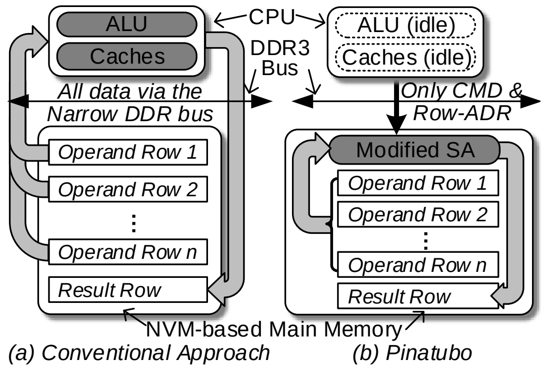

Figure 2: Overview: (a) Computing-centric approach, moving tons of data to CPU and write back. (b) The proposed Pinatubo architecture, performs n-row bitwise operations inside NVM in one step.

We propose Pinatubo to accelerate the bitwise operations inside the NVM-based main memory. Fig. 2 shows the overview of our design. Conventional computing-centric architecture in Fig. 2 (a) fetches every bit-vector from the memory *sequentially*. The data walks through the narrow DDR bus and all the memory hierarchies, and finally is executed by the limited ALUs in the cores. Even worse, it then needs to write the result back to the memory, suffering from the data movements overhead again. Pinatubo in Fig. 2 (b) performs the bit-vector operations inside the memory. Only commands and addresses are required on the DDR bus, while all the data remains inside the memory. To perform bitwise operations, Pinatubo activates two (or more) memory rows that store bit-vector simultaneously. The modified SA outputs the desired result. Thanks to in-memory calculation, the result does not need the memory bus anymore. It is then written to the destination address thought the WD directly, bypassing all the I/O and bus.

Pinatubo embraces two major benefits from PIM architecture. First, the reduction of data movements. Second, the large internal bandwidth and massive parallelism. Pinatubo performs a memory-row-length (typical 4Kb for NVM) bit-vector operations. Furthermore, it supports multi-row operations, which calculate multi-operand operations in one step, bringing the equivalent bandwidth $\sim$1000$\times$ larger than the DDR3 bus.

4. ARCHITECTURE AND CIRCUIT DESIGN

In this section, we first show the architecture design that enables the NVM main memory for PIM. Then we show the circuit modifications for the SA, LWL driver, WD, and global buffers.

4.1 From Main Memory to Pinatubo

Main memory has several physical/logic hierarchies. *Channels* runs in parallel, and each channel contains several *ranks* that share the address/data bus. Each rank has typical 8 physical *chips*, and each chip has typical 8 *banks* as shown in Fig. 3 (a). Banks in the same chip share the I/O, and banks in different chips work in a lock-step manner. Each bank

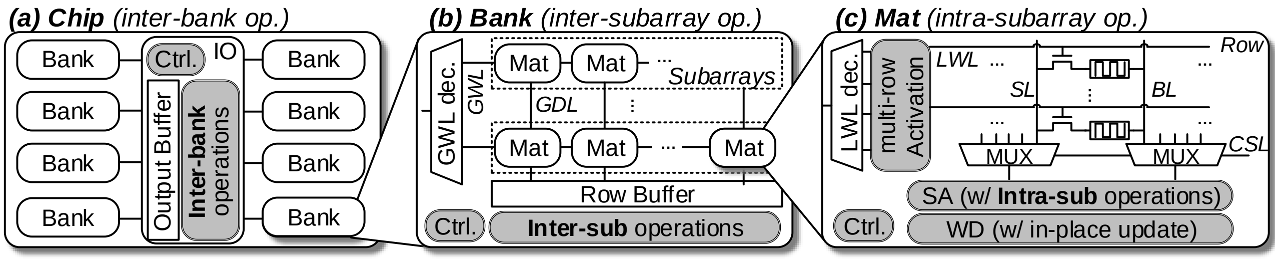

Figure 3: The Pinatubo Architecture. Glossary: Global WordLine (GWL), Global DataLine (GDL), Local WordLine (LWL), SelectLine (SL), BitLine (BL), Column SelectLine (CSL), Sense Amplifier (SA), Write Driver (WD).

has several *subarrays*. As Fig. 3 (b) shows, Subarrays share the GDLs and the global row buffer. One subarray contains several *MATs* as shown in Fig. 3 (c), which also work in the lock-step manner. Each Mat has its private SAs and WDs. Since NVM's SA is much larger than DRAM, several (32 in our experiment) adjacent columns share one SA by a MUX.

