This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3003982, IEEE Access

ReGra: Accelerating Graph Traversal Applications Using ReRAM with Lower Communication Cost

HAOQIANG LIU[1], QIANG-SHENG HUA[1], (MEMBER, IEEE), HAI JIN[1], (FELLOW, IEEE), AND LONG ZHENG.[1], (Member, IEEE)

[1]National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China

Corresponding author: Qiang-Sheng Hua (e-mail: qshua@ hust.edu.cn).

ABSTRACT There is a growing gap between data explosion speed and the improvement of graph processing systems on conventional architectures. The main reason lies in the large overhead of random access and data movement, as well as the unbalanced and unordered communication cost. The emerging metal-oxide resistive random access memory (ReRAM) has great potential to solve these in the context of processing-in-memory (PIM) technology. However, the unbalanced and irregular communication under different graph organizations is not well addressed. In this paper, we present a PIM graph traversal accelerator using ReRAM with a lower communication cost named ReGra. ReGra optimizes the graph organization and communication efficiency in graph traversal. Benefiting from high density and efficient access of ReRAM, graphs are organized compactly and partitioned into processing cubes by the proposed Interval-Block Hash Balance (IBHB) method to balance graph distribution. Moreover, remote cube updates in graph traversal are converged into batched messages and transferred in a concentrated period via the custom circular round communication phase. This eliminates irregular and unpredictable inter-cube communication and overlaps partial computation and communication. Comparative experiments with previous work like Tesseract and RPBFS show that ReGra achieves better performance and yields a speedup of up to 2.2×. Besides the communication cost is reduced by up to 76%. It also achieves an average reduction in energy consumption of 70%.

INDEX TERMS processing-in-memory, resistive memory, ReRAM, architecture, communication

I. INTRODUCTION

Graphs are widely used to characterize and analyze the real-world relationship in a wide variety of applications that become data-intensive, such as social networks analysis, natural language processing, machine learning, recommendation, etc. However, the existing approaches are inadequate for explosive data growth, regardless of algorithms or hardware architectures. The CPU-based or GPU-based systems, like Frog [1], GraphChi [2], and SIMD-X [3], work in graph processing but there still exists a growing gap between data explosion and improvement of algorithms and architectures. Therefore more attention is being paid on large-scale graph processing, such as graph traversal.

The main bottleneck in graph processing is derived from the "memory wall" [4]. Specifically, a large number of random access to graph data leads to high dynamic random access memory (DRAM) cost. And the large data movements lead to severe pressure of bandwidth as well as energy cost. The processing-in-memory (PIM) technology is a potential and efficient way to reduce the latency and cost of data access and movement, as well as alleviate the bandwidth bottleneck.

PIM is a technology that puts logic circuits into or near memories, so the memory bandwidth can scale up well with the memory capacity increase ("memory-capacity-proportional") [5]–[7]. Besides the resistive random access memory (ReRAM) is an emerging non-volatile memory with appealing features for efficient memory access and even computation. With proper data organization and auxiliary components [8]–[10], ReRAM crossbars can efficiently process graphs and solve the issue of limited bandwidth and data movement.

Another problem that comes with PIM architecture is communication among processing cubes. In PIM architecture, multiple cubes are usually used for graph storage and calculation. On the one hand, unbalanced graph distribution may increase the time overhead in the critical path and coarse-grained data partitioning can cause this situation [11]. On the other hand, because of random data access, the inter-cube communication is unpredictable and irregular, which means that the time of communication, the amount and size of messages, and the destinations are not sure until the communication occurs. GraphH [6] and GraphQ [7] propose some solutions, but not for compactly stored data organization and the graph data partition is not well-suited in ReRAM-based architectures.

In this work, a ReRAM-based PIM architecture for graph traversal called ReGra is proposed. This accelerator is composed of ReRAM-based cubes with computation and communication components. Each cube processes the partial graph stored within it. The characteristics of graph traversal algorithms and features of the ReRAM crossbar such as high density and efficient access are taken into consideration, thus a compactly organized and efficiently accessed data representation is adopted. Besides, the Interval-Block Hash Balance (IBHB) method is proposed to balance graph distribution, which brings about more balanced inter-cube communication. Moreover, remote cube updates in graph traversal are transferred by messages according to destination vertices. Most importantly, the updated values to the same cube are converged into batched messages and transferred in a concentrated period and in an organized way through a custom circular round communication phase. And this eliminates the irregular and unpredictable inter-cube communication and thus reduces communication and cost.

The detailed contributions of this paper are concluded as follows:

- A ReRAM-based PIM accelerator for graph traversal is proposed. Graphs are compactly organized according to special features of the ReRAM crossbar with better access efficiency.
- The Interval-Block Hash Balance (IBHB) algorithm is proposed to balance graph distribution and inter-cube communication.
- The custom circular round communication mechanism eliminates irregular and unpredictable inter-cube communication and overlaps partial communication with computation. It reduces communication and cost.
- The well-designed cache alleviates the efficiency of memory access and efficient communication promotes the reduction of energy consumption. Therefore ReGra is energy-efficient.

The paper is organized as follows. Section II gives the background of PIM-based graph processing like graph traversal and basis of ReRAM. In Section III the architecture, the graph partition method, and the communication mechanism of ReGra are described. And Section IV presents a concrete process of graph traversal and implementation. Section V evaluates the architecture by different comparative experiments. And some related work is listed in Section VI. The last Section VII concludes this paper.

II. BACKGROUND AND MOTIVATION

A. GRAPH PRESENTATION AND PARTITION

Graphs are usually presented in two formats: adjacency list and adjacency matrix. The latter can be calculated using approaches of matrices, with more space consumption, especially in real-world sparse graphs. On the contrary, the adjacency list saves the space with some loss of computing convenience. This is acceptable in scenes of graph traversal (e.g. breadth-first search, BFS). All the neighbors of a vertex are stored compactly and continuously, and then easily accessed. RPBFS [8] presents a compressed sparse row (CSR) format to achieve a high compression ratio. This paper makes some changes with consideration of data partition. An example graph and its CSR format presentation are shown in Figure 1. Green dotted arrows simplify the coordinates which indicate the starting position.

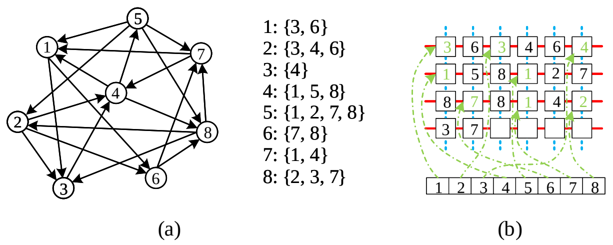


FIGURE 1. CSR format example. (a) The example graph and its adjacency list; (b) CSR representation in ReRAM crossbars.

As for graph partition, there are two typical approaches: edge-cut and vertex-cut [11], [12]. The vertex-cut approach focuses on edges and evenly distributes edges to processing units and spans vertices replicas. The edge-cut approach is adopted to partition vertices evenly instead of edges. However, the tools [13], [14] perform poorly due to complex algorithms and they are too balanced to sacrifice computing convenience. GraphH [6] and NXgraph [15] proposed two simplified approaches with destination-sort partition, but the performance is limited.

