Accelerating Graph Computations on 3D NoC-enabled PIM Architectures

DWAIPAYAN CHOUDHURY, LIZHI XIANG, ARAVIND SUKUMARAN RAJAM, ANANTH KALYANARAMAN AND PARTHA PRATIM PANDE

Washington State University, Pullman, WA

Graph application workloads are dominated by random memory accesses with poor locality. To tackle the irregular and sparse nature of computation, ReRAM-based Processing-in-Memory (PIM) architectures have been proposed recently. Most of these ReRAM architecture designs have focused on mapping graph computations into a set of multiply-and-accumulate (MAC) operations. ReRAMs also offer a key advantage in reducing memory latency between cores and memory by allowing for processing-in-memory (PIM). However, when implemented on a ReRAM-based manycore architecture, graph applications still pose two key challenges – significant storage requirements (particularly due to wasted zero cell storage), and significant amount of on-chip traffic. To tackle these two challenges, in this paper we propose the design of a 3D NoC-enabled ReRAM-based manycore architecture. Our proposed architecture incorporates a novel crossbar-aware node reordering to reduce ReRAM storage requirements. Secondly, its 3D NoC-enabled design reduces on-chip communication latency. Our architecture outperforms the state-of-the-art in ReRAM-based graph acceleration by up to 5x in performance while consuming up to 10.3x less energy for a range of graph inputs and workloads.

CCS CONCEPTS • Computer systems organization ~ Architectures ~ Other architectures ~ Special purpose systems

Additional Keywords and Phrases: Processing-in-Memory, Vertex Reordering, Graph Analytics, ReRAM, Small World NoC.

1 INTRODUCTION

Graphs have become ubiquitous in several data-driven applications and machine learning workflows, as they offer an effective way to model networked behavior in both the natural world and human-engineered systems. However, with steep increases in both the volume of observable data and the diversity in applications, scalable processing for graph workloads on emerging manycore platforms remains a challenge. While CPU- and GPU-based manycore platforms continue to be used for executing graph applications, poor locality in graph structures and irregular data access patterns pose significant challenges. Skewed vertex degree distributions of real-world graphs make it nearly impossible to maintain high locality in graph structures, causing repeated accesses to vertex neighborhoods or random walk traversals to incur a high volume of cache misses. Furthermore, the deep memory hierarchies in conventional manycore architectures (such as CPUs and GPUs) exacerbate the cost of data movement [1].

Resistive random-access memory (ReRAM)-based Processing-in-Memory (PIM) modules, offer an effective way to address the high memory bandwidth requirement of graph analytics by integrating the computing logic in the memory. The ReRAM crossbars can store the adjacency matrix of a graph and the computation in most graph primitives can be decomposed into multiply-and-accumulate (MAC) operations, which are supported by ReRAM. However, most real-world graphs are sparse–i.e., with far fewer number of nonzero cells than the zero cells–causing significant wastage in the storage across the ReRAM crossbars (as only nonzeros contribute to meaningful computation). One way to reduce storage as well as improve locality in the distribution of nonzeros is through vertex (re)ordering [2]. By assigning similar ranks to vertices that are also neighbors on the graph, reordering techniques can effectively cluster the nonzero cells along the main diagonal of the adjacency matrix. While this increased density of nonzero

1084-4309/2022/1-ART1 $15.00
http://dx.doi.org/10.1145/3564290

cells can reduce wasted storage on ReRAMs, current vertex reordering schemes are not fully equipped to maximize on this potential as they do not consider the crossbar structure of ReRAMs [3]. Secondly, current ReRAM-based approaches [2][3] also do not support an efficient communication backbone between ReRAM-based processing elements (PEs). Graph computations frequently feature irregular memory accesses including long range traffic between PEs, which could degrade overall performance and energy efficiency.

In this paper, we address the above limitations of ReRAM-based graph acceleration by presenting the design of an efficient 3D Network-on-Chip (NoC)-enabled ReRAM manycore accelerator for graph analytics. The main contributions are as follows:

1) (Software-level) To improve performance and reduce storage for ReRAM-based graph applications, we propose an efficient crossbar-aware vertex reordering-based approach.
2) (Hardware-level) To reduce communication latency for irregular graph workloads, we present the design of a 3D NoC architecture that optimizes ReRAM block placement on the manycore platform.
3) (Evaluation) We present a thorough evaluation of our proposed architecture on various real-world graph inputs using different graph operations – namely, PageRank, Single Source Shortest Path (SSSP), Connected Components (CC), BFS and Triangle counting. Our proposed framework significantly outperforms existing state-of-the-art ReRAM-based graph accelerators both in terms of execution time and energy consumption.

2 BACKGROUND AND RELATED WORK

Designing specialized manycore architectures for graph analytics has been an area of active research in recent years. Though CPU and GPU-based manycore computing have been used, the data movement due to irregular memory accesses limits performance and energy efficiency. One possible way is to modify the organization of caches and partition them into multiple planar layers in a 3D structure to improve the cache hit rate [4]. DRAM-based Hybrid Memory Cube (HMC) is another way to enhance performance of graph accelerators [5][24][25]. However, the deep memory hierarchies in these architectures degrade the overall performance.

Performance of most of the current ReRAM-based accelerators is limited by the sparsity and lack of locality in graph structures [6][7]. To this end, vertex reordering techniques can help by clustering non-zero elements in graph adjacency matrix [2][3]. Yet, in almost all existing ReRAM-based graph accelerators either reordering techniques are unaware of crossbar structure, or the crossbar bounded property does not utilize the benefit introduced by the clustering of non-zero entries.

Another factor influencing performance is the cost of data movement. An efficient communication backbone for inter-PE exchanges is critical; however, existing ReRAM-based graph accelerators do not support such efficient and scalable on-chip communication [3]. An optimized placement of the PEs and suitable network-on-chip (NoC) design have been shown to significantly improve the overall latency and energy efficiency, including for graph analytics [8]. However, these NoC architectures do not consider ReRAM-based PEs. Design of 3D NoC-based ReRAM architecture for training graph neural networks (GNN) involving dense weight matrices has been proposed [15]. The dense computations make the GNN workloads different from the sparsity seen in graph workloads. Hence, in this paper, we bridge the gap in

the state-of-the-art of ReRAM-based graph accelerators by designing a crossbar-aware vertex reordering scheme (software-level) complemented with an optimized NoC architecture (hardware-level) to achieve high performance and energy efficiency. We postulate that optimizing solely at either the software-level or at hardware-level will be inadequate as the gains achieved at one layer can be lost in the other if left unoptimized. In contrast, our software-hardware design is better positioned to generate significant performance gains because of its complementary nature – i.e., reducing data movement and storage requirement using software, while reducing communication latency using hardware.

3 VERTEX REORDERING

Preliminaries: Graph computations involve traversing the input sparse adjacency matrix corresponding to the graph. Since it is only the nonzero values of the matrix that contribute to work, reducing the zero storage becomes an important consideration. One way to achieve this is to rearrange the rows and columns of the adjacency matrix such that the concentration of nonzero cells is "clustered" in only some regions of the matrix, so that the vast remaining sections of the matrix, which have only zero cells need not be stored.

Vertex (re)ordering is an effective way to perform such a clustering [9]. Given an input graph $G = (V, E)$ with n vertices (in V) and m edges in E, the goal is to compute a linear ordering $\Pi : i \longrightarrow [1, n]$, for every vertex $i \in V$, such that the average linear gap distance in Π between any two neighbors $(i, j) \in E$ is minimized. The assignment $\Pi(i)$ is also referred to as the rank of vertex i. We refer to the original input ordering as the graph's *natural ordering* ($\Pi(i) = i$, for each $i \in V$). The process of taking a natural ordering and producing a different vertex ordering is referred to as "reordering". Several heuristics are used to generate reordering [10]. These schemes range from light-weight (e.g., degree-based) to more heavy-weight (window- and partitioning-based) schemes [9]. However, most existing node reordering algorithms are designed assuming a more traditional parallel platform (multicores, cluster computing) and remain oblivious to the ReRAM crossbar structure. Two recently proposed ReRAM-based graph accelerators (GraphSAR[3]and Spara [2]) leveraged vertex reordering techniques which help to outperform several well-known previous investigations (e.g., GraphR [6] and HyVE [7]) making them appropriate as baselines to consider.