According to the physical address of the operand rows, Pinatubo performs three types of bitwise operations: intra-subarray, inter-subarray, and inter-bank operations.

Intra-subarray operations. If the operand rows are all within one subarray, Pinatubo performs intra-subarray operations in each MAT of this subarray. As shown in Fig. 3 (c), the computation is done by the modified SA. Multiple rows are activated simultaneously, and the output of the modified SA is the operation result. The operation commands (e.g., AND or OR) are sent by the controller, which change the reference circuit of the SA. We also modify the LWL driver is also implemented to support multi-row activation. If the operation result is required to write back to the same subarray, it is directly fed into the WDs locally as an in-place update.

Inter-subarray operations. If the operand rows are in different subarrays but in the same bank, Pinatubo performs inter-subarray operations as shown in Fig. 3 (b). It is based on the digital circuits added on the global row buffer. The first operand row is read to the global row buffer, while the second operand row is read onto the GDL. Then the two operands are calculated by the add-on logic. The final result is latched in the global row buffer.

Inter-bank operations. If the operand rows are even in different banks but still in the same chip, Pinatubo performs inter-bank operations as shown in Fig. 3 (a). They are done by the add-on logic in the I/O buffer, and have a similar mechanism as inter-subarray operations.

Note that Pinatubo does not deal with operations between bit-vectors that are either in the same row or in different chips. Those operations could be avoided by optimized memory mapping, as shown in Section 5.

4.2 Peripheral Circuitry Modification

SA Modification: The key idea of Pinatubo is to use SA for intra-subarray bitwise operations. The working mechanism of SA is shown in Fig. 5. Different from the charge-based DRAM/SRAM, the SA for NVM senses the resistance on the BL. Fig. 5 shows the BL resistance distribution during read and OR operations, as well as the reference value assignment. Fig. 5 (a) shows the sensing mechanism for normal reading (Though the SA actually senses currents, the figure presents distribution of resistance for simplicity). The resistance of a single cell (either R_{low} or R_{high}) is compared with the reference value ($R_{\text{ref-read}}$), determining the result between "0" and "1". For bitwise operations, an example for a 2-row OR operation is shown in Fig. 5 (b). Since two rows are activated simultaneously, the resistance on the BL is the parallel connection of two cells. There could be three situations: $R_{\text{low}}||R_{\text{low}}$ (logic "1","1"), $R_{\text{low}}||R_{\text{high}}$ ("1","0"), and $R_{\text{high}}||R_{\text{high}}$ ("0","0")[2]. In order to perform OR operations, the SA should output "1" for the first two situations and output "0" for the last situation. To achieve this, we simply shift the reference value to the middle of $R_{\text{low}}||R_{\text{high}}$ and $R_{\text{high}}||R_{\text{high}}$, denoted as $R_{\text{ref-or}}$. Note that we assume the variation is well controlled so that no overlap exists between "1" and "0" region. In summary, to compute AND and OR, we only need to change the reference value of the SA.

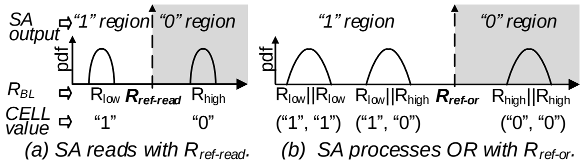

Figure 5: Modifying Reference Values in SA to Enable Pinatubo.

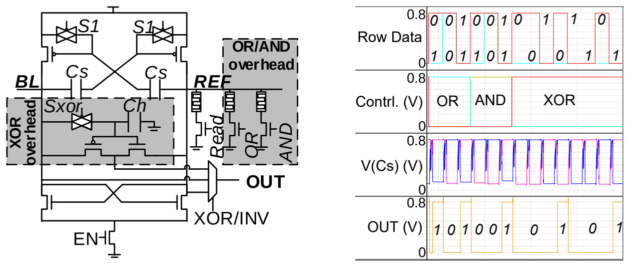

Figure 6: Current Sense Amplifier (CSA) Modification (left) and HSPICE Validation (right).

