GraphiDe: A Graph Processing Accelerator leveraging In-DRAM-Computing

Shaahin Angizi and Deliang Fan
Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816
angizi@knights.ucf.edu,dfan@ucf.edu

ABSTRACT

In this paper, we propose *GraphiDe*, a novel DRAM-based processing-in-memory (PIM) accelerator for graph processing. It transforms current DRAM architecture to massively parallel computational units exploiting the high internal bandwidth of the modern memory chips to accelerate various graph processing applications. *GraphiDe* can be leveraged to greatly reduce energy consumption and latency dealing with underlying adjacency matrix computations by eliminating unnecessary off-chip accesses. The extensive circuit-architecture simulations over three social network data-sets indicate that *GraphiDe* achieves on average 3.1× energy-efficiency improvement and 4.2× speed-up over the recent DRAM based PIM platform. It achieves ~59× higher energy-efficiency and 83× speed-up over GPU-based acceleration methods.

ACM Reference format:
Shaahin Angizi and Deliang Fan. 2019. GraphiDe: A Graph Processing Accelerator leveraging In-DRAM-Computing. In *Proceedings of Great Lakes Symposium on VLSI 2019, Tysons Corner, VA, USA, May 9–11, 2019 (GLSVLSI '19),* 6 pages.
https://doi.org/10.1145/3299874.3317984

1 INTRODUCTION

Nowadays, reaching high bandwidth of graph processing on top of Von-Neumann architectures suffers from various challenges [6], such as long memory access latency, intensified congestion at I/Os, humongous data communication energy, and large leakage power consumption for storing graph parameters that result in over 90% bandwidth degradation on CPU-DRAM hierarchy [18]. In order to tackle these challenges, Processing-in-Memory (PIM), as a potentially viable way to solve the memory wall challenge, has been put forward [4, 11, 16]. The key idea of PIM is to realize computation units inside memory to process data by leveraging the inherent parallel computing mechanisms and exploiting large internal memory bandwidth. Therefore, total memory bandwidth for computation units scales well by increase memory capacity leading to a significant reduction in latency and energy overheads of data communication [3]. PIM architectures ideally should be capable of performing bulk bit-wise operations which is needed in many graph processing applications [12]. However, this has been limited to basic logic operations such as `AND`, `OR` and `XOR` so far [12, 16], which are not necessarily applicable to a wide variety of tasks except by imposing multi-cycle operations [5, 16] or large in-memory computational units [11] to realize specific functions such as addition.

The proposals for exploiting SRAM-based [2, 9] PIM architectures can be found in recent literature. However, PIM in context of main memory (DRAM- [3, 11, 15–17]) has drawn much more attention mainly due to larger memory capacities and off-chip data transfer reduction as opposed to SRAM-based PIM. Ambit [16] shows DRAM-based graph processing acceleration by realizing a majority function between every three rows and so can implement 2-input logic after saving operand data in reserved rows to avoid data-overwritten. GraphH [6] and Graphpim [14] present new designs based on Hybrid Memory Cube (HMC) to accelerate large-scale graph processing tasks at architectural level.

From graph processing algorithm perspective, network topology analysis can help us better understand the intricate connectivity of complex networks in practical problems. For instance, degree centrality is often used to measure the importance of a vertex. In social networks, people with more connections tend to have more significant influence in the community. The matching index is another basic topology parameter characterizing the similarity between two vertices in a network. It measures the ratio of common neighbors for pairs of vertices. Evaluation of these network properties plays an essential part in potential applications, such as social network analysis and traffic flow control. The main ***goal*** of this paper is to develop a parallel and energy-efficient PIM architecture that could simultaneously work as main memory and realize a high performance accelerator for such data-intensive graph processing applications. The main contributions of this paper are summarized as follows: (1) We propose a novel DRAM-based in-memory accelerator, *GraphiDe*, based on set of novel microarchitectural and circuit-level schemes. *GraphiDe* can perform any bulk bitwise operation inside DRAM exploiting DRAM structure, and therefore requiring low cost on top of commodity DRAM chip area. (2) We provide case studies of how important graph processing workloads can be partitioned and mapped to our architecture and how they can benefit from it. (3) We evaluate our proposed scheme using a variety of real-world social network graph data compared with other state-of-the-art accelerators i.e. DRAM, HMC, and GPU.