GraphSAR [3] proposes vertex reordering technique where the rank of each vertex is assigned in an incremental order depending on their location in the original graph input file. More specifically, while loading the original edge list, an index is assigned to each new vertex starting from 0. For example, if vertices 1 and 3 are the vertices of the first edge listed in the input file, then these two vertices are renumbered as 0 and 1 respectively (i.e., $1 \longrightarrow 0$ and $3 \longrightarrow 1$). Subsequently, any vertex to be encountered for the first time is assigned the next unallocated vertex rank in an incremental fashion. This implies that the vertex reordering will depend on the order in which the list of edges is provided at input. Figs. 1 (a) and (b) show the adjacency matrices of the original and the reordered graph while using GraphSAR. Considering the ReRAM crossbar of size 2x2 (for illustration purpose only), we can see that the number of active blocks for original and reordered graph from Figs. 1 (a) and (b). In this example, the GraphSAR scheme reduces the number of active blocks from twelve in the natural ordering to eleven in the GraphSAR ordering. This reordering scheme is oblivious to the underlying crossbar configuration.

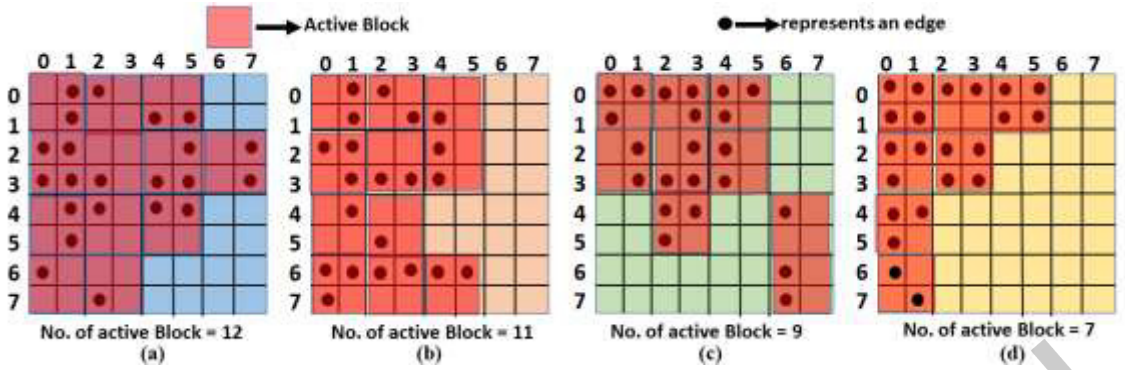

Fig 1: The adjacency matrices for (a) original (Natural) and reordered graph for (b) GraphSAR, (c) Spara and (d) CARE.

Spara [2] uses different graph formats (compressed sparse row (CSR) and column (CSC)) to determine the ranks for destination and source vertices of an edge. Starting from an initial vertex, it searches its destination vertex set based on the CSR-formatted graph. Next, each node in the destination vertex set is analyzed one-by-one to obtain a new source vertex set based on the CSC representation. Based on that source vertex set, it then finds the new destination vertex set until it reaches the bounded threshold, which directly depends on the crossbar size. Fig. 1 (c) shows the adjacency matrix of the reordered graph using Spara. Considering the threshold as two for illustration purpose, we can see from Fig. 1 (c) that the number of active blocks is nine in the reordered graph by Spara, whereas it is twelve for the original graph. Hence, clustering the edges by reordering results in the reduction of the number of active crossbars.

However, the crossbar-aware feature does not fully exploit the advantage introduced by the clustering, leading to suboptimal use of ReRAM-based architectures. Moreover, Spara and GraphSAR both rely on sequentially processing the vertices to determine the new vertex labels. This makes both these two algorithms inherently sequential. Hence, we present a new crossbar-aware vertex reordering scheme called CARE that improves the clustering factor of the adjacency matrix and thus reduces the total number of "active blocks."

Crossbar-Aware Vertex Reordering (CARE) Algorithm: A matrix block of size X^*X is considered "*active*" if it contains at least one non-zero cell. The objective of the CARE algorithm is to minimize the total number of active blocks (via reordering of rows and columns), which in turn reduces the execution time, storage requirement, and power consumption.

Terminology: For a given adjacency matrix A, a row panel of size l starting at row r is a slice of A, which includes all rows from r to $r+l-1$. Let *col_seg(j, r, l)* denote a column segment of a given column j of length l starting at the cell at row r, i.e., the contiguous slice $A[r:r+l-1, j]$. A column segment *col_seg(j, r, l)* is considered "active" if at least one of its cells is a non-zero. Similarly, a 2D block of matrix A is considered "active" if at least one of its cells is a non-zero. Let $active_{col(p)}$ represent the set of column IDs of all the non-zero cells in row p. We define the similarity of two rows, p and q, using the Jaccard similarity of the active columns, i.e.,

$$J(p,q) = \frac{|active_{col(p)} \cap active_{col(q)}|}{|active_{col(p)} \cup active_{col(q)}|} \quad (1)$$

The CARE algorithm is based on the following main ideas: (i) for a crossbar of size *X*, the number of active blocks is positively correlated with the number of active column segments; (ii) grouping rows with a high similarity can reduce the total number of active column segments; and (iii) empty 2D blocks within a row panel can be safely ignored.

Row ordering: Building on these ideas, CARE first tries to reorder rows with high similarity together and then reorders the columns to minimize the total number of active 2D blocks. First, rows are reordered so that similar rows are assigned contiguous row ids. Jaccard similarity can be used to group and reorder the rows; however, such an approach is expensive ($O(n^2\sigma)$; n = #rows, σ = average #nnz per row). Alternatively, a light-weight approach is to sort the rows based on the number of non-zeros – intuitively, vertex (row) pairs that share a high Jaccard similarity also need to have similar degrees (i.e., a necessary but not sufficient condition). Once the rows are reordered, the set of rows is partitioned into row panels of size *X*. Fig. 2 depicts phase 1 of our algorithm. Fig. 2 (a) shows the original adjacency matrix *A*. Fig. 2 (b) shows the state of *A* after row sorting. Fig. 2 (c) shows the conceptual view of *A* after the *X*-way row panel split.

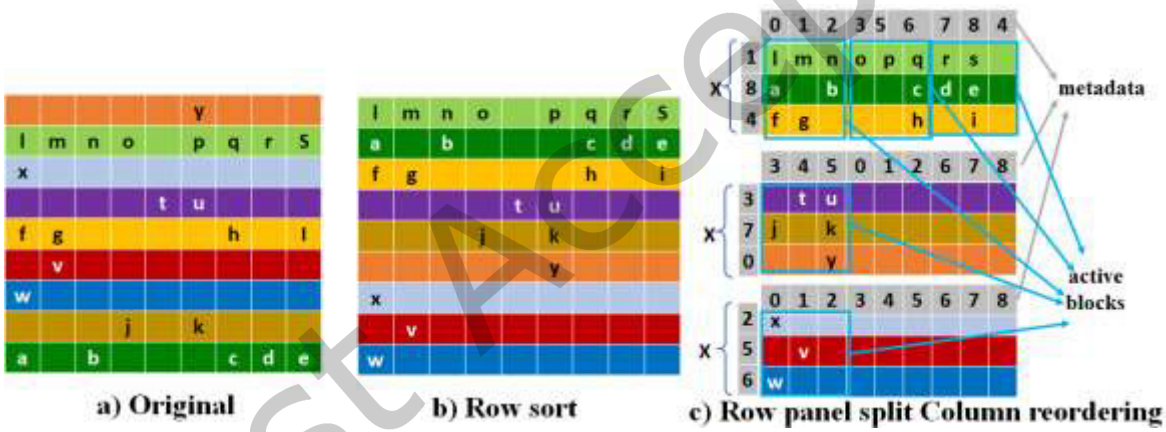

Fig. 2: Overview of CARE reordering algorithm (X represents crossbar size)

Algorithm 1: CARE reordering algorithm.

