# HyVE: Hybrid Vertex-Edge Memory Hierarchy for Energy-Efficient Graph Processing

Tianhao Huang, Guohao Dai, Yu Wang, Huazhong Yang
Department of Electronic Engineering, TNLIST, Tsinghua University, Beijing, China
{hth14, dgh14}@mails.tsinghua.edu.cn, {yu-wang, yanghz}@tsinghua.edu.cn

***Abstract*—High energy consumption of conventional memory modules (e.g., DRAMs) hinders the further improvement of large-scale graph processing's energy efficiency. The emerging metal-oxide resistive random-access memory (ReRAM) and ReRAM crossbar have shown great potential in providing the energy-efficient memory module. However, the performance of ReRAMs suffers from data access patterns with poor locality and large amounts of written data, which are common in graph processing. In this paper, we propose a Hybrid Vertex-Edge memory hierarchy, HyVE, to avoid random access and data written to ReRAM modules. With data allocation and scheduling over vertices and edges, HyVE reduces memory energy consumption by 69% compared with conventional memory system in graph processing. Moreover, we adopt a bank level power-gating scheme to further reduce the stand-by power. Our evaluations show that the optimized design achieves at least 2.0x improvement of energy efficiency compared with DRAM-based designs.**

## I. Introduction

Graphs are widely used to represent relationships among real-world entities in various domains, such as social networks, webpage hyperlinks, etc. Ever-growing demands for large-scale graph data processing in data centers and mobile devices pose challenges to both speed and energy efficiency. Compared with graph processing frameworks developed on clusters or a single PC [1–5], accelerators [6–12] concentrate more on the energy efficiency. Most graph processing applications are memory-bound. Therefore, with well-designed scheduling of data access, graph processing accelerators show huge potential on energy efficiency without performance compromises (e.g., Graphicionado [6] achieves 50x energy efficiency compared with CPU).

However, the energy efficiency of accelerators is currently negatively influenced by modern high-performance but energy-hungry memory modules. Power breakdown results [13] show that over 60% of energy is consumed by memory for PageRank. In Graphicionado [6], the energy consumption of memory is even higher (90%). Memory energy usage, which has been a problem in CPU-based platforms, poses a more serious bottleneck on graph processing accelerators.

High energy usage of main memory modules (e.g., DRAMs) is closely associated with the type and organization of memory cells. The recent advancement in memory technology offers possible solutions to some of these challenges. One of the most promising solutions is Resistive Random-Access Memory (ReRAM) [14, 15]. ReRAM is a non-volatile memory (NVM) with much lower static energy, similar read delay but much higher write delay compared with DRAMs. ReRAM alone is not sufficient for an energy-saving graph processing system without performance degradation due to the data access patterns of poor locality and huge amount of write traffic. Thus, proper memory hierarchy design and data scheduling become the key to exploit advantages of ReRAMs in graph processing accelerators.

In this paper, we propose a **Hy**brid **V**ertex-**E**dge memory hierarchy, HyVE, for efficient graph processing. In HyVE, different types of graph data are stored in either ReRAM or conventional memories according to different access patterns. A memory controller is responsible for handling various requests from high-level accelerator logic. Techniques like vertices replacement scheduling and bank-level power-gating are introduced to HyVE to further improve the energy efficiency. The contributions of this paper include:

- **Hybrid memory hierarchy with data scheduling**. We propose a hybrid memory hierarchy, HyVE, composed of both emerging ReRAMs and conventional memories (e.g., DRAMs and SRAMs) to achieve higher energy efficiency in graph processing. Vertices and edges in the graph are stored into different memories according to their distinct data access patterns. Accesses to these data are well-scheduled to fully take advantages of different types of memories.
- **Efficient power-gating scheme**. Based on HyVE, we propose an aggressive power-gating scheme for graph processing on ReRAMs to reduce background energy. It is made feasible by the non-volatility of ReRAMs and nature of the edge-centric graph processing model.
- **Extensive experiments**. We have conducted extensive experiments to evaluate the performance of HyVE and HyVE with power gating. Critical trade-off decisions are made in terms of ReRAM cell bit length and SRAM capacity. Our design finally achieves at least 2.0x better energy efficiency compared with conventional memory systems.