2 PROCESSING-IN-DRAM BACKGROUND

A DRAM cell basically consists of two elements, a capacitor (storage) and an Access Transistor (AT) (Fig. 1b Ⓑ). The drain and gate of the AT is connected to the Bit-line (*BL*) and Word-line (*WL*), respectively. DRAM cell encodes the binary data by the charge of the capacitor. It represents logic '1' when the capacitor is full-charged, and logic '0' when there is no charge. Technically, accessing data from a DRAM's sub-array (write/read) has three consecutive steps

GLSVLSI '19, May 9–11, 2019, Tysons Corner, VA, USA

ACM ISBN 978-1-4503-6252-8/19/05…$15.00
https://doi.org/10.1145/3299874.3317984

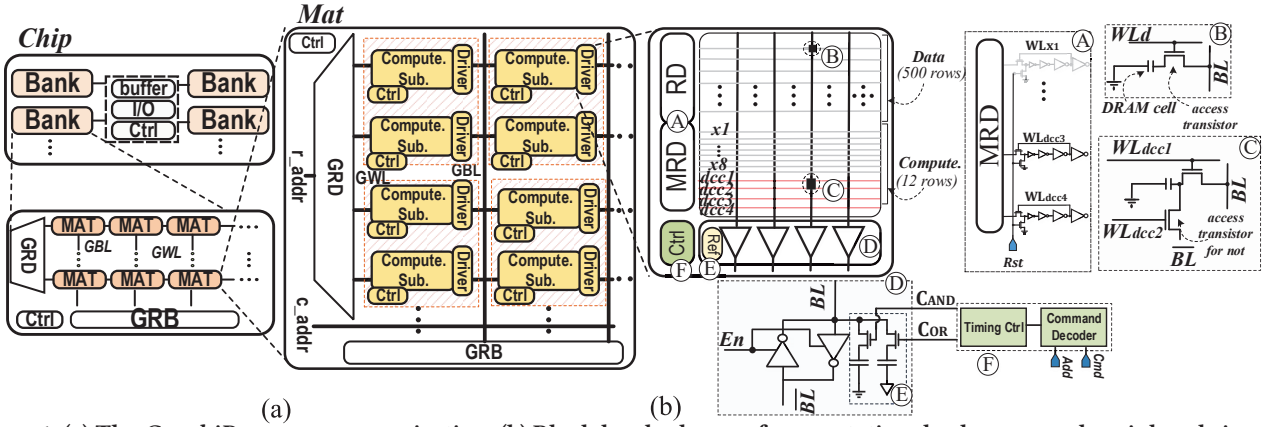

Figure 1: (a) The *GraphiDe* memory organization, (b) Block level scheme of computational sub-array and peripheral circuitry.

[15, 16]. During the first step (i.e. ACTIVATE), activating the specified row, data is copied from the DRAM cells to the Sense Amplifiers (SA). Then in the next step, such data can be accessed from the SAs by a READ or WRITE command. In addition, multiple READ/WRITE commands can be issued to an identical row. Afterwards, the activated row is precharged by issuing PRECHARGE command.

Following the aforementioned DRAM mechanism, series of circuit-level design have been presented, that enable design opportunity of applying PIM strategy to DRAM cell arrays. *RowClone* [17] presents an innovative mechanism to realize fast and efficient copy operation within DRAM sub-arrays without need to send the data to the processing unit. In this scheme, two back-to-back ACTIVATE commands to the source and destination rows without PRECHARGE command in between, leads to a multi-kilo byte in-memory copy operation. *Ambit* [16] extends the idea by realizing a 3-input majority function (Maj3)-based operation in memory by issuing ACTIVATE command to three rows simultaneously followed by a single PRECHARGE command so-called *Triple Row Activation (TRA)* method. Considering one row as the control, initialized by '0'/'1', Ambit can readily implement in-memory AND2/OR2 function. The NOT function has been also carried out in different works employing Dual-Contact Cells (DCC) [13, 16] with issuing two back-to-back ACTIVATE commands. DCC (Fig. 1b Ⓒ) first activates the WL_{dcc1} of input DRAM cell, and reads the data out to the SA through *BL*. It then activate WL_{dcc2} to connect $\overline{BL}$ to the same capacitor and so writes the negated result back to DCC. *Dracc* [7] implements a carry look-ahead adder by enhancing Ambit to accelerate convolutional neural networks.

3 GRAPHIDE DESIGN

GraphiDe is designed to be an independent, high-performance, energy-efficient accelerator based on main memory architecture. The main memory rank is a set of DRAM chips. Each chip is divided into multiple Banks. Banks within the same chip typically share I/O, buffer and banks in different chips working in a lock-step manner. Each bank consists of multiple memory matrices (mats). The general mat organization of *GraphiDe* is shown in Fig. 1a. Each mat consists of multiple computational memory sub-arrays connected to a Global Row Decoder (GRD) and a shared Global Row Buffer (GRB). According to the physical address of operands within memory, *GraphiDe*'s Controller (Ctrl) is able to configure the sub-arrays to perform data-parallel intra-sub-array computations. Our design is motivated by Ambit [16] PIM method, which leverages charge sharing among different rows to perform logic operations, but with significant modifications. We divide the *GraphiDe*'s sub-array row space into two distinct regions as depicted in Fig. 1b: 1-Data rows (500 rows out of 512) that include the typical DRAM cells Ⓑ connected to a regular Row Decoder (RD), and 2- Computation rows (12), connected to a Modified Row Decoder (MRD) (Fig. 1b Ⓐ), which enables bulk bitwise in-memory operations between copied operands. Eight computational rows ($x1, ..., x8$) include typical DRAM cells and four rows ($dcc1, ..., dcc4$) are allocated to special DCCs Ⓒ enabling NOT function in every sub-array. In the following, we propose dual-row in-memory AND-OR and addition operations that further enhance Ambit to perform graph processing tasks.