Input: Adj. matrix $A[1:n][1:n]$ for $G(V, E)$;
Crossbar size X; Number of panels $p = \lceil \frac{n}{X} \rceil$
Output: Π_r: row reorder;
$\Pi_c[1:p]$: column reorderings (one per panel)

```
1  CARE(A, X)
2      Reorder_Rows(A)
3      Reorder_Cols(A, X, p)
4  Reorder_Rows(A)
5      Π_r ← Degree sort rows in non-ascending order;
6      return Π_r
7  Reorder_Cols(A, X, p)
8      Block partition the reordered rows (Π_r) into p
         panels of size X rows each
9      for each row panel P do
10         Initialize col_id ← 0;
11         for each column slice j ∈ [1, n] within P do
12             if there exists at least one nonzero in j then
13                 Π_c[P][col_id] ← j
14                 Increment col_id
15         Assign empty columns to the unused column
             ids in Π_c[P]
16     return Π_c
```

Column ordering: The second step of the approach reorders the columns. While existing approaches reorder columns, we reorder the column segments within each row panel (without explicitly renumbering the column ids), allowing for a better clustering of nonzero cells. For each row panel, we find the list of active column segments. Each such active column is then reordered such that the first active column is placed in column 0, the second active column in column 1, and so on. In other words, all the active columns are grouped together and moved to the left side, leaving the non-active columns grouped together to the right side. A separate array per row panel is used to indicate the column id (metadata). Fig. 2 (c) shows the state after reordering. The blue boxes represent the active blocks. Clustering subgraphs with active elements in left columns helps to discard the inactive blocks placed in the right side of the adjacency matrix. As ReRAM crossbars store active blocks only, it reduces storage requirement. Moreover, the locality improvement by the CARE reordering scheme brings a vertex closer to its neighboring vertices and thus decreases the on-chip traffic. Fig. 1 (d) shows the adjacency matrix of the reordered graph by using CARE. Here, the value of X is considered as two for illustration purpose. We can see from Figs. 1 (a), (b), (c) and (d) that the number of active blocks is twelve, eleven and nine for natural, GraphSAR and Spara respectively, whereas the number of active blocks for CARE is seven. However, though CARE potentially reduces the number of active vertices, when irregular graph workloads are mapped onto a ReRAM-based manycore architecture, inter-PE communication is significant. It should be noted that any ReRAM-based architecture must be divided into multiple ReRAM tiles with bounded crossbar size. Hence, inter-PE traffic is inevitable. Therefore, to reduce communication latency for irregular graph workloads, we present the design of a 3D NoC that optimizes ReRAM-based PE placement on the manycore platform in the next section.

4 OVERALL ARCHITECTURE

In this section, we present the key attributes of our proposed architecture including the ReRAM-based tile (4.1) and NoC (4.2).

4.1 Tiled Architecture

Vertex In ReRAM-based accelerators, the adjacency matrix of the input graph is stored across the ReRAM cells, and graph computations are decomposed into a set of MAC operations that are performed based on Ohm's and Kirchhoff's current laws. By applying a voltage into the word line and sensing the resultant current along the bit-line, we implement the product of the input voltage and the cell conductance. Along with the product, the sum is obtained through the current summation over the bit-lines. Each row computes a product by streaming in the multiplicand via the word-line Digital to Analog Converter (DAC). The overall system consists of multiple ReRAM processing elements (PEs), where each PE contains several ReRAM tiles. Each ReRAM tile is composed of several crossbars and the associated peripherals [15].

We use a simple strategy to map each active block (blue boxes in Fig. 2 (c)) to ReRAM tiles. Each active block is assigned by a sequential id S and is mapped to the a unique tile (i,j), where

$$0 < i,j < \sqrt{N} \; and \; i = S \bmod \sqrt{N} \; \; and \; j = \mathrm{S} \, / \sqrt{N} \qquad (2)$$

4.2 NoC-Based Communication Backbone

When irregular graph workloads are mapped onto a ReRAM-based manycore architecture, inter-PE communication is significant. To analyze the effects of inter-PE communication, we considered three graph applications, viz. PageRank, Single Source Shortest Path (SSSP) and Connected Components (CC). Six different datasets (Table 1) considered in this work, are taken from the Stanford Network Analysis Platform [18] and the Network Repository [19].

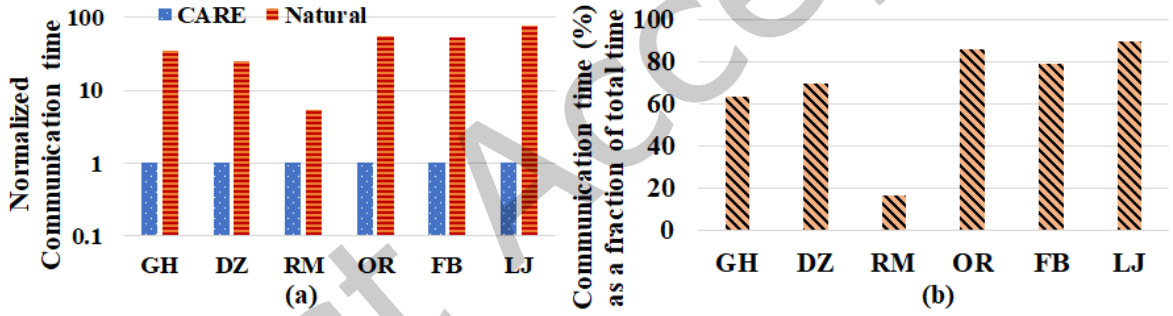

Fig. 3: PageRank communication analysis: (a) Factor of increase in communication time for Natural ordering relative to CARE. (b) Contribution of communication to the total execution time for CARE.

Fig. 3(a), Fig. 4(a), and Fig. 5 (a) show the normalized time needed for inter-PE communication with the natural ordering and CARE reordering scheme with PageRank, SSSP and CC, respectively. It is evident from Fig. 3(a), Fig. 4(a) and Fig. 5 (a) that locality improvement by CARE achieves significant reduction (25.2x to 76.1x) in on-chip communication time compared to the natural ordering except for the RM (5.3x) dataset. The reduction in savings is least for RM because it is a road network with a uniform degree distribution, and consequently there is relatively less to be gained in locality through reordering relative to natural ordering. All other inputs (GH, DZ, OR, FB and LJ), which have power-law degree distribution characteristics, demonstrate larger savings with the CARE ordering. Though CARE reduces the overall communication cost compared to natural, it still has significant amount of inter-PE data traffic. Fig. 3(b),

Fig. 4(b), and Fig. 5 (b) show the contribution of inter-PE communication in total processing time for a 2D Mesh NoC-based manycore architecture incorporating CARE. We can see from Fig. 3(b), Fig. 4(b), and Fig. 5 (b) that even after applying CARE, the contribution of inter-PE communication to total execution time for all the datasets is high (63.4% to 89.7%) except for RM (16.6%). This motivates the need for designing an efficient NoC for inter-PE communication (even with CARE).

