PIM-GraphSCC: PIM-based Graph Processing using Graph's Community Structures

Newton, *Student Member, IEEE,* Virendra Singh, *Member, IEEE,* and Trevor E. Carlson, *Senior Member, IEEE*

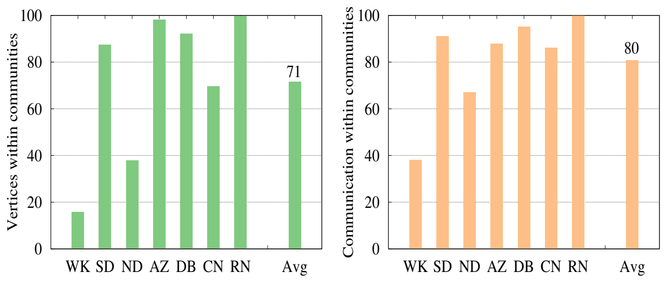
Abstract—Graphs are used to store relationships on a variety of topics, such as road map data and social media connections. Processing this data allows one to uncover insights from its structure. However, while analyzing graphs with traditional processors, the graph connectivity can result in irregular memory access patterns and thus poor data locality that can result in low performance. Processing-in-Memory (PIM) is an attractive alternative for graph processing, as it can reduce data movement by bringing the computation closer to the data itself. While PIM-based techniques have been shown to improve graph processing performance, there is still room for improvement, as critical bottlenecks exist when connecting multiple PIM-based accelerators into larger clusters. Although a number of recent proposals have aimed to reduce inter-accelerator data movement, their techniques have generally overlooked the potential to optimize how the graph's connectivity can lead to a more efficient hardware mapping. In fact, many real-world graphs have a small percentage of high-degree nodes that connect widely to a large number of other nodes. By clustering these nodes into communities, one can more efficiently map them to hardware, minimizing expensive inter-accelerator communication, a key performance bottleneck in these accelerators. To capitalize on this observation, we propose PIM-GraphSCC, the first PIM-based graph processor that exploits a graph's connectivity to significantly reduce communication over critical resources: the inter-accelerator links. By partitioning graphs into communities, PIM-GraphSCC provides a community-aware graph partitioning scheme that reduces inter-accelerator data movement by up to 93% compared to modern graph processing schemes.

Index Terms—Processing-in-Memory, 3D Stacked Memory, HMC, Graph Processing, Strongly-connected components.

1 Introduction

In recent years, graph processing has received significant interest as it forms the basis of many commonly used techniques to analyze relational data. However, irregular memory accesses and the low compute-to-communication ratio that exists in many graph algorithms [1], has limited their performance on CPU-centric architectures where data locality is the key for both performance and efficiency. As an alternative, memory-centric architectures [2] have been developed in an attempt to overcome these limitations. Processing-in-Memory (PIM)-based architectures enable fast delivery of data because of the proximity (and low latency) of compute to memory (unlike traditional architectures which rely on caching and locality to achieve the same goal). Previously proposed techniques [2], [3] follow a vertex-centric approach and use a sequential graph partitioning technique to distribute a graph among multiple memory structures (like High-Bandwidth Memory (HBM) or Hybrid Memory Cube (HMC)). However, such an approach generates a significant amount of inter-accelerator communication, stressing the limited inter-accelerator bandwidth, a critical bottleneck for performance [3].

In an attempt to improve performance in spite of the limited inter-accelerator bandwidth available, prior solutions [2], [3] have proposed techniques to reduce inter-accelerator communication. While effective, these solutions take a vertex-centric approach to distribute each vertex independently. The key insight of our work is that, while these techniques are effective, many real-world graphs, like social networking, road networks, etc., do not behave like synthetically generated random graphs. Real-world graphs, instead, have a strong community structure in the underlying data [1], that operate according to a power law [4],



(a) Vertices (b) Communication

Fig. 1: Percentage of vertices in communities and communication among them.

where *clusters of vertices* naturally form throughout a graph. Naive sequential placement of vertices, as used in prior solutions, will not allow the hardware to take advantage of this natural community structure. However, by viewing a graph as a collection of clusters, instead of an unorganized collection of vertices, it is possible to reduce inter-accelerator communication (the key bottleneck in today's PIM designs) to improve performance and efficiency.