Fig. 6 (a) shows the corresponding circuit modification based on the CSA [8] introduced in Section 2. As explained above, we add two more reference circuits to support AND/OR operations. For XOR, we need two micro-steps. First, one operand is read to the capacitor C_h. Second, the other operand is read to the latch. The output of the two add-on transistors is the XOR result. For INV, we simply output the differential value from the latch. The output is selected among READ, AND, OR, XOR, and INV results by a MUX. Fig. 6 (b) shows the HSPICE validation of the proposed circuit. The circuit is tested with a large range of cell resistances from the recent PCM, STT-MRAM, and R-eRAM prototypes [23].

Multi-row Operations: Pinatubo supports multi-row operations that further accelerate the bitwise operations. A multi-row operation is defined as calculating the result of multiple operands at one operation. For PCM and ReRAM

[2] "||" denotes production over sum operation.

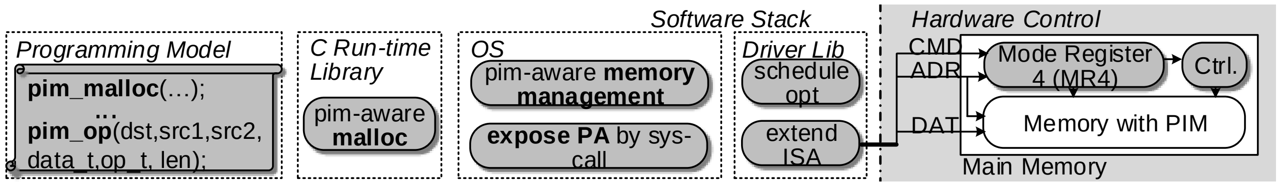

Figure 4: Pinatubo System Support.

which encode R_{high} as logic "0", Pinatubo can calculate n-row OR operations[3]. After activating n rows simultaneously, Pinatubo needs to differentiate the bit combination of only one "1" that results in "1", and the bit combination with all "0" that results in "0". This leads to a reference value between $R_{\text{low}}||R_{\text{high}}/(n-1)$ and R_{high}/n. This sensing margin is similar with the TCAM design [17]. State-of-the-art PCM-based TCAM supports 64-bit WL with two cells per bit. Therefore we assume maximal 128-row operations for PCM. For STT-MRAM, since the ON/OFF ratio is already low, we conservatively assume maximal 2-row operation.

LWL Driver Modification: Conventional memory activates one row each time. However, Pinatubo requires multi-row activation, and each activation is a random-access. The modifications of the LWL driver circuit and the HPSICE validation are shown in Fig. 7. Normally, the LWL driver amplifies the decoded address signal with a group of inverters. We modify each LWL drive by adding two more transistors. The first transistor is used to feed the signal between inverters back and serves as a latch. The second transistor is used to force the driver's input as ground. During the multi-row activation, it requires to send out the RESET signal first, making sure that no WL has latched anything. Then every time an address is decoded, the selected WL signal is latched and stuck at VDD until the next RESET signal arrives. Therefore, after issuing all the addresses, all the corresponding selected WL are driven to the high voltage value.

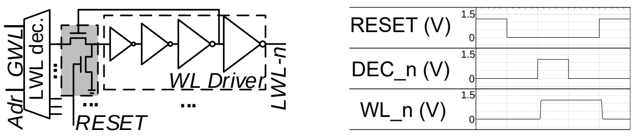

Figure 7: Local Wordline (LWL) Driver Modification (left) and HSPICE Validation (right).

WD Modification: Fig. 8 (a) shows the modification to a WD of STT-MRAM/ReRAM. We do not show PCM's WD since it is simpler with unidirectional write current. The write current/voltage is set on BL or SL according to the write input data. Normally, the WD's input comes from the data bus. We modify the WD circuit so that the SA result is able to be fed directly to the WD. This circuit bypasses the bus overhead when writing results back to the memory.

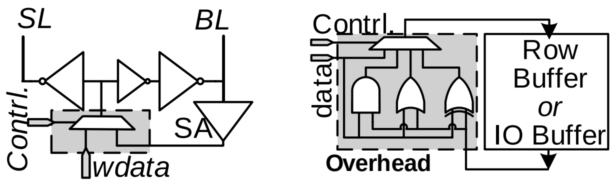

Figure 8: (a) Modifications to Write Driver (WD). (b) Modifications for Inter-Sub/Bank Operations

[3] Multi-row AND in PCM/ReRAM is not supported, since it is unlikely to differentiate $R_{\text{low}}/(n-1)||R_{\text{high}}$ and R_{low}/n, when $n > 2$.

