RAGra: Leveraging Monolithic 3D ReRAM for Massively-Parallel Graph Processing

Yu Huang, Long Zheng, Xiaofei Liao, Hai Jin, Pengcheng Yao, and Chuangyi Gui
Service Computing Technology and System Lab/Cluster and Grid Computing Lab/Big Data Technology and System Lab
Huazhong University of Science and Technology, Wuhan, 430074, China
Email: {yuh, longzh, xfliao, hjin, pcyao, chygui}@hust.edu.cn

***Abstract*—With the maturity of monolithic 3D integration, 3D ReRAM provides impressive storage-density and computational-parallelism with great opportunities for parallel-graph processing acceleration. In this paper, we present RAGra, a 3D ReRAM-based graph processing accelerator, which has two significant technical highlights. First, monolithic 3D ReRAM usually has the complexly-intertwined feature with shared input wordlines and output bitlines for different layers. We propose novel mapping schemes, which can guide to apply different graph algorithms into 3D ReRAM seamlessly and correctly for *exposing* the inherently-irregular parallelism of 3D ReRAM. Second, consider the sparsity of real-world graphs, we further propose a row- and column-mixed execution model, which can filter invalid subgraphs for *exploiting* the massive parallelism of 3D ReRAM. Our evaluation on 8-layer stacked ReRAM shows that RAGra outperforms state-of-the-art planar (2D) ReRAM based graph accelerator GraphR by 6.18× performance improvement and 2.21× energy saving, on average. In particular, RAGra significantly outperforms GridGraph (a typical CPU-based graph system) by up to 293.12×.**

I. Introduction

Graph processing has been widely used as an effective means to understand the associated relationship for many real-world applications. Several graph processing-specific accelerators [1]–[4] can often greatly boost graph processing with more significant improvement in both efficiency and energy. Nevertheless, these earlier efforts construct little for compute logic, still performing one operation at a time. In contrast, by following the *matrix-vector multiplication* (MVM) paradigm, emerging *resistive random access memory* (ReRAM) allows performing multiple operations simultaneously per cycle with substantial potential parallelism.

As the rapid development of three-dimensional (3D) monolithic integration technologies, 3D ReRAM gradually has compelling storage-density and impressive computational parallelism [5], [6]. Currently, 3D ReRAM devices can be integrated in two types of architectures, from horizontal stacking [5] to vertical stacking [6]. In this work, we particularly focus on 3D horizontal ReRAM (3D-HRAM) [5] since it has a relatively-mature fabrication technology [7].

Nevertheless, leveraging emerging 3D HRAM for parallel graph processing acceleration remains tremendously challenging. First, although 2D ReRAM-based solution [8] provides an access to map graph algorithms into the matrix-formatted crossbar, it still suffers the complexly-intertwined feature of 3D ReRAM. Simply extending to use 2D mapping for 3D-HRAM will lead to incorrect (intermediate and final) results. Second, even if graph algorithms can be mapped into 3D-HRAM seamlessly and correctly with massive parallelism, there still remains a significant gap to fully exploit this parallelism for graph analytics on sparse real-world graphs.

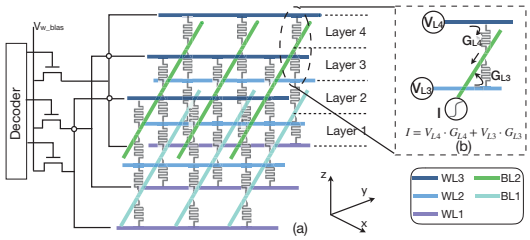

Fig. 1. Monolithic 3D-ReRAM Basics. (a) 3D-HRAM architecture with four layers stacked, and (b) A computational instance of 3D-HRAM.

In this paper, we present RAGra, a novel 3D ReRAM-based accelerator, for massively-parallel graph processing. RAGra has the following key designs. First, by fully considering the accumulative feature of 3D-HRAM's outputs, we categorize existing graph algorithms into `add`- and `non-add`-featured patterns according to their `reduce` operations by using different mapping schemes for parallelism exposure. Second, we present a novel row- and column-mixed execution model, which can filter invalid subgraphs to fully exploit the computational parallelism of 3D-HRAM.

