ELP2IM: Efficient and Low Power Bitwise Operation Processing in DRAM

Xin Xin[1], Youtao Zhang[2], and Jun Yang [1]

[1]Electrical and Computer Engineering Department, University of Pittsburgh

[2]Department of Computer Science, University of Pittsburgh

xix59@pitt.edu, zhangyt@cs.pitt.edu, juy9@pitt.edu

ABSTRACT

Recently proposed DRAM based memory-centric architectures have demonstrated their great potentials in addressing the memory wall challenge of modern computing systems. Such architectures exploit charge sharing of multiple rows to enable in-memory bitwise operations. However, existing designs rely heavily on reserved rows to implement computation, which introduces high data movement overhead, large operation latency, large energy consumption, and low operation reliability.

In this paper, we propose ELP^{2}IM, an efficient and low power processing in-memory architecture, to address the above issues. ELP^{2}IM utilizes two stable states of sense amplifiers in DRAM subarrays so that it can effectively reduce the number of intra-subarray data movements as well as the number of concurrently opened DRAM rows, which exhibits great performance and energy consumption advantages over existing designs. Our experimental results show that the power efficiency of ELP^{2}IM is more than 2$\times$ improvement over the state-of-the-art DRAM based memory-centric designs in real application.

1. INTRODUCTION

Modern system performance is hindered by memory subsystem, known as "the memory wall", due to the high cost of data movement [1, 2]. Particularly, for big-data applications, the limited bandwidth of the off-chip bus between memory and processor cannot meet the increasing demand of data. A significant amount of power during data movement is consumed, leading to system energy inefficiency. To tackle this problem, near-data processing (NDP) has been proposed as a solution. In DRAM-based NDP, logic units (LU) are built close to but outside DRAM arrays. Examples of NDP include HMC [3, 4], Automata [5] and DRISA [6]. NDP delivers good performance due to the removal of volume data movement and high processing speed in LUs. However, high speed of LUs is attributed to their high complexity, which usually occupy large die area and decreases the density of memory. For example, the adder in DRISA takes 51% of area, and the routing matrix in Automata occupies about 30% of the chip [5]. Complex LUs on a separate layer of an HMC are constrained by not only area but also power consumption. Moreover, large area of LUs increases the complexity of DRAM technology, as LUs and DRAM cells are usually not compatible in technology process [7].

Alternatively, true in-memory computing, a.k.a. processing-in-memory (PIM), directly integrates logic functions into memory arrays. PIM keeps integrity of memory and realizes high-bandwidth calculation with light area overhead. Note that NDP and PIM are not contradicting approaches. PIM embeds operations inside memory array, offloading those logic from NDP design. Recently, a DRAM-based PIM architecture, termed Ambit [8], has been proposed for bulk bitwise operations. It implements basic logic operations based on an inter-cell charge sharing mechanism, which can exploit full internal bandwidth inside DRAM. It provides great opportunities to accelerate applications with large amount of bitwise operations [9, 10, 11]. Nevertheless, we observed that its performance is still impeded by several aspects.

First, each logic operation in Ambit requires triple-row activation (TRA) which imposes high power consumption that challenges inherent power constraint, e.g. t_{FAW}, and reduces bank level parallelism [12, 13]. The frequent copies among different memory rows in each logic operation generally double or even triple the number of row activation in one access. In other words, when a memory array is performing a logic operation, there is little to no power left for other banks to perform regular memory accesses. Second, each logic operation in Ambit requires many commands, or long latency, giving that each command in DRAM is already a long-latency operation. For example, an XOR operation requires 7 commands (or DRAM cycles), totaling $\sim$363ns (DDR3-1600 [14]). Third, Ambit requires to reserve a group of rows, typically 8, near sense amplifiers for each calculation. Even though those reserved rows can be used for storing data, it is still necessary to migrate them when Ambit switches to calculation mode. Those are additional capacity and latency overhead in the array. Fourth, charge sharing among the cells opened by TRA can become unreliable with process variation and bitline coupling effect. This is because those cells may not have equal capacitance which could lead to less than ideal charge sharing [15]. Finally, the coupling effect worsen the situation by narrowing the sensing margin, as we will quantify in our experiments.

To address the aforementioned problems, we propose a new technique by creating a new state, termed pseudo-precharge state, in DRAM access. In this state, bitline voltage is regulated by sense amplifier (SA) instead of the precharge logic. Logic operation can be implemented by using the regulated

2378-203X/20/$31.00 ©2020 IEEE
DOI 10.1109/HPCA47549.2020.00033

bitline to access the next source cell.

Pseudo-precharge based approach offers two benefits: First, it provides the opportunity to implement an in-place logic operation, which completes the calculation directly in the destination cells, instead of the reserved cells. Thereby, it reduces data copy to the reserved cells, which further diminishes operation cycles and the reserved rows. For the basic logic operation, we shorten the average latency by up to 1.23×. Second, based on the method, charge sharing in each cycle just involves one cell, which is similar to the regular DRAM access. Thereby, it reduces the number of row activations and can maintain the same sensing margin as regular DRAM. In real applications, we save up to 2.45× row activations, thereby expanding bank level parallelism by 2.45×. Significant improvement in reliability is also achieved. The contributions of this paper are summarized as follows:

- We present a lightweight mechanism to implement bulk bitwise operation in DRAM, termed ELP^{2}IM, which efficiently reduces the number of commands for each kind of logic operation.
- ELP^{2}IM not only lowers the power consumption, but also relieves power constraint problem by reducing the number of activated rows of each command.
- ELP^{2}IM also significantly saves the reserved space, only retaining one reserved row, and improves the operation reliability.
- We propose two strategies considering different sizes of DRAM subarray. We also categorize six primitives for ELP^{2}IM, and further improve the performance by better scheduling of those primitives.

2. BACKGROUND AND RELATED WORK

2.1 Overview of DRAM

A DRAM chip traditionally adopts hierarchical structure from cell matrices to subarrays and to banks. While one subarray horizontally consists of many cell matrices, a bank has multiple subarrays connected using the global bitlines. Each bank can be operated by the memory controller independently. Each cell matrix comprises a large number of cells which are built by an access transistor and a capacitor (1T1C). Multiple cells are connected by a local bitline with a sense amplifier (SA) which also acts as a row buffer [8, 16].

Figure 1(a) illustrates the internal structure of one column in the cell matrix. The SA is constructed as a latch using two CMOS inverters, which has two voltage inputs (at node 1 and 2) and one enable EN signal. As we discuss next, we provide different voltages at node 1 and 2 at different times to enable the precise memory access. The precharge unit (PU) is powered by a Vdd/2 source and controlled by the EQ signal. The DRAM bitline usually possesses a noticeable parasitic capacitor, which exhibits large impact on access speed.

Figure 1(b) indicates an access sequence in DRAM. It can be divided into two states: precharge and activate [17]. In the precharge state, the PU is enabled to set both the bitline pair and the SA to Vdd/2. The activate state can be further divided into three phases — access, sense, and restore. In the access

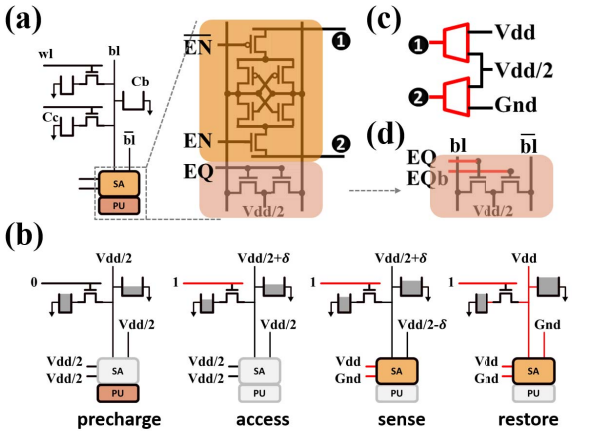

Figure 1: (a) Concise structure of a DRAM column, (b) Basic states (or phases) in a DRAM access, (c) SA supply voltage controller, (d) Precharge unit with separate control signals

phase, the wordline is enabled to share the charges in the cell capacitor with the bitline, which leads to a voltage variation on the bitline. In the sense phase, the SA is enabled to sense the slight variation on the bitline. Finally, in the restore phase, the SA continuously charges the bitline and the cells until their voltage reaches Vdd or Gnd. The DRAM subarray may return to the precharge state afterwards.

2.2 Related Work

Recently, several DRAM based PIM designs, such as such as Rowclone, Ambit, Dracc, ROC etc., have been proposed [8, 18, 19, 20]. They implement bulk bitwise operation intra-subarray with high throughput and efficiency.