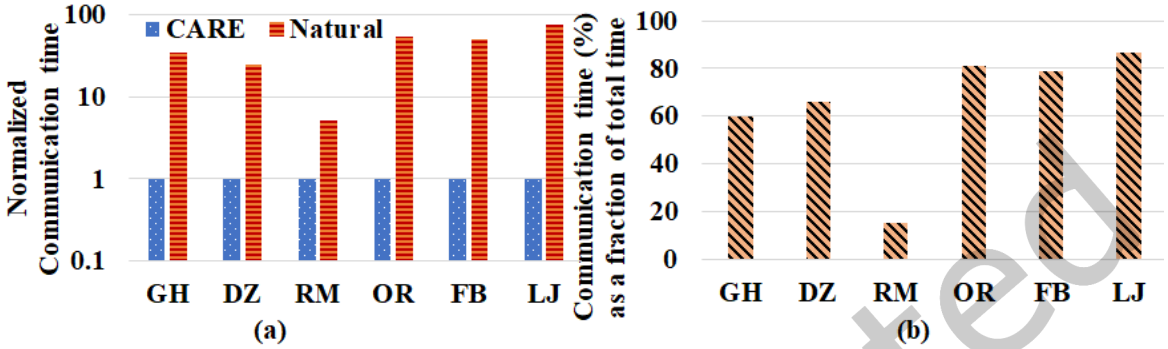

Fig. 4: SSSP communication analysis: (a) Factor of increase in communication time for Natural ordering relative to CARE. (b) Contribution of communication to the total execution time for CARE.

Traffic and network topology: Most of the graph workloads follow the Gather-Apply-Scatter (GAS) model, where processing each vertex includes (a) gathering values from incoming neighbors, (b) generating new value and (c) scattering that to all outgoing neighbors [3]. Hence, graph operations on ReRAM-based accelerators are expected to predominantly give rise to many-to-few traffic patterns. This many-to-few traffic pattern involves long-range communication, which degrades the overall performance. Conventional 2D Mesh NoC architectures are not suitable for this kind of traffic [11]. It has already been shown that either by inserting long-range shortcuts in a regular Mesh to induce small-world effects or by adopting power-law based small-world connectivity, we can achieve significant performance gain and lower energy dissipation compared to traditional multi-hop Mesh networks [11]. Therefore, we design a small-world network based NoC (SWNoC) where the links between routers are established following a power law distribution for the graph applications under consideration. However, when a small-world network is implemented in a 2D structure, there will be multiple physically long wires connecting the largely separated PEs. Ultimately, this will give rise to high timing and energy overheads. However, when a small-world NoC is implemented using 3D integration, the largely separated PEs in a 2D structure can be placed in different planar dies and connected using vertical links. Fig. 6 shows the overall architecture and the illustration of the 3D SWNoC based ReRAM-based manycore architecture. As shown in Fig. 6, relatively longer planner links, can be converted to shorter vertical links by placing the communicating PEs in two planar dies. Hence, it reduces the timing and energy costs [11]. Therefore, in this work, we design a 3D SWNoC to enhance the overall performance.

Table 1: Input statistics of the graph datasets used in our experiments.

Input graph (label)	No. vertices	No. edges
musae_Github (GH)	37,699	289,003
gemsec-Deezer (DZ)	41,773	125,826
road_luxembourg-osm (RM)	114,598	119,667
com-Orkut (OR)	2,937,612	20,959,854
socfb-A-anon (FB)	3,097,165	23,667,394
soc-LiveJournal1 (LJ)	4,847,571	68,993,773

Placement: Due to massive data parallelism, ReRAM based manycore architectures are typically optimized to achieve high throughput. Reducing the average hop count reduces latency, making PEs available for more computation and thereby improving the throughput of computation. Additionally, load balancing across the NoC is used to further enhance throughput [12]. Minimizing the standard deviation of hop count will achieve load balancing by reducing the congestion along various paths. Hence, we compare designs (θ) with different PE and link placements via the degree of achievable load balancing in the NoC, i.e., using mean $M(\theta)$ and standard deviation $SD(\theta)$ of the hop count, as given by:

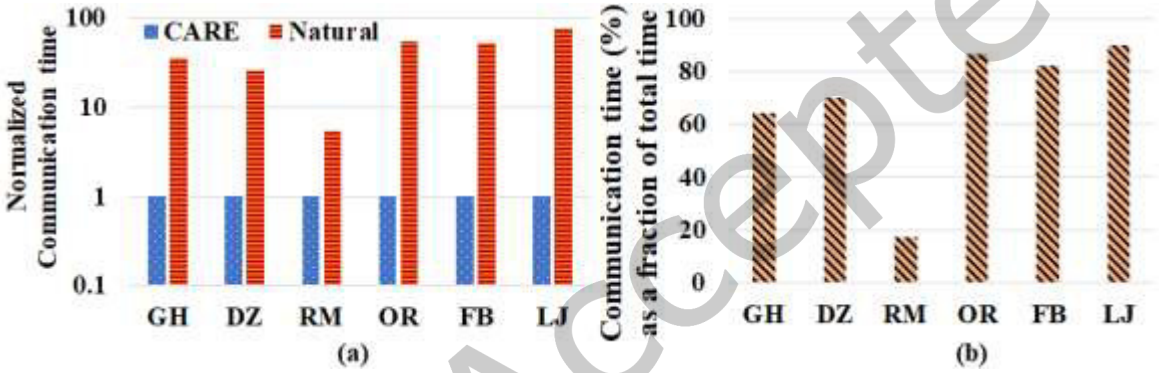

Fig. 5: CC communication analysis: (a) Factor of increase in communication time for Natural ordering relative to CARE. (b) Contribution of communication to the total execution time for CARE.

$$M(\theta) = \frac{1}{L} * \sum_{i=1}^{C} \sum_{j=1}^{C} h_{ij} \qquad (3)$$

$$SD(\theta) = \sqrt{\frac{1}{L} \sum_{i=1}^{C} \sum_{j=1}^{C} \left(h_{ij} - M(\theta)\right)^2} \qquad (4)$$

where C and L represents the number of PEs and the number of links respectively, in the overall architecture, h_{ij} is the number of hops from PE i to PE j. Therefore, designing the optimized SWNoC boils down to a multi-objective optimization (MOO) problem where both $M(\theta)$ and $SD(\theta)$ are minimized to maximize the achievable throughput. We can represent the MOO-formulation as follows:

$$P^* = \left\{ \theta^* \mid \theta^* \in \arg\max_{\theta} f(M(\theta), SD(\theta)) \right\} \qquad (5)$$

where, P^* is the set of Pareto optimal designs. We choose the design (θ^*) from the set of Pareto optimal designs where the throughput is maximum. The optimization problem is solved by using the popular simulated annealing (SA) based multi-objective heuristic, AMOSA [13] as it can find a high-quality solution with optimized placement of PEs and links in a reasonable time.