B. PIM-BASED GRAPH PROCESSING

Processing-in-memory technology integrates computing logic circuits into memory banks to achieve memory-capacity-proportional bandwidth and reduce data movements. With the development of 3D stacking technology, PIM technique is gaining more attention in graph processing. Usually, real-world graphs have millions of vertices and a much larger number of edges, therefore it is challenging to analyze and process with high performance.

Considering that it is more suitable for thinking and programming, graph algorithms are normally described in vertex-centric programs. The edges of a vertex are usually stored together. And it features "Think Like a Vertex(TLAV)" [16] as the way human thinks of the graph problem. That is to say, we access edges through the corresponding vertex and then process the value. Taking graph traversal (e.g. BFS) as an example, when a vertex is processed, all its neighbors are accessed to generate the frontiers for next level traversal, and then the neighbors in frontier will be accessed and processed sequentially.

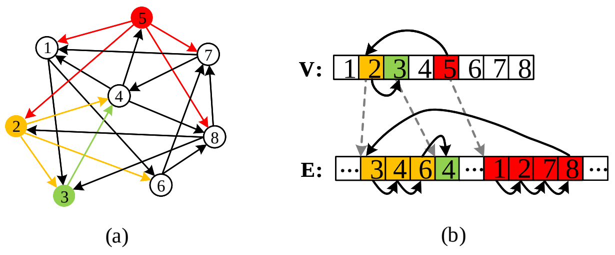


FIGURE 2. Vertex-centric access pattern. (a) Example graph and three active vertex; (b) access process of partial graph.

Figure 2(b) shows an example of vertex-centric grapah access pattern. In the vertex-centric algorithms, edges of a vertex are accessed sequentially but globally edges are accessed randomly because of random neighbors. Graphs are usually partitioned into several cubes in PIM-based architectures. In Tesseract [5] each cube processes the vertices stored in the cube and cooperates with other cubes via message passing based communication to update the value of all vertices. RPBFS [8] modifies vertex status by accessing shared memory, and both memory conflicts and underutilized bandwidth exist in it.

Inter-cube communication needs to be taken seriously. Figure 3 depicts the inter-cube communication in BFS from the perspective of the adjacency matrix, in which a dot represents an edge. All the vertices are divided into three cubes and the overlapping area of the green and yellow pane is an edge that connects cube 0 and cube 1 ($v_i \rightarrow v_j$). Inter-cube communication happens between cube 0 and 1 and messages are sent from cube 0 and values are updated in cube 1 (yellow and red pane). Messages can be sent at any uncertain time and uncertain number. For the receiver, if facing the challenge of irregular messages, the computation within the cube may be interrupted. ReGra is inspired by previous work and reasonable changes are made.

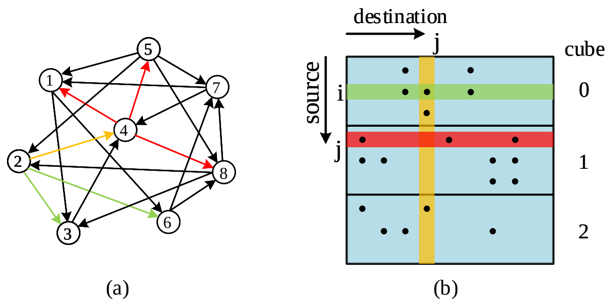


FIGURE 3. Schematic diagram of inter-cube communication. (a) Example graph; (b) inter-cube communication in BFS.

C. RERAM-BASED GRAPH PROCESSING

The metal-oxide resistive random access memory(ReRAM) is an emerging non-volatile memory. It consists of a sandwich-like structure, which includes the top and bottom metal electrodes and a metal-oxide switch between two electrodes [17]. The metal-insulator-metal structure of a ReRAM cell is demonstrated in Figure 4(a). By applying an appropriate external voltage across the ReRAM cell, it can switch between a high resistance state (HRS) and a low resistance state (LRS), and this can be used to represent binary “0” and “1”. The I-U curve in Figure 4(b) shows more information about this property.

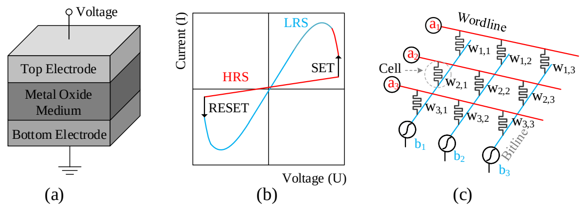


FIGURE 4. ReRAM basics. (a) structure of ReRAM cell; (b) I-U curve of ReRAM cell; (c) conceptual view of ReRAM crossbar.

ReRAM cells can be organized by a dense and area-efficient array named crossbar structure [18], as shown in Figure 4(c). When values are programmed to cells and reasonable input voltages are applied to wordlines, a sum of dot products can be got from the bitlines. There is a lot of work of matrix-vector multiplication utilizing this feature of ReRAM crossbars([9], [10], [18]). Graphs are translated to a matrix in multiplication operations with the loss of space efficiency. Another way is for efficient storage if the input voltage of a certain wordline is set to “1” (this means a read voltage), then the whole values stored in a line of ReRAM cells can be read in bitlines. It is much convenient to fetch continuous data in memory, such as all neighbors of a vertex. RPBFS [8] utilizes these features and stores the adjacency list in a compact and continuous way, the performance is improved by bank-level parallelism. However, there remains data conflict and scalability issues because of competitive accesses to shared memory. Inter-cube communication is a better way in large scale graph processing.

III. LOW-COMMUNICATION REGRA OVERVIEW

A. ARCHITECTURE OVERVIEW

The size of the crossbar is usually hard to grow with the size increase of graph in large scale graph processing. So several memory banks make up cubes and cubes interconnect with each other to accommodate the entire graph. Figure 5 depicts ReGra architecture. All of the ReRAM crossbar cubes are interconnected in a memory-centric network [19]. The topology among memory cubes is dragonfly offering higher bandwidth and throughput. In addition, a series of periphery circuits and components are applied to ReRAM crossbars, to efficiently access graph values.

Here introduce some basic components attached to ReRAM crossbars, which are ordinary components for driving ReRAM crossbars to work properly. **ReRAM crossbar** stores the graph vertices and edges. It mainly includes two aspects: One is the compactly and continuously organized

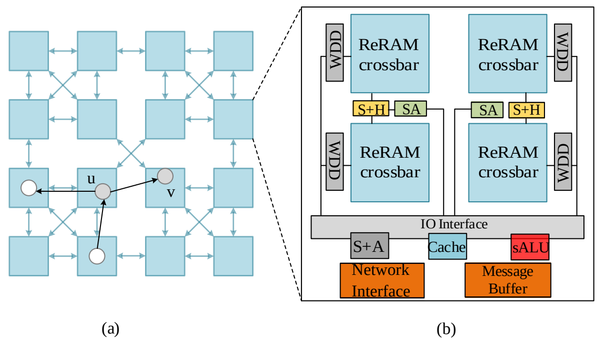


FIGURE 5. ReGra architecture. (a) The schematic overview of ReGra topology; (b) details inside a ReRAM cube.

adjacency list of vertices that are partitioned into this cube; another is the starting location coordinates of each adjacency list. **Word Decoder and Driver (WDD)** is the wordline decoder and writing driver and it is used to access the graph data stored in the ReRAM crossbars. **Sample and Hold (S+H)**, as the name suggests, it samples the analog currents and holds it until the current is converted into digital values. Another component is **Sense Amplifier(SA)**. SA seems like the analog-to-digital converter, which can provide high precision. And we share SA between ReRAM banks. Besides there are still several components playing unusual roles, and a detailed description of the components is followed:

- **Shift and Add (S+A).** S+A unit is applied to get a higher computing resolution(e.g. we can unite several ReRAM cell units to get a 32-bit number from the ReRAM crossbars).
- **simple ALU (sALU).** The sALU performs some more complicated arithmetic and logic operations. It is a 32 bit ARM Cortex-A5 processor with a FPU running at 1 GHz, the same as Tesseract [5].
- **Message Buffer.** It is used for the message passing based communication. And it consists of a receive buffer and a send buffer. And the network interface supports communication.