2.2.1 RowClone

RowClone [18] implements data copy between different rows in the same subarray. When DRAM completes the restore phase, the SA can retain the state if precharge is not enabled. Therefore, row-to-row data copy can be implemented by sequentially activating the target row after completing the restore phase of the source row. This row copy method contains two back-to-back activate states and one precharge state, referred as an Activate-Activate-Precharge (AAP) primitive in [18]. The AAP is close to 2× latency of a regular Activate-Precharge (AP) access, as activate state occupies a large proportion of the access time. To reduce the AAP latency, Ambit adds an extra row decoder for the reserved rows. Thereby, two rows in the same subarray but belong to different decoder domains can be activated simultaneously. This separated row decoder strategy offers the opportunity to overlap the two Activates in AAP, hence, significantly saving the time. To distinguish from AAP, we define the overlapped Activate-Activate-Precharge as oAAP in this article. Although the overlapped AAP (oAAP) is only 4 ns longer than AP (49 ns), it aggravates the burden on charge pump, which drives the wordlines.

2.2.2 Ambit

Ambit exploits the analog operation of DRAM technology to perform bitwise operations. The critical step in the mech-

anism is to activate three DRAM cells on the same bitline at the same time [8], termed Triple-Row Activate (TRA). Thereby, if any two (or all) of the three cells, termed A, B, and C, are '1's, the voltage on the bitline will be above Vdd/2 after charge sharing. Otherwise, the voltage will drop below Vdd/2 when only one or none of them is '1'. The result can be written as: R = AB + BC + AC. If we define C = 1 in advance, the operation will perform A OR B. Likewise, A AND B can be performed with C = 0. For completeness, the NOT operation is implemented in a modified dual-connected cell, where one additional access transistor is attached. Benefiting from the high parallel actions of DRAM cells in a row, logic computation of large bit-vectors could be significantly accelerated.

Directly implementing TRA with cell A, B, and C is infeasible because general row decoder cannot activate three rows simultaneously. This also destroys the original data. Therefore, Ambit requires a group of reserved rows with special row decoder for TRA. For example, to implement an OR operation, it first copies two variables A and B to the reserved rows. Then it copies C, which is '1' as a preceding definition for OR logic. Finally, it calculates the result via TRA. This process involves four cycles in total. For more complex logic, it will consume more cycles. Therefore, performance of Ambit is impeded by the redundant data movement process, which copies A, B, and C before the true calculation.

In addition, Ambit also suffers from power constraint, reliability, and overhead of reserved rows, which has been introduced in Section 1.

2.2.3 Other related works

Deng *et al.* modify several subarray rows with extra transistors to improve the performance of Ambit based accelerator for CNN calculation [19]. However, the multiple cycles of an operation still hinder the speed. For example, it takes 13 cycles to complete an ADD calculation, which amounts to $\sim 630ns$ with 49ns cycle time [8]. In addition, the additional transistors reduces area efficiency, which also increases design complexity for DRAM fabrication process.

Li *et al.* propose a DRAM-based configurable accelerator, termed Drisa [6]. Different from previous works, Drisa completes logic operation by directly integrating logic gates in subarray, thereby, exploiting the full internal bandwidth. For the design of logic part, it presents several strategies, such as NOR gate, mixed gates, and even adders. The area overhead varies based on the complexity of integrated logic gates. In the case studies, the author claims the NOR or mixed design is more efficient. But even for the simplest NOR based design, it still increases 24% area overhead.

2.3 Structure Comparison

Among the aforementioned studies, we choose Ambit as the representative design for bitwise operation, because of its relatively good efficiency and less modifications. Figure 2 provides a general view of regular DRAM, Ambit, and ELP^{2}IM. Compared to the structure of regular DRAM [21], Ambit requests a special row decoder, which can simultaneously pull up triple wordlines, to serve the reserved the rows, termed B-group. In the B-group, the last four rows are modified into the two dual-contact cells, which provide buffers for NOT operation. There is less modification in ELP^{2}IM. Only one dual-contact-cell row, controlled by a small driver, is attached in each array.

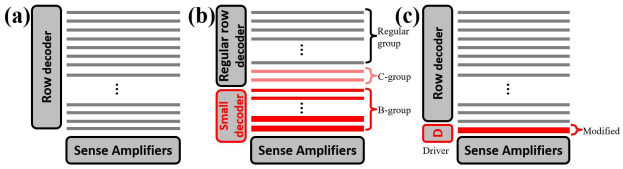

Figure 2: Structure of regular DRAM(a), Ambit(b), and ELP^{2}IM(c)

Our design is built on open-bitline architecture, because all modern DRAM utilizes the open-bitline to achieve high density [22]. The detailed analysis of modification and related overhead will be discussed in Section 5.2.

3. DESIGN OF ELP^{2}IM

In this paper, we propose ELP^{2}IM, a pseudo-precharge approach, to complete logic operations through direct charge sharing with bitlines, which greatly reduces the latency and lowers the power consumption of DRAM based PIM operations. We next motivate the design with two observations, elaborate the design details, and analyze different execution strategies.

3.1 Observations

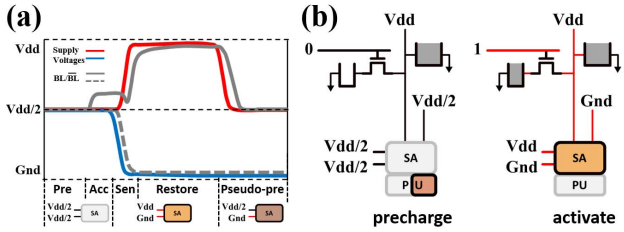

Figure 3: (a) Pseudo-precharge state, (b) Bitline overwrite effect

3.1.1 Pseudo-precharge States

We exploit two stable yet non-traditional DRAM states, referred to as *pseudo-precharge states*. As shown in Figure 1(b), the supply voltage of an SA is switched periodically with the help of a voltage controller (Figure 1(c)). In particular, the SA has its input voltages switched from Vdd and Gnd at node 1 and 2, respectively, at activation state to the same Vdd/2 at prechage state. Having Vdd/2 supply voltage helps to suppress the leakage power and improve the subsequent sensing accuracy.

Similar to that in recent studies [23], we study the SA workflow and observe that, if in precharge state, only one of the two supplies shifts back to Vdd/2 while the other keeps its voltage level, the output of SA adjusts accordingly due to rail-to-rail output character of the CMOS inverters [24]. That is, the SA stays at a stable state. Figure 3(a) shows an example, during activate state, bitline and $\overline{\text{bitline}}$ are charged to Vdd and Gnd respectively. If only Vdd shifts to Vdd/2, then bitline will follow the change, while $\overline{\text{bitline}}$ will remain the same. To distinguish this voltage shifting process from precharge state, we term it as pseudo-precharge state.

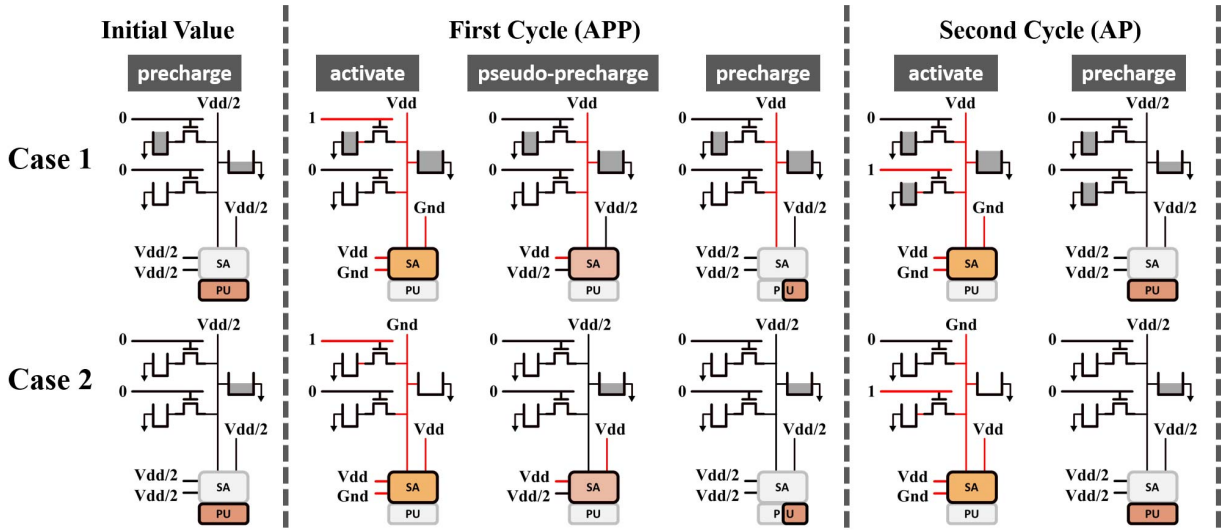

Figure 4: OR Operation of two cases: '1'+'0' (Case 1) and '0'+'0' (Case 2)

Since the voltage shifting technique is compatible with regular CMOS circuit design and incurs small overhead, it has been adopted as structure optimization to suppress leakage induced power in latches or registers [23, 25, 26]. Instead of reducing power consumption, we leverage this technique to assist logic operation in DRAM, and what we change is only the control sequence for the voltage shift of SA.