## II. Background and Motivation

### A. Graph Processing Models

Graph algorithms ranging from Breadth-First Search (BFS) to PageRank can be described with the general Gather-Apply-Scatter (GAS) model. In the GAS model, different computations are done in the apply phase, depending on the

This work was supported by National Key Research and Development Program of China (No. 2017YFA0207600), National Natural Science Foundation of China (No.61373026, 61622403, 61621091) and Joint fund of Equipment pre-Research and Ministry of Education (No. 6141A02022608).

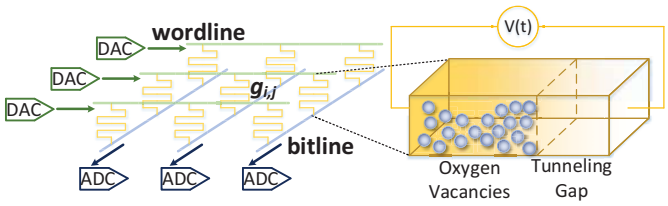



Fig. 1. Structure of the ReRAM Crossbar Array (mat).

implemented graph algorithms. In a shared-memory machine, we have two simplified methods to implement GAS:

- Vertex-centric: Iterating over active vertices. All outgoing neighbors' properties are updated using the local property.
- Edge-centric: Iterating over edges. For each edge, the destination is updated using the source's property.

X-Stream [5] first proposed the edge-centric model and pointed out that edge-centric algorithm has better spatial locality than the vertex-centric one. Edge-centric algorithm achieves sequential access over edges at the cost of random accesses over the vertices whose size is orders of magnitude smaller than edges. Later graph processing frameworks either adopt X-Stream-like algorithm or just sort the edges to improve data locality. As a result, memory access patterns of graph processing consist of sequential read over edges and random read/write over vertices.

Proper data partitioning also plays an important role in the performance of graph processing. By adopting a 2D interval-block partitioning method, previous systems [16, 17] only need to update one part of the graph at a time. Vertices are partitioned according to indexes, and edges are partitioned according to source/destination vertices. The locality is ensured in this way.

### B. Challenges of Graph Processing Accelerators

Hardware accelerators are inherently more efficient than general-purpose platforms as the computation units, datapaths and memory accesses are tailored to certain applications. Besides, since the performance bottleneck is memory bandwidth for graph processing accelerators, researchers have been trying to employ high-bandwidth memory devices or large on-chip memory to improve the design, such as Hybrid Memory Cube (HMC) [8, 13] and eDRAM [6].

With such efforts, the energy consumption of computation units and logic is reduced significantly. However, they also expose memory as the most energy-hungry component in the system. The basic structure of conventional memories (mostly DRAM) is hard to be optimized for low power and maintained comparable performance at the same time. In other words, conventional memories have become a serious bottleneck of further energy efficiency improvement in graph processing accelerators (e.g., the energy consumption of memory is 90% in Graphicionado [6]). We must seek an energy-economical alternative to DRAM and integrate it into the existing systems seamlessly.

### C. ReRAM Technology

ReRAM refers to the non-volatile memory that uses varying metal-oxide cell resistances to store information. A single-level ReRAM cell (SLC) has two states – low and high resistance states. Different states are switched by applying an external voltage of certain polarity, magnitude, and duration. A multiple-level cell (MLC) [18] uses $2^N$ resistance levels to represent $N$ bits of data, and thus involves finer tuning of switching voltages and sensing schemes. The most efficient way of organizing ReRAM array is the crossbar structure (Fig. 1). In comparison with the conventional DRAM, ReRAM benefits from low static power and non-volatility. Versus other NVM technologies such as phase change memory, ReRAM benefits from superior endurance ($> 10^{10}$) [14], no resistance drift and lower energy usage for write operations.

Long write latency is the main hurdle for ReRAM to be a viable alternative to the main memory. To overcome the drawback, previous work [19] proposed microarchitecture level optimizations along with a smart scheduling algorithm, finally achieving almost the same performance as DRAM under various benchmarks.

Apart from treating ReRAM as a substitute of DRAM, some accelerators exploit the ReRAM crossbar to implement matrix-vector multiplications. GraphR [12] proposes a ReRAM-based graph processing accelerator by taking advantage of multiple ReRAM crossbars, or graph engines (GEs), to carry out graph computations in parallel. However, the small size of GE will incur unbearable overhead on vertex data traffic between ReRAM and registers as datasets become larger (the largest one used in GraphR has only ∼5M vertices and ∼100M edges). The proposed architecture would also be impractical for algorithms like Single Source Shortest Path, because the number of resistance levels of current MLCs is far from enough to represent the vertex data.