B. INTERVAL-BLOCK HASH BALANCE GRAPH PARTITION

The number of vertices and edges in a cube affects the execution time within it, which in turn affects the running time of the entire program. On the one hand, it takes time to process the values updates and communication messages. On the other hand, the imbalanced graph partition may lead to longer critical path execution time. Considering that graphs are stored in CSR format and that neighbors may be continuous in BFS [8], we propose the interval-block hash balance partition algorithm to distribute graph blocks evenly into cubes.

Partition is based on vertex id. Before that, vertex indices are mapped to continuous indices to achieve convenient computation and efficient access. And this change has nothing to do with the graph property. This optimization operation is often called degreeing [15]. In ReGra, the adjacency list of each vertex is organized continuously and a read operation to the ReRAM crossbar may fetch several lists. Such operations can ensure a feasible spatial locality. Given this, it is not a good idea to divide the graph by destination vertex id and partition the graph vertex-by-vertex. The graph will be divided into blocks with intervals consisting of consecutive IDs, and then allocated to each cube. This is different from some of the previous approaches [19]. To obtain balanced-size partition blocks, we first generate much more blocks than the number of cubes, and then the blocks are modularized and merged to a certain cube. The IBHB process is depicted in Figure 6. Edges move with vertices, so they are omitted in the figure.

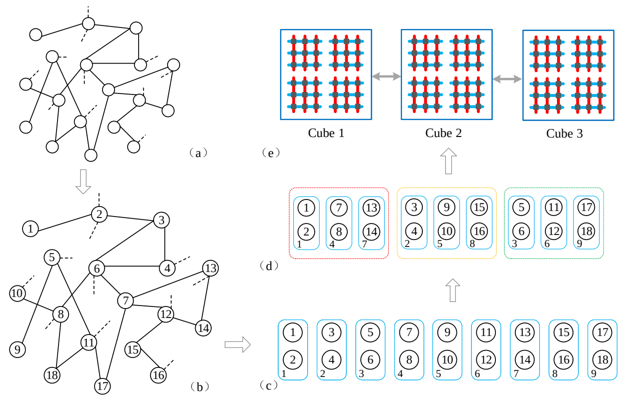


FIGURE 6. Illustration of IBHB process. (a) Example graph and cubes; (b) degreeing; (c) interval block partition; (d) hash and merge blocks to cubes.

As is shown in the example process in Figure 6, there are three cubes involved in the calculation. It is a very straightforward method to partition all the vertices evenly into three cubes. Actually in IBHB, vertices id are divided into nine blocks at the interval length of 2. The block size is minimized and more blocks are generated. Then the blocks are numbered and then hashed to the three cubes. We take a simpler mapping scheme for partition efficiency. Moreover, this simple way helps each cube manage and access the blocks.

Although a smaller block size can reduce the size gap among blocks and cubes, it is worth discussing that the more graph is divided into blocks, the more edge-cut will appear. The latter will increase the number of communications. However, a larger granularity of partition will cause imbalance and increase communication time. So we need to make a compromise when dividing the blocks. And it is described in the experiment section.

C. BATCHED COMMUNICATION VIA MESSAGE PASSING

In ReGra each ReRAM cube is limited to access and update its local memory only. Therefore a proper and efficient communication mechanism is necessary for cubes to communicate with each other, transferring vertices and edges values. A message-passing based communication mechanism is proposed in ReGra. Figure 5(a) shows an example of this process. It is convenient for vertex u to send a message to vertex v or request an update for its property. The message containing concrete computing operation is processed by the remote ReRAM cube.

Message passing based communication is a fundamental way for cubes to communicate with each other. In ReGra, this is encapsulated in different remote procedure calls (RPC) so that the communication is transparent to programmers. The two typical approaches are: blocking RPC and non-blocking RPC. This is obvious, but it also has different effects. In blocking procedure call, the caller sends a message to the callee and waits for the result returned by the callee. When the callee receives the call from any remote cube, it gets messages from receive buffer and processes immediately. A similar process happens in the non-blocking procedure call, except that callee needs not to process immediately, and there is no return value and the caller does not need to wait for the result, which increases communication and computation efficiency.

In graph traversal programs, most inter-cube communication involves transferring messages of updates to destination vertices, and then the receiver processes the messages and updates the vertices properties. That is to say, the sender is not responsible for updating the vertices properties in remote cubes. Therefore non-blocking is a better way, and that is exactly what we adopt in ReGra. This simplifies communication and makes it more efficient. In addition, we can make use of non-blocking communication to overlap the process of receive buffer and send buffer.

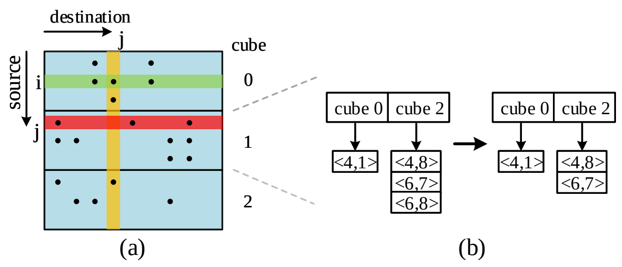


FIGURE 7. Batched message communication. (a) Inter-communication among three cubes; (b) the converged message queues.

Another pro of the non-blocking procedure call is that it makes the batch process more flexible. As shown in Figure 7(a), after the vertex i (green pane) is processed, vertex 4 and 6 in cube 1 will join the frontier in the next traversal level. Next, when processing remote updates, vertex 7 and 8 are going to receive update messages, and both of them are in cube 2, so all the messages in the queue will be gathered naturally (shown in Figure 7(b)). And the batched message will be passed to cube 2 through the subsequent circular round message passing mechanism. Specifically considering BFS, the valid status updates to destination vertex 8 will be activated by only one vertex, as we specify that vertex 4 is processed before vertex 6 in the example graph. So the queue of cube 2 can be further reduced as illustrated in Figure 7(b).

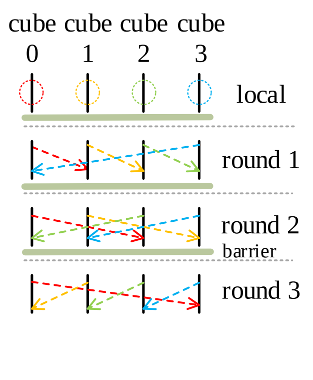
In the example, $N-1$ batched messages will be generated in each cube to be sent for other cubes, and the number that received from other cubes is also $N-1$, where N is the number of cubes. In ReGra, 16 cubes are commonly used, thereby $N=16$. In every level of traversal, each cube will generate messages to the other 15 cubes, by the circular round message transferring phase which is depicted in the following paragraph.