In this work, we demonstrate that PIM-based graph processing systems can benefit significantly from the adoption of a community-centric paradigm. Our study of seven graphs (see Section 4.1) demonstrates that: (1) **on an average more than 70% of vertices are part of various communities present in the graph** (Figure 1a), and (2) **communication among these communities account for 80% of the total communication** (Figure 1b). Hence, our proposed solution, *PIM-GraphSCC*, utilizes these two key observations to build a community-aware approach for distributing a graph on the PIM's memory system, thus reducing the inter-accelerator data movement by up to 93%.

2 Background and Motivation

2.1 Stacked Memory Architecture Designs

Hybrid Memory Cubes, or HMCs, are 3D stacked memories that can be easily extended with PIM capabilities. A single

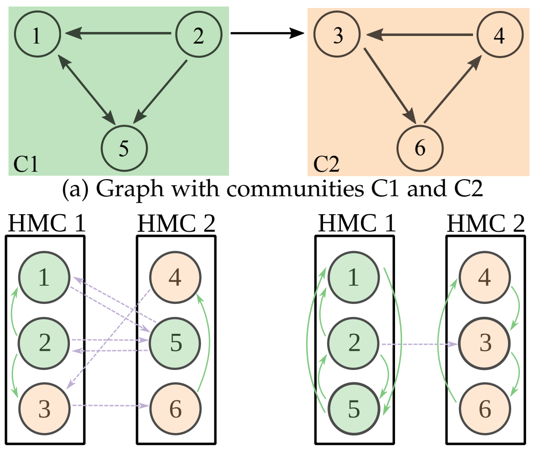
• *Authors are with Indian Institute of Technology Bombay and National University of Singapore.*
E-mail: {newton, tcarlson}@comp.nus.edu.sg, viren@ee.iitb.ac.in

HMC consists of a logic layer at the base, with memory controllers and processing capabilities, and up to 8 stacked memory layers on top organized into 32 vertical skyscraper-like vaults [5]. HBM, or High-Bandwidth Memory, an alternative high-density memory technology, also shares the same principle of 3D stacking of memory layers. Such devices like HMCs provide two types of connectivity: 1) *Internal (or intra-cube) connections* between the logic and the memory layers; and 2) *External (or inter-cube) connections* between one HMC and another, or to the host processor. An HMC offers an internal bandwidth of 320 GB/s along with an external bandwidth of 480 GBs/s [5]. The logic layer can support general purpose cores to enable PIM functionality.

Apart from PIM capability, HMC-based systems also have *memory-capacity-proportional* bandwidth, enabling their performance to scale with an increase in the memory capacity (through the use of additional, interconnected, HMCs), unlike conventional systems. However, HMC-based systems have their own set of challenges. The first challenge is to partition a graph in a balanced way for an HMC's memory system, an NP-complete problem [6]. The second challenge is to organize a graph's data to fully utilize the HMC's internal bandwidth, and minimize external communication. Prior solutions have shown that a system's performance can improve with a reduction in external communication (typically through the use of better data organization). Tesseract [2] and GraphP [3], for example, have used METIS-generated balanced graph partitioning and a source-cut technique, respectively, to reduce external communication. However, there is still scope for a further reduction in communication as prior works have not considered a graph's community structure while partitioning it.