Taking characteristics of both ReRAM and graph algorithms into consideration, our design mainly focuses on the storage role of ReRAM. Note that HyVE will not restrict specific graph computation logic implementations.

## III. Hybrid Memory Hierarchy for Energy-efficient Graph Processing

There is a significant difference between two major types of memory access in graph processing, according to Section II-A. Edge data access is essentially a sequential read of the main memory, while vertex data access involves fine-grained random read and write operations. Moreover, storage space of edges is generally 10x∼100x larger than that of vertices in natural graphs. Given these facts, we divide the off-chip main memory into two functional parts, edge memory and vertex memory. Finally, we provide a solution to integrate the memory hierarchy with higher-layer computation logic.

### A. Edge Memory

Edge memory is designed exclusively for sequential read of a large amount of edges. During the algorithm initialization, the edge data go through a one-shot preprocessing step and are written into the memory. An edge only contains its source and destination indices, possibly plus a constant edge weight. All data will remain unchanged. Therefore, edge memory can be

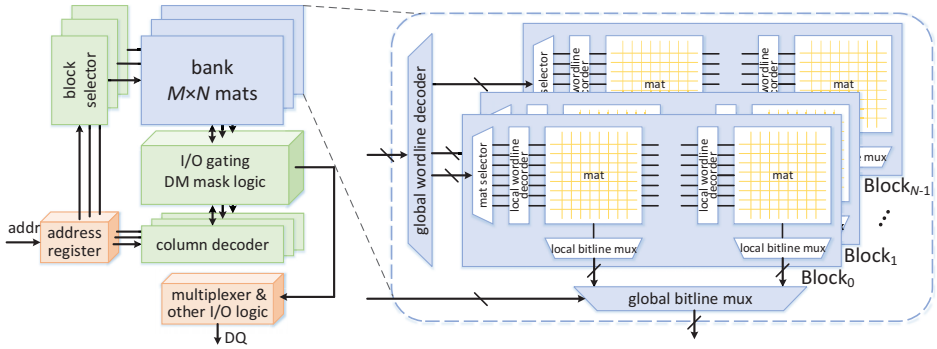



Fig. 2. The on-chip organization of a ReRAM memory: a chip consists of several banks, each bank consists of several mats (ReRAM crossbar).

treated as a large-capacity read-only streaming device during the program execution.

In the perspective of energy efficiency, main memory constructed by ReRAM cells is the most suitable one for this role. Huge volumes of data and the need for high memory bandwidth cancel the chances of SRAM or flash memory respectively. DRAM has enough bandwidth, but as Section II-B presents, a large area of DRAM chip will incur substantial refresh and standby power, becoming the energy bottleneck of the whole system. With close read timing parameters, ReRAM-based memory has comparable read bandwidth against DRAM. Limited write bandwidth of ReRAM will not cause obvious delay since the data write only happens during initialization. More importantly, ReRAM-based memory reduces static power and improves the area efficiency because refresh mechanism is no longer necessary. Our baseline design, a ReRAM chip shown in the left side of Fig. 2, is organized the same way as commodity DRAM counterparts.

To extract more energy efficiency from large-volume edge memory, we exploit the sub-bank level interleaving and non-volatility of ReRAM cells. The right side of Fig. 2 illustrates the detailed structure of a ReRAM *bank*. A bank is made up of *rows* and *columns* of *mats* rather than simply a large ReRAM array. For a read operation, several selected mats generate output data. Similar to the bank interleaving scheme, sub-bank level interleaving utilizes independent mats to improve sequential bandwidth and increase row hit rate. However, interleaving also means that multiple active working units (mats or banks) and a larger amount of background energy usage. For the edge memory, we adopt sub-bank interleaving and avoid bank interleaving. Finer grained interleaving allows more banks to be put into power-saving state and also provides with enough bandwidth. Furthermore, due to the non-volatility of ReRAM and sequential read pattern, a bank level power-gating scheme is adopted to completely remove the leakage power, which is explained in Section IV.