3.1.2 Overwrite Process

Our other observation is on a stable yet non-traditional cell write operation. During precharge state, if only bitline is charged to Vdd/2, while bitline keeps its value, the following accessed cell will be overwritten by the bitline value. Since the bitline capacitor (Cb) is generally much larger (2∼4×) than the cell capacitor (Cc) [27, 30], it dominates the bitline voltage level during charge sharing. Figure 3(b) shows an example, where logic '1' on bitline is reserved during precharge. In the next activate state, the accessed cell is overwritten by '1'. Note, to enable the overwriting process, we need to modify the precharege unit, as shown in Figure 1(d) — we split the EQ signal to two such signals, EQ and EQb, such that we can control each side independently.

For DRAM subarrays that have short bitlines, e.g., due to small capacity [28, 29], Cb may be close to, or even smaller than, Cc, which prevents the overwriting process as we discuss above. In this paper, we are to develop a simple strategy to enable reliable data overwrite by comparing the difference between bitline and bitline. We will elaborate the details in Section 4.

In summary, the above two observations expose stable yet non-traditional states and memory operations in DRAM, which provide promising opportunities that can be exploited to speed up DRAM based processing-in-memory operations.

3.2 The Basic Idea

Intuitively, ELP^{2}IM takes a two step approach to exploit the pseudo-precharge states and overwrite operation — it first regulates the charge level of bitline capacitor (Cb) in the pseudo-precharge state; and then leverages the regulated Cb to influence the next cycle DRAM access (overwrite or not), thereby completes logic operations.

Figure 4 illustrates how to implement a 'two-cycle' (not two normal memory cycles) OR operation with two specific examples, '1'+'0' and '0'+'0' operations in case 1 and 2 respectively. The initial values saved in the two cells are '1' and '0' in case 1 ('0' and '0' in case 2). For case 1, logic '1' is accessed in the activate state during the first cycle. Then a pseudo-precharge state is inserted, where the Gnd of SA is switched to Vdd/2. As explained in section 3.1, under this condition, the bitline can hold its voltage level of Vdd. In the following precharge state, only bitline is charged to Vdd/2 by PU, which is similar to the second observation. Thereby, in activate state of the second cycle, the accessed cell is overwritten by the reserved bitline value. For case 2, logic '0' is accessed in the activate state during the first cycle. In the next state, bitline is driven to Vdd/2, because '0' will be regulated to Vdd/2 by the pseudo-precharge approach. In the following precharge state, bitline is charged to Vdd/2 by PU. Thereby, at the beginning of the second cycle, both bitline and bitline are Vdd/2. The second cell can be regularly sensed.

Other cases, such as '0'+'1' and '1'+'1', can be explained in the same way. To sum up, if logic '1' is accessed from the first cycle, the cell in the second cycle will be overwritten. Otherwise, the second cell can be regularly accessed. This procedure is corresponded to an OR logic and the final result is restored to the second cell.

Similarly, the AND operation can be conducted by dropping Vdd to Vdd/2 and retaining Gnd in the pseudo-precharge state. Therefore, logic '0' read from the first cycle can overwrite the second cycle cell. Otherwise, logic '1' from the first cycle will be charged to Vdd/2, which will not influence the cell access in the second cycle. For completeness, we build the dual-connected cell at the bottom of a column for NOT operation, the same design as in Ambit.

One character of this pseudo-precharge based method is that the charge sharing in each cycle just involves one cell,

which is similar to the regular access. This character guarantees a proper sense margin during access phase, thereby ensuring the reliability. It also reduces the number of activated rows per cycle, thereby saving the power and reserved rows induced by multiple rows activation in previous work. Another character of pseudo-precharge based method is that it offers the opportunity to complete an AND or OR operation with just 'two-cycle'. This can be applied to optimize the operation cycles for compound logic, such as XOR and XNOR, thereby reducing the average latency for bitwise operation. These advantages in the aspects of reliability, power, and latency will be further indicated in the article.

3.3 Implementing the Logic Operations

As discussed above, in the 'two-cycle' access, irregular control sequence is inserted in the first cycle, as shown in figure 4. Hence, we propose an activate-pseudo_precharge-precharge (APP) primitive to implement the first cycle. Different from the regular activate-precharge (AP) access, APP actually divides the precharge state into two parts: First regulating bitline with the shifted SA voltage. Then charging bitline via PU. We find pseudo-precharge time is 20∼30% longer than precharge time, and more detail analysis will be indicated in section 6.1. In the article, we take the conservative timing parameter (30%). In summary, the 'two-cycle' access (termed, APP-AP sequence) is only ∼18% longer than two regular cycles access (termed, AP-AP sequence), based on DDR3-1600.

The APP-AP can process an OR (or AND) logic operation where one operand shares the same address with the destination, which can be represented by A=f(A,B). For destination and operands occupying different addresses, C=f(A,B), we leverage the Rowclone mechanism to first copy one operand to the destination row (C=A), then executing the APP-AP sequence (C=f(C,B)). As mentioned above, the Rowclone copy operation can be completed by either AAP or oAAP.

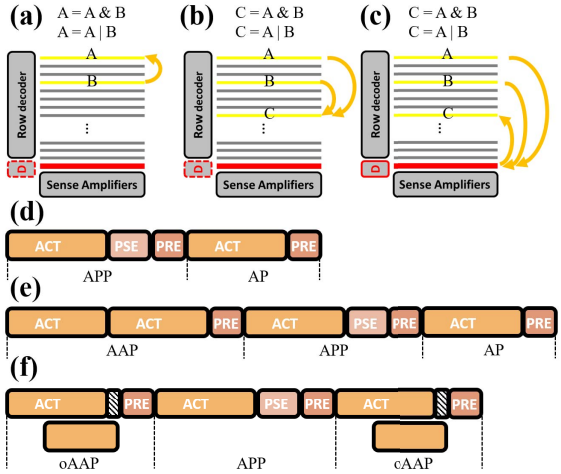

Figure 5: Three execution strategies and related primitive sequence for basic AND and OR operations. (a), (b), and (c) are related to (d), (e), and (f), respectively.

Here we present three execution strategies for basic AND and OR operations. As shown in Figure 5(a), the first one is the APP-AP sequence for the type of A=f(A,B). Figure 5(d) illustrates the related timing sequence of AAP-AP. For type of C=f(A,B), there are two strategies. As shown in Figure 5(b), an AAP-APP-AP sequence can be applied to implement logic operation within one decoder domain. The corresponding timing sequence is shown in Figure 5(e). In Figure 5(c), an oAAP-APP-oAAP sequence can be applied to complete logic operation with the help of reserved dual-connected row, which owns a separate wordline driver. The corresponding timing sequence is shown in Figure 5(f).

Among these three strategies, APP-AP has the smallest latency, but limited by the A=f(A,B) equation type. Latency of AAP-APP-AP is longer than oAAP-APP-oAAP, but saving the power of activating extra wordlines and transferring data. Therefore, AAP-APP-AP can be applied to a high throughput mode, where DRAM bank-level parallelism is limited by power constraint. oAAP-APP-oAAP can be designed for reduced latency mode, which explores the reserved row to accelerate DRAM logic operations.

4. IMPROVEMENT OF ELP²IM

The proposed ELP²IM introduced above still depends on the precondition that Cb is significantly lager than Cc. To accelerate DRAM operation, many designs prefer short bitline to reduce access latency [28, 29, 30]. Under this condition, Cb/Cc becomes smaller, which could harm the reliability of ELP²IM and even induce errors. Here we introduce an alternate strategy to avoid the influence of the ratio (Cb/Cc).

We further explore approaches to reduce the operation latency of ELP²IM. By leveraging the row-buffer decoupling approach [31], we overlap the pseudo-precharge state with precharge state. By trimming down the redundant operation, we reduce the latency of activation state.