D. CIRCULAR ROUND COMMUNICATION

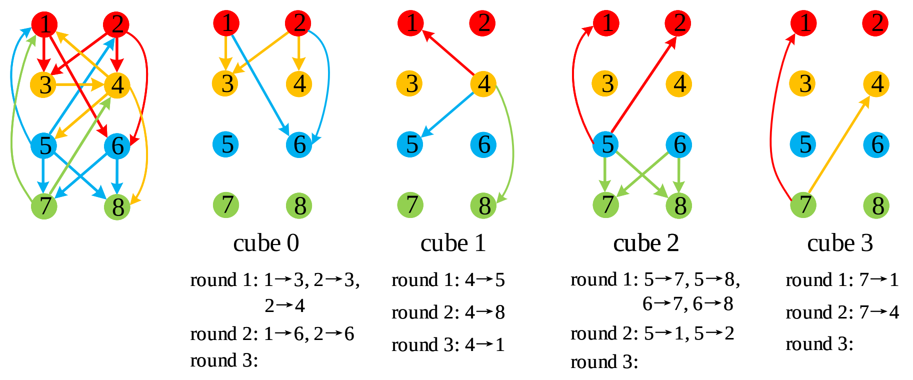
As mentioned above, inter-cube communication is a necessity in ReGra architecture. The batched messages reduce the number of communication, but the timing of the messages being transferred is irregular and unpredictable and the destination cubes may be interrupted. A custom circular round communication mechanism is proposed to solve the problem and eliminate the irregular and unpredictable communication. This is inspired by GraphH [6], but more optimization work is introduced. One is the different representations of graph organization, and another is the influence of new ReRAM hardware.

Circular round communication makes cubes transfer messages among each other in a determined order within a certain period. After multiple rounds of circular transferring, every cube will receive messages from all other $N-1$ cubes, where N means the number of cubes. Only one cube is the destination for each cube in each round, therefore $N-1$ rounds are needed in a circular round communication. Figure 8(a) depicts the process of circular round message passing. In the first round, messages are transferred from cube 0 to cube 1, and cube 1 to cube 2, ... , and so on. Each serial number of destination cube is the serial number of the current cube “plus 1". And in round 2, each cube performs “plus 2" operation and gets the serial number of destination cube in this round. Comprehensively, it can easily deduce the rest procedure from the “plus n" operation in the round n. As N cubes are adopted, all messages can be transferred to their destination in $N-1$ rounds. And it is worthwhile to mention that the result of “plus n" may exceed N, so a “mod n" operation to get the correct destination cube is also necessary. This is the “circular" means. An example of circular round communication is shown in Figure 8(b), where we suppose all the vertices are in the frontier and all edges are processed and updated. All the vertices are partitioned into four cubes with different colors. The edges to be remotely processed in each round are colored by the color of their destination vertices and detailed descriptions are under the figure.

It is easily achieved in destination-vertex based graph organization. The data for the graph is already sorted by destination vertex, which can be used directly in the communication phase. However the edges are continuously and compactly stored by source vertices, and it is difficult to get the right batched messages by destination vertices in each



(a)



(b)

FIGURE 8. Circular round communication. (a) Circular round communication mechanism; (b) a detail case.

round like the destination-vertex based block partition. So all the neighbors of a vertex are fetched in the round 0 and stored in the corresponding buffer list for each cube, and a similar destination-vertex based block partition is formed. And in the following each round, we get the message from the buffer and transfer it. Compared with graphH, it adds some buffer overheads. But the overall improvement is considerable.

E. EFFICIENT CACHE FOR RERAM OPERATIONS

The graph data is stored in ReRAM banks by continuously compressed sparse row, one vertex by another. Additionally starting location of each adjacency list is recorded in the same bank to help determine the scope of the neighbors (Figure 9). With the features of ReRAM crossbars read operation, it is very convenient to get all the neighbors of a vertex, and only several read operations to the whole wordline. But there are a lot of global random accesses in vertex-centric graph algorithms (Figure 2(b)). We not only need to use cache to reduce random memory access but also need to design better cache according to the characteristics of graph traversal.

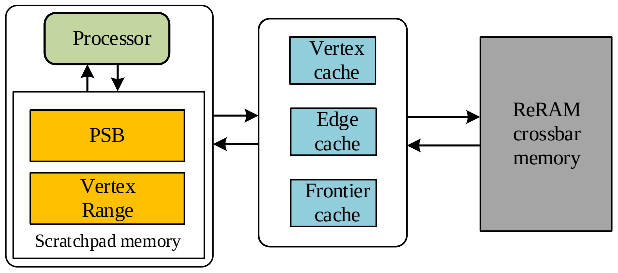


FIGURE 9. The cache in ReGra architecture.

The partial status bitmap of vertices stored in the cube is accessed and updated frequently. Besides, the vertex range of each cube is also frequently accessed to determine the destination cube, but it is constant. Both of them are stored in scratchpad memory. The latency is shorter than L1 cache and it is not interfered with by the cache replacement policy. For large graphs, the bitmap can not fit into scratchpad memory, and flush operation is conducted to flush the bitmap to memory. At the meanwhile, the operations to check and update the bitmap will be divided into small partitions, and it ensures the requested vertices are in the scratchpad memory.

As for the adjacency list, the cache is also necessary to buffer the neighbors of continuous vertices. During the accessing, the required data is first checked in the cache. If the target data is hit, whether the cache line contains the whole adjacent list is also a matter that must be considered. If the whole data is not found in the cache, at least one line of ReRAM crossbar is accessed and cached according to the data position or range. Limited by the size of the cache capacity, the cache replacement strategy used here is First In First Out (FIFO). Conceptual cache components are shown in Figure 9.

IV. GRAPH TRAVERSAL ON REGRA

BFS and Depth First Search (DFS) are two basic graph traversal algorithms. In this section, BFS is applied to ReGra. We take advantage of the convenience of this architecture to bring better performance to the algorithm. There are some minor changes to the conventional BFS algorithm but it is transparent to the interface user. Moreover, other traversal algorithms such as Single Source Shortest Paths (SSSP) and Minimum Spanning Tree (MST) are discussed.

A. GRAPH PREPARATION

All cubes are equal units for traversing the graph and updating the status of vertices in its partial state bitmap. The vertex range of partial status bitmap is limited within the vertex range stored in that cube. As for those outside neighbor vertices, the update message is transferred in the communication phase, which is transparent to programmers. And in this work, only static graphs whose vertices and edges are not changed are studied. A graph is statically mapped to each cube.