3.1 Dual-row in-memory logic

With a careful observation on Ambit's TRA method, we notice that it imposes an excessive latency and energy to memory chip which could be alleviated by rethinking about the process. Given R=A*op*B function (*op* ∈ AND2/OR2), Ambit takes 4 consecutive steps to calculate the result as: 1-RowClone data of row A to row $x1$ (Copying first operand to a computation row to avoid data-overwritten), 2-RowClone of row B to $x2$, 3-RowClone of ctrl row to $t3$ (Copying initialized control row to a computation row), 4-TRA and RowClone data of row $x1$ to R row (Computation and Writing-back the result). As a matter of fact, every RowClone command imposes ~80ns [17], therefore TRA method takes averagely 320ns to perform in-memory operations. Our key idea to perform dual-row bit-line computing in *GraphiDe* is still based on majority function but by selecting different thresholds (references) when performing the charge sharing between selected memory cell(s). The proposed reconfigurable SA, as depicted in Fig. 1b Ⓓ, consists of a regular SA with two

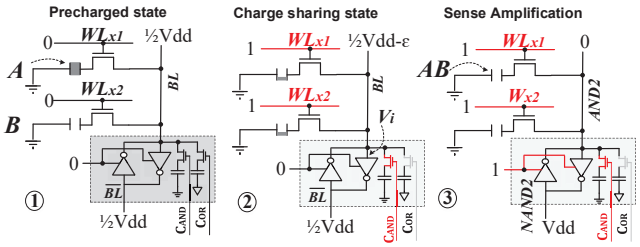

Figure 2: Realization of AND2 function within *GraphiDe*.

back-to-back inverters connected to two fixed reference-capacitor branches (E) that can be selected by control bits (C_{AND}, C_{OR}) by the sub-array's Ctrl (F). This design basically forms a capacitive voltage divider between two selected cells by MRD and the activated reference (connected to either GND or Vdd), driving a CMOS inverter, to implement AND2 or OR2 functions, respectively.

GraphiDe's Dual Row Activation method (DRA) eliminates the need for the third RowClone step in Ambit's AND2/OR2 operations and saves two initialized memory rows used for controls per sub-array at the cost of adding two low-overhead reference capacitors in SA unit. Fig. 2 shows the realization of AND2 operation in *GraphiDe*'s sub-array. Consider A and B operands are RowCloned from Data rows to $x1$ and $x2$ rows (1) and both BL and $\overline{BL}$ are precharged to $\frac{V_{dd}}{2}$. The DRA simultaneously activates two WLs and the corresponding reference (C_{AND}) for charge-sharing (2). During sense amplification (3), with the similar capacitance (C_c) of memory cells and the reference, input voltage of first inverter (V_i) in SA is simply derived as $V_i = \frac{n.V_{dd}}{C}$, where n denotes the number of DRAM cells storing logic '1' and C represents the total number of unit capacitors (C_c) connected to the inverter. Thus, the inverter acts as a threshold detector by amplifying deviation from $\frac{V_{dd}}{2}$ and realizes a NAND2 function on $\overline{BL}$ and consequently AND2 function (AB) on BL. *GraphiDe* can perform such DRA-based operations in ~240ns by eliminating the need for the third RowClone step in Ambit's operations. In this work, we use Ambit's TRA method just to directly realize in-memory majority function (Maj3) and AND2/OR2 operations are realized through DRA method.

3.2 In-memory adder

Here, we also propose Quintuple Row Activation method (QRA), as an extension for the TRA method, realizing 5-input (Maj5) operation. In this method, *GraphiDe*'s MRD (Fig. 1b (A)) helps to activate five WLs, simultaneously. During the precharged state as shown in Fig. 3 (1), both BL and $\overline{BL}$ are connected to $\frac{V_{dd}}{2}$. By activating the five WLs ($WLx1$ to $WLx5$), the memory cells storing input operands start to charge sharing (2). In this case, since three of the five cells are initially in the charged state, charge sharing results in a positive deviation on the BL. Therefore, by activating the Enable (En), such deviation from $\frac{V_{dd}}{2}$ is amplified (3) and the SA drives the BL to Vdd and accordingly, fully charges all the five cells. Based on Maj3 and Maj5 schemes, we now present a parallel in-DRAM computation and mapping method for addition (add) operation to accelerate a wide spectrum of graph processing tasks. Assume D_i, D_j, and D_k as input operands, the carry-out (C_{out}) of the Full-Adder (FA) can be generated through $MAJ3(D_i, D_j, D_k) = D_iD_j + D_iD_k + D_jD_k$ using TRA method. Moreover, the Sum can be readily carried out through $MAJ5(D_i, D_j, D_k, \overline{C_{out}}, \overline{C_{out}})$ with only writing back the $\overline{C_{out}}$ into memory (leveraging two DCC rows) and then applying QRA method. This will be further elaborated in subsection 3.3.