4.1 Alternative Strategy

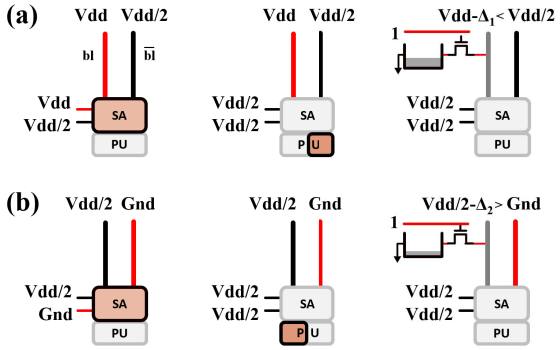

Figure 6: (a) Regular strategy, (b) alternative strategy of the Pseudo-precharge state, precharge state, and access phase in APP. (Note that the bitline pair in figure is not the physical layout in open-bitline architecture)

For ELP²IM with small Cb, the worst case for an OR operation is that ELP²IM reads logic '1' first and then senses logic '0', which is the case of '1'+'0'. Similarly, the worst case for an AND operation is '0'×'1'. Figure 6(a) shows the pseudo-precharge, precharge, and access phases in the worst case of OR operation. It accesses '1' in the first cell, and the pseudo-precharge retains '1' on bitline. After precharge, it enables the second cell, which stores '0', to perform the

charge sharing between Cb ('1') and Cc ('0'). Taking it to an extreme condition, where Cb < Cc, the charge sharing will produce a voltage level below Vdd/2, leading to a wrong result of the OR operation.

To solve the problem, we regulate $\overline{\text{bitline}}$, instead of bitline in the pseudo-precharge state. Because $\overline{\text{bitline}}$ and bitline are complementary, it requires to regulate $\overline{\text{bitline}}$ with the complementary voltage level of bitline. For example, if bitline was regulated to Vdd in previous pseudo-precharge state, $\overline{\text{bitline}}$ should be regulated to Gnd now. If bitline was regulated to Vdd/2, $\overline{\text{bitline}}$ should be regulated to Vdd/2 now. This complementary state of $\overline{\text{bitline}}$ can be achieved by changing voltage shift of SA in complement. As shown in Figure 6(b), the supply voltages of SA are Vdd/2 and Gnd in the pseudo-precharge state, which are opposite to the Vdd and Vdd/2 in Figure 6(a). In the following precharge state, only bitline is charged to Vdd/2 (Figure 6(b)), also opposite to the precharge state in Figure 6(a). Finally, in the access phase of Figure 6(b), charge is shared between Vdd/2 ('1/2') in Cb and '0' in Cc, and the resulting voltage level is above '0', which means voltage on bitline is larger than $\overline{\text{bitline}}$. Based on the fact that SA is a differential amplifier, it will generate a correct result of logic '1' on bitline.

It is provable that other conditions of OR operation, '1'+'1', '0'+'1', and '0'+'0', are still true for this complementary pseudo-precharge strategy. Symmetrically, AND operation can also be correctly executed using this strategy.

4.2 Optimizing Operation Sequence

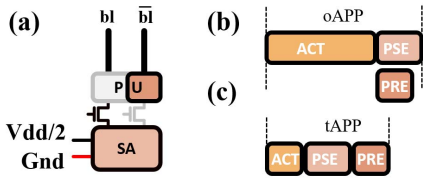

Figure 7: (a) Enable pseudo-precharge and precharge state simultaneously based on row-buffer decoupling approach (b) overlapped APP primitive (c) trimmed APP primitive

4.2.1 Leverage Isolation Architecture

Integrating an isolation transistor on bitline has been widely exploited in prior work [17,24,30,31], which can significantly improve the performance of DRAM with limited overhead. ELP^{2}IM can also gain benefit from these isolation strategies. Seongil *et al.* propose a row-buffer decoupling mechanism to relieve the latency penalty induced by precharge in open-page policy [31]. In the design, an isolation transistor is attached between SA and PU. With the isolation transistor, SA and PU can be enabled at the same time. As shown in 7(a), this approach can be adopted by ELP^{2}IM to overlap the pseudo-precharge and precharge state. We define the overlapped APP primitive as oAPP (Figure 7(b)). Taking advantage of oAPP, ∼21% latency can be saved compared to a regular APP (in case of DDR3-1600).

4.2.2 Leverage Restore Truncation

During the process of logic operations, many intermediate data will not be reused. Thereby, it is unnecessary to restore these data when accessing them. Based on the mechanism from [32], we trim down the restore phase in activate state. Specifically, after the sense phase, the pseudo-precharge state will be directly set up. We define the trimmed APP primitive as tAPP (Figure 7(c)). Leveraging tAPP, ∼31% latency can be saved compared to a regular APP (in case of DDR3-1600).

4.2.3 Explore More Buffers

In some cases, there is more than one copy of a variable in one Boolean expression. For example, each variable in the Boolean median operation, given by the expression AB+AC+BC, has more than one copy. Because ELP^{2}IM implements logic expression in the granularity of basic AND, OR, and NOT operations, any complex logic expression is required to be decomposed into the basic operations and executed sequentially. This would induce multiple accesses to one variable. Therefore, it is important to simplify the Boolean expression to the minimized form and explore more buffers for the reused data. In the following example of XOR operation, we elaborate the improvement step by leveraging oAPP primitive, tAPP primitive and additional buffers.

4.3 Example: XOR operation

Table 1 concludes 6 primitives applied in ELP^{2}IM. The timing parameters are based on DDR3-1600. In the following example, these 6 primitives are permuted to obtain an optimized operation.

Table 1: Primitives of ELP^{2}IM (DDR3-1600)

Primitive	Meaning	Time
AP	Activate-Precharge	49 ns
AAP	Activate-Activate-Precharge	84 ns
oAPP	overlapped Activate-Activate-Precharge	53 ns
APP	Activate-Pseudoprecharge-Precharge	67 ns
oAPP	overlapped Activate-Pseudoprecharge-Precharge	53 ns
tAPP	trimmed Activate-Pseudoprecharge-Precharge	46 ns

An exclusive OR (XOR) equation can be expressed as C=A$\overline{\text{B}}$+$\overline{\text{A}}$B, including two AND operations and one OR operation. As ELP^{2}IM can complete the AND and OR with a 'three-cycle' oAAP-APP-oAAP sequence, it takes at most three oAAP-APP-oAAP sequences, which is ∼519 ns, to implement XOR operation. In the following analysis, we show steps how to trim down the latency to ∼297 ns.

To execute C=A$\overline{\text{B}}$+$\overline{\text{A}}$B, as shown by the first sequence in Figure 8(a), we first use an oAAP-APP-oAAP subsequence to calculate C=$\overline{\text{A}}$B. In the second step, we leverage another oAAP-APP-AP to execute R=A$\overline{\text{B}}$, where R is the reserved dual-connected row. Thirdly, we use an APP-AP subsequence to obtain the result C=C+R. However, the AP in the second step and the APP in the third step both access to R. Thereby, they can be merged to one APP, as shown in the second sequence of Figure 8(a). Now the whole process only includes 7 primitives, which is ∼409 ns. Taking a further step, the merged APP finally stores the result A$\overline{\text{B}}$, which is a intermediate data in the XOR operation. Therefore, the restore phase in the APP can be trimmed down, which further reduces the latency to ∼388 ns, as shown in the sequence 3 of Figure 8(a).

Up to now, the optimized operation sequence still accesses variable A and B two times, as shown by the first and second oAAP-APP segments in the sequence 3. In order to better indicate the following optimization, we rearrange sequence 3 to sequence 4 (Figure 8(a)), which maintains the same

number of primitives. Sequence 5 is the optimized version of sequence 4, which leverages the row-buffer decoupling mechanism and substitutes APP with oAPP. It reduces the XOR operation latency to ∼346 ns. In sequence 5 (or 4), the second and third primitive both access B. The difference is that the second primitive copies data B, while the third one retains data B in form of regulated voltage on bitline. If there is another buffer, we can merge these two primitives into one, as shown in sequence 6 of Figure 8(a). Figure 8(b) shows the layout of data cell (A, B, C) and reserved cell (R0, R1) on the open-bitline architecture. Figure 8(c) indicates the commands corresponding to sequence 6. The variables in the '( )' are the rows that the command activates. For example, oAAP activates two rows, thereby two variables exist in the '( )'. Following the '( )' is the calculation done by the command. For an oAPP, the accessed variable will be regulated to either Vdd/2 ($\frac{1}{2}$) or Gnd (0) for an AND operation (Vdd (1) or Vdd/2 ($\frac{1}{2}$) for an OR operation). Therefore, we mix Boolean variable and number $\frac{1}{2}$ in the expression to indicate the regulation done by the oAPP in Figure 8(c). This additional buffer based approaching consists 6 primitives, finally reducing the XOR operation latency to ∼297 ns.