### B. Vertex Memory

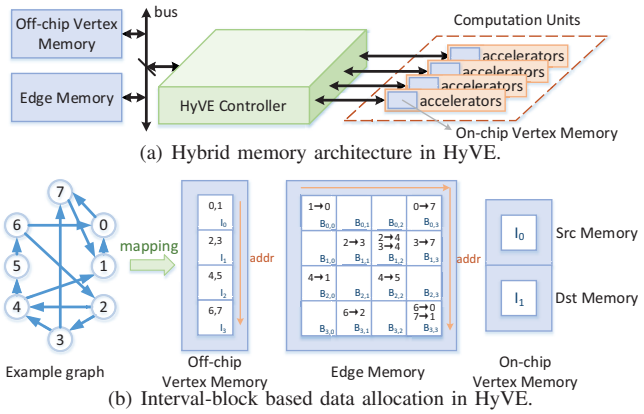
Unlike edge memory, vertex memory has to handle frequent random read/write traffic. Direct random accesses to DRAM or ReRAM modules, however, are far less efficient than sequential accesses. To deal with this problem, a fast small-capacity on-chip SRAM with data scheduling is employed. Computation units can issue consecutive read/write requests to SRAM without waiting for extra clock cycles.

On-chip SRAM alone is not sufficient for global vertex random access since normally-sized SRAM cannot accommodate the whole vertex data of most large natural graphs. According to previous research of graph partitioning and scheduling [16], global randomness can be alleviated using interval-block based partitioning. The partitioning is based on indexes: Interval $I_i$ contains the $i$-th disjoint vertex set; Block $B_{ij}$ contains the edges whose source belongs to $I_i$ and destination belongs to $I_j$. In summary, we adopt two types of vertex memory in HyVE, on-chip and off-chip vertex memory. Off-chip vertex memory is used to store a complete copy of vertex data while on-chip vertex memory only loads a partition of vertices for processing.

Before the edge memory is streaming one partition of edges out, the corresponding intervals should have been scheduled on-chip. Each accelerator can directly access the on-chip vertex memory, which consists of a source vertex section and a destination vertex section. When the current partition has been traversed, on-chip vertex memory will write the modified vertex data back to off-chip vertex memory and update to next interval(s). All operations on off-chip memory are sequential. HyVE adopts DRAM as the off-chip vertex memory because of its high write bandwidth compared with other candidates. Vertex memory has much smaller capacity than edge memory, so the static power is not the main optimization target here.

### C. Hybrid Vertex-Edge Memory Hierarchy

Providing an interface to integrate the hybrid memory modules with existing accelerators is a feasible way to improve the energy efficiency of many graph processing accelerators. The main abstraction layer is a hybrid memory controller responsible for address mapping, edge buffering and vertex data scheduling. ReRAM-based edge memory and DRAM-based off-chip vertex memory, located in the lowest layer, are connected to the controller via a dual-channel bus. SRAM-based vertex memory is recommended to be placed on chip to minimize the delay. When fetching edges from edge memory, the memory controller keeps checking the current partition and newly coming edges, in case vertex data scheduling is triggered. During scheduling on-chip vertex memory access requests are stalled. Upwards, the hybrid memory controller is the interface to accept memory requests from the computation logic. The data allocation based on interval-block partitioning is shown in Fig. 3(b).




(a) Hybrid memory architecture in HyVE.

(b) Interval-block based data allocation in HyVE.

Fig. 3. Architecture and data allocation in HyVE.

## IV. Efficient Bank Level Power-gating Scheme for Non-volatile Edge Memory

Power-gating is a widely used technology to put idle logic circuits into powered-off states to reduce the leakage power of CMOS. However, power-gating has following limitations:

- Extra storage may be needed to save circuit states and recover data.
- Transitions between power-gated and active states may lead to extra time overhead. Frequent transitions will incur performance penalty.
- Relevant control logic and the power gates must be introduced. Many large power gates may incur chip area penalty.

Then we consider two types of memories in HyVE, vertex memory and edge memory. Unpredictable and randomized access pattern to the vertex memory makes it impossible to adopt such power-gating scheme. Rather, sequential read from the edge memory means that only a small region of the edge memory is kept busy regularly. With the purpose of further improving the energy efficiency of HyVE, it is reasonable to apply power-gating to the edge memory.

In order to implement power-gating properly, the three limitations should be considered. Non-volatility of ReRAMs makes it free from the first limitation. The other two limitations are relevant to memory access patterns and granularity of the power-gating scheme. The sequential access pattern of edges in HyVE makes the transition between two states predictable and infrequent. Moreover, since HyVE does not apply bank interleaving, it can make efficient use of bank-level power-gating (BPG) scheme. As in Fig. 4, we adopt the BPG controller to implement the BPG scheme in the edge memory. BPG controllers will power down most of the banks except those which have been accessed recently. Active banks which are not issued commands in a fixed period of time are also powered down by BPG controllers. In terms of sequential access, usually only one bank per chip is active. We could thus place one single power gate (header or footer) to be in charge of the whole bank. This design incurs little overhead on power gates, or low area penalty.