3.3 ISA support

While *GraphiDe* is meant to be an independent high-performance and energy-efficient accelerator, we need to expose it to programmers and system-level libraries to utilize it. From a programmer perspective, *GraphiDe* is more of a third party accelerator that can be connected directly to the memory bus or through PCI-Express lanes rather than a memory unit, thus it is integrated similar to that

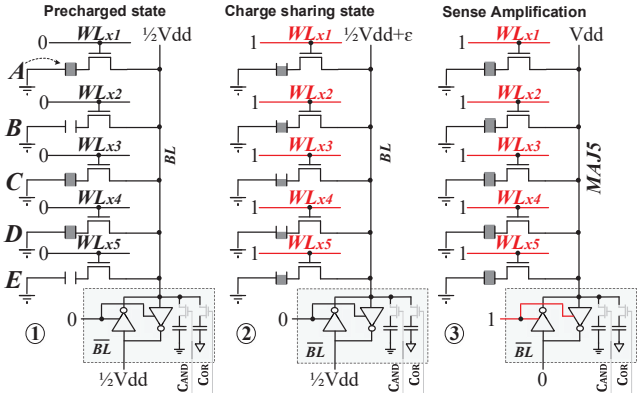

Figure 3: Realization of MAJ5 function within *GraphiDe*.

of GPUs. Therefore, a virtual machine and ISA for general-purpose parallel thread execution need to be defined similar to PTX [1] for NVIDIA. Accordingly, the programs are translated at install time to the *GraphiDe* hardware instruction set discussed here to realize the functions tabulated in Table 1. The micro and control transfer instructions are not discussed here.

GraphiDe is developed based on ACTIVATE-ACTIVATE-PRECHARGE command referred to as AAP primitives. As thoroughly explained in Ambit [16], most bulk bitwise operations involve a sequence of AAP commands. There are five types of AAP primitives supported by *GraphiDe* that only differ from number of activated source or destination rows, 1- AAP (src, des) that runs the following commands sequence: ACTIVATE source address; ACTIVATE destination address; PRECHARGE. This is manly used for copy and NOT functions as indicated Table 1. 2- AAP (src, des1, des2) that is designed to copy the result of an operation simultaneously to two destination rows. 3- AAP (src1, src2, des, Ctrl) that performs DRA method by activating two source addresses along with a control input ('0' for C_{AND}/ '1' for C_{OR}) and then writes back the result on the destination address. 4- AAP (src1, src2, src3, des) that performs TRA method by activating three source rows simultaneously and writing back the MAJ3 or MIN3 result on the destination address. 5- AAP (src1, src2, src3, src4, src5, des) that performs QRA method on five sources and write the result back to the destination address.

Table 1: The basic functions supported by *GraphiDe*.

Function	Operation	Command Sequence	AAP Type
copy	$D_r \leftarrow D_i$	AAP(D_i, D_r)	1
NOT	$D_r \leftarrow \overline{D_i}$	AAP(D_i, $dcc2$) AAP($dcc1$, D_r)	1 1
AND2	$D_r \leftarrow D_i.D_j$	AAP(D_i, $x1$) AAP(D_j, $x2$) AAP($x1$, $x2$, D_r, 0)	1 1 3
OR2	$D_r \leftarrow D_i + D_j$	AAP(D_i, $x1$) AAP(D_j, $x2$) AAP($x1$, $x2$, D_r, 1)	1 1 3
XOR2	$D_r \leftarrow D_i \oplus D_j$	AAP(D_i, $x1$, $dcc2$) AAP(D_j, $x2$, $dcc4$) AAP($x1$, $dcc3$, $x4$, 0) AAP($x2$, $dcc1$, $x5$, 0) AAP($x4$, $x5$, D_r, 1)	2 2 3 3 3
Addition	$Sum \leftarrow D_i \oplus D_j \oplus D_k$ $C_{out} \leftarrow MAJ3(D_i, D_j, D_k)$	AAP(D_i, $x1$) AAP(D_j, $x2$) AAP(D_k, $x3$) AAP($x1$, $x2$, $x3$, C_{out}) AAP(C_{out}, $dcc2$, $dcc4$) AAP($x1$, $x2$, $x3$, $dcc1$, $dcc2$, Sum)	1 1 1 4 2 5

In order to implement the addition-in-memory, as shown in Table 1, three AAP-type1 commands first copy the three input data rows to computational rows ($x1, x2, x3$). Then, C_{out} is generated by AAP-type4 and written back to the designated data row. Again, C_{out} row is readout and its inversion is copied to two DCC rows ($dcc2$ and $dcc4$) with AAP-type2. Eventually, AAP-type5 command activates five rows to implement *Sum* function.