Global Buffers Modification: To support inter-subarray and inter-bank operations, we have to add the digital circuits to the row buffers or IO buffers. The logic circuit's input is the data from the data bus and the buffer. The output is selected by the control signals and then latched in the buffer, as shown in Fig. 8 (b).

5. SYSTEM SUPPORT

Fig. 4 shows an overview of Pinatubo's system design. The software support contains the programming model and run-time supports. The programming model provides two functions for programmers, including the bit-vector allocation and the bitwise operations. The run-time supports include modifications of the C/C++ run-time library and the OS, as well as the development of the dynamic linked driver library. The C/C++ run-time library is modified to provide a PIM-aware data allocation function. It ensures that different bit-vectors are allocated to different memory rows, since Pinatubo is only able to process inter-row operations. The OS provides the PIM-aware memory management that maximizes the opportunity for calling intra-subarray operations. The OS also provides the bit-vector mapping information and physical addresses (PAs) to the PIM's run-time driver library. Based on the PAs, the dynamic linked driver library first optimizes and reschedules the operation requests, and then issues extended instruction for PIM [3]. The hardware control part utilizes the DDR mode register (MR) and command. The extended instructions are translated to DDR commands and issued through the DDR bus to the main memory. The MR in the main memory is set to configure the PIM operations.

6. EXPERIMENT

In this section, we compare Pinatubo with state-of-the-art solutions and present the performance and energy results.

6.1 Experiment Setup

The three counterparts we compare are described below:

SIMD is a 4-core 4-issue out-of-order x86 Haswell processor running at 3.3GHz. It also contains a 128-bit SIMD unit with SSE/AVX for bitwise operation acceleration. The cache hierarchy consists of 32KB L1, 256KB L2, and 6MB L3 caches.

S-DRAM is the in-DRAM computation solution to accelerate bitwise operations [22]. The operations are executed by charges sharing in DRAM. Due to the read-destructive feature of DRAM, this solution requires copying data before calculation. Only 2-row AND and OR are supported.

AC-PIM is an accelerator-in-memory solution, where even the intra-subarray operations are implemented with digital logic gates as shown in Fig. 8 (b).

The *S-DRAM* works with a 65nm 4-channel DDR3-1600 DRAM. *AC-PIM* and *Pinatubo* work on 1T1R-PCM based main memory whose tRCD-tCL-tWR is 18.3-8.9-151.1ns [9]. *SIMD* works with DRAM when compared with *S-DRAM*,

Vector:	pure vector OR operations.
dataset:	*e.g. 19-16-1(s/r) means 2^{19}-lentgh vector, 2^{16} vectors, 2^1-row OR ops (sequntial/random access)*
Graph:	bitmap-based BFS for graph processing [5].
dataset:	*dblp-2010, eswiki-2013,amazon-2008 [1]*
Database:	bitmap-based database (Fastbit [26]) application.
dataset:	*240/480/720 number of qurarying on STAR [2]*

Table 1: Benchmarks and Data Set

and with PCM when compared with *AC-PIM* and *Pinatubo*. Note that the experiment takes 1T1R PCM for a case study, but Pinatubo is also capable to work with other technologies and cell structures.

The parameters for *S-DRAM* are scaled from existing work [22]. The parameters for *AC-PIM* are collected from synthesis tool with 65nm technology. As to parameters for Pinatubo, the analog/mixsignal part, including SA, WD, and LWL, is extracted from HSPICE simulation; the digital part, including controllers and logics for inter-subarray/bank operations, is extracted from the synthesis tool. Based on those low-level parameters, we heavily modify NVsim [11] for the NVM circuit modeling, and CACTI-3DD [9] for the main memory modeling, in order to achieve high-level parameters. We also modify the PIN-based simulator Sniper [7] for SIMD processor and the NVM-based memory system. We develop an in-house simulator to evaluate the *AC-PIM*, *S-DRAM*, and Pinatubo. We show the evaluation benchmarks and data sets in Table 1, in which *Vector* only has OR operation while *Graph* and *Database* contain all AND, OR, XOR, and INV operations.