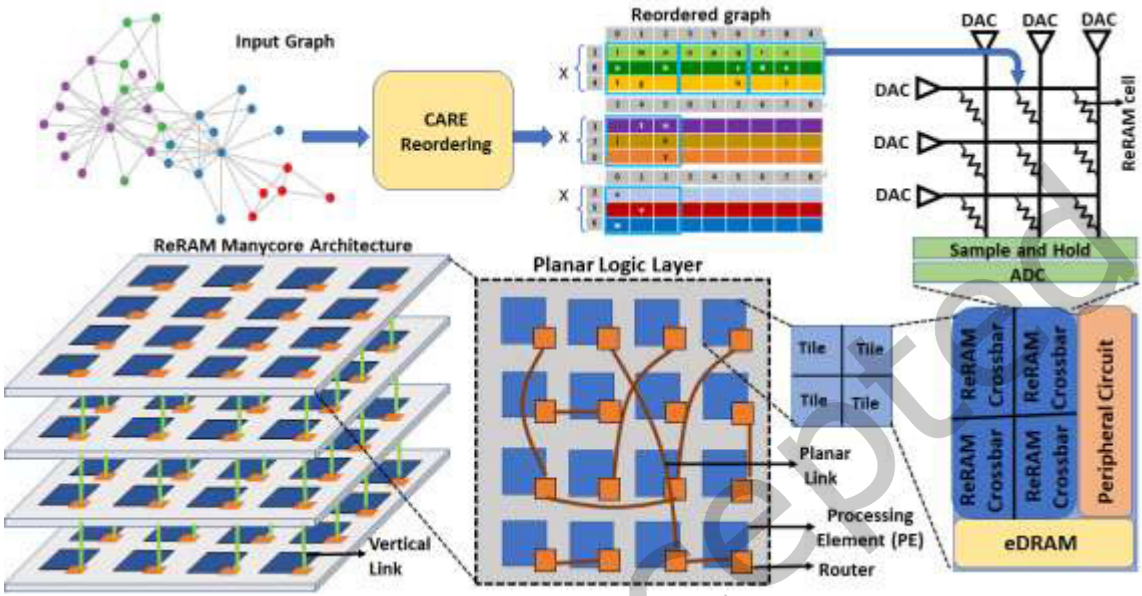

Fig. 6: Illustration of the overall architecture.

5 EXPERIMENT RESULTS

Experimental Setup: We use NVSim [17] in conjunction with BookSim [16] to evaluate the performance of the proposed ReRAM-based manycore architecture. We leverage Booksim [16] for implementing different NoC architectures considered in this work. In the proposed architecture, each PE has four tiles. Each tile contains 96 crossbars (128x128) and associated peripheral circuits such as ADC, DAC, etc. along with eDRAM. The capacity of eDRAM is considered as 36 MB in our proposed design. The value of LRS and HRS are 14.7 KΩ and 167 KΩ respectively. Here, we assume ReRAMs that can store 2-bits per cell. Each PE takes up 0.37 mm^2 of area [14]. The architecture requires multiple such ReRAM PEs for storage as well as computation, to accommodate the large sizes of input graphs. For implementing 2D Mesh, 1024 PEs are arranged in an 32x32 grid pattern. Considering a 20mmx20mm die, the length of each inter-router link is 0.625mm. The overall system runs at the clock frequency of 2.5 GHz. Considering this clock frequency, a 0.625 mm link can be traversed in one cycle. In our proposed 3D SWNoC architecture, 1024 PEs are equally partitioned into four planar layers. Each layer is of size 10mmx10mm (considering same area as the 2D system). Within each layer, 256 PEs are placed in 16x16 grid pattern. In the SWNoC architecture, there are planar links longer than 0.625mm. The longer links are divided into multiple pipelined stages where each stage is of length 0.625 mm. Hence, multiple cycles are necessary to traverse these links. All the vertical links connecting the planar layers are traversed in one cycle. BookSim determines the overall NoC latency.

We use the PE and memory characteristics along with total NoC latency in NVSim to determine the overall energy consumption and execution time. We evaluate the performance of the manycore architecture incorporating CARE with respect to two state-of-the-art ReRAM-based graph accelerators, GraphSAR [3] and Spara [2]. We choose GraphSAR and Spara as these are the state-of-the-art architectures that outperform other previously developed techniques such as GraphR [6] and HyVE [7]. More specifically, GraphSAR achieves 4.43x energy reduction and 1.85x speedup with respect to GraphR and 1.29x speedup and 2.18x energy reduction compared with HyVE on an average. On the other hand, Spara outperforms GraphR and GraphSAR by 8.21x and 5.01x in terms of performance, and by 8.97x and 5.68x in terms of energy savings, respectively. Therefore, as GraphSAR and Spara already demonstrated the comparative performance analysis with respect to other state-of-the-art counterparts, we refrain from repeating those results in this paper for brevity. Table 1 shows all the inputs used for the full system performance analysis. Table 2 shows the specifications of the proposed 3D manycore ReRAM architecture. For thermal evaluation, we model the overall architecture in 3-D-ICE simulator based on various parameters e.g., layer thickness, thermal conductivity, etc. as listed in [20].

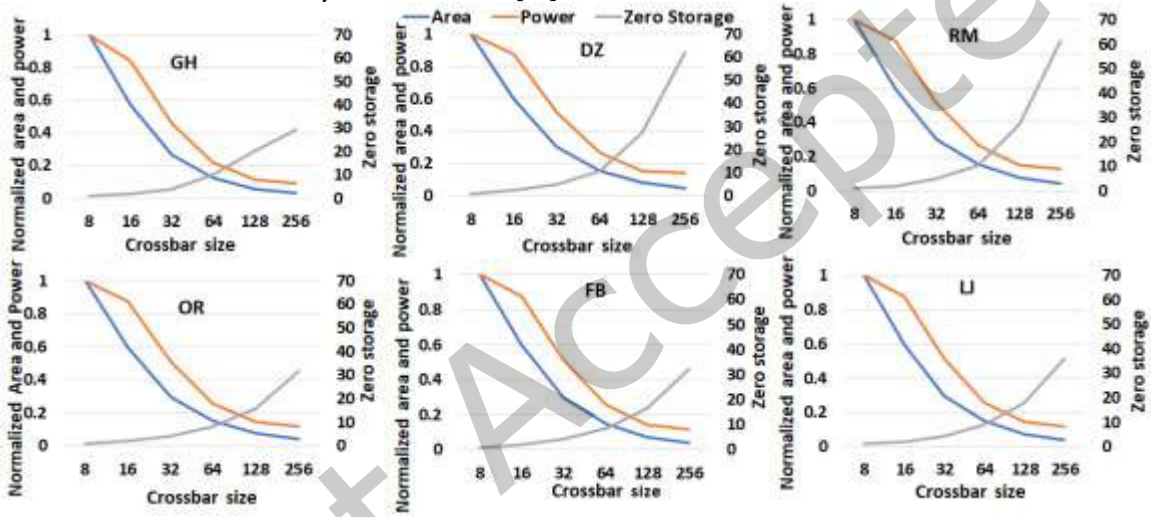

Fig. 7: Area-Power-Zero storage trade-offs for different crossbar configurations.

Table 2: Specifications of the proposed 3D manycore ReRAM architecture

No. of planar layers	4
No. of total PEs	1024
Area of each PE	0.37 mm^2
Area of each planar layer	10 x 10 mm^2
Clock frequency	2.5 GHz
Value of LRS	14.7 KΩ
Value of HRS	167 KΩ
Capacity of eDRAM	36 MB
ReRAM cell size	2-bits per cell

5.1 Selection of Crossbar Size

While storing the graph in crossbars, the adjacency matrix is decomposed into multiple non-overlapping $N \times N$ segments to map on to $N \times N$ shaped ReRAM crossbars. Current graph PIM architectures use relatively small crossbars (8×8) to reduce the storage of zeros [3]. However, this also negatively impacts the area and power as those terms are dominated by peripheral circuits [14]. To reduce area and power, as well as to minimize the overall number of required ReRAM crossbars, a larger size becomes more desirable, and experimentation is needed to evaluate this tradeoff. In other words, when considering total area, power and zero storage, it boils down to two choices: (a) smaller size implies a greater number of crossbars and fewer zeros, and (b) larger size implies fewer crossbars and more zeros. We conducted an experiment to evaluate this tradeoff with multiple inputs. Fig. 7 shows the normalized area, power and zero storage by varying the crossbar size from 8×8 to 256×256 for all the graph datasets considered in this work. We can see that the area and power continuously decrease with increasing crossbar size. However, beyond 128×128 both area and power show saturating trends, while the zero storage increases. Consequently, we select the 128×128 crossbar configuration as our default for all our experiments. This also implies setting the value of parameter parameter X in CARE reordering (Fig. 2) to 128.