3.4 Reliability

We performed an extensive circuit-level simulations following the Ambit's approach [16] to study the effect of process variation on both DRA and QRA methods considering a worst-case scenario variation in all components (cell/BL/WL capacitance and transistor). We ran Monte-Carlo simulation with 45nm PTM library [21] (DRAM cell parameters were taken from Rambus [8] model) under 10000 trials and increased the amount of variation from ±0% to ±20% for each method. Table 2 shows the percentage of the test error in each variation. We observe that even considering a significant ±10% [16] variation, the percentage of erroneous DRA or QRA across 10000 trials is just 0.12% and 0.39% which is consistent with what Ambit reports. Therefore, *GraphiDe* shows an acceptable reliability in performing PIM operations. Note that DRA method is less vulnerable to capacitance variation effects as opposed to TRA, due to its third fixed-voltage branch. By scaling down the transistor size, the process variation effect is expected to get worse [16, 17]. Since *GraphiDe* is mainly developed based on existing DRAM structure and operation with slight modifications, different methods currently-used to tackle process variation can be also applied for *GraphiDe* (e.g., spare rows). Besides, just like Ambit, *GraphiDe* chips that fail testing due to DRA, TRA, and QRA methods can potentially be considered as regular DRAM chips alleviating DRAM yield.

Table 2: Process variation analysis.

Variation	±0%	±5%	±10%	±20%
DRA	0.00%	0.00%	0.12%	11.43%
QRA	0.00%	0.08%	0.39%	18.92%

Regarding the error correction, many ECC-enabled DIMMs rely on calculating some hamming code at the memory controller and use it to correct any soft errors. Unfortunately, such a feature is not available for *GraphiDe* as the data being processed are not visible to the memory controller. Note that this issue is common across all PIM designs. To overcome this issue, *GraphiDe* can potentially augment each row with additional ECC bits that can be calculated and verified at the memory module level or bank level. Augmenting *GraphiDe* with reliability guarantees is left as future work.

3.5 Virtual memory

GraphiDe has its own ISA with operations that can potentially use virtual addresses. To use virtual addresses, *GraphiDe*'s Ctrl must have the ability to translate virtual addresses to physical addresses. While in theory this looks as simple as passing the address of the page table root to *GraphiDe* and giving *GraphiDe*'s Ctrl the ability to walk the page table, it is way more complicated in real-world designs. The main challenge here is that the page table can be scattered across different DIMMs and channels, while *GraphiDe* operates within a memory module. Furthermore, page table coherence issues can arise. The other way to implement translation capabilities for *GraphiDe* is through memory controller pre-processing of instructions being written to *GraphiDe* instruction registers. For instance, if the programmer writes instruction AAP *add*0, *add*1, then the memory controller intercepts the virtual addresses and translate them into physical addresses. Note that most systems have near memory controller translation capabilities, mainly to manage IOMMU and DMA accesses from I/O devices. One issue that can arise is that some operations are appropriate only if the resulting physical addresses are within specific plane, e.g., within the same bank. Accordingly, the compiler and the OS should work together to ensure that the operands of commands will result physical addresses that are suitable to the operation type. To avoid the complexity of virtual memory when using *GraphiDe*, system architects can opt for designating a continuous physical range that can be used by *GraphiDe* and the user/application can use physical addresses for operands. Directly operating on physical addresses can limit multi-tasking on *GraphiDe*, however, we leave supporting multi-tasking in *GraphiDe* through virtual memory support as future work.

4 APPLICATION: GRAPH PROCESSING

The *GraphiDe*'s parallel operations can be easily utilized to accelerate a wide variety of graph processing tasks. For the sake of limited space, we briefly explain two widely-used tasks so-called *matching index* and *degree centrality*.

Matching index: The matching index $M_{i,j}$ quantifies the *similarity* between two vertices (V_i and V_j) based on the number of common neighbors shared by vertices as $(\frac{\sum \text{common neighbors}}{\sum \text{total number of neighbors}})$. The main task here is to find the common and total number of neighbors which can be implemented and accelerated by *GraphiDe*. Fig. 4 provides a straightforward example to elucidate the mapping and acceleration method of *GraphiDe*. Initially, the sample four-vertex network is converted to adjacency matrix and stored in 4 consecutive rows of sub-array. To find the common neighbors of two particular vertices (e.g. V1, V2), *GraphiDe* performs parallel AND2 on the rows and SA's outputs determine the matches (here, V4). In addition, the total number of neighbors is found by performing OR2 operation on the same rows. Then, *GraphiDe*'s add operation can readily process the summation operation as explained earlier. Afterwards, the only remaining operation is to divide these numbers that can be done utilizing a off-chip processing unit to generate corresponding index matrix.