In Figure 1, we can see the CSR format representation of the graph and the neighbors are stored continuously in crossbars. We adopt the IBHB approach to evenly partition

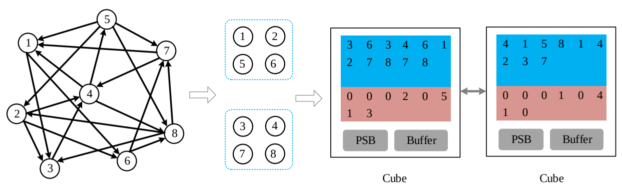


FIGURE 10. Graph is mapped into crossbars wiht CSR format and coordinates.

graph, which is a pre-processing. After the pre-processing, the vertices IDs are no more continuous in each cube. Fortunately, we adopt a simple module hash, and virtual continuous id can be obtained by one-step calculation, and then data access is performed. The formula is:

$$cid = id - l \cdot [(s-1) + i \cdot l]. \tag{1}$$

The cid means fake continuous id, and s means first vertex id in each block, and i is the serial number of a vertex in each block, as vertex is calculated one by one. and l is the block length. The results can be buffered for convenience. As depicted in Figure 10, the graph is partitioned into two cubes, each contains four vertices and their edges. When all the neighbors of a vertex are to be accessed, the index $[cid*2, cid*2+1]$ of starting location array (pink panes in Figure 10) stores the starting location of its adjacency list. The continuously subsequent starting location, which is stored in the index $[cid*2+3, cid*2+4]$, is also needed to determine the boundary of neighbors. When reading data from ReRAM crossbars, it usually reads a complete line and puts data into the cache, and then gets the required neighbors from the cache.

B. GRAPH TRAVERSAL

In the BFS algorithm, starting from a source vertex s, all other vertices in the graph are traversed and checked whether it is reachable from s. This is a hierarchical or level centric traversal, starting from the source vertex and spreading to farther vertices in turn, and finally completing the global traversal. A schematic diagram of this process is shown in Figure 11(a), where gray dotted lines indicate different traversal levels, and Figure 11(b) depicts a complete traversal process in the example graph.

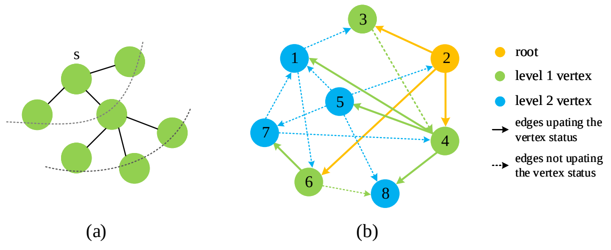

s
● root
● level 1 vertex
● level 2 vertex
edges upating the vertex status
edges not upating the vertex status
(a)
(b)

FIGURE 11. Example of level traversal of BFS.

The pseudocode process of BFS in ReGra is described in Algorithm 1. From the perspective of the algorithm process, we follow the conventional top-down way. The current frontier queue (CQ) and status bitmap of vertices in the range of a cube (PSB) are maintained, which are cores in BFS. What must be mentioned here is that the vertex bitmap is part of the whole vertices, therefore it is essential to use the displacement from the first vertex in a cube as an index for the bitmap. Other variables are introduced in section IV-A (Graph Preparation), and the description in the algorithm pseudocode is also very intuitive. Thus no more detailed explanation is given here.

Algorithm 1 BFS in ReRAM based cubes

```
Input: source vertex $v_s$
Preparation:
 1: The set of vertices in each cube $V_i$
 2: The set of edges in each cube $E_i$
 3: The current frontier queue $CQ$
 4: The frontier queue for next level $NQ$
 5: The queue for batched message $BMQ$
 6: All neighbors of a vertex $NBQ$
 7: Partial status bitmap $PSB$
 8: $CQ \leftarrow v_s$
 9: for $v_j \in V_i$ do
10:     $PSB[j] = 0$
11: end for
12: $PSB[s] = 1$
Graph Traversal:
 1: while $CQ \neq \phi$ do
 2:     $NQ \leftarrow \phi$
 3:     for $v_u \in CQ$ do
 4:         $NBQ = GetNeighbors(u, E_i)$
 5:         $BMQ = SplitRemoteVertex(NBQ, V_i)$
 6:         $//NBQ^{'}$ contains vertices in the range of cube
 7:         for $v_k \in NBQ^{'}$ do
 8:             if $PSB[k] = 0$ then
 9:                 $PSB[k] = 1$
10:                 $NQ \leftarrow NQ \cup v_k$
11:             end if
12:         end for
13:         $BatchExecuteRemote(BMQ, NQ, PSB)$
14:     end for
15:     $CQ \leftarrow NQ$
16: end while
```

In the traversal, several necessary variables are initialized to enhance the breadth-first traversal. On the one hand, the partial neighbors of a vertex, which are not in the range of the cube, are split from the NBQ, and these vertices are to be sent to remote cubes. The local cube is responsible for the remaining vertices, which are in $NBQ^{'}$. As for these vertices, their visited status is checked first. If the vertex is not accessed, its status is updated and it will also join the NQ for the next traversal level. On the other hand, the vertices in the

BMQ are preparing for batched messages to remote cubes. All edges are organized by source vertex-centric way, it is convenient to get the whole neighbors in ReRAM crossbars. But at the same time, it requires more operations to gather edges by their destination vertices. In BMQ, for each cube a list of active vertices is established. Each list maintains the split vertices that are in the same cube. In the circular round communication phase, one of the lists is transferred to the destination cube and then traversed to generate frontier for next level traversal.

There are three special functions in Algorithm 1. **GetNeighbors** and **SplitRemoteVertex** are simple, as the name suggests and discussed above. One more thing to note is that the cache and buffer in these functions. We introduced the cache adopted in ReGra, such as the scratchpad memory. They are transparent and can not be accessed directly in programming, what we operate in the function is a memory address, but for processor it is data in the cache. Next the focus shifts to the third function **BatchExecuteRemote**.

Algorithm 2 circular round communication

```
The local cube identifier $localId$
The remote cube identifier variable $remoteId$
Messages to be sent in $sendBuf$
Messages received from other cubes $recvBuf$
function BatchExecuteRemote($BMQ, NQ, PSB$)
    for $roundId = 1 \to cubeNums - 1$ do
        $remoteId = (localId + roundId)\%cubeNums$
        InitBatch($recvBuf, sendBuf, BMQ$)
        SendBatch($remoteId, sendBuf$)
        RecvBatch($recvBuf$)
        for $v_k \in recvBuf$ do
            if $PSB[k] = 0$ then
                $PSB[k] = 1$
                $NQ \leftarrow NQ \cup v_k$
            end if
        end for
        barrier()
    end for
    $CQ \leftarrow NQ$
end function
```

Algorithm 2 shows the process of **BatchExecuteRemote**. The core is custom circular round communication among cubes. This function constantly sends and receives messages. In the beginning of each round, the $recvBuf$ is cleaned and the $sendBuf$ is filled with vertices list in BMQ with the right destination cube (denoted by $remoteId$). The calculation formula of $remteId$ has been deduced in the previous paragraph. The $SendBatch$ is an asynchronous operation for sending batched messages. And the local cube is not blocked therefore the local cube can overlap communication and computation. What concerned is the received data. It must exist in $recvBuf$ before the subsequent calculation. There is a kind of synchronization in this procedure. After receiving messages, these vertices are checked and updated to generate the frontier for next level traversal. In conclusion, CQ contains the result of local computation and remote updates, which is the global status of each vertex in the level.

Graph data is stored in a continuous CSR format. This format is more useful to level-centric graph traversal algorithms like BFS, rather than depth recursive algorithms like DFS. In addition to BFS, other algorithms such as SSSP and MST can also be applied to ReGra architecture. Better performances are also obtained under the proposed format. As for unweighted graphs, SSSP traverses a graph in a level centric way. In this case, a similar process and data structure to BFS can be used. For weighted graphs, the weight of each edge is first stored in the compressed adjacent list, and more runtime data structures are needed. The distance array is critical in SSSP, and it records the current shortest distances between each vertex and the source vertex and it should be updated at each traversal level. In summary, ReGra can achieve better performance when processing level-centric graph traversal.