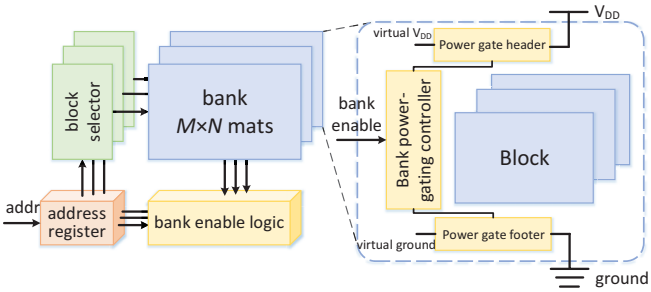



Fig. 4. Bank level power-gating in HyVE: structures in yellow are extra units dedicated to power-gating (compared with Fig. 2).

## V. Evaluations

### A. Experiment Setup

Five typical graph datasets shown in Table I are used for our evaluation. PageRank (PR), Breadth-first Search (BFS), and Connected Components (CC) are the evaluated algorithms. We use MTEPS/W (Millions of Traversed Edges Per Second per Watt) as the metric of throughput in this Section.

TABLE I
Graph datasets used in evaluation

| Datasets | #Vertices | #Edges |
|---|---|---|
| com-youtube (YT) | 1.16 million | 2.99 million |
| wiki-talk (WK) | 2.39 million | 5.02 million |
| as-skitter (AS) | 1.69 million | 11.1 million |
| live-journal (LJ) | 4.85 million | 69.0 million |
| twitter-2010 (TW) | 41.7 million | 1,470 million |

For the performance evaluation, we designed a custom cycle-accurate simulator based on the single-board version of [9] and HyVE. The energy and timing models of SRAM and ReRAM were generated using NVSim [20], while those of DRAM were generated using Micron System Power Calculators [21]. Since NVSim allows quite a few optimization targets and configurations, different combinations are attempted to acquire the most energy-efficient one (Table II). We also use the Intel Processor Counter Monitor (PCM) [22] to measure the performance on a hexa-core Intel i7 CPU running at 3.3GHz, to demonstrate graph processing on conventional CPU-based architecture as the baseline. The software program running on the CPU is similar to NXgraph [17] except that we resident all data in the main memory in our experiments.

### B. Design Decisions

*1) ReRAM Output Bits:* NVSim [20] provides diverse optimization models for a given ReRAM design. To optimize the energy efficiency of the whole system, we compare two optimization directions, energy-optimize (minimize the energy consumption per read operation) and latency-optimize (minimize the working period) in Table II. We compare the energy efficiency by joule per bit, which means the lowest power used to read one bit from ReRAM. The energy-optimized model with 512 bits output achieves the optimal energy efficiency[1], we choose this model in following designs.

[1]NVSim [20] can only provide simulation results for output bits smaller than 512 bits.

TABLE II
POWER CONSUMPTION UNDER DIFFERENT BANK CONFIGURATIONS

| | | energy (pJ) | period (ps) | power/bit (mW/bit) |
|---|---|---|---|---|
| energy-optimized | 64bits | 20.13 | 1221 | 0.26 |
| | 128bits | 33.87 | 1983 | 0.13 |
| | 256bits | 57.31 | 1983 | 0.11 |
| | **512bits** | **102.07** | **1983** | **0.10** |
| latency-optimized | 64bits | 381.47 | 653 | 9.13 |
| | 128bits | 378.57 | 590 | 5.01 |
| | 256bits | 382.37 | 590 | 2.53 |
| | 512bits | 660.23 | 527 | 2.45 |

*2) ReRAM Cell bits:* ReRAM cell could have more than two states in MLC chips. Since NVSim cannot model the behavior of MLC ReRAM chip, we made necessary modification to NVSim according to the parallel sensing scheme [23] to calculate the power and delay in MLC chips. With these parameters our simulator generates the results in Figure 5. Although MLC chips could increase the density and the internal bandwidth of the memory, extra sense amplifiers in MLC chips bring a non-negligible amount of energy usage. Therefore, SLC turns out to outperform MLC in HyVE. SLC ReRAM is adopted in later evaluations.