The rest of this paper is organized as follows. We present the background in Section II. Section III overviews the RAGra architecture. Section IV elaborates mapping schemes for different graph algorithms. We discuss sparsity-aware execution model in Section V. Section VI shows the results. We conclude this work in Section VII.

II. Background

A. Graph Processing

Graph is generally organized by vertices, edges, and modifiable, user-defined properties. By following vertex-centric programming model, graph processing is usually easy to use, parallel and scale [1], but suffers from excessive random accesses. MVMs have been witnessed useful to achieve vertex programming productivity with HPC performance for graph processing [8]. By following MVM ideology, ReRAM enables to integrate MVM with in-situ computation, substantial parallelism, and considerable energy saving [8], [9].

B. Monolithic 3D-ReRAM Basics

As one of most important 3D integration technologies, Fig. 1(a) shows the schematic of 3D-HRAM, which has the planar layer-stacked structure. All odd and even layers of wordlines share the same decoder for saving area and energy, so they share the same input vectors such as $WL3$ and $WL1$ in Fig. 1(a). In particular, the adjacent layers often share the

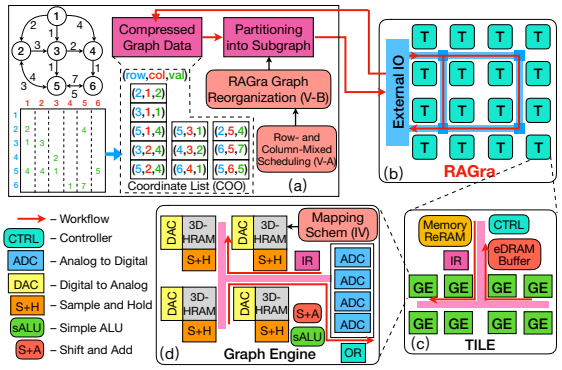

Fig. 2. Architectural overview of RAGra with (a) graph reorganization, (b) hierarchical structures, (c) tile, and (d) graph engine framework

wordlines or bitlines for the super-linear increment on storage density as the number of layers increases. This complexly-intertwined feature induces a unique computational paradigm with (minimum computational kernel) $I = V_u \cdot G_u + V_l \cdot G_l$ where V_u and V_l are the input voltages. G_u and G_l are the cell conductance of upper and lower adjacent layers. I is accumulative current in the shared bitline.

III. The RAGra Architecture

This section overviews the RAGra architecture. We first elaborate some key components for better understanding the subsequent workflow description and analysis.

A. Basic Components

As shown in Fig. 2, the RAGra chip comprises external I/O and a set of tiles. Each tile is connected with an on-chip concentrated-mesh [9]. Memory ReRAM stores graph datasets from off-chip storage. The controller converts the coordinate list (COO) to matrix format and also checks whether graph is converged. GEs have a few 3D-HRAMs connected with a shared bus and other peripheral circuits to perform MVMs. The eDRAM buffer aims to cache the intermediate output for reducing the number of writes to ReRAM.

B. Workflow of RAGra

RAGra works as the red arrows go in Fig. 2. It first partitions the input graph into many subgraphs and loads them into tiles in the matrix form. We then present novel mapping schemes to expose the massively-irregular parallelism of 3D-HRAM as will be discussed in Section IV. Afterwards, we further present a sparsity-aware execution model, which can filter the invalid subgraphs during graph iteration to exploit a maximum degree of effective parallelism as will be detailed in Section V. Finally, the new results update the off-chip storage.

IV. Mapping Graph Algorithms Into 3D-HRAM

This section elaborates how graph algorithms can be mapped into 3D-HRAM seamlessly and correctly. The goal is to expose the inherently-irregular parallelism of 3D-HRAM to accelerate graph algorithms. By considering the accumulative feature of 3D-HRAM for output results, we categorize existing graph algorithms into two kinds according to their `reduce` operation that is used for vertex update.