V. EVALUATION

A. METHODOLOGY

We simulate the ReGra architecture with zSim [20] and NVMain [21]. zSim is a scalable x86-64 multicore simulator, and we modify it with an interconnection model, memory, on-chip networks, and other hardware components. The computing unit in ReGra is single-issue in-order cores the same as Tesseract. And each core has 32 KB L1 cache for instructions and 64 KB L1 data cache. The simulation frequency of computing core is 1 GHz. The scratchpad memory in each core is set to 64 KB, and it is enough to accommodate the vertex range of each cube, the frontiers in each traversal level as well as the vertex status bitmap. As for cube configuration, we use 16 cubes connected with the Dragonfly topology (Fig.4.(a)).

As for memory, the detailed configuration of ReRAM needs to be explained. Each crossbar contains 1024×1024 ReRAM cells, and the parameters of ReRAM cell model is the same as RPBFS [8](derived from GraphR [10]). The HRS/LRS resistances are set to 2.5 $M\Omega$ and 50 $K\Omega$, read voltage and write voltage are 0.7V and 2V and thus the current of HRS/LRS are 2 μA and 40 μA respectively. Moreover, the energy cost of read and write are 1.59 pJ and 5.33nJ, and the read and write latencies are 29.31 ns and 50.88ns. And we also refer to [22] to evaluate the energy consumption of ReGra components.

Table 1 shows six different real-world datasets in our evaluation. These all come from Stanford Large Network Dataset Collection (SNAP) [23], including social networks from Slashdot, Twitter as well as other types of graph such as co-purchasing, web graph, and road network. The soc-Slashdot(SS) has the least vertex number but the maximum ratio of edge to vertex, briefly it is the densest graph in the six datasets. And the counterpart is wiki-Talk(WT), the most sparse one. The twitter-higgs(TT) becomes the largest graph. All the graphs are stored in the ReRAM crossbars in a CSR

TABLE 1. Graph datasets used in experiment

Datasets	Vertices	Edges	Type
soc-Slashdot (SS)	82K	0.95M	social network
amazon0312 (AZ)	400K	3.2M	product co-chasing network
twitter-higgs (TT)	456K	14.86M	social network
web-google (WG)	875K	5.11M	web graph
roadNet (RA)	1.97M	5.53M	road network
wiki-Talk (WT)	2.39M	5.02M	communication

format. During pre-processing, the adjacency list of each vertex is sorted and mapped to different cubes according to vertices ranges. These operations do not change the result of graph traversal algorithms, and it is not involved in the result comparison because of the small cost.

B. EVALUATION RESULTS

1) IBHB partition method

IBHB divides the graph into more small blocks than the cube numbers and then merges those blocks into the cubes through a simple hash mapping. After IBHB partition, the graph is evenly distributed into the cubes, which is more balanced than dividing the graph into blocks whose number is the same as that of cubes. IBHB is a relatively simple method and requires very little cost. We apply the algorithm to the six graphs with different block numbers and calculate the number of edges in the inter-cube communication. The results are shown in Figure 12. The abscissa indicates the number of divided blocks, and N indicates the number of cubes. It can be seen that dividing too many blocks results in more communication, especially the more partitions. When partitioned into 8N or 16N blocks, the number of communications increases even more. But the communication time has different performance. Figure 13 shows the inter-cube communication time of different blocks. According to the results, when the partition ratio is 4, the graph traversal performs best. Because the balance is more appropriate at this time. This ratio is also used in our experiments.

2) Execution time compared with RPBFS

RPBFS utilizes ReRAM crossbars to traverse graphs in parallel. On this basis, we carried out further research. The message passing mechanism is used to transfer edges to be checked and synchronize traversal level. Even further work is to alleviate the irregular and unpredictable message that could have been avoided among cubes, which is the circular round communication among cubes. Hence we compared the performance of ReGra with that of RPBFS. Besides, the performance is measured in execution time. In BFS it means traversal starts from giving a source vertex and ends till the frontier queue is empty. And in SSSP it means traversal starts from the source vertex and ends till all the shortest paths

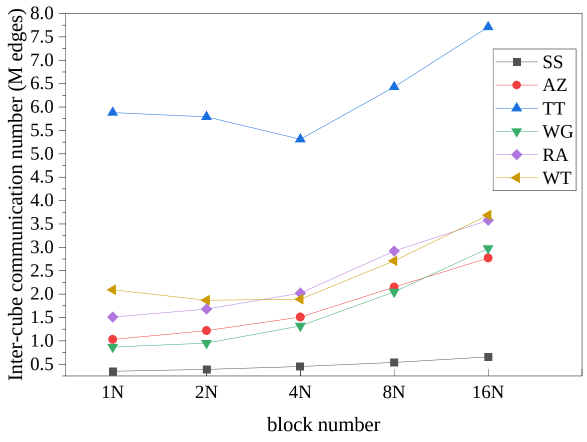


FIGURE 12. Inter-cube communication number of different block size.

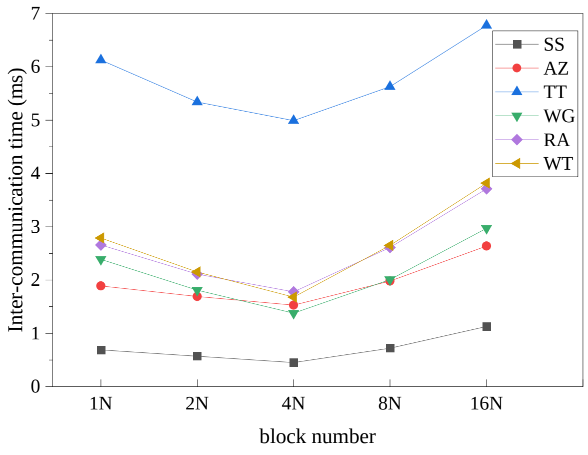


FIGURE 13. Inter-cube communication time of different block size.

are found. The results of the execution time speedup ratio comparison are shown in Figure 14.

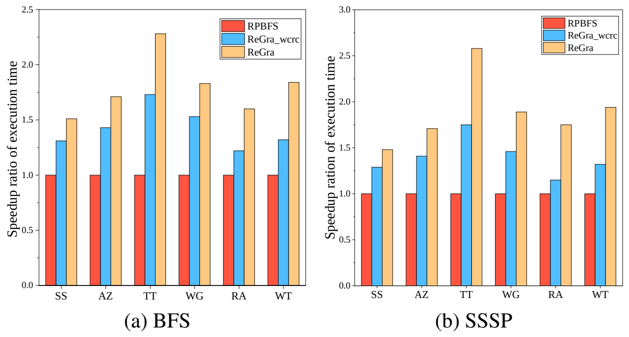
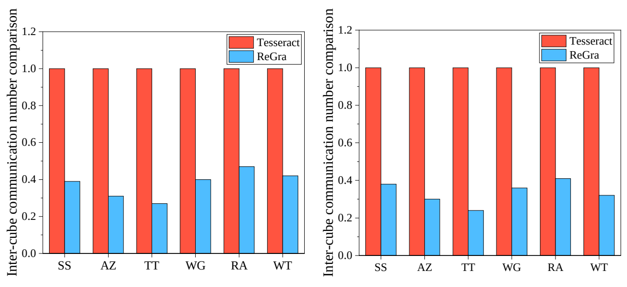


FIGURE 14. The speedup ratio of execution time comparison of three architectures. The result is normalized to RPBFS.