6.2 Performance and Energy Evaluation

Fig. 9 shows Pinatubo's OR operation throughput. We have four observations. First, the throughput increases with longer bit-vectors, because they make better use of the memory internal bandwidth and parallelism. Second, we observe two turning points, *A* and *B*, after which the speedup improvement is showed down. *Turning point A* is caused by the sharing SA in NVM: bit-vectors longer than 2^{14} have to be mapped to columns the SA sharing, and each part has to be processed in serial. *Turning point B* is caused by the limitation of the row length: bit-vectors longer than 2^{19} have to be mapped to multiple ranks that work in serial. Third, Pinatubo has the capability of multi-row operations (as the legends show). For n-row OR operations, larger n provides larger bandwidth. Fourth, the y-axis is divided into three regions: the below DDR bus bandwidth region which only includes short bit-vectors' result; the memory internal bandwidth region which includes the majority of the results; and the beyond internal bandwidth region, thanks to the multi-row operations. DRAM systems can never achieve beyond memory internal bandwidth region.

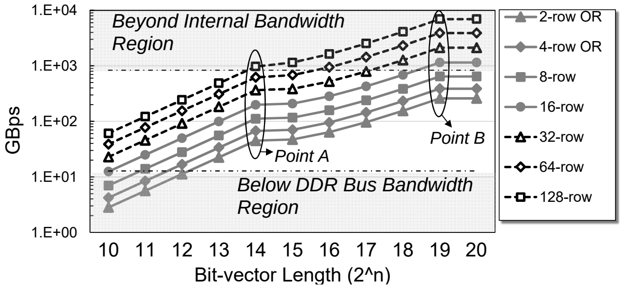

Figure 9: Pinatubo's Throughput (GBps) for OR operations.

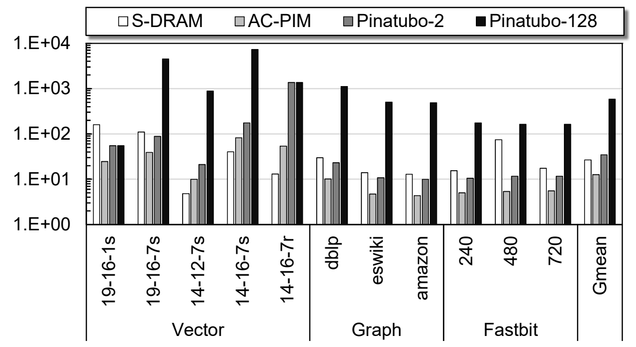

Figure 10: Speedup Normalized to *SIMD* Baseline.

We compare both Pinatubo of 2-row and 128-row operation with two aggressive baselines in Fig. 10, which shows the speedup on bitwise operations. We have three observations: First, *S-DRAM* has better performance than *Pinatubo-2* in some cases with very long bit-vectors. This is because DRAM-based solutions benefit from larger row buffers, compared with the NVM-based solution. However, the advantage of NVM's multi-row operations still dominates. *Pinatubo-128* is 22× faster than *S-DRAM.* Second, the *AC-PIM* solution is much slower than *Pinatubo* in every single case. Third, multi-row operations show their superiority, especially when intra-subarray operations are dominating. An opposite example is *14-16-7r*, where all operations are random accesses and it is dominated by inter-subarray/bank operations, so that *Pinatubo-128* is as slow as *Pinatubo-2.*

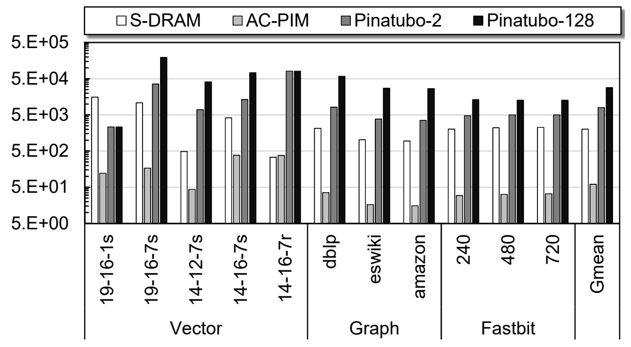

Figure 11: Energy Saving Normalized to *SIMD*.