- `Add`-*featured graph algorithm*: This type reduces the vertex update with `add` operation such as PageRank (PR). Their reduce operations are naturally fit for the seamless mapping of accumulative operation in 3D-HRAM (Section IV-A).
- `Non-add`-*featured graph algorithm*: This type reduces the vertex update with non-add operation such as SSSP. Mapping for this type into 3D-HRAM are indirect and should be carefully designed and modified (Section IV-B).

We particularly note that those graph algorithms for checking the reachability with `non-add` feature (e.g., BFS and CC) can be interestingly turned into an `add`-featured pattern for exposing the inherent parallelism of 3D-HRAM (Section IV-C).

A. Mapping `add`-featured Graph Algorithm

Consider that the outputs of 3D-HRAM come from the sum of the upper and lower layers, and `add`-featured graph algorithms reduce the edge by a series of addition operations. This highly-consistent fact enables to expose the full-layer parallelism of 3D-HRAM.

Let us take PR as an `add`-featured case to illustrate how it can be mapped into 3D-HRAM. First, we schedule subgraphs (vertically) in a column-major order to ensure that adjacent layers handle the edges associated with the same destination vertex. In the yellowed part in Fig. 3(a), destination vertex j_4 is associated with edges from i_2, i_3 in Layer1 and i_4, i_5 in Layer2. They can be thus designed to share bitlines for outputting the reduction results of these edges. Second, we schedule subgraphs (horizontally) in a row-major order to ensure that the wordlines of all odd and even layers come from the same source vertices. For instance, both Layer1 and Layer4 have the same source vertices i_2 and i_3.

B. Mapping `non-add`-featured Graph Algorithm

`Non-add`-featured graph algorithms have no inherent add-reducing feature for the addition operation of current accumulation. It is intuitively difficult for simultaneous subgraph processing by directly using in adjacent layers with shared bitlines. For instance, the `reduce` operation for SSSP is a `min` comparison. Thence, only one layer of the adjacent layer with shared bitline can process the subgraph in a single time slot. Fortunately, we find that the edge processing of SSSP is an addition operation. We are therefore motivated to fully exploit 3D-HRAM to accelerate this part for high efficiency. Hence, we present to extract the afore-mentioned `add` operations that are compatible with bitline-shared accumulative feature of 3D-HRAM. Fig. 3(c) shows the mapping. Since comparing the results of the edge reduction affects the correctness, in each time slot we allow to handle only one row in the crossbar.

C. Turning `non-add` Pattern into `add` Pattern

Interestingly, we find that a class of `non-add`-featured graph algorithms that checks the reachability of unweighted graphs can be turned into an `add`-featured pattern for achieving higher parallelism. For instance, for BFS, its `reduce` is originally a `min` comparison. Unlike prior work that makes a couple of `compute` and `comparison` operations per edge [8], we exploit the observation that all edges associated to the same vertex can be computed (i.e., accumulated) in advance, and then do only once final `comparison` operation. This is correct in the final result for this type of graph algorithms. We present to exploit this insight in Fig. 3(d).