Partitioning a graph among HMCs generates edge-cuts between these HMCs. Each edge-cut result in the generation at least one inter-cube communication. Thus, reducing edge-cuts reduces inter-cube or external communication. The focus of this work is to take advantage of the natural communities in graphs (referred to as Strongly Connected Components (SCCs)) to reduce edge-cuts. In our experiments, we observed that on average 80% of vertex communication occurs *within the communities* of the graph (See Figure 1b). Thus, if the partitioning strategy partitions the graph such that all the vertices in a community are mapped to the same HMC, the vertex communication within these vertices will be *intra*-cube communication, thus utilizing HMC's large internal bandwidth. In contrast, if the strategy is community-oblivious, it can map the vertices within a community to different HMCs, incurring expensive *inter*-cube communication and thus stressing the limited external bandwidth. Thus, this work proposes a **Community-Aware Graph Partitioning, or CAGP strategy**, to harness the benefit of communities.

As an example (with details shown in Figure 2), we can illustrate how CAGP has the potential to reduce inter-cube communication, the main bottleneck for PIM architectures handling graph-workloads. An example graph with 6 vertices and arrows depicting the direction of vertex communication is shown in Figure 2a. If these vertices are sequentially distributed among the 2 HMCs (as is typically done in previous works to distribute the workload and bandwidth



(a) Graph with communities C1 and C2

(b) Sequential partitioning (c) CAGP

Fig. 2: Intra- and Inter-cube communication in *Sequential* and *Community-aware* graph partitioning (CAGP) strategies. Intra-cube communication have increased from 3 to 8, and inter-cube communication have reduced from 6 to 1 in CAGP in comparison to sequential partitioning.

utilization), the resulting assignment (shown in Figure 2b) demonstrates the potential for a large number of inter-HMC (external) communication requests. In contrast, CAGP considers the communities C1 and C2 present in the graph and partitions the communities accordingly (Figure 2c). Due to this mapping, the resulting inter-cube communication (depending on the algorithm used) can be $1/6^{th}$ that of the sequential partitioning case. Apart from reducing inter-cube communication, CAGP has also increased the intra-cube communication from 3 to 8, thus utilizing the large internal bandwidth. Hence, CAGP benefits in effective utilization of both internal and external connectivity. In the following sections of this work, we will present an argument for why CAGP is effective at reducing inter-cube communication.

2.2 Different Strategies of Graph Partitioning

To understand the effectiveness of CAGP in partitioning real-world graphs, we conducted a comparison study of two graph partitioning strategies to understand CAGP's impact on the *inter-cube* communication, a key bottleneck in PIM-based graph processing. The first strategy, *source-cut partitioning* [3], distributes a graph among multiple HMCs using sequential partitioning (e.g. vertexID % NUM_HMCs). It then follows the *source-cut* technique to assign a vertex and all its incoming edges on the same HMC. However, as it follows the vertex-centric approach, it is oblivious to

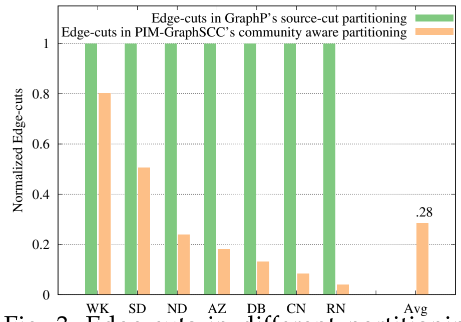


Fig. 3: Edge-cuts in different partitioning strategies

the clusters of vertices naturally formed in the graph and hence misses the opportunity to further reduce inter-cube communication. Our proposed strategy, *community-aware partitioning*, (a) first identifies various communities, then (b) assigns communities to HMCs such that an entire community is assigned to the same HMC (if possible), (c) and then follows source-cut replication to further reduce inter-cube communication. As shown in Figure 3, *community-aware partitioning* is successful in reducing edge-cuts by more than 70% for CAGP in comparison to *source-cut partitioning*. As mentioned previously, reducing edge-cuts leads to a reduction in inter-cube communication.

Our strategy of CAGP is generic and can also be used for graph processing on other 3D-stacked memory interfaces (e.g. HBM-based PIM systems). In the case of HBM, CAGP can be used to map the graph on different DRAM layers and independent channels of HBMs, for efficient usage of limited channel bandwidth.