As shown in Figure 14, the execution time of RPBFS is used as a baseline to compare the execution time speedup ratio of ReGra. And the performance speedup of BFS and SSSP is clearly shown in Figure 14. It is worthwhile to mention that the column labeled with “ReGra_wcrc" means the mechanism of custom optimized circular round communication is not applied in the experiment. The communication in "ReGra_wcrc" is the same as Tesseract. ReGra performs up to 2.3× speedup in BFS and 2.6× speedup in SSSP than RPBFS. In the meantime, without circular round communication, the performance of ReGra is affected, especially in the large graph such as TT. It is visual in Figure 14 that for smaller graphs the performance difference between RPBFS and ReGra can be small and reasonable. That is because the small graph is distributed in several cubes and the execution parallelism of architecture is not high and communication is less, and the blocked data interferes more with the calculation of small scale graphs.

3) Communication compared with Tesseract

In RPBFS, communication is used to transfer commands and frontiers and synchronize the status between the master bank and graph banks. Shared memory is adopted to maintain status information for all vertices. ReGra improves the way the vertex status is maintained and the communication mechanism adopted in ReGra is inspired by Tesseract. In this section, we compare the communication efficiency between ReGra and Tesseract, evaluating the circular round communication for good communication improvement. The evaluation result is depicted in Figure 15.



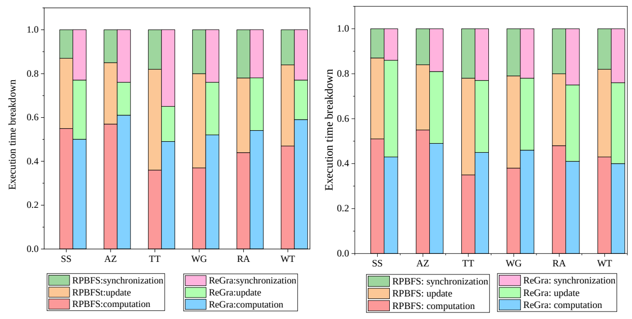
(a) BFS (b) SSSP

FIGURE 15. The communication efficiency of ReGra compared with Tesseract.

As illustrated in Figure 15, the data transferred among cubes in ReGra is far less than that of Tesseract. The total number of bytes of data cross cubes is normalized to that of Tesseract. ReGra reduces the inter-cube communication by around 60% on average for all six data sets, and up to 80% in SSSP. Firstly, the architecture of ReGra is simpler than that of Tesseract, and the statistical caliber is limited to inter-cube communication, which is more favorable for ReGra. More importantly, the batched message and circular round communication make inter-cube communication more regular and reduce the total number of communication. And the more balanced data distribution has also improved communication. A graph like AZ which has good locality performs well in communication efficiency because most updates happen in the local cube.

4) Execution time breakdown

The above is a comparative experiment, reflecting the improvement of ReGra. Figure 16 shows the execution time breakdown of BFS and SSSP in RPBFS and ReGra. We mainly divide the traversal execution time into three parts: computation, update, and synchronization. The three parts exist in both architectures, with the major difference being the update process. In RPBFS, all cubes access the shared memory and update the status of vertices. Otherwise, in ReGra, each cube maintains partial vertices status and communicates with other remote cubes to transfer updates. The consistency of the global status of vertices and the level of graph traversal are ensured in the synchronization phase.



(a) BFS (b) SSSP

FIGURE 16. The execution time breakdown of RPBFS and ReGra.

As shown in Figure 16, ReGra takes more time in update operation than RPBFS, benefiting from a partially overlap of computation and communication. On the contrary, the synchronization process seems to take longer than RPBFS. However, because the total execution of ReGra is less than that of RPBFS, it takes less time in synchronization indeed. Besides, a small-scale graph takes more time to update status in ReGra because of graph partition.

5) Energy and Area

Figure 17 illustrates the energy cost of Tesseract and ReGra, and the results are normalized to Tesseract. As we can see, ReGra saves around 60% energy on average than Tesseract in BFS and 70% on average in SSSP. The mechanism of batched message passing and circular round communication eliminates the irregular and unpredictable message in Tesseract, thus the context switch is reduced. The dynamic consumption is benefited from this and the ReRAM operations do a favor either. In large scale graphs like TT, it takes a lot in memory access and communication, so the energy saved in the large scale graph is more obvious.

Figure 18 shows the breakdown of the device area of ReGra, and only one cube is considered. ReRAM crossbars make up 14.5%, which is more space-saving than DRAM. Otherwise, the size of the Sense Amplifier is slightly larger, even if it is shared among multiple cubes. Other devices like simple ALU, cache, and message buffer make up 27.9% of the whole area. In addition to the small area of the ReRAM

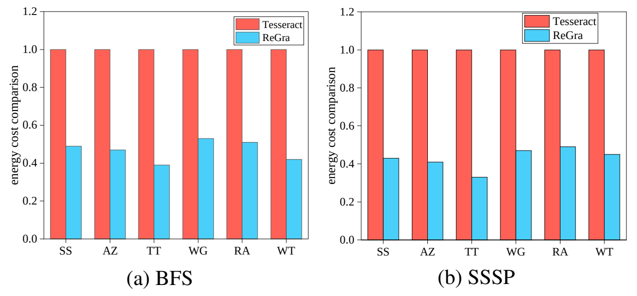


FIGURE 17. The energy cost of ReGra compared with Tesseract.

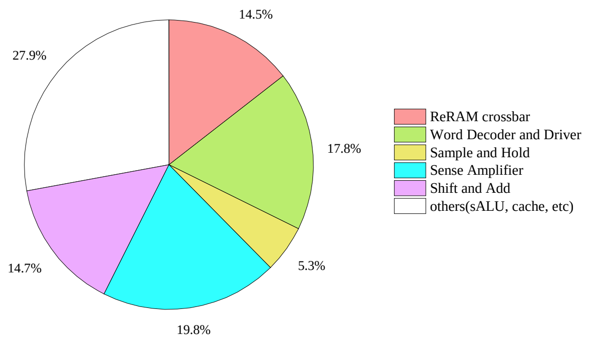


FIGURE 18. Area breakdown of ReGra.

crossbar, the read-write power is also very advantageous. The power of the ReRAM crossbar accounts for only 14% of the power of the whole cube. While in Tesseract, the power account of HMC is far more than this ratio, reaching 29%.

VI. RELATED WORK

There are lots of PIM based graph accelerators([5]–[7], [24]–[26]). Tesseract [5] utilizes the 3D-stacked HMC to offer enough memory bandwidth, achieving a “memory-capacity-proportional" feature. It is a well-known baseline for a series of later works. GraphP [19] is proposed to fully consider data organization and data partition as well as the architecture to achieve lower communication. Open questions of irregular data movement and communication still exist in previous accelerators, graphH [6] and graphQ [7] take the idea of round execution but with different implementations under the architectures and destination-based graph partition as well as the graph organization. These are all related to Tesseract, and they are also vertex-centric graph processing architectures. Another way of graph processing is edge-centric. It stores and accesses edge sequentially, and it is first proposed by X-stream [27] to get a better locality of graph processing. GraphSAR [9] adopts a hybrid-centric execution model with separate consideration of multiple edges blocks and one edge blocks, and it proposes a sparsity-aware scheme.