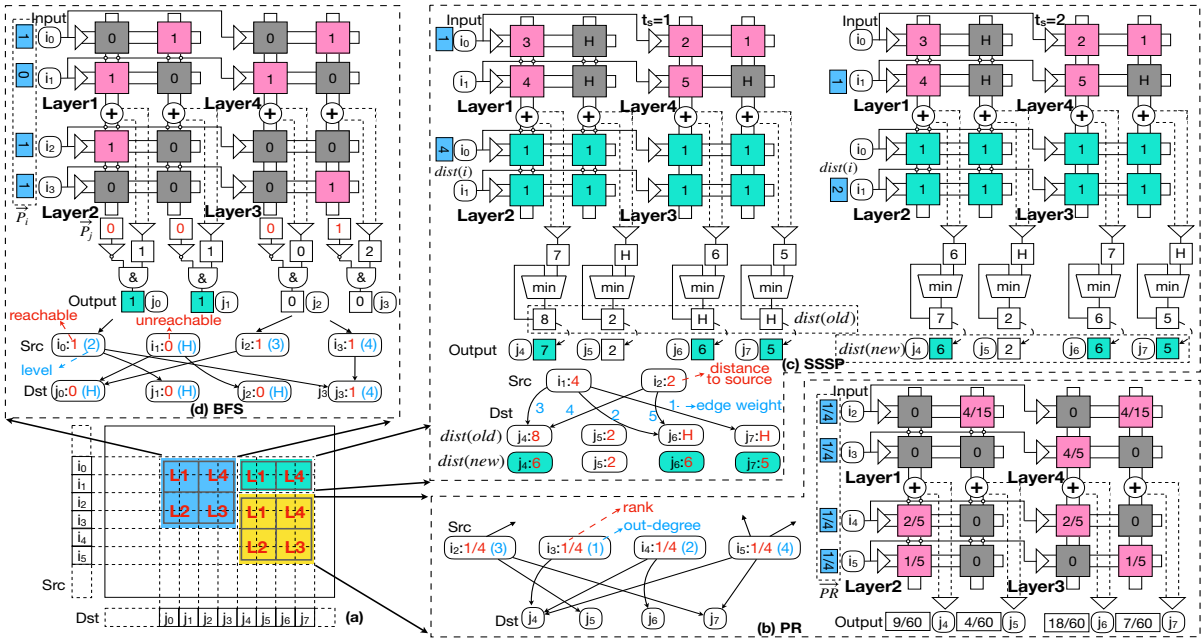

Fig. 3. Mapping different graph algorithms into 3D-HRAM. (a) A graph-induced adjacent matrix; (b) PageRank; (c) SSSP; and (d) BFS. We flat 3D-HRAM into a plane diagram for facilitating the description and comprehension. Sub-matrix labeled with L_i indicates it will be mapped into the ith Layer. **H** inside a given cell indicates no edge connected two vertices with a reserved maximum conductance value.

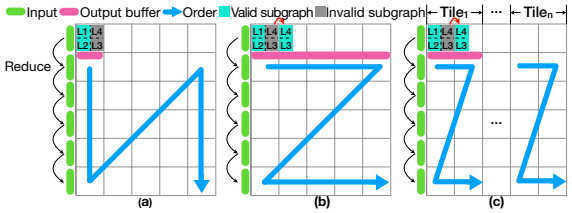

Fig. 4. Different execution model with (a) column-major order, (b) row-major order, and (c) our row- and column-mixed order

V. Sparsity-aware Execution Model

3D-HRAM can be architected to expose massive parallelism for graph algorithms in principle, but there still exists a significant gap to fully exploit this parallelism for graph datasets in practice. We present a sparsity-aware execution model to make 3D-HRAM-based graph processing performant.

A. Row- and Column-Mixed Scheduling

Graph sparsity indicates that many invalid subgraphs loaded into 3D-HRAM are essential with no valuable elements. We hence have an insight to enhance the performance of 3D-HRAM-based graph processing by filtering invalid subgraphs.

GraphR [8] schedules subgraphs in a column-major order (in Fig. 4(a)). Filtering invalid subgraphs in column-major order needs to add extra inputs, which is essentially in conflict with the shared input feature of 3D-HRAM. Although the row-major schedule order (in Fig. 4(b)) can satisfy the requirement, it greatly increases the capacity and the area of on-chip buffer. We present a row- and column-mixed execution model (in Fig. 4(c)) that can not only enable to filter invalid subgraphs but also allow to maintain a small buffer size.

B. Graph Reorganization

We note that our row- and column-mixed execution model enforces new memory patterns. Following COO format, RAGra may suffer from superfluous random accesses. We also present to use a reasonable storage sequence to reduce random access overhead. We propose to merge pairwise subgraphs vertically. Finally, we sort merged subgraphs by source vertex so that they can be scheduled sequentially inside the tile.