3 PIM-GraphSCC Architecture

Our PIM-GraphSCC architecture consists of a cluster of 16 HMCs interconnected by the HMC's external links. In this section, we first describe how communities are detected in a graph and then explain how CAGP assigns the vertices on different HMCs.

3.1 Detecting Communities

One of the key features of PIM-GraphSCC is its potential to utilize the graph's community structures. Figure 2a shows a graph G represented as a set of vertices V and a set of edges E, with $G = (V, E)$. Prior works [2], [3] have used this straight-forward representation of the graph for its partitioning. However, as also shown in Figure 2a, the graph G also consists of communities C1, and C2. Eight out of nine edges of G are *within* these two communities. Hence, it is important to detect such communities to further reduce inter-cube communication. The Kosaraju-Sharir (KS) algorithm [7], which we use in this work, is used to detect communities in a graph with linear time-complexity $O(V + E)$. By definition, two vertices (a, b) of a graph are said to be strongly connected, $(a \sim b)$, if there is a path from $a \rightarrow b$, and also, a path from $b \rightarrow a$. In the context of graphs, it means that two vertices are part of a community if they mutually communicate with each other. Each community is a set a vertices and is *mutually exclusive* from other communities. We detect such communities during a preprocessing stage.

Graph Preprocessing: Preprocessing is common for most graph processing systems. In PIM-GraphSCC, three steps are performed in the preprocessing stage. (1) The graph is converted from an edge-list text format to a binary format, e.g., CSR (Compressed Sparse Row). (2) The CSR format is processed by the KS algorithm to detect communities present in the graph. (3) The graph is partitioned among HMCs. Steps (1) and (3) are also performed in prior works, and only the step (2) is specific to PIM-GraphSCC. While partitioning the graph among HMCs, PIM-GraphSCC attempts to assign all the vertices of a single community on a single cube. However, as the graph partitions are balanced to evenly distribute the workload, as in [3], this constraint can result in assigning vertices of a large community to multiple HMCs, thus generating edge-cuts. While detecting communities can take a bit longer (up to 3 seconds for the largest graph i.e RoadNet-CA in comparison to 1 seconds by [3]), we see that this time, while short, is amortized over multiple runs of the graph and is a *one-time cost*. As per our analysis, the preprocessing time for the graphs under study is less than 1.5% of the simulated runtime and is only 0.06% of the runtime for RoadNet-CA, the largest graph.

Preprocessing a graph using the KS algorithm partitions it into multiple SCCs (i.e., $SCC_{0...i} : 0 \leq i \leq N$) and *non-SCC-vertices* (the vertices that are *not* part of any SCC). As one of the goals of partitioning (as in [3]) is even distribution of the workload, we first find J, the maximum number of vertices that can be mapped on an HMC for balanced distribution. Next, we iterate over all of the SCCs. If the number of vertices in the given SCC is $\leq J$, the SCC is mapped on a single available HMC. Otherwise, the SCC is evenly distributed on K ($\leq T$, the total number of HMCs) available HMCs. After mapping all of the SCCs, non-SCC-vertices are sequentially mapped on the remaining available HMCs. As per our mapping algorithm, if the number of vertices mapped on an HMC is $\leq J$, the remaining SCCs or non-SCC-vertices can be mapped on the same HMC until the number of vertices mapped on the HMC reaches J.

4 Simulation Framework and Evaluation

4.1 Simulation Methodology

Our architecture consists of 16 HMCs (each with 32 vaults and each vault having an in-order core) connected in a 2-D mesh topology. Each HMC has four quadrants, and the vaults within each quadrant are connected in a star topology. We model a closed page policy [5], with a static DRAM access latency of 33.6 ns (t_{RAS} = 22.4 ns, t_{RP} = 11.2 ns). Each serial link has an 8-cycle latency, including 3.2 ns for SerDes [8]. We assume a 3 cycle router latency and 1 cycle wire communication delay [8]. We model this configuration in *Booksim* [9] with 4 virtual channels and a 32-entry packet buffer size. We studied four widely used graph algorithms (their implementation taken from [10]): PageRank (PR), Single-Source Shortest Path (SP), Vertex Cover (VC), and Breadth-First Search (BF).