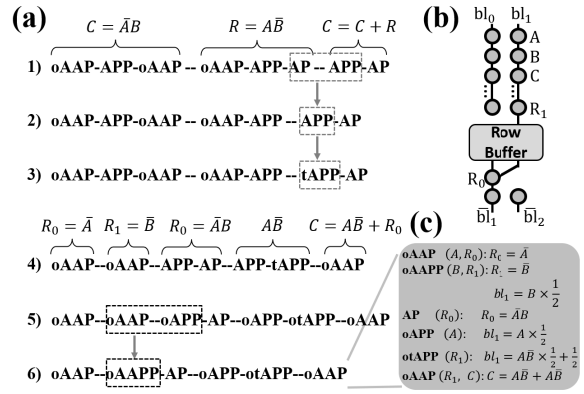

Figure 8: Optimization for primitive sequence of XOR operation

Note that sequence 6 requires two reserved rows. In the article, unless otherwise specified, we only utilize sequence 5 as the optimized primitive sequence for XOR operation. One character of all the primitives in Figure 8(a) is that they only request single or double row activation during each access. This is more friendly to the charge pump inside DRAM, which has limited output power [12].

5. DISCUSSION

In this section, we discuss the modification and related cost of ELP^{2}IM. There are two types of modifications: control sequence modification and hardware modification. We adjust control sequences by integrating new primitives in memory controller. We discuss the hardware change mainly in subarray level.

5.1 Integrating with System

There are 5 new primitives, AAP, APP, oAAP, oAPP, and tAPP, in Table 1. Memory controller of ELP^{2}IM has to be integrated with new control modes to support these primitives. The new modes can be implemented by adjusting the sequence of the three control signals: SA enable control, SA supply power control, and PU enable control. For example, APP primitive prolongs the enable time and adjusts the supply voltages of SA after activate state. In precharge state, it activates one side of the EQ pair.

For different operations, we adopt different permutations of the primitives to reduce cycles. This can be realized by a configurable memory controller, where specific primitive sequence can be buffered in the controller. The form of the primitive has been introduced in Figure 8(c),

$prmt([dst], src)$

where prmt is the type of primitives, dst is the destination address, src refers to the source address.

5.2 Hardware Cost

Modern DRAM is sensitive to modification, especially, in the DRAM array cells [22]. Compared to other designs, ELP^{2}IM induces less modification to DRAM array and reduces area overhead.

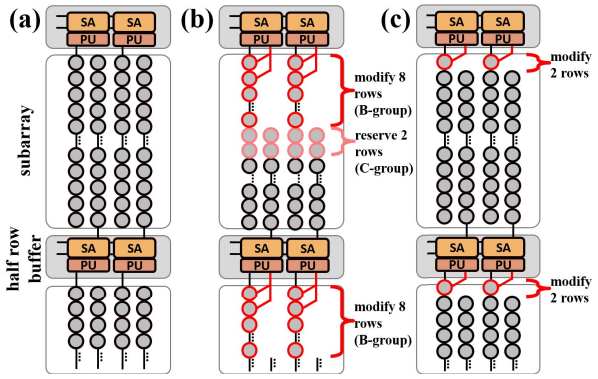

Figure 9: Hardware cost comparison of regular DRAM(a), Ambit(b), and ELP^{2}IM(c) in open-bitline architecture

Figure 9 shows the cost of array cells in different designs. Ambit has to retain 6 rows for B-group and 2 rows for C-group to implement the TRA-based calculation. Specifically, the 6 rows in B-group occupies 8 physical rows, because it includes two dual-connected-cell rows. What's more, the B-group also lowers the cell density, which means half of the allocated region will be empty or not used. ELP^{2}IM relieves this cost since it only allocates 1 dual-connected-cell row. ELP^{2}IM also has light change in PU, which separates the control signal of EQ. This will not hurt the array density, because it only influences the layout of metal layer.

To implement the oAPP primitive, the isolation transistor has to be attached on each bitline. Based on the evaluation from [31], the area overhead of isolation transistor is only ∼0.8% with 512×512 matrix size. Combing the 1 reserved row, the total array overhead of ELP^{2}IM is still 22% less than Ambit under open-bitline architecture.

Besides the modification in DRAM array, light change in the peripheral circuit, such as the drive strength enhancement of Vdd/2 supply and the separate wordline driver for the reserved row [33], would be required to better support the logic operation in ELP^{2}IM.

6. EVALUATION

In the evaluation of ELP^{2}IM, we first conduct circuit-level simulation to verify the timing and accuracy of logic operation in ELP^{2}IM. Then we analyze the performance and power efficiency in basic logic operations. Finally, we implement three case studies to evaluate the efficiency of ELP^{2}IM in real applications.

6.1 Circuit-level Simulation

We use H-spice for circuit-level simulation. The parameters are derived from Rambus power model [34, 35], which includes information such as circuit capacitances, resistances, and transistor dimensions. The timing of control signals in the circuit agrees with DDR3-1600.

6.1.1 Timing Simulation

In the simulation, we first analyze the timing of pseudo-precharge state. As the function of pseudo-precharge is similar to precharge, both intending to charge bitlines (or bitline) to Vdd/2, they are supposed to take the same time. However, based on our simulation, pseudo-precharge actually consumes a longer time. This is because the drive strength of SA is reduced when the supply (difference between supply voltages) is suppressed. Thankfully, the reduction is not significant, because the transistors in SA, different from the access transistors in DRAM cell, are built with low threshold (Vth) to improve sense and restore speed. Based on the fact that the Vth of transistors in SA is 25∼30% of Vdd [36, 37, 38], the drive strength of SA with half Vdd supply is reduced by 11∼23% in our simulation. On the other hand, the SA with half Vdd supply is used in pseudo-precharge state, which only charges the bitline. While in activation state, SA with regular supply is required to charge both bitline and cell capacitor. Subtracting the portion of charging cell capacitors, pseudo-precharge state is 13∼20% shorter than the restore time in activate state and 20∼30% longer than precharge state.

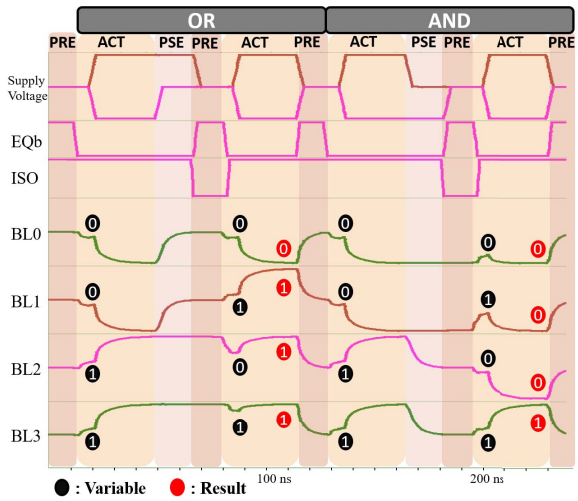

Figure 10: Waveform of ELP^{2}IM in OR, AND operations

Figure 10 shows the waveform of two APP-AP sequences, which are also critical steps for other complex primitive sequences. The first APP-AP executes an OR operation. In this operation, ELP^{2}IM accesses the first data during activate state. The data value can be recognized from the small variation of bitline at the beginning of each activate state. Then the pseudo-precharge state is switched on. If bitline is '0', it will be regulated to Vdd/2, otherwise, it will remain '1'. The following precharge state does not influence the voltage level of bitline, because it only drives the bitline, which is not shown in the waveform, to Vdd/2. Finally, in the second activate, the voltage change will not always follow the small variations induced by the second data cell. It actually complies with the result of an OR operation. The AND operation is executed in the same way, except '1' will be regulated to Vdd/2 in pseudo-precharge state.

6.1.2 Reliability Analyzing

The reliability of ELP^{2}IM, Ambit, and regular DRAM under different process variations (PV) is studied. The bitline coupling effect is also taken into consideration. To conduct the simulation, we first identify the worst case of each device. For ELP^{2}IM, bitlines that are pseudo-precharged to Vdd/2 have higher error possibility. For Ambit, inconsistent values in TRA, such as '101' or '010', tend to form a 'weak 1' or 'weak 0' after charge sharing, and possibly make the calculation failed. The worst case for coupling effect under open bitline structure are the data patterns that alternate between '0' and '1' in wordline direction [39, 40]. Second, given that PV can be categorized into systematic and random variations, it is difficult to determine every parameter of the circuit is influenced by which variation [41, 42]. We carry out Monte-Carlo simulations in two extreme conditions, where variations are all systematic or all random. Any other condition is the intermediate case between these two.