TABLE I
Graph Datasets

Graph	$\|V\|$	$\|E\|$	Description
WikiVote(WV)	7.1K	103K	Wikipedia Network
Slashdot(SD)	82K	948K	Slashdot Zoo Social Network
WebGoogle(WG)	0.88M	5.1M	Web Graph from Google
Pokec(PK)	1.6M	30M	Pokec Social Network
LiveJournal(LJ)	4.8M	69M	LiveJournal Social Network
Orkut(OK)	3.0M	106M	Orkut Social Network

VI. Evaluation

This section evaluates the effectiveness and efficiency of RAGra. Table I shows the real-world graph datasets from [10]. We benchmark three widely-used graph algorithms.

We use DESTINY [11] to model the performance and energy for the ReRAM part. We build RAGra based on [8] with read and write latency by 29.31ns and 50.88ns. Read and write energy consumption is by 1.08pJ and 3.91pJ. We conservatively assume the 4-bit *multi-level cell* (MLC) for computation. The default settings for crossbar size, $\#layers$, $\#GEs$ per tile, and $\#tiles$ are 8, 8, 32, and 16, respectively. We set the latency and energy of Analog/Digital converters according to [12]. All buffers and on-chip interconnects are modeled using CACTI 6.5 [13] at 32nm scale.

A. Overall Performance

We first investigate the overall performance of RAGra against state-of-the-art 2D ReRAM-based graph accelerator GraphR [8], and a CPU-based graph system GridGraph [14] on a server with 2×Intel Xeon CPU E5-2680v2 at 2.80 GHz, and 128GB DRAM. Fig 5 depicts the normalized results.

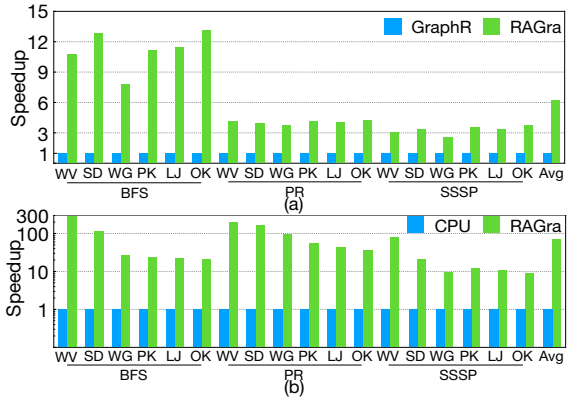

Fig. 5. Normalized performance results of RAGra against (a) GraphR, and (b) GridGrpah on a high-end CPU platform

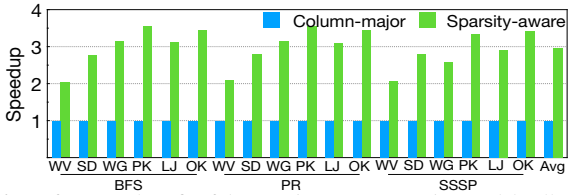

Fig. 6. Performance benefit of the sparsity-aware execution model. All results are normalized to the column-major model used in GraphR.

Compared to GraphR, RAGra is featured by improving computational parallelism. As a result, RAGra outperforms GraphR with an overall average speedup by 6.18× on average. BFS has up to 13.19×, which is significantly higher than PR and SSSP due to the full exposure of parallelism after transformation. Compared to CPU-based graph system, RAGra significantly outperforms GridGraph on a high-end 2×10-core CPU with an average speedup by 67.61×. Particularly for BFS on WV dataset, its speedup can be as much as 293.12×.

B. Benefits from Sparsity-aware Execution Model

We also compare sparsity-aware execution mode with the column-major execution order used in GraphR. As shown in Fig 6, it can offer considerable performance speedup by $2.05\times \sim 3.55\times$. The main reason accounting for this is RAGra can filter invalid subgraphs processing to fully exploit the parallelism potential of 3D-HRAM.

C. Scalability

We also evaluate the scalability of RAGra with an increasing number (4/8/16) of layers. As it can be seen that the average speedup for 8-layer and 16-layer implementation are 1.70× and 3.15× respectively as shown in Fig 7. RAGra provides an approximately-linear as the number of layers increases.