Performance Evaluation: We evaluate the cycles spent (to measure speed-up) and the number of intra-cube and inter-cube messages sent (to measure communication reduction) while running the graph algorithm with different graph inputs mentioned in Table 1. Due to the long simulation times, we simulate a single iteration of the application and record the run time in cycles, similar to [2].

TABLE 1: Graph Dataset

Graph	#V	#E
Wiki-Vote (WK)	7.1 K	0.1 M
Slashdot (SD)	82 K	0.9 M
Notre Dame (ND)	326 K	1.5 M
Amazon (AZ)	262 K	1.2 M
DBLP (DB)	326 K	1.6 M
CNR (CN)	326 K	3.2 M
RoadNet-CA (RN)	1965 K	2.7 M

We compare our results to GraphP, a state-of-the-art graph partitioning strategy for PIM-based graph processing system. While a newer work, GraphQ [11] outperforms GraphP (through its techniques of batched and overlapped inter-cube communication), those enhancements are orthogonal to GraphP's source-cut partitioning technique. As GraphQ still uses sequential partitioning, it is complementary to our CAGP partitioning technique.

4.2 Evaluation

Figure 4a shows that PIM-GraphSCC outperforms GraphP by 2.7× on an average (up to 11×), and Figure 4b

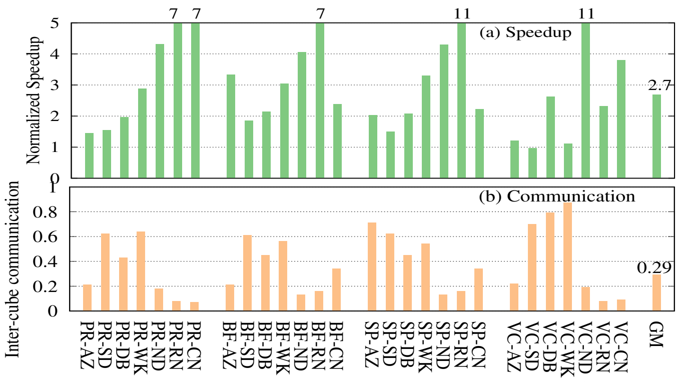


Fig. 4: *Speedup* (reduction in run time) and *Inter-cube communication* in PIM-GraphSCC normalized to GraphP

shows that, on average, inter-cube communication for PIM-GraphSCC is 71% (up to 93%) lower than GraphP. Figure 4a also shows that there is a performance trend. As explained later in the section, we find that the performance benefits for PIM-GraphSCC follow the reduction in bandwidth utilization of the inter-cube links. The trend is more evident from the performance of graphs for the PR application. PR is an *all-active* application, i.e., all of its vertices are active in each iteration, whereas BF, and SS enable only partial vertices and edges as graph traversal ends at the searched vertex.

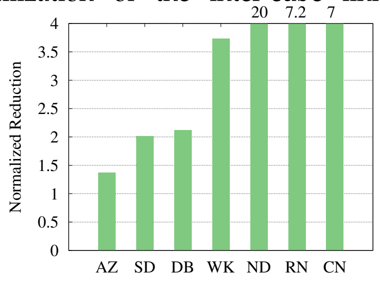


Fig. 5: Normalized reduction in VC buffers getting full.