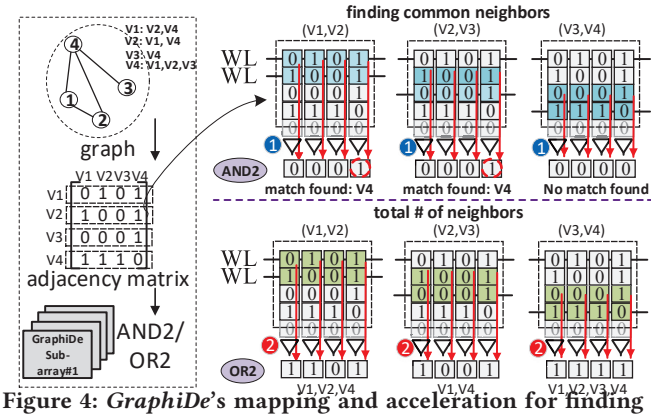

Figure 4: *GraphiDe*'s mapping and acceleration for finding matching index.

Degree centrality: One of the most important graph processing tasks is degree centrality. This task deals with massive number of

add operation which basically counts the number of valid links connected to a vertex. Fig. 5 shows an intuitive example of hardware mapping and acceleration of such operation performed by *GraphiDe* for a small graph. Initially, the designated graph is converted to adjacency matrix and mapped to consecutive rows of *GraphiDe*'s sub-arrays. Now, in the first step, every three rows are activated through *WL*s sequentially (here, (1) and (2)) to perform parallel add operation based on command sequence tabulated in Table 1 and generate initial Carry (C) and Sum (S) bits. In the second step, the results are written back to the memory reserved space. Then, next steps ((3) and (4)) only deal with multi-bit addition of resultant data starting bit-by-bit from the LSBs of the two words continuing towards MSBs. There are 2 steps for every bit-position computation. In the first step of (3), 2 *WL*s (accessing to LSBs of 6 elements) and one *WL* (accessing the reserved row initialized by zero) are enabled to generate the sum and carry. The SAs use these 3 words to generate sum and carry. During second step, two *WL*s are activated to save back the sum and carry bits. This process continues to MSB. At the end, the degree of each vertex is stored in memory (e.g. 4 determines the degree of vertex 1 in Fig. 5).

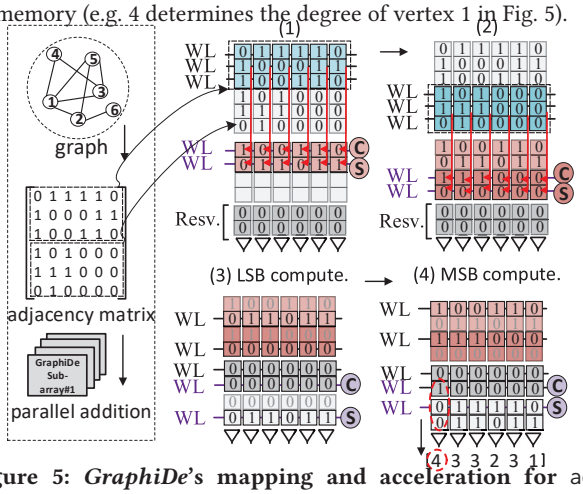

Figure 5: *GraphiDe*'s mapping and acceleration for add-based graph processing operations. Here we take degree centrality computation as an example.

5 EXPERIMENTAL RESULTS

We compare *GraphiDe* with other possible graph processing acceleration solutions based on DRAM, HMC, and GPU. We configure the *GraphiDe*'s memory sub-array with 512 rows and 256 columns, 4×4 mats (with 1/1 as row/column activation) per bank organized in H-tree routing manner, 16×16 banks (with 1/1 as row/column activation) and 512Mb total capacity. As mentioned earlier, enlarging the chip area provides a higher performance for *GraphiDe* and other PIM designs due to the increased number of computational sub-arrays, though the die size directly impacts the chip cost. Therefore, an identical physical memory size (512Mb) is considered for all implementations henceforth. To evaluate the performance of accelerators, we take three social network data-sets as tabulated in Table 3. Then, we map and run three graph processing tasks i.e. degree centrality, matching index, and Breadth First Search (BFS) on them that seek most of *GraphiDe*'s operations.

Table 3: Social Network data-sets.