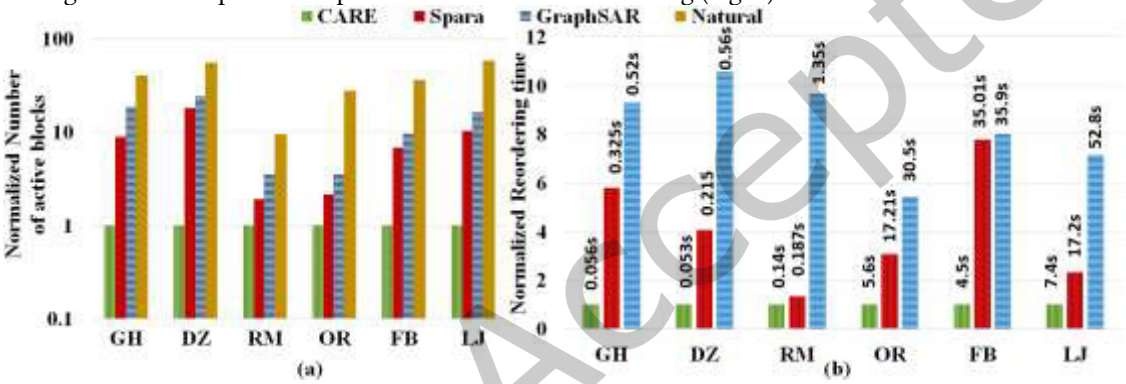

Fig. 8: (a) Normalized no. of active blocks (relative to CARE) for Spara, GraphSAR and natural, (b) Normalized reordering time of CARE, GraphSAR and Spara.

5.2 Performance of Reordering Scheme

Due to the sparsity in most real-world graph datasets, vertex reordering schemes help to reduce number of active blocks (i.e., matrix blocks with at least one non-zero element). Hence, we compare the number of active blocks generated using CARE to that of GraphSAR, Spara, and natural orderings. Fig. 8 (a) shows that all the ReRAM-based accelerators (CARE, GraphSAR and Spara) outperform natural. Furthermore, we observe that CARE significantly outperforms GraphSAR and Spara, by up to 23.8x and 18.3x respectively. Note that the storage improvements achieved expectedly vary with inputs as it is tied to the structural organization of the underlying graphs. For instance, CARE reduces the number of active blocks for RM by 9.5x compared to natural, whereas the gains are varying from 27.7x to 58.4x for the other social media datasets (e.g., GH, DZ, OR, FB and LJ) considered in this work. It should be noted that the reduction in the number of active blocks for RM is much lower than the other social media datasets with power-law characteristics. This is because the RM dataset's natural ordering already had a good locality to start with.

Hence, there is less room for improving the locality towards the goal of reducing the number of active blocks.

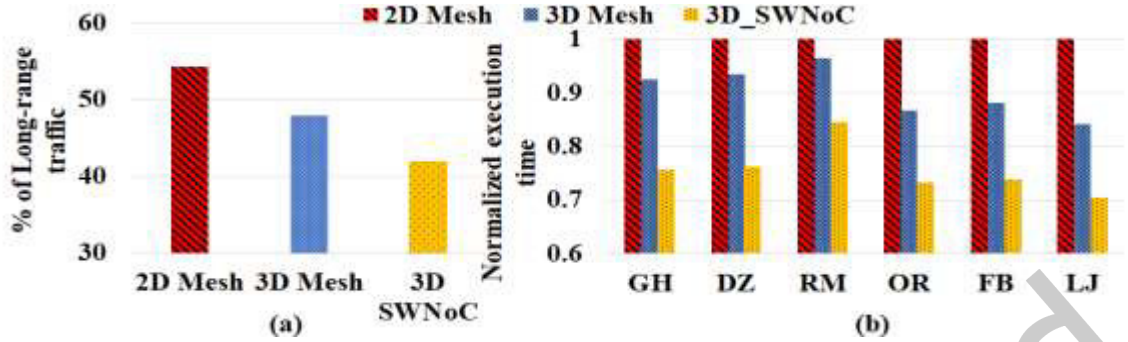

Fig. 9. (a) Percentage of total traffic that is long-range, under different NoC architectures for PageRank with GH, (b) Normalized execution time w.r.t 2D Mesh for PageRank.

Next, to assess the cost of the reordering time (preprocessing), we compared the reordering times for CARE, GraphSAR, and Spara (Fig. 8 (b)). In all the cases, the CARE reordering times were the smallest, with the other two schemes taking considerably longer (up to over 10x more in some cases). In summary, these experiments demonstrate that CARE reordering outperforms both GraphSAR and Spara on both zero storage as well as preprocessing cost.

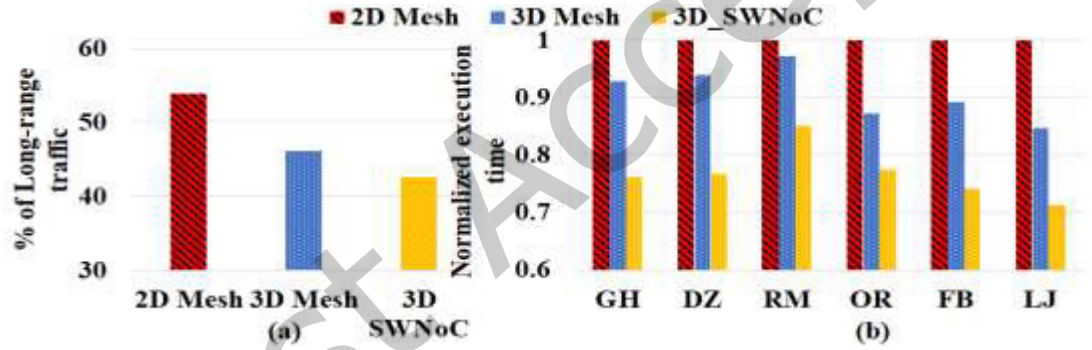

Fig. 10. (a) Percentage of total traffic that is long-range, under different NoC architectures for SSSP with GH, (b) Normalized execution time w.r.t 2D Mesh for SSSP.

5.3 Full System Performance Evaluation

Using CARE as the chosen reordering scheme, we analyzed hop count distribution of the proposed 3D SWNoC and compared that with 2D and 3D Mesh-based designs. For evaluation purposes, we refer to traffic with more than three 2D mesh hops as "long-range". Fig. 9(a), Fig. 10(a), and Fig. 11(a) show the percentage of long-range traffic for the 2D Mesh, 3D Mesh and 3D SWNoC. The traffic shown is for PageRank, SSSP and CC, respectively with GH input as an example. We observe similar characteristics for other datasets as well. The results show that long-range traffic for 3D SWNoC is 40% of total traffic, compared to 57% and 47.3% for 2D Mesh and 3D Mesh respectively. It should be noted that introduction of

the 3D structure helps in the reduction of long-range traffic due to the presence of shorter vertical links. As mentioned earlier the PEs that are separated by long distance on a 2D Mesh can be placed in different planar layers and connected through vertical links. As vertical links are smaller in length compared to their planar counterparts, they can establish one-hop data exchange. Hence, it is evident from Fig. 9(a), Fig. 10(a), and Fig. 11(a) that the traffic within three hops has been increased and the amount of long-range traffic has been reduced. Fig. 9(b), Fig. 10(b), and Fig. 11(b) show the normalized execution time for the three NoC architectures running PageRank, SSSP and CC. Fig. 9(b), Fig. 10(b), and Fig. 11(b) show that 3D SWNoC achieves the lowest execution time, and that improvement is also input-dependent. For instance, more savings are achieved on the power-law graphs with 3D SWNoC (26% to 32.5% savings) than with RM (17%). As mentioned above, the reduction in savings is least for RM because it is a road network with a uniform degree distribution.