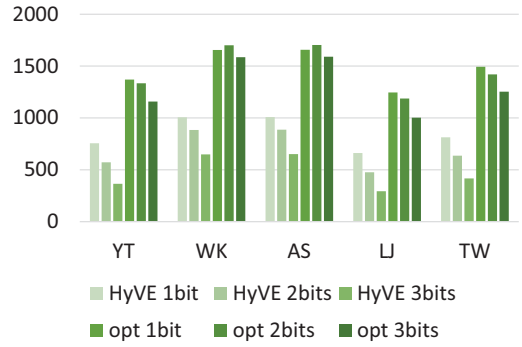



Fig. 5. Energy efficiency (MTEPS/W) using different ReRAM cells (We adopt power-gating scheme introduced in Section IV in 'opt-' configurations).

*3) SRAM Capacity:* To evaluate the influence of variant SRAM sizes, we extract the parameters from SRAM of capacity 4MB, 8MB, 16MB, 32MB and then simulate the energy efficiency of HyVE. The result is shown in Table III. Although larger capacity of SRAM reduces the data traffic between memories, increasing leakage power, read/write energy and delay have to be taken into consideration. Besides, the most efficient SRAM configuration is related to the size of input graphs. For small datasets with small block numbers like YT, WK and AS, 4MB SRAM is sufficient to achieve high energy efficiency. For larger datasets, 4MB SRAM loses its advantage due to frequent data scheduling. In conclusion, for graph datasets with 1M to 50M vertices, 8MB on-chip SRAM turns out to be the sweet spot.

TABLE III
ENERGY EFFICIENCY VARYING SRAM SIZES (MTEPS/W)

| | | without power-gating | | | | with power-gating | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 4MB | 8MB | 16MB | 32MB | 4MB | 8MB | 16MB | 32MB |
| BFS | TW | 806 | **1187** | 976 | 887 | 1544 | **1831** | 1517 | 1200 |
| | LJ | 1096 | **1121.6** | 950 | 871 | **1913** | 1774.2 | 1491 | 1188 |
| | AS | 1099 | **1130** | 1049 | 852 | **1916** | 1777.6 | 1584 | 1174 |
| | WK | 700 | 927.9 | **1061** | 784 | 1389 | **1568.4** | 1595 | 1122 |
| | YT | 871 | 1045.5 | **1100** | 803 | 1632 | **1694.1** | 1629 | 1137 |
| CC | TW | 581 | 756 | **868** | 794 | 1204 | 1370 | **1413** | 1131 |
| | LJ | 949 | **1006** | 994 | 834 | **1735** | 1655 | 1535 | 1161 |
| | AS | 969 | **1008** | 1000 | 820 | **1759** | 1657 | 1540 | 1149 |
| | WK | 552 | 662 | **720** | 704 | 1156 | 1246 | **1250** | 1054 |
| | YT | 767 | 813 | **874** | 733 | 1488 | **1493** | 1417 | 1079 |
| PR | TW | 340 | **755.9** | 598 | 573 | 707 | **1370** | 942.3 | 775 |
| | LJ | 649 | **1005.8** | 752 | 632 | 1148 | **1654.7** | 1078 | 815 |
| | AS | 674 | **1009.1** | 731 | 618 | 1179 | **1656.7** | 1060 | 803 |
| | WK | 312 | **662.3** | 546 | 523 | 660 | **1245.7** | 886 | 684 |
| | YT | 524 | **813.4** | 666 | 546 | 987 | **1437.5** | 993 | 748 |

## C. HyVE Performance

*1) Energy Efficiency of HyVE:* We compare the energy efficiency results among different memory hierarchies, including acc+DRAM, acc+SRAM+DRAM, acc+HyVE. Data scheduling in these three configurations is the same. For acc+SRAM+DRAM and acc+HyVE, one SRAM chip size is set to 8 MB. Performance measured on the physical machine (CPU+DRAM) is the baseline. For different graph datasets, different partition numbers are used to fit into the SRAM. Power-gating scheme is not tested for now.

Fig. 6 shows the comparison results. Acc+HyVE achieves 1.5x and 4.0x energy efficiency improvement on average over acc+SRAM+DRAM and acc+DRAM configurations respectively. Besides, acc+HyVE achieves 2 orders of magnitude (114.4x on average) improvement over CPU-based baseline.