Using CAGP, SCCs are mapped to reduce inter-cube communication. Due to limited inter-cube bandwidth, there remains significant network congestion. As the reduction in the number of edge-cuts is specific to a graph (and independent of the algorithm), it alone can not explain the performance benefits completely. We observed that the interplay between two factors, namely *inter-cube edge cut reduction* and *inter-cube communication reduction*, determines the resultant performance for a graph. In order to study reduced network congestion in PIM-GraphSCC, we studied the occupancy of virtual channel buffers in the network (See Figure 5). We found that the limited inter-cube bandwidth (manifested as full packet buffers, See Figure 5) remained a dominant factor for graphs AZ, SD, DB, and WK. The graphs AZ and DB did not achieve their potential for higher performance related to their edge-cut reduction because of the high inter-cube bandwidth requirements. This could indicate that a NoC design to tolerate network congestion due to the limited inter-cube bandwidth of links at the SCC-mapped HMCs might alleviate such performance degradation. Overall, the CAGP technique is able to reduce more than 90% edge-cuts in these two graphs.

5 Related Work

PowerGraph [12] exploits the structure of graph programs for distributed graph processing. Their graph partitioning scheme is based on reducing vertex-cuts, while this work focuses on reducing edge-cuts. Gemini [13] attempts to improve the distributed graph processing performance by using locality-aware chunk-based graph partitioning; we use community-aware partitioning in this work. Tesseract [2] proposed the usage of PIM-based systems for graph processing and used message-passing for reducing communication among HMCs. However, since it is oblivious to the data organization of the graph, it generates a significant amount of messages that puts pressure on the the HMC's limited link bandwidth. GraphP [3] improves upon Tesseract by specifically considering the graph's data organization by using a source-cut technique. However, GraphP is oblivious to community structures present in the graph, and thus, it also has high inter-cube communication. HATS [1] is among the first proposals to use information in the graph's communities. However, it uses the community structure in the graph to improve data locality in the cache memories. In contrast, this work uses the information of communities to reduce the edge-cuts in the graph. PIM-GraphSCC shows that the community structures present in the graph can significantly reduce inter-cube communication.

6 Conclusion

PIM-based graph processing has opened new opportunities for high performance graph processing. However, inter-cube communication is very expensive in such architectures due to limited external bandwidth. Hence, reducing inter-cube communication is performance-critical. We demonstrate that our work has a potential to drastically reduce such communication by using *Community Aware Graph Partitioning* (CAGP). Our proposed solution, *PIM-GraphSSC*, capitalizes on CAGP to improve performance by up to 11× in comparison to the recent PIM-based graph processing system, GraphP. It also reduces inter-cube communication, the critical bottleneck in PIM-based systems, by up to 93%.

References

[1] A. Mukkara, N. Beckmann, M. Abeydeera, X. Ma, and D. Sanchez, "Exploiting locality in graph analytics through hardware-accelerated traversal scheduling," in *MICRO*, 2018.
[2] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," *ISCA*, 2016.
[3] M. Zhang, Y. Zhuo, C. Wang, M. Gao, Y. Wu, K. Chen, C. Kozyrakis, and X. Qian, "GraphP: Reducing communication for PIM-based graph processing with efficient data partition," in *HPCA*, 2018.
[4] P. Boldi, M. Rosa, M. Santini, and S. Vigna, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," in *World Wide Web*, 2011.
[5] Hybrid Memory Cube Consortium, "Hybrid Memory Cube Specification 2.1," 2015.
[6] K. Andreev and H. Racke, "Balanced graph partitioning," *Theory of Computing Systems*, vol. 39, no. 6, pp. 929–939, 2006.
[7] M. Sharir, "A strong-connectivity algorithm and its applications in data flow analysis," *Computers & Mathematics with Applications*, vol. 7, no. 1, pp. 67–72, 1981.
[8] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *PACT*, 2015.
[9] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. J. Dally, "Booksim 2.0 user's guide," *Standford University*, 2010.
[10] GeeksForGeeks, https://www.geeksforgeeks.org/.
[11] Y. Zhuo, C. Wang, M. Zhang, R. Wang, D. Niu, Y. Wang, and X. Qian, "GraphQ: Scalable PIM-based graph processing," in *MICRO*, 2019.
[12] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *OSDI*, 2012.
[13] X. Zhu, W. Chen, W. Zheng, and X. Ma, "Gemini: A computation-centric distributed graph processing system," in *OSDI*, 2016.