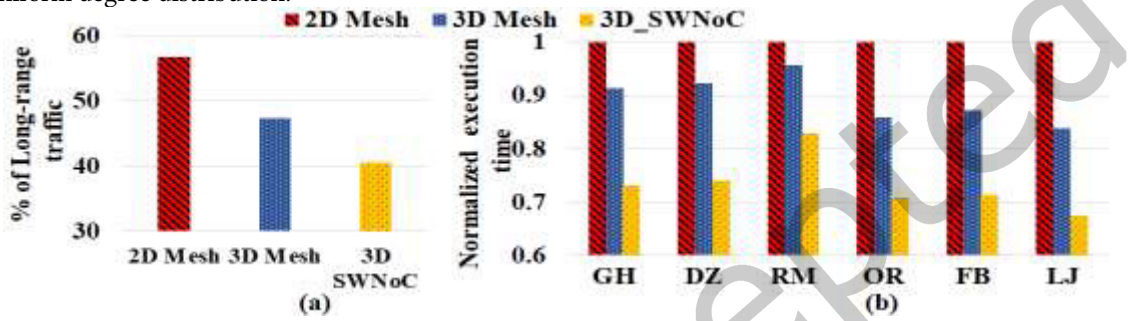

Fig. 11. (a) Percentage of total traffic that is long-range, under different NoC architectures for CC with GH, (b) Normalized execution time w.r.t 2D Mesh for CC.

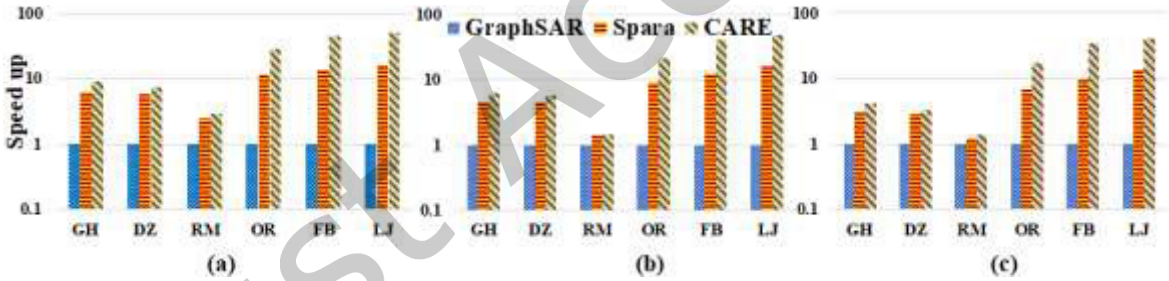

Fig. 12. Speedup of Spara and CARE in total execution time compared to GraphSAR for (a) PageRank, (b) SSSP and (c) CC.

In Fig. 12 and Fig. 13, we compare the speedups achieved in full-system execution time by the proposed manycore architecture using CARE and Spara against GraphSAR (baseline). To ensure a fair and direct comparison, we tested the graph applications with the datasets reordered by Spara and GraphSAR reordering schemes, on our proposed architecture. The full-system execution time includes the computation time, inter-PE communication time and the data transfer time from the host. It should be noted that we show preprocessing cost separately from this analysis because we execute the reordering process once offline and then the reordered graph is being used multiple times for various applications (e.g. SSSP, PageRank, CC). Hence, the preprocessing cost is being amortized over multiple runs of the

accelerator. The same strategy is adopted in other related works such as GraphSAR and Spara as well. For comparing these preprocessing times, we have reported the preprocessing timings, which are in seconds, for CARE along with GraphSAR and Spara in Fig. 8 (b). It should be noted that reordering not only helps to achieve speed up in graph computation on ReRAM, but also reduces the storage requirement. For example, the CARE reordering takes 89.2x more time than the processing time of PageRank with LJ on ReRAM-based manycore system. However, by paying that amount of preprocessing cost once, the proposed reordering scheme, CARE reduces on-chip storage requirement by 58.4x and achieves 84.2x speed up compared to that of natural (i.e., without any preprocessing). This speed up compared to natural is valid for executing the application on the ReRAM-based system as many times as the user requires. So, we pay the one-time offline processing cost to achieve this huge speed up multiple times.

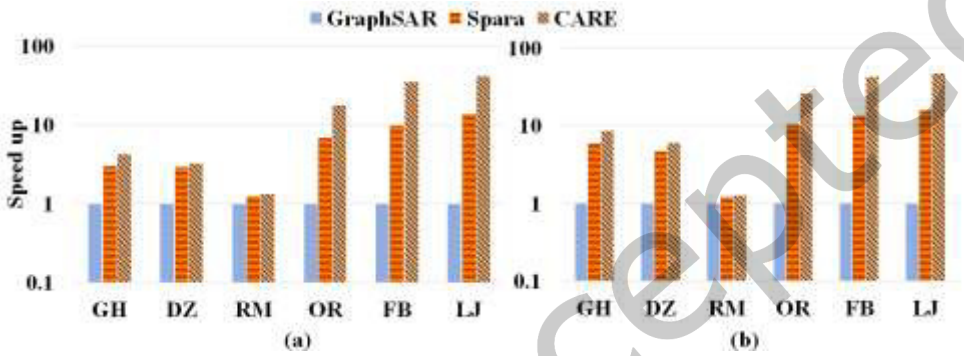

Fig. 13. Speedup of Spara and CARE in total execution time compared to GraphSAR for (a) BFS, (b) Triangle counting.

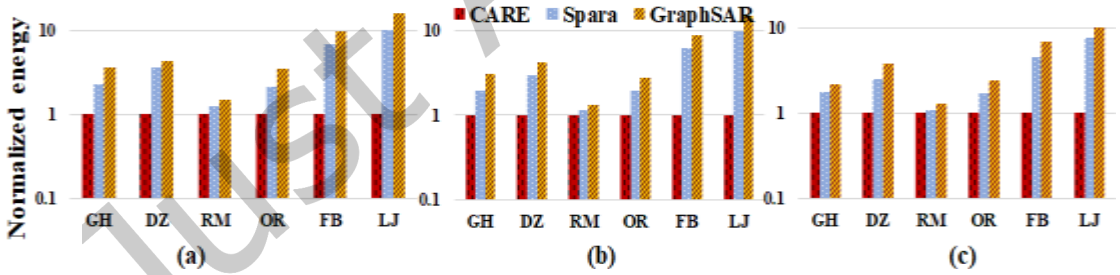

Fig. 14. Normalized energy of (a) PageRank, (b) SSSP and (c) CC for CARE, Spara and GraphSAR.

We can see from Fig. 12 and Fig. 13 that the SWNoC-enabled manycore architecture incorporating CARE achieves highest speedup (2.87x to 49.4x) compared to GraphSAR. This was clearly superior to the speedups achieved by Spara, which ranged from 1.3x to 16.3x. Another key observation is that the speedup gains realized by the proposed architecture with CARE is higher for the larger datasets (where it matters

more) – e.g., CARE achieves peak speedups (41x to 49.4x) for the largest input tested (LJ: 4.8M vertices and 68.9M edges). This is because CARE significantly reduces the number of active blocks and on-chip data movement, and thereby also reducing the time taken for data transfer, on-chip traffic, and computation. Furthermore, if we were to exclude the contribution from the inter-PE on-chip communication in total execution time, the speedup for CARE with respect to GraphSAR would reduce (ranging from 1.29x to 14x) relative to its counterpart with communication time (2.87x to 49.4x). Excluding on-chip communication compromises the achievable performance gain significantly, which reinforces the necessity of designing an efficient NoC.