Dataset	Nodes	Edges	Graph Information
ego-Facebook	4,039	88,234	profiles & friends lists from Facebook [10]
dblp-2010	326,186	1,615,400	scientific collaboration network
amazon-2008	735,323	5,158,388	similarity among books reported by Amazon store

5.1 Accelerators' setup

GraphiDe: To evaluate the performance of *GraphiDe* as a new PIM platform, a comprehensive circuit-architecture assessment framework and two in-house simulators are developed. 1- At the circuit level, we developed *GraphiDe*'s sub-array with new peripheral circuity (SA, MRD, etc.) in SPICE 45nm PTM library [21] to verify the proposed design methods and achieve the performance parameters. 2- An architectural-level simulator is built on top of Cacti [20]. The circuit level results were then fed into our simulator. It can change the configuration files corresponding to different array organization and report performance metrics for AAP-based PIM operations. The memory controller circuits are designed and synthesized by Design Compiler [19] with a 45nm industry library. 3- A behavioral-level simulator is developed in Matlab to calculate the latency and energy that *GraphiDe* spends on different graph processing tasks. In addition, it has a mapping optimization framework to maximize the performance according to the available resources. Real world graph consists of millions of vertices and edges that need to be processed. To efficiently map such graphs into *GraphiDe* architecture, graph partitioning methods are used. Here, we adopt interval-block partitioning method to balance workloads of each *GraphiDe*'s chip and maximize parallelism. We use hash-based method [6] to split the vertices into M intervals and then divide edges into M^2 blocks. ***DRAM:*** We developed an Ambit-like [16] accelerator for graph processing. Ambit implements logic function using capacitor-based majority functions. We accordingly modified CACTI [20] for evaluation of DRAM's solution. The controllers were synthesized in Design Compiler [19]. ***Baseline HMC:*** We used a conventional architecture presented in [14] using HMC as main memory without instruction offloading functionality. Due to the lack of space, we refer the readership to above-mentioned papers for the detailed configuration of each accelerator. ***GPU:*** We used the NVIDIA GTX 1080Ti Pascal GPU. The energy was measured with NVIDIA's system management interface. We scaled the achieved results by 50% to exclude the energy consumed by cooling, etc.

5.2 Energy and Delay

Figure 6 shows normalized energy consumption of the four accelerators on various graph processing tasks. *GraphiDe* achieves the highest energy-efficiency in different tasks compared to others owning to its low-energy and reduced-cycle operations. We observe that *GraphiDe* consumes on average 3.1× less energy than that of Ambit accelerator. The main reason here is the energy-efficiency of basic operations in *GraphiDe*; as discussed earlier, *GraphiDe* can finish the operations (such as addition) in less number of cycles using DRA and QRA methods. Fig. 6 shows that *GraphiDe* solution saves on average 3.9× and 58.6× energy compared to that of HMC and GPU solutions, respectively. It is worth pointing out that HMC and GPU designs are not capable of implementing fast bulk addition and therefore impose excessive energy consumption to memory chip in addition-intensive tasks such as degree centrality analysis. To realize such operation in Ambit platform, we considered multi-cycle majority-based implementation presented in [5].

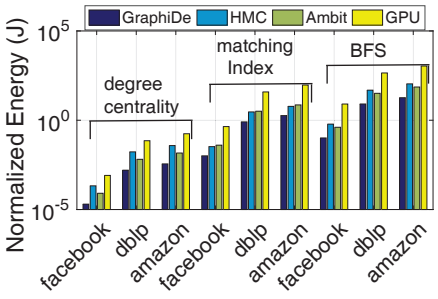

Figure 6: Normalized log-scaled energy of accelerators.

However, HMC solution shows a relatively equal energy-efficiency in matching index task compared to Ambit.

Figure 7 plots execution time of the *GraphiDe* and other accelerators on different graph processing tasks. We observe that *GraphiDe* solution is on average 4.2× faster than that of Ambit solution and 5.6× faster than HMC. This is mainly because of fast and parallel in-memory operations of *GraphiDe*, specifically for implementing add operation. Additionally, we see that *GraphiDe* is 83.4× faster that GPU solution. As can be seen, this directly translates to large performance improvements for the discussed applications that seek bulk bitwise operations.

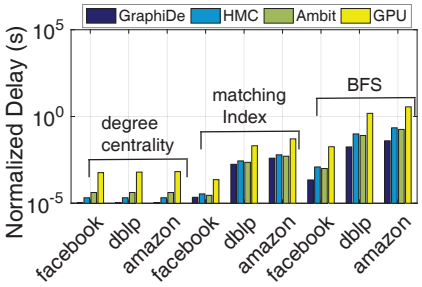

Figure 7: Normalized log-scaled delay of accelerators.

5.3 Memory bottleneck

Figure 8a reports the Memory Bottleneck Ratio (MBR), i.e. the time fraction at which the computation has to wait for data and on-/off-chip data transfer obstructs its performance (memory wall happens) running matching index task on two data-sets. The evaluation is performed according to the peak throughput for each platform considering number of memory access. The results show the *GraphiDe*'s efficiency for solving memory wall issue. We observe that *GraphiDe* along with other PIM solutions spend less than ~20% time for memory access and data transfer. However, GPU accelerator spends more than 90% time waiting for the loading data. The less memory wall ratio can be interpreted as the higher Resource Utilization Ratio (RUR) for the accelerators which is plotted in Fig. 8b. We observe that *GraphiDe* can efficiently utilize up to 70% of its computation resources. Overall, PIM solutions demonstrates a high ratio, which reconfirms the results reported in Fig. 8a.