*2) Bank Level Power-gating:* HyVE adopts aggressive bank level power-gating by powering down most of ReRAM banks. We validated the improvement of energy efficiency brought by such scheme. Experimental results in Fig. 7 show that the aggressive power-gating in HyVE yields at least 1.3x better results than acc+HyVE in general, which demonstrates the effectiveness of our power-gating design. The proportion is 2.0x if we compare acc+HyVE-opt and acc+SRAM+DRAM.

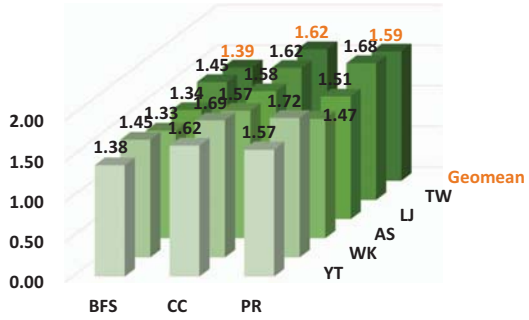



Fig. 7. Energy efficiency improvement by adopting power-gating in HyVE.

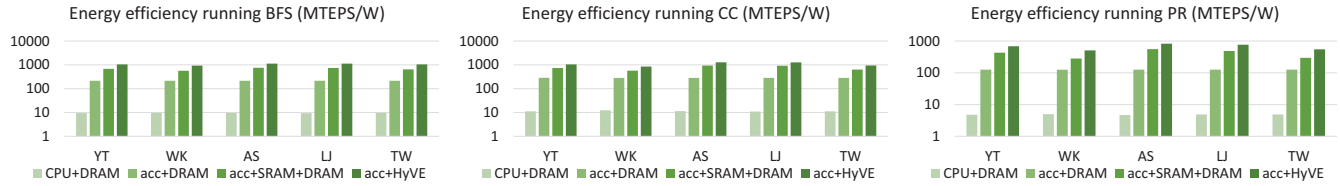



Fig. 6. Energy efficiency (MTEPS/W) comparison between HyVE and other configurations.

*3) Energy Consumption Breakdown:* As mentioned in Section I, on-chip and off-chip memory takes up more than 60% of overall energy consumption in systems without HyVE. We illustrate the energy consumption breakdown under three configurations, acc+SRAM+DRAM (SD), acc+HyVE (HyVE), and acc+HyVE+power-gating (opt) in Fig. 8. By adopting ReRAM in HyVE, the energy consumption of the whole memory system reduces by 51% and 69% under two HyVE configurations compared with SD configuration. The memory subsystem accounts for 59% and 49% of the total energy consumption in HyVE and opt, while the percentage of SD is 76%. The obvious drop in the edge memory energy consumption is the main reason for overall energy savings. Thus, HyVE can significantly reduce the energy consumption in graph processing systems.

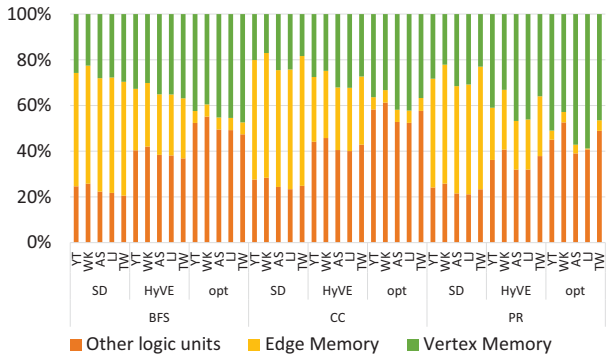


Fig. 8. Energy consumption breakdown.

*4) Absolute System Performance:* We compare the execution time of acc+SRAM+DRAM (SD) and acc+HyVE to see how HyVE can possibly influence the absolute system performance. Fig. 9 shows the normalized measurement data. It can be observed that there exist a slight increase of execution time if HyVe is adopted. The geometry mean of performance degradation for the three tested algorithms are merely 1.9%, 2.5% and 15.1% respectively. It should be noted that the performance results are closely associated with design decisions of the selected ReRAM. Thus, with our choice of key parameters in ReRAM modules, HyVE is able to yield profitable energy efficiency results and incur no significant performance penalty.