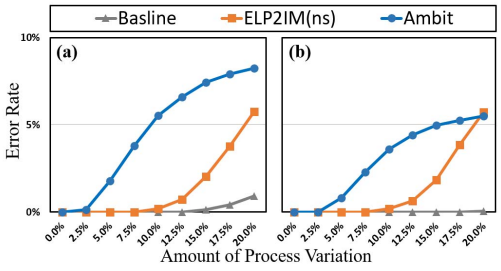

Figure 11: Error rate of ELP^{2}IM, Ambit, and regular DRAM

Figure 11 shows the simulation result. ELP^{2}IM exhibits lower error rate, especially under random PV (Figure 11(a)). The higher error rate of Ambit can be ascribe into two aspects. First, TRA approach originally reduces the bitline voltage sensing margin, because mismatch of the triple activated cells could induce more voltage deviation. Under systematic PV (Figure 11(b)), the triple cells are tend to be identical, and the error rate is suppressed. Second, the TRA may provide different variation scale on bitline, which aggravates the coupling effects. For example, a 'weak 0' could be driven close to Vdd/2 by its neighbouring 'strong 1's, which are generated by three '1' cells in TRA. In the simulation, the bitline shared (coupled) capacitor is set to 15% of bitline capacitor [40, 43, 44]. ELP^{2}IM can avoid the above problems that

Ambit suffers, because it only accesses one cell during the charge sharing process. On the other hand, even if the bitline coupling effect is aggravated in pseudo-precharge state, it can be avoided by the complementary pseudo-precharge strategy introduced in section 4.1. In the strategy, ELP^{2}IM regulates voltage on bitline, which is physically allocated in a different subarray.

Note that error rate of ELP^{2}IM is still higher than regular DRAM. This is reasonable, as any modification could break the original balance and induce inaccuracy. The inaccuracy of ELP^{2}IM mainly comes from the mismatch of Vdd/2. Because pseudo-precharge state may set Vdd/2 on bitline, and the following precharge state will set Vdd/2 on bitline. The two Vdd/2 voltages are charged via different paths, which may induce slight mismatch. Therefore, in the design of ELP^{2}IM, it is important to keep the supply voltage nodes at SA and PU equal when delivering Vdd/2.

ELP^{2}IM is also facing the challenge of error correction, which is a common problem faced by all bitwise-operation based PIMs (no matter using what kind of technologies, such as SRAM, DRAM, NVM, etc. [6, 45, 46, 47]), because the traditional error correcting code (ECC) is not compatible with bitwise logic operation. Further extensive research would be needed to tackle this problem. However, even without novel error checking method, bitwise PIM is still a promising architecture for error tolerant scenarios such as approximate computing or neural network acceleration.

6.2 Performance and Overhead Analysis

We implement ELP^{2}IM, Ambit and Drisa_nor in DDR3-1600 to evaluate the latency and power consumption when carrying out basic logic operations [14]. Note that DDR3-1600 is just an example, other type of DRAM is also compatible with the aforementioned designs.

Among the three designs, ELP^{2}IM shows the smallest latency in almost all the basic operations, as shown in Figure 12(a), which can be attributed to the less commands it takes. For example, to implement an AND operation, it only takes 3 primitives, as indicated in section 3.3, while Ambit requires 4 primitives. Drisa_nor consumes even longer time, excepting the NOR operation, because it needs to transfer any other logic to a compound of NOR logic, which induces more commands. On average, ELP^{2}IM is 1.17× faster than Ambit, and 1.12× than Drisa_nor. Note that ELP^{2}IM does not gain much benefit on XOR and XNOR operations, because it only has one reserved row. If there is one more buffer in ELP^{2}IM, the improvement can be 1.23× and 1.16× over Ambit and Drisa_nor.

We estimates power consumption based on spice simulation and the power parameters collected from Micron DDR3 DRAM power datasheet [48]. As the logic operation is conducted inside the subarray, which does not include read/write process, the background power and activate power become the main part in power consumption. In Figure 12(b), Drisa consumes more power as the additional logic gates and latches greatly increase background power. For ELP^{2}IM, the activate power of APP increases by ∼31% compared to the regular AP primitive. For Ambit, the multiple wordlines activation in TRA also increases power, ∼22% for each wordline, which can be attributed to the low power efficiency of charge pump. Although in Figure 12(b), ELP^{2}IM does not show much benefit on power consumption compared to Ambit, only 3% better, it still retains large potential in saving power. As it can include more power-saving primitives, such as AAP and AP, in the calculation. In the following case studies, the power of ELP^{2}IM is 17%∼27% less than Ambit.

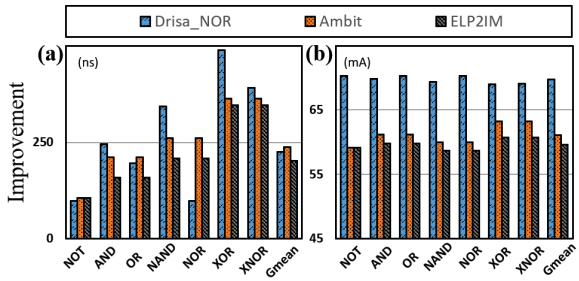

Figure 12: Latency and power of Drisa_nor, Ambit, and ELP^{2}IM in complementing basic logic

What's more, the increased part of power consumption in Ambit is more critical, because it is limited by the power delivery network and charge pumps, which have limited capacity to simultaneously drive multi-wordline in different banks. Therefore, in consideration of power constraint, the throughput of Ambit will significantly drop below ELP^{2}IM.

For the area overhead, ELP^{2}IM takes even less change than Ambit, as discussed in the 5.2. While Drisa_nor modifies DRAM with 24% area overhead, which is a big challenge for DRAM manufacturing.

To sum up, ELP^{2}IM is the most efficient design in aspects of latency, power, throughput, and area. Adding logic circuit directly in subarray level could improve specific operation in the design of Drisa_nor, but cannot bring general benefit when compared to ELP^{2}IM and Ambit.

6.3 Improvement for Applications

We build an in-house simulator to calculate the latency, throughput, and power consumption of ELP^{2}IM on several applications. We configure a regular DRAM module with 8 banks. The baselines are Ambit, Drisa_nor, and Kaby-Lake CPU [49]. As shown in case study of Bitmap [50] and Bitweaving [51], we first implement ELP^{2}IM in the light-modified DRAM designs, whose main purpose is to restore data and thereby is sensitive to reserved space overhead. Therefore, these designs should be close to commodity DRAM, and the power constraint and power efficiency are important factors. Secondly, we implement ELP^{2}IM in two accelerator designs, where DRAM is built in a application-specific way and performance is the first order consideration. Thereby, we construct ELP^{2}IM with two reserved rows to buffer more data and reduce processing latency.

6.3.1 Case Study: Bitmap

Bitmap indices [50] can offer fast analytics in specific applications for databases. Traditionally, the bulk bitwise operation occupies a significant time in execution of Bitmap indices. Thereby, the DRAM-based bitwise accelerators are promising candidates for Bitmap. Here we impose Bitmap on an application which tracks the activities of a large number of

users (16 million). The tracking case is to count the number of users who were active every week for the past w weeks, and the number of male users who were active each of the past w weeks. The case can be divided into two parts: bulk bitwise operation, which is executed in ELP^{2}IM, and count operation, which is performed with the help of CPU.

In the case study of Bitmap, we compare ELP^{2}IM with 3 Ambit designs which are configured with different reserved space. The baseline is the throughput of implementing the whole case in CPU. We first conduct the experiment without power constraint, which means DRAM can activate all 8 banks simultaneously (even it is not realistic). Then we set on power constraint to evaluate the performance again.

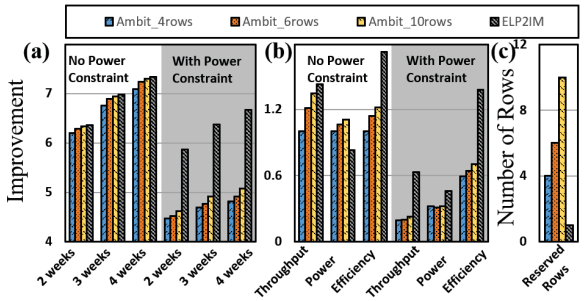

Figure 13: (a)System throughput improvement offered by ELP^{2}IM and Ambit with different configurations of reserved space, (b)ELP^{2}IM and Ambit device performance, excluding CPU, (c)reserved space

The result is shown in Figure 13, where (a) indicates the whole system performance, and (b) shows the average performance of DRAM device. Figure 13(c) shows the number of reserved rows in the two designs. We draw three conclusions from the experiment. First, by allocating more reserved rows, system performance of Ambit is improved, as shown in Figure 13(a). However, the improvement is not linear with the increment of reserved rows. The throughput gains a significant improvement when the number of reserved rows are raised from 4 to 6, but the growth is much slower when reserved rows are increased from 6 to 10. What's more, we find that even Ambit is allocated more than 10 reserved rows, it cannot catch up ELP^{2}IM. Second, under condition of power constraint, the device throughput of Ambit is greatly reduced, which drops up to ∼83% (Figure 13(b)). ELP^{2}IM is also influenced, because the number of activated banks in the same activate window is now decreased to the half, from 8 to 4. However, its device throughput only drops 56%, which is quite close to the ratio of decreased banks, 1/2. Third, even allocated with different number of reserved rows, the device throughput of Ambit (Figure 13(b)) tends to be the same under power constraint, implying more reserved space cannot offer much benefit under such condition.