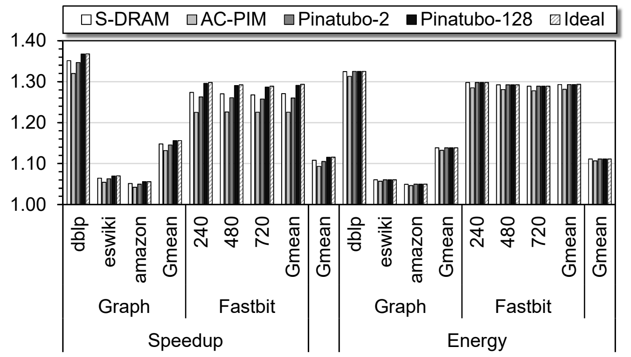

Figure 12: Overall Speedup and Energy Saving Normalized to *SIMD* Baseline.

Fig. 11 shows the energy saving result. The observations are similar with those from speedup: *S-DRAM* is better than *Pinatubo-2* in some cases but worse than *Pinatubo-128* on average. *AC-PIM* never has a change to save more energy then any of the other three solutions, since both *S-DRAM* and *Pinatubo* rely on high energy efficient analogy computing. On average, Pinatubo saves 2800× energy for bitwise operations, compared with SIMD processor.

Fig. 12 shows the overall speedup and energy saving of Pinatubo in the two real-world applications. The *ideal* legend represents the result with zero latency and energy spent on bitwise operations. We have three observations. First, Pinatubo almost achieves the ideal acceleration. Second, limited by the bitwise operations' proportion, Pinatubo can improve graph processing applications by 1.15× with 1.14× energy saving. However, it is data dependent. For the *eswiki* and *amazon* data set, since the connection is "loose", it has to spend most of the time searching for an unvisited bit-vector. For *dblp*, it has 1.37× speedup. Third, for the database applications, it achieves 1.29× overall speedup and energy saving.

6.3 Overhead Evaluation

Fig. 13 shows the area overhead results. As shown in Fig. 13 (a), Pinatubo incurs insignificant area overhead only 0.9%. However, *AC-PIM* has 6.4% area overhead, which is critical to the cost-sensitive memory industry. *S-DRAM* reports ∼0.5% capacity loss, but it is for DRAM-only result and orthogonal with Pinatubo's overhead evaluation. Fig. 13 (b) shows the area overhead breakdown. We conclude that the majority area overhead are taken by inter-subarray/bank operations. For intra-subarray operations, XOR operations takes most of the area.

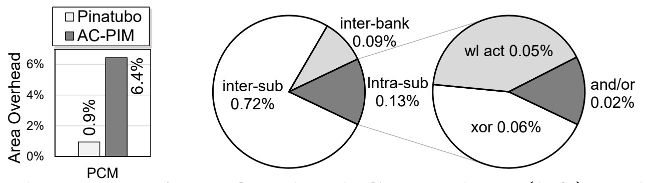

Figure 13: Area Overhead Comparison (left) and Breakdown (right).

7. RELATED WORK

Pinatubo distinguishes itself from PIM work, in-DRAM computing work, and logic-in-memory work. First, different from other PIM work, Pinatubo does not need 3D integration [18], and does not suffer from logic/memory coupling problem [19] either. Pinatubo benefits from NVM's resistive-cell feature, and provides cost and energy efficient PIM. Although ProPRAM [25] leverages NVM for PIM, it uses NVM's lifetime enhancement peripheral circuits for computing, instead of the NVM's character itself. Moreover, bitwise AND/OR is not supported in ProRPAM, and it is computing with digital circuit while Pinatubo takes advantage of high energy-efficient analog computing. Second, based on charge sharing, in-DRAM bulk bitwise operations is proposed [22]. However, it suffers from read destructive problem so that operand copy is required before computing, incurring unnecessary overhead. Also, only maximal 2-row operations are supported. Third, there is other work using NVM for logic-in-memory functionality such as associate memory [17, 12]. Recent studies also take use of ReRAM crossbar array to implement IMPLY-based logic operations [16, 13]. However, none of them are necessarily fitted to the PIM concept: they use the memory technique to implement processing unit, but the processing unit appears as either a co-processor or a stand-alone accelerator. They still pay for the expensive data fetching from the memory and are limited by the memory bus bandwidth.