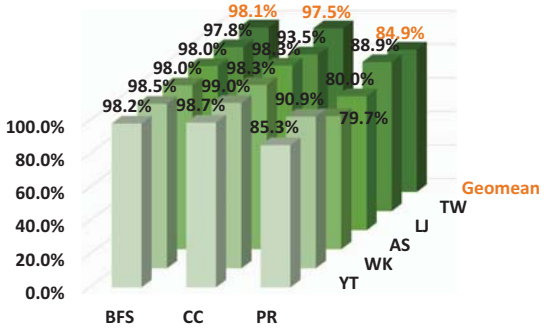


Fig. 9. Execution time comparison between SD and HyVE (SD/HyVE).

## VI. Conclusion

In this paper we present HyVE, a novel hybrid memory hierarchy which takes advantage of ReRAM technology. The organization of HyVE and a possible method to integrate HyVE with existing systems are discussed thoroughly. We also put forward a power-gating scheme to optimize our design. Extensive experiments on various datasets show that HyVE brings more than 2.0x energy efficiency improvements compared with common DRAM-based accelerators, and 2 orders of magnitude improvement compared with the CPU-DRAM architecture. The results indicate that HyVE is a promising solution for designing memory hierarchy of more efficient graph processing accelerators.

## References


[1] Grzegorz Malewicz et al. Pregel: a system for large-scale graph processing. In *SIGMOD*, pages 135–146. ACM, 2010.

[2] Yucheng Low et al. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

[3] Aapo Kyrola et al. Graphchi: Large-scale graph computation on just a pc. In *OSDI*, pages 31–46. USENIX, 2012.

[4] Xiaowei Zhu et al. Gemini: A computation-centric distributed graph processing system. In *OSDI*, pages 301–316. USENIX, 2016.

[5] Amitabha Roy et al. X-stream: Edge-centric graph processing using streaming partitions. In *SOSP*, pages 472–488. ACM, 2013.

[6] Tae Jun Ham et al. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics. In *MICRO*, pages 1–13. IEEE, 2016.

[7] Muhammet Mustafa Ozdal et al. Energy efficient architecture for graph analytics accelerators. In *ISCA*, pages 166–177. IEEE, 2016.

[8] Junwhan Ahn et al. A scalable processing-in-memory accelerator for parallel graph processing. In *ISCA*, pages 105–117. IEEE, 2015.

[9] Guohao Dai et al. Foregraph: Exploring large-scale graph processing on multi-fpga architecture. In *FPGA*, pages 217–226. ACM, 2017.

[10] Tayo Oguntebi et al. Graphops: A dataflow library for graph analytics acceleration. In *FPGA*, pages 111–117. ACM, 2016.

[11] Guohao Dai et al. Fpgp: Graph processing framework on fpga a case study of breadth-first search. In *FPGA*, pages 105–110. ACM, 2016.

[12] Linghao Song et al. GraphR: Accelerating graph processing using ReRAM. *arXiv preprint arXiv:1708.06248*, 2017.

[13] Mingyu Gao et al. Practical near-data processing for in-memory analytics frameworks. In *PACT*, pages 113–124. IEEE, 2015.

[14] J Joshua Yang et al. Memristive devices for computing. *Nature nanotechnology*, 8(1):13–24, 2013.

[15] Meng-Fan Chang et al. 19.4 embedded 1mb reram in 28nm cmos with 0.27-to-1v read using swing-sample-and-couple sense amplifier and self-boost-write-termination scheme. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 332–333. IEEE, 2014.

[16] Xiaowei Zhu et al. GridGraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning. In *ATC*, pages 375–386. USENIX, 2015.

[17] Yuze Chi et al. NXgraph: an efficient graph processing system on a single machine. In *ICDE*, pages 409–420. IEEE, 2016.

[18] Shyh-Shyuan Sheu et al. A 4mb embedded slc resistive-ram macro with 7.2 ns read-write random-access time and 160ns mlc-access capability. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pages 200–202. IEEE, 2011.

[19] Cong Xu et al. Overcoming the challenges of crossbar resistive memory architectures. In *HPCA*, pages 476–488. IEEE, 2015.

[20] Xiangyu Dong et al. NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE TCAD*, 31(7), 2012.

[21] Micron system power calculators. https://www.micron.com/support/tools-and-utilities/power-calc.

[22] Intel pcm. https://software.intel.com/en-us/articles/intel-performance-counter-monitor/.

[23] Cong Xu et al. Understanding the trade-offs in multi-level cell ReRAM memory design. In *DAC*, pages 1–6. IEEE, 2013.