6.3.2 Case Study: Table Scan

Table scan [52] is a common operation in a memory-based database management system. It sequentially reads the database and checks the columns for the validity of a predicate of a query. It usually takes many cycles to evaluate simple predicates. For example, a database query Q1 can be written as the following:

$$Q1: SELECT\ \ COUNT(*)\ \ FROM\ \ R\ \ WHERE\ \ R.a < C1$$

where R.a < C1 is a simple LESS THAN predicate. It involves a significant number of comparison and increment operations. [51] proposes the BitWeaving method to parallelize comparisons for multiple words. It permutes each word to store it in a memory column. Hence, the same bit in multiple words can be compared with bulk bitwise operation.

In the Table Scan case study, Drisa_nor is studied in the comparison with ELP^{2}IM and Ambit. Given Table Scan is the databases application, where memory capacity is the major consideration, ELP^{2}IM, Ambit and Drisa_nor (even Drisa_nor has much area overhead) are regarded as light-modified designs which are sensitive to the overhead of reserved rows and under the limitation of power constraint. We use the same calculation strategy as Bitmap. CPU executes the count operation in BitWeaving. Ambit, Drisa_nor, or ELP^{2}IM implements bulk bitwise operations.

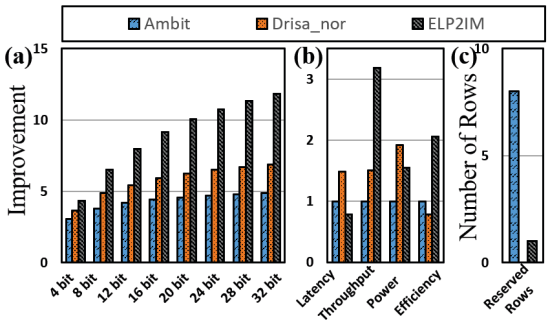

Figure 14: (a)Throughput improvement over our baseline CPU offered by Ambit, Drisa_nor, and ELP^{2}IM for table scan (b)DRAM device performance, (c)reserved space

In the evaluation, ELP^{2}IM has the highest throughput, as shown in Figure 14(a), which can be attributed to the small latency and high bank-level parallelism. What's more, the improvement of ELP^{2}IM grows up quickly with the increase in data width, because in BitWeaving method, the proportion of CPU-implemented counting is reduced when data width is extended. However, Ambit cannot gain such obvious improvement because its parallelism is impeded by power constraint. Figure 14(b) is the average device performance of Ambit, Drisa_nor, and ELP^{2}IM, excluding the part of CPU. ELP^{2}IM still maintains the best performance in all the aspects. Although Drisa_nor is integrated with the high-speed logic gates near row buffers, its latency is still the largest. However, the throughput of Drisa_nor outperforms Ambit, because Ambit is hindered by the multiple row activates under power constraint. Figure 14(c) indicates that Ambit and ELP^{2}IM are allocated with 8 rows and 1 row respectively. As Drisa_nor is attached with additional latches near row buffers, it does not require reserved rows, but in cost of even more area. In conclusion, ELP^{2}IM consumes the least resource and achieves the best performance.

6.3.3 Case Study: CNN

Several studies use bitwise operation to accelerate convolu-

tional neural network (CNN) applications, such as Dracc [19] and NID [53, 54]. We perform these two designs as case studies to evaluate the efficiency of our proposed ELP^{2}IM. Given that CNN accelerator is more focused on calculation performance, it would be acceptable for DRAM to be added with more circuit to break the original limitation in DRAM, even though at some cost of density [55, 56]. Thereby, we do not set the limitation of power constraint in the simulation. For ELP^{2}IM, we exploit the sequence 6 in Figure 8 for XOR operation. For Ambit, we allocate a large number of reserved rows, which helps reduce the calculation latency.

Dracc builds in-memory adders by modifying the subarray with limited number of transistors. Therefore, besides the bitwise operation, Dracc also realizes word-wise addition in subarray level. Leveraging the integrated adder, Dracc can be applied to the ternary weight neural networks (TWNs), which replaces full precision weights with ternary weights ('-1', '0' or '+1'). Thereby, it transfers dot product calculation to addition operations [57, 58]. Dracc also proposes several operation modes, such as high throughput mode, single frame mode, and low power mode, to meet different application requirements.

Table 2: Application in Dracc

	Lenet5	Cifar10	Alexnet	VGG16	VGG19
Ambit (FPS)	7697.4	6008.4	84.8	4.8	4.1
ELP^{2}IM (FPS)	8329.5	6850.5	96.4	5.4	4.6
Improvement	1.08×	1.14×	1.14×	1.13×	1.13×
Drisa_nor (FPS)	6107.2	3889.7	55.5	3.2	2.7
Improvement	0.79×	0.65×	0.66×	0.68×	0.66×

Dracc implements addition based on Ambit approach. It divides an addition operation into several basic logic steps, which can also be realized by ELP^{2}IM and Drisa_nor. In our simulation, we exploit the three designs to realize the adder in Dracc separately. We first optimize the command sequence to achieve the minimum operation cycles. Then we run TWNs in the high throughput mode of Dracc, which can fully exploit the hardware resources.

Table 2 shows the result of Dracc based on the three approaches. Given that there are only 13 commands (including two new propagation and shift commands, which cannot be optimized) for the addition operation in Dracc, the space left for optimization is limited. ELP^{2}IM still improves the performance by 12% (on average). Note that this improvement is achieved by the shorter latency of ELP^{2}IM, instead of higher parallelism, as we have removed the limitation of power constraint. In the opposite, Drisa_nor impedes the performance by 31%. Depending on the proportion of data computation and data movement in different neural networks, the improvement (or reduction) changes.

NID realizes binary CNN inside DRAM. It implements XOR and count operations, which are the dominant calculations in binary CNN [59, 60], by exploiting Ambit approach. To implement the count operation, NID firstly permutes each word and stores it in column-wise. Then it decomposes the count operation into minimum number of AND and XOR operations, thereby realizing the count with bulk bitwise operation. NID is embedded with accumulators and comparators in peripheral area to assist the processing of other layers, such as max pooling and normalization.

Table 3: Application in NID

	Lenet5	Alexnet	Resnet18	Resnet34	Resnet50
Ambit (FPS)	7525.1	227.1	9.5	4.7	1.4
ELP^{2}IM (FPS)	9958.7	252.6	12.4	6.1	1.7
Improvement	1.32	1.11×	1.31×	1.31×	1.25×
Drisa_nor (FPS)	5497.1	205.9	7.1	3.5	1.1
Improvement	0.73×	0.91×	0.74×	0.74×	0.79×

In our experiment, we apply Ambit, ELP^{2}IM, and Drisa_nor to implement the XOR and count operations in NID. Then we compare the performance obtained by the three methods, as shown in table 3. On average, ELP^{2}IM achieves 1.26× throughput. This can be attributed to the fact that ELP^{2}IM is initially faster than Ambit in basic logic operations. Thus it is also faster in the compound Boolean function. The other implicit reason is that ELP^{2}IM is better at optimizing command sequence, because it contains 6 different primitives, which makes the optimization quite flexible. Meanwhile, the count operation in NID contains a large number of cycles, offering a wide space for optimization. Therefore, ELP^{2}IM achieves significant improvement in NID. Drisa_nor is neither faster than Ambit in basic operations nor flexible in the optimization of command sequence. Therefore, it loses performance by 22% on average.

7. CONCLUSION

In conclusion, we present ELP^{2}IM, which provides a solid improvement for bitwise operation using DRAM technology. It retains the benefit and breaks the limitation of state-of-the-art work. It is based on a lightweight pseudo-precharge state, which offers the opportunity to implement in-place operation and improves calculation accuracy.

Benefiting from the mechanism, ELP^{2}IM improves the performance by reducing the number of primitives and intra-subarray data movement. ELP^{2}IM lowers the power consumption by diminishing the number of simultaneously activated rows. Meanwhile, ELP^{2}IM achieves such significant improvement even with less reserved space, relieving the modification on DRAM.

Our simulation result shows that, in bitmap and table scan application, ELP^{2}IM achieves up to 3.2× throughput improvement in consideration of power constraint. Even without the limitation of power constraint, ELP^{2}IM still achieves up to 1.26× throughput in CNN applications.

8. ACKNOWLEDGMENTS

This work is supported in part by US National Science Foundation #1422331, #1535755, #1617071, #1718080, #1725657, #1910413, The authors thank the anonymous reviewers for their constructive comments.