5.4 Area overhead

GraphiDe is developed on top of Ambit [16] (with the area overhead of <1%). We have modified the controller and MRD circuits as well as SAs by adding two reference branches per column. Such enhanced SAs and peripheral circuitry in *GraphiDe*'s sub-array occupy less than 15% of area. Therefore, the overall area overhead of *GraphiDe* is ~1.3% over the commodity DRAM.

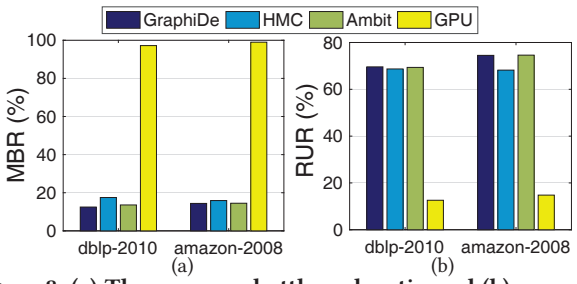

Figure 8: (a) The memory bottleneck ratio and (b) resource utilization ratio.

6 CONCLUSION

In this paper, we presented *GraphiDe*, which transforms current DRAM sub-arrays to massively parallel computational units to reduce energy consumption dealing with graph processing tasks and eliminate unnecessary off-chip accesses. The simulation results on three social network data-sets show *GraphiDe* can roughly achieve 3.1× energy-efficiency improvement and 4.2× speed-up over the recent processing-in-DRAM platform. It achieves ~59× higher energy-efficiency and 83× speed-up over GPU-based acceleration methods.

ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation under Grant No. 1740126 and Semiconductor Research Corporation nCORE.

REFERENCES

[1] 2018. Parallel Thread Execution ISA Version 6.1. (2018). http://docs.nvidia.com/cuda/parallel-thread-execution/index.html
[2] Shaizeen Aga et al. 2017. Compute Caches. In *HPCA*. IEEE, 481–492.
[3] Junwhan Ahn et al. 2016. A scalable processing-in-memory accelerator for parallel graph processing. *Computer Architecture News* 43 (2016).
[4] Shaahin Angizi et al. 2018. CMP-PIM: an energy-efficient comparator-based processing-in-memory neural network accelerator. In *55th DAC*. ACM, 105.
[5] Shaahin Angizi, Zhezhi He, et al. 2017. Design and evaluation of a spintronic in-memory processing platform for non-volatile data encryption. *IEEE TCAD* (2017).
[6] Guohao Dai et al. 2018. GraphH: A Processing-in-Memory Architecture for Large-scale Graph Processing. *IEEE TCAD* (2018).
[7] Quan Deng et al. 2018. DrAcc: a DRAM based accelerator for accurate CNN inference. In *55th DAC*. ACM, 168.
[8] 2010. DRAM Power Model. https://www.rambus.com/energy/. [n. d.]. ([n. d.]).
[9] Charles Eckert et al. 2018. Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks. *arXiv preprint arXiv:1805.03718* (2018).
[10] Jure Leskovec and Julian J Mcauley. 2012. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*. 539–547.
[11] Shuangchen Li et al. 2017. DRISA: A DRAM-based Reconfigurable In-Situ Accelerator. In *Micro*. ACM, 288–301.
[12] Shuangchen Li, Cong Xu, et al. 2016. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *DAC*. IEEE.
[13] Shih-Lien Lu et al. 2015. Improving DRAM latency with dynamic asymmetric subarray. In *MICRO*. IEEE, 255–266.
[14] Lifeng Nai et al. 2017. Graphpim: Enabling instruction-level pim offloading in graph computing frameworks. In *HPCA*. IEEE, 457–468.
[15] Vivek Seshadri et al. 2015. Fast bulk bitwise AND and OR in DRAM. *IEEE Computer Architecture Letters* 14 (2015).
[16] Vivek Seshadri et al. 2017. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *Micro*. ACM, 273–287.
[17] Vivek Seshadri, Yoongu Kim, et al. 2013. RowClone: fast and energy-efficient in-DRAM bulk data copy and initialization. In *46th Micro*. ACM, 185–197.
[18] Linghao Song et al. 2018. GraphR: Accelerating graph processing using ReRAM. In *HPCA*. IEEE, 531–543.
[19] Synopsys Design Compiler. Product Version 14.9.2014. Synopsys, Inc. [n. d.]. ([n. d.]).
[20] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P Jouppi. 2008. *CACTI 5.1*. Technical Report. Technical Report HPL-2008-20, HP Labs.
[21] Wei Zhao and Yu Cao. 2006. New generation of predictive technology model for sub-45nm design exploration. In *ISQED*. IEEE Computer Society, 585–590.