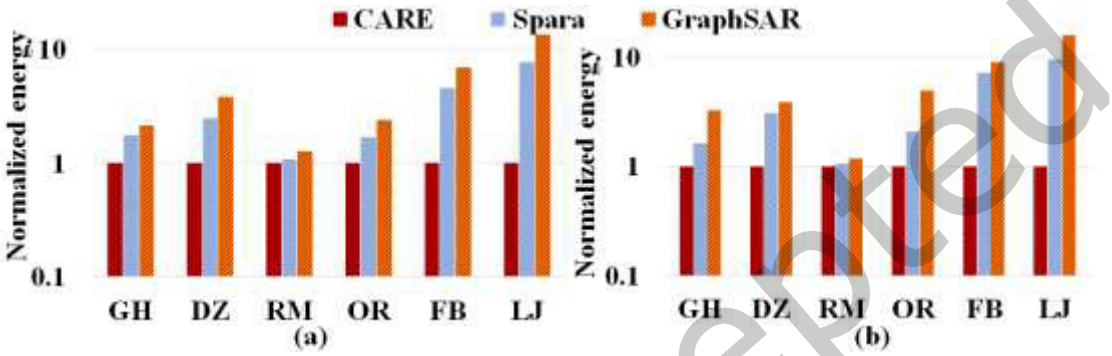

Fig. 15. Normalized energy of (a) BFS, (b) Triangle Counting for CARE, Spara and GraphSAR.

Figs. 14 (a), (b), and (c) illustrate the comparison of full-system energy consumption for CARE with respect to GraphSAR and Spara for PageRank, SSSP, and CC respectively. Similarly, Figs. 15 (a) and (b) show the comparison of full-system energy consumption for CARE with respect to GraphSAR and Spara for BFS and Triangle counting respectively. Fig. 14 and Fig. 15 show that CARE consumes 1.3x to 16.3x less energy compared to GraphSAR and 1.1x to 10.3x less energy compared to Spara for the inputs tested. As mentioned above, CARE is more efficient in reducing the number of active blocks and on-chip data movement. Also, due to high energy efficiency of ReRAM-based PEs, the peak temperature of the system remains below 85°C for all the configurations tested.

6 CONCLUSION

In this paper, we presented a 3D manycore ReRAM architecture well suited for accelerating graph applications. We introduce a crossbar-aware node reordering scheme called CARE that reduces the storage requirement and on-chip traffic volume on ReRAM. However, even after applying CARE, the contribution of inter-PE communication in total execution time for all the datasets is high (63.4% to 89.7%) except for RM (16.6%), which motivates the need of an efficient NoC for inter-PE communication. To reduce latency of communication on the chip, we presented an optimized 3D SWNoC architecture that reduces the network latency incurred by the long-range traffic in graph workloads. This combination of CARE reordering in software coupled with SWNoC yields drastic reductions in both runtime and energy cost,

also consistently outperforming two state-of-the-art ReRAM-based graph accelerators. We have also demonstrated that the speed up and energy improvement of our proposed architecture vary with the datasets. For social network inputs, vertex reordering and NoC architecture, both contribute noticeably to the overall performance and energy improvement whereas they are comparatively less for Road Map. This dichotomy in the results goes to show that the input characteristics could have a pronounced impact on what can be achieved through the combination of CARE reordering in software coupled with the proposed 3D SWNoC-based manycore ReRAM architecture.

ACKNOWLEDGMENTS

This work was supported by the US National Science Foundation (NSF) grant CCF-1815467.

REFERENCES

[1] K A. Kalyanaraman and P. Pande. 2019. A Brief Survey of Algorithms, Architectures, and Challenges toward Extreme-scale Graph Analytics. Proc. Design, Automation & Test in Europe Conference & Exhibition. 1307-1312.

[2] L. Zheng et al. 2020. Spara: An Energy-Efficient ReRAM-Based Accelerator for Sparse Graph Analytics Applications. Proc. IEEE International Parallel and Distributed Processing Symposium (IPDPS). 696-707.

[3] G. Dai et al. 2019. GraphSAR: a sparsity-aware processing-in-memory architecture for large-scale graph processing on ReRAMs. Proc. Asia and South Pacific Design Automation Conference. 120-126.

[4] A. A. Maashri et al. 2009. 3D GPU architecture using cache stacking: Performance, cost, power and thermal analysis. IEEE International Conference on Computer Design, Lake Tahoe, CA. 254-259.

[5] M. M. Ozdal et al. 2016. Energy Efficient Architecture for Graph Analytics Accelerators. Proc. International Symposium on Computer Architecture, pp. 166–177.

[6] L. Song et al. 2018. GraphR: Accelerating Graph Processing Using ReRAM. Proc. IEEE International Symposium on High Performance Computer Architecture. 531-543.

[7] T. Huang et al. 2018. HyVE: Hybrid vertex-edge memory hierarchy for energy-efficient graph processing. Proc. Design, Automation & Test in Europe Conference & Exhibition, pp. 973-978.

[8] K. Duraisamy et al. 2017. Accelerating graph community detection with approximate updates via an energy-efficient NoC. 54th ACM/IEEE Design Automation Conference (DAC). 1-6.

[9] R. Barik et al. 2020. Vertex Reordering for Real-World Graphs and Applications: An Empirical Evaluation. IEEE International Symposium on Workload Characterization (IISWC). 240-251.

[10] I. Safro, D. Ron and A. Brandt. 2009. Multilevel algorithms for linear ordering problems. ACM J. Exp. Algorithmics 13, Article 4.

[11] S. Das et al. 2017. Design-Space Exploration and Optimization of an Energy-Efficient and Reliable 3-D Small-World Network-on-Chip. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. vol. 36, no. 5, 719-732.

[12] B. K. Joardar et al. 2019. Learning-Based Application-Agnostic 3D NoC Design for Heterogeneous Manycore Systems. IEEE Transactions on Computers, vol. 68, no. 6, 852-866.

[13] S. Bandyopadhyay et al. 2008. A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA. IEEE Transactions on Evolutionary Computation. vol. 12, no. 3, 269-28.

[14] A. Shafiee et al. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). 14-2.

[15] A. I. Arka et al. 2021. ReGraphX: NoC-enabled 3D Heterogeneous ReRAM Architecture for Training Graph Neural Networks. Design, Automation & Test in Europe Conference & Exhibition (DATE). 1667-1672.

[16] N. Jiang et al. 2013. A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator. In Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software. 86-96.

[17] Xiangyu Dong et al. 2014. Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory. In Emerging Memory Technologies. Springer. 15–50.

[18] http://snap.stanford.edu/; data accessed: September 2021

[19] http://networkrepository.com/; data accessed: September 2021

[20] A . Sridhar et al. 2010. 3D-ICE: Fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling. Proc. IEEE/ACM Int. Conf. Comput.-Aided Des., 2010, pp. 463–470.

[21] W. Lee et al., “Varistor-type bidirectional switch (JMAX > 107 A/cm2, selectivity~104) for 3D bipolar resistive memory arrays,” in Proc. Symp. VLSI Technol., Jun. 2012, pp. 37–38

[22] L. Zhang et al. 2015. On the Optimal ON/OFF Resistance Ratio for Resistive Switching Element in One-Selector One-Resistor Crosspoint Arrays. IEEE Electron Device Letters, vol. 36, no. 6, pp. 570-572.

[23] T. Schultz, R. Jha, M. Casto and B. Dupaix. 2020. Vulnerabilities and Reliability of ReRAM Based PUFs and Memory Logic. IEEE Transactions on Reliability, vol. 69, no. 2, pp. 690-698.

[24] D. Choudhury, A. S. Rajam, A. Kalyanaraman and P. Pande. 2022. High-Performance and Energy-Efficient 3D Manycore GPU Architecture for Accelerating Graph Analytics. ACM Journal on Emerging Technologies in Computing Systems. Volume 18, Issue 1, January 2022, Article No: 18, pp 1–19.

[25] D. Choudhury, R. Barik, A. S. Rajam, A. Kalyanaraman and P. Pande. 2021. Software/Hardware Co-design of 3D NoC-based GPU Architectures for Accelerated Graph Computations. ACM Transactions on Design Automation of Electronic Systems.