The emerging non-volatile memory such as ReRAM brings a new chance to solve graph processing. Several works have been proposed ([8]–[10], [28], [29]). GraphR [10] stores graphs with ReRAM crossbars and constructs several ReRAM-based graph engine to parallelly accelerate processing. RPBFS [8] compactly stores the adjacency list of vertices to achieve better read performance and it uses shared memory to maintain the status of vertices. And RPBFS is the baseline inspiring our work, the shared memory scheme in RPBFS leads to scalability problems and access conflicts, as well as irregular communication. Based on comprehensive comparison to previous work, we adopt message passing instead of shared memory access to increase the scalability. A well designed and custom circular round communication mechanism reduces irregular and unpredictable communication among cubes. In general, ReGra is an accelerator that specializes in graph traversal but not limited to graph traversal.

VII. CONCLUSION

In this work, we propose an efficient ReRAM based PIM accelerator with lower communication costs. On the one hand, ReGra partitions graphs into ReRAM cubes through IBHB approach, and stores graphs in a special CSR format to utilize the feature of ReRAM and improve memory access efficiency, and the balanced graph partition reduces the influence of communication. Moreover PIM technology provides enough bandwidth and alleviates the pressure of data movements. On the other hand, the updates in graph traversal are divided into local updates and remote updates. In remote updates values are converged into batched messages and transferred via the custom circular round communication mechanism in a concentrated period and in an organized way. This overlaps the communication with computation and eliminates irregular and unpredictable inter-cube communication. Therefore communication and its cost are significantly reduced. Benefiting from these efficient approaches, the performance of ReGra with BFS and SSSP achieves a performance speed of up to 2.2×, as well as a communication reduction of up to 76% and energy cost reduction of up to 70%.

ACKNOWLEDGMENT

This work is supported in part by National Key Research and Development Program of China under Grant No. 2018YFB1003203, the National Natural Science Foundation of China Grant No. 61972447, and the Fundamental Research Funds for the Central Universities No. 2019kfyXKJC021.

REFERENCES

[1] X. Shi, X. Luo, J. Liang, P. Zhao, S. Di, B. He, and H. Jin, “Frog: Asynchronous graph processing on gpu with hybrid coloring model,” IEEE Transactions on Knowledge and Data Engineering, vol. 30, no. 1, pp. 29–42, 2017.
[2] A. Kyrola, G. Blelloch, and C. Guestrin, “Graphchi: Large-scale graph computation on just a {PC},” in Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12), 2012, pp. 31–46.
[3] H. Liu and H. H. Huang, “Simd-x: Programming and processing of graph algorithms on gpus,” in 2019 USENIX Annual Technical Conference (USENIX ATC 19). Renton, WA: USENIX Association, Jul. 2019, pp. 411–428. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/liu-hang

[4] S. A. McKee et al., "Reflections on the memory wall." in Conf. Computing Frontiers, 2004, p. 162.

[5] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," ACM SIGARCH Computer Architecture News, vol. 43, no. 3, pp. 105–117, 2016.

[6] G. Dai, T. Huang, Y. Chi, J. Zhao, G. Sun, Y. Liu, Y. Wang, Y. Xie, and H. Yang, "Graphh: A processing-in-memory architecture for large-scale graph processing," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 4, pp. 640–653, 2018.

[7] Y. Zhuo, C. Wang, M. Zhang, R. Wang, D. Niu, Y. Wang, and X. Qian, "Graphq: Scalable pim-based graph processing," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. ACM, 2019, pp. 712–725.

[8] L. Han, Z. Shen, D. Liu, Z. Shao, H. H. Huang, and T. Li, "A novel reram-based processing-in-memory architecture for graph traversal," ACM Transactions on Storage (TOS), vol. 14, no. 1, p. 9, 2018.

[9] G. Dai, T. Huang, Y. Wang, H. Yang, and J. Wawrzynek, "Graphsar: a sparsity-aware processing-in-memory architecture for large-scale graph processing on rerams," in Proceedings of the 24th Asia and South Pacific Design Automation Conference. ACM, 2019, pp. 120–126.

[10] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Graphr: Accelerating graph processing using reram," in 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2018, pp. 531–543.

[11] M. Onizuka, T. Fujimori, and H. Shiokawa, "Graph partitioning for distributed graph processing," Data Science and Engineering, vol. 2, no. 1, pp. 94–105, 2017.

[12] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12), 2012, pp. 17–30.

[13] G. Karypis and V. Kumar, "Multilevelk-way partitioning scheme for irregular graphs," Journal of Parallel and Distributed computing, vol. 48, no. 1, pp. 96–129, 1998.

[14] F. Pellegrini and J. Roman, "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," in International Conference on High-Performance Computing and Networking. Springer, 1996, pp. 493–498.

[15] Y. Chi, G. Dai, Y. Wang, G. Sun, G. Li, and H. Yang, "Nxgraph: An efficient graph processing system on a single machine," in 2016 IEEE 32nd International Conference on Data Engineering (ICDE). IEEE, 2016, pp. 409–420.

[16] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010, pp. 135–146.

[17] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–oxide rram," Proceedings of the IEEE, vol. 100, no. 6, pp. 1951–1970, 2012.

[18] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in ACM SIGARCH Computer Architecture News. IEEE Press, 2016, pp. 27–39.

[19] M. Zhang, Y. Zhuo, C. Wang, M. Gao, Y. Wu, K. Chen, C. Kozyrakis, and X. Qian, "Graphp: Reducing communication for pim-based graph processing with efficient data partition," in 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2018, pp. 544–557.

[20] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," in ACM SIGARCH Computer architecture news, vol. 41, no. 3. ACM, 2013, pp. 475–486.

[21] M. Poremba, T. Zhang, and Y. Xie, "Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems," IEEE Computer Architecture Letters, vol. 14, no. 2, pp. 140–143, 2015.

[22] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," ACM SIGARCH Computer Architecture News, vol. 44, no. 3, pp. 14–26, 2016.

[23] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[24] M. M. Ozdal, S. Yesil, T. Kim, A. Ayupov, J. Greth, S. Burns, and O. Ozturk, "Energy efficient architecture for graph analytics accelerators," in 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). IEEE, 2016, pp. 166–177.

[25] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "Graphpim: Enabling instruction-level pim offloading in graph computing frameworks," in 2017 IEEE International symposium on high performance computer architecture (HPCA). IEEE, 2017, pp. 457–468.

[26] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2016, pp. 1–13.

[27] A. Roy, I. Mihailovic, and W. Zwaenepoel, "X-stream: Edge-centric graph processing using streaming partitions," in Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, 2013, pp. 472–488.

[28] K. Iwabuchi, H. Sato, Y. Yasui, K. Fujisawa, and S. Matsuoka, "Nvm-based hybrid bfs with memory efficient data structure," in 2014 IEEE International Conference on Big Data (Big Data). IEEE, 2014, pp. 529–538.

[29] Y. Huang, L. Zheng, X. Liao, H. Jin, P. Yao, and C. Gui, "Ragra: Leveraging monolithic 3d reram for massively-parallel graph processing," in 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2019, pp. 1273–1276.