8. CONCLUSION

In this paper, a processing-in-NVM architecture for bulk bitwise operations is proposed. The computation makes use of NVM's resistive-cell feature and achieves high performance and energy efficiency with insignificant area overheads. Experimental results show that the proposed architecture achieves ∼500× speedup and ∼28000× energy saving on bitwise operations, and 1.12× overall speedup, 1.11× overall energy saving on data intensive graph processing and database applications with real-world data.

9. REFERENCES

[1] Laboratory for web algorithmics. http://law.di.unimi.it/.
[2] The star experiment. http://www.star.bnl.gov/.
[3] J. Ahn et al. Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *ISCA*, pages 336–348. ACM, 2015.
[4] J. Ahn et al. A scalable processing-in-memory accelerator for parallel graph processing. In *ISCA*, pages 105–117. ACM, 2015.
[5] S. Beamer et al. Direction-optimizing breadth-first search. In *SC*, pages 1–10, Nov 2012.
[6] J. Bruce et al. Fast and inexpensive color image segmentation for interactive robots. In *IROS*, volume 3, 2000.
[7] T. E. Carlson et al. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *SC*, pages 52:1–52:12. ACM, 2011.
[8] M.-F. Chang et al. An offset-tolerant fast-random-read current-sampling-based sense amplifier for small-cell-current nonvolatile memory. *JSSC*, 48(3):864–877, March 2013.
[9] K. Chen et al. Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory. In *DATE*, pages 33–38, 2012.
[10] G. De Sandre et al. A 90nm 4mb embedded phase-change memory with 1.2v 12ns read access time and 1mb/s write throughput. In *ISSCC*, pages 268–269, Feb 2010.
[11] X. Dong et al. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *TCAD*, 31(7):994–1007, July 2012.
[12] Q. Guo et al. Ac-dimm: associative computing with stt-mram. In *ISCA*, pages 189–200. ACM, 2013.
[13] S. Hamdioui et al. Memristor based computation-in-memory architecture for data-intensive applications. In *DATE*, pages 1718–1725, 2015.
[14] S. W. Keckler et al. Gpus and the future of parallel computing. *IEEE Micro*, (5):7–17, 2011.
[15] B. C. Lee et al. Architecting phase change memory as a scalable dram alternative. In *ISCA*, pages 2–13. ACM, 2009.
[16] H. Li et al. A learnable parallel processing architecture towards unity of memory and computing. *Scientific reports*, 5, 2015.
[17] J. Li et al. 1 mb 0.41 um 2t-2r cell nonvolatile tcam with two-bit encoding and clocked self-referenced sensing. *JSSC*, 49(4):896–907, April 2014.
[18] R. Nair et al. Active memory cube: A processing-in-memory architecture for exascale systems. *IBM Journal of Research and Development*, 59(2/3):17:1–17:14, March 2015.
[19] D. Patterson et al. A case for intelligent ram. *IEEE Micro*, 17(2):34–44, 1997.
[20] J. T. Pawlowski. Hybrid memory cube (hmc). In *Hot Chips*, volume 23, 2011.
[21] M. Pedemonte and other. Bitwise operations for gpu implementation of genetic algorithms. In *GECCO*, pages 439–446. ACM, 2011.
[22] V. Seshadri et al. Fast bulk bitwise and and or in dram. *CAL*, PP(99):1–1, 2015.
[23] K. Suzuki et al. The non-volatile memory technology database (nvmdb). Technical Report CS2015-1011, UCSD, May 2015.
[24] K. Tsuchida et al. A 64mb mram with clamped-reference and adequate-reference schemes. In *ISSCC*, pages 258–259, 2010.
[25] Y. Wang et al. Propram: Exploiting the transparent logic resources in non-volatile memory for near data computing. In *DAC*, pages 47:1–47:6. ACM, 2015.
[26] K. Wu. Fastbit: an efficient indexing technology for accelerating data-intensive science. In *Journal of Physics*, volume 16, page 556, 2005.
[27] J. Zhao et al. Memory and storage system design with nonvolatile memory technologies. *IPSJ*, 8(0):2–11, 2015.
[28] P. Zhou et al. A durable and energy efficient main memory using phase change memory technology. In *ISCA*, 2009.