9. REFERENCES

[1] O. Villa, *et al.*, "Scaling the Power Wall: A Path to Exascale," in *SC*, 2014.

[2] S. McKee, *et al.*, "Reflections on the Memory Wall," in *CF*, 2004.

[3] B. Akin, *et al.*, "Data Reorganization in Memory Using 3D-stacked DRAM," in *ISCA*, 2015.

[4] H. Asghari-Moghaddam, *et al.*, "Chameleon: Versatile and Practical near-DRAM Acceleration Architecture for Large Memory Systems," in *MICRO*, 2016.

[5] A. Subramaniyan, *et al.*, "Parallel Automata Processor," in *ISCA*, 2017.

[6] S. Li, *et al.*, "DRISA: A DRAM-based Reconfigurable In-Situ Accelerator," in *MICRO*, 2017.

[7] Y. Kim, *et al.*, "Assessing merged DRAM/logic technology," in *INTEGRATION, the VLSI journal*, 27, 2, 179-194, 1999.

[8] V. Seshadri, *et al.*, "Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.

[9] Boroumand, Amirali, *et al.*, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *ASPLOS*, 2018.

[10] D. E. Knuth, *et al.*, "The Art of Computer Programming. Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams," 2009.

[11] K. Wu, *et al.*, "Compressing Bitmap Indexes for Faster Search Operations," In *SSDBM*, 2002.

[12] M. Shevgoor, *et al.*, "Quantifying the relationship between the power delivery network and architectural policies in a 3D-stacked memory device." in *MICRO*, 2013.

[13] N. Chatterjee, *et al.*, "Architecting an energy-efficient dram system for gpus," in *HPCA*, 2017.

[14] JEDEC. DDR3 SDRAM Standard, JESD79-3D. `http://www.jedec.org/sites/default/files/docs/JESD79-3D.pdf`, 2009.

[15] P. Nair, *et al.*, "ArchShield: Architectural Framework for Assisting DRAM Scaling by Tolerating High Error Rates," in *ISCA*, 2013.

[16] T. Zhang, *et al.*, "Half-DRAM: A High-bandwidth and Low-power DRAM Architecture from the Rethinking of Fine-grained Activation," in *ISCA*, 2014.

[17] K. K. Chang, *et al.*, "Low-cost Inter-linked Subarrays (LISA): Enabling Fast Inter-subarray Data Movement in DRAM." In *HPCA*, 2016.

[18] V. Seshadri, *et al.*, "RowClone: Fast and Energy-efficient in-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.

[19] Q. Deng, *et al.*, "DrAcc: A DRAM Based Accelerator for Accurate CNN Inference," in *DAC*, 2018.

[20] X. Xin, *et al.*, "ROC: DRAM-based Processing with Reduced Operation Cycles," in *DAC*, 2019.

[21] T. Zhang, *et al.*, "CREAM: A concurrent-refresh-aware DRAM memory architecture," In *HPCA*, 2014.

[22] S. Lu, *et al.*, "Improving DRAM Latency with Dynamic Asymmetric Subarray," in *MICRO*, 2015.

[23] Fl. Krisztián, *et al.*, "Drowsy caches: simple techniques for reducing leakage power," in *Computer Architecture News*, 2002.

[24] B. Keeth, *et al.*, "DRAM Circuit Design: Fundamental and High-Speed Topics (2nd ed.," *Wiley-IEEE Press*, 2007.

[25] P. Salvador, *et al.*, "Exploiting temporal locality in drowsy cache policies," in *Proceedings of the 2nd conference on Computing frontiers*, 2005.

[26] F. Brendan, *et al.*, "Drowsy cache partitioning for reduced static and dynamic energy in the cache hierarchy," in *International Green Computing Conference Proceedings*, 2013.

[27] Y. Hamamoto, *et al.*, "Overview and future challenges of floating body RAM (FBRAM) technology for 32 nm technology node and beyond," in *Solid-State Electronics journal*, 53, 7, 676-683, 2009.

[28] T. Ting, *et al.*, "23.9 An 8-channel 4.5 Gb 180GB/s 18ns-row-latency RAM for the last level cache," In *ISSCC*, 2017.

[29] Y. H. Son, *et al.*, "Reducing memory access latency with asymmetric dram bank organizations," In *ISCA*, 2013.

[30] D. Lee, *et al.*, "Tiered-latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.

[31] O. Seongil, *et al.*, "Row-buffer Decoupling: A Case for Low-latency DRAM Microarchitecture,"In *ISCA*, 2014.

[32] X. Zhang, *et al.*, "Restore truncation for performance improvement in future DRAM systems," in *HPCA*, 2016.

[33] G. Fredeman, *et al.*, "17.4 A 14nm 1.1 Mb embedded DRAM macro with 1ns access." In *ISSCC*, 2015.

[34] DRAM Power Model, `https://www.rambus.com/energy/`, 2010.

[35] T. Vogelsang, *et al.*, "Understanding the Energy Consumption of Dynamic Random Access Memories," in *MICRO*, 2010.

[36] H. Kang, *et al.*, "A Sense Amplifier Scheme with Offset Cancellation for Giga-bit DRAM" *Semiconductor Technology and Science journal*, 7, 2, 67-75, 2007.

[37] S. Chung, *et al.*, "Method and System for DRAM Sensing." *US Patent* 7,369,425, 2008.

[38] T. Na, *et al.*, "Comparative study of various latch-type sense amplifiers." *IEEE Transactions on VLSI journal*, 22, 2, 425-429, 2013.

[39] S. Seyedzadeh, *et al.*, "Mitigating bitline crosstalk noise in dram memories." In *MEMSYS*, 2017.

[40] Y. Konishi, *et al.*, "Analysis of coupling noise between adjacent bit lines in megabit DRAMs." *Solid-State Circuits journal*, 24, 1, 35-42, 1989.

[41] B. Zhao, *et al.*, "Process variation-aware nonuniform cache management in a 3D die-stacked multicore processor," in *IEEE Transactions on Computers journal*, 62, 11, 2252-2265, 2013.

[42] A. Agrawal, *et al.*, "Mosaic: Exploiting the spatial locality of process variation to reduce refresh energy in on-chip eDRAM modules," in *HPCA*, 2014.

[43] Y. Nakagome, *et al.*, "The impact of data-line interference noise on DRAM scaling," *Solid-State Circuits journal*, 23, 5, 1120-1127, 1988.

[44] J. Liu, *et al.*, "An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms," *Computer Architecture News journal*, 41, 3, 60-71, 2013.

[45] C. Eckert, *et al.*, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *ISCA*, 2018.

[46] S. Li, *et al.*, "Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories," in *DAC*, 2016.

[47] M. Imani, *et al.*, "FloatPIM: In-Memory Acceleration of Deep Neural Network Training with High Precision," in *ISCA*, 2019.

[48] Micron, "MT41J256M16HA-125 Data Sheet," `http://www.micron.com/products/dram/`.

[49] 7th Generation Intel Core Processor Family for S Platforms, Vol. 1, Datasheet. `https://www.intel.com/content/www/us/en/processors/core/7th-gen-core-family-desktop-s-processor-lines-datasheet-vol-1.html`.

[50] C. Chan, *et al.*, "Bitmap Index Design and Evaluation," in *SIGMOD Rec. journal*, 27, 2, 355-366, 1998.

[51] Y. Li, *et al.*, "BitWeaving: Fast Scans for Main Memory Data Processing," in *SIGMOD*, 2013.

[52] W. Thomas, *et al.*, "Vectorizing Database Column Scans with Complex Predicates," in *ADMS*, 2013.

[53] J. Sim, *et al.*, "NID: Processing Binary Convolutional Neural Network in Commodity DRAM," in *ICCAD*, 2018.

[54] H. Kim, *et al.*, "NAND-Net: Minimizing Computational Complexity of In-Memory Processing for Binary Neural Networks," in *HPCA*, 2019.

[55] L. Jinag, *et al.*, "XNOR-POP: A processing-in-memory architecture for binary Convolutional Neural Networks in Wide-IO2 DRAMs," in *ISLPED*, 2017.

[56] S. Li, *et al.*, "SCOPE: A stochastic computing engine for dram-based in-situ accelerator," in *MICRO*, 2018.

[57] F. Li, *et al.*, "Ternary weight networks." arXiv preprint arXiv:1605.04711, 2016.

[58] C. Zhu, *et al.*, "Trained ternary quantization." arXiv preprint arXiv:1612.01064, 2016.

[59] A. Renzo, *et al.*, "YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights," in *ISVLSI*, 2016.

[60] C. Matthieu, *et al.*, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *NIPS*, 2015.