D. Energy Results

We finally evaluate energy consumption of RAGra against GraphR in Fig 8. Overall, RAGra with 8 layers shows superior energy-efficiency over GraphR with a single layer. The energy saving is by 2.21× on average for two main reasons. First, due to the fact that all odd and even layer wordlines share the DAC, this reduces roughly $\frac{\#layers}{2}$-times digital-to-analog signal operations compared to the 2D structure. Second, two-layer crossbar shares bitlines. This roughly reduces a half of the analog-to-digital signal operations.

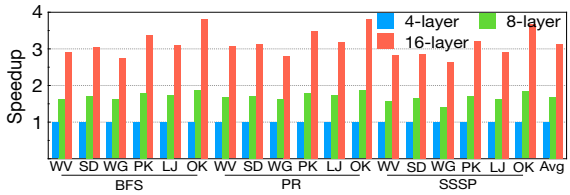

Fig. 7. Performance characterization with the different number of layers. All results are normalized to the one adopted with 4 layers.

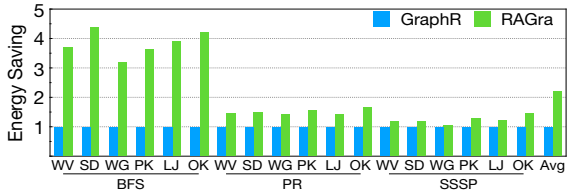

Fig. 8. Normalized results of energy saving for RAGra with respect to GraphR

VII. Conclusion

In this paper, we present RAGra, a first 3D ReRAM-based graph processing accelerator, which enables to leverage irregular parallelism of 3D-ReRAM for massively-parallel graph analytics. Results on an 8-layer stacked horizontal ReRAM show that RAGra outperforms state-of-the-art GraphR by 6.18× speedup and 2.21× energy saving. In addition, RAGra can also offer up to 293.12× speedup over GridGraph on a high-end CPU platform.

Acknowledgment

This paper is supported by the National Key Research and Development Program of China (2018YFB1003500), NSFC (61825202, 61832006, 61702201, 61628204) and China Postdoctoral Science Foundation (2018T110765 and 2018M630862). To whom the correspondence should be addressed to Long Zheng.

References

[1] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, "Graphicionado: A High-Performance and Energy-Efficient Accelerator for Graph Analytics," in *MICRO'16*.

[2] M. M. Ozdal *et al.*, "Energy Efficient Architecture for Graph Analytics Accelerators," in *ISCA'16*.

[3] M. Zhang *et al.*, "GraphP: Reducing Communication for PIM-based Graph Processing with Efficient Data Partition," in *HPCA'18*.

[4] P. Yao, L. Zheng, X. Liao, H. Jin, and B. He, "An Efficient Graph Accelerator with Parallel Data Conflict Management," in *PACT'18*.

[5] Y. C. Chen, H. Li, W. Zhang, and R. E. Pino, "3D-HIM: A 3D High-density Interleaved Memory for Bipolar RRAM Design," in *ISNA'11*.

[6] W. Chien *et al.*, "Multi-Layer Sidewall WOx Resistive Memory Suitable for 3D ReRAM," in *VLSIT'12*.

[7] M. Mao, S. Yu, and C. Chakrabarti, "Design and Analysis of Energy-Efficient and Reliable 3-D ReRAM Cross-Point Array System," *VLSI'18*.

[8] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "GraphR: Accelerating Graph Processing Using ReRAM," in *HPCA'18*.

[9] A. Shafiee *et al.*, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ISCA'16*.

[10] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford Large Network Dataset Collection," http://snap.stanford.edu/data.

[11] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "DESTINY: A Tool for Modeling Emerging 3D NVM and eDRAM caches," in *DATE'15*.

[12] B. Murmann, "ADC performance survey 1997-2018," https://web.stanford.edu/ murmann/adcsurvey.html.

[13] "CACTI," http://www.hpl.hp.com/research/cacti/.

[14] X. Zhu, W. Han, and W. Chen, "GridGraph: Large-Scale Graph Processing on a Single Machine Using 2-Level Hierarchical Partitioning." in *ATC'15*.