Journal of Electrical and Computer Engineering Innovations
(JECEI)
Journal homepage: http://www.jecei.sru.ac.ir

Research paper

NodeFetch: High Performance Graph Processing Using Processing in Memory

M.A. Mosayebi, M. Dehyadegari*

Department of Computer Systems Architecture, Faculty of Computer Engineering, K. N. Toosi University of Technology, Tehran, Iran.

Article Info

Article History:
Received 24 April 2020
Reviewed 24 June 2020
Revised 14 September 2020
Accepted 15 November 2020

Keywords:
Graph processing
Hybrid Memory Cube (HMC)
Processing in memory
Hardware Accelerator

*Corresponding Author's Email Address:
dehyadegari@kntu.ac.ir

Abstract

Background and Objectives: Graph processing is increasingly gaining attention during era of big data. However graph processing applications are highly memory intensive due to nature of graphs. Processing-in-memory (PIM) is an old idea which revisited recently with the advent of technology specifically the ability to manufacture 3D stacked chipsets. PIM puts forward to enrich memory units with computational capabilities to reduce the cost of data movement between processor and memory system.
This approach seems to be a way of dealing with large-scale graph processing, considering recent advances in the field.
Methods: This paper explores real-world PIM technology to improve graph processing efficiency by reducing irregular access patterns and improving temporal locality using HMC.
We propose NodeFetch, a new method to access nodes and their neighbors while processing a graph by adding a new command to HMC system.
Results: Results of our simulation on a set of real-world graphs point out that the proposed idea can achieve 3.3x speed up in average and 69% reduction of energy consumption over the baseline PIM architecture which is HMC.
Conclusion: Most of the techniques in the field of processing-in-memory, hire methods to reduce movement of data between processor and memory. This paper proposes a method to reduce graph processing execution time and energy consumption by reducing cache misses while processing a graph.

Introduction

Graph processing is increasingly gaining attention during era of big data. While graph algorithms are varied, but most of them have expensive memory accesses. There are an important reason which makes graph algorithms memory intensive. Graph structure is irregular and therefore access to nodes are irregular and difficult to predict which leads to poor temporal locality [1]. So Numerous techniques for large-scale graph processing have been proposed in the literature that address the memory bandwidth and data movement problems [2]-[4]. Hardware accelerators have proven successful in achieving significant speedup and energy efficiency in comparison to general purpose processors [5]-[6]. Many graph accelerators hire different techniques to decrease movement of data between memory and the host processor to achieve noticeable performance improvements [5]-[10]. Processing in memory (PIM) is an old idea which introduced few decades ago by academia. Back then it could not gain enough attention due to several issues such as complexity and fabrication technology. Recently PIM has been revisited by both industry and academia. Micron proposed Hybrid memory cube (HMC) as one solution to hire PIM in real word.

HMC is one of the most promising DRAM systems which is a true 3D stacked DRAM. HMC contains of multiple DRAM dies on the top of a logic die. Several reaches have been made based on HMC to improve graph computation performance. For example, HMC-MAC proposed a PIM architecture based on the hybrid memory cube (HMC) that adds a MAC operation to HMC to accelerate graph and NN applications. Enhanced Tesseract is another example which propose an idea for large scale graph processing based on HMC. In this paper, we explore real-world PIM technology to improve graph processing efficiency by reducing irregular access patterns and improving temporal locality. We are proposing NodeFetch, a new method to access node neighbours while processing a graph. Our research follows the HMC 2.1 specification [11].

Background

This section contains the background knowledge on PIM accelerators and graph processing.

A. Graph Processing

Based on nature of a graph, graph processing applications suffer from several issues such as random access patterns, poor locality and unbalanced workloads [1]. Two main issues of graph processing are:

- Neighbours of a given node might be somewhere else in memory causing cache miss.
- Accessing nodes are unpredictable and depends of the shape of input graph and graph algorithm.

In this Paper we are aiming to clearly address these issues and propose a novel architecture to properly fix them.

B. Processing in Memory

Processing in memory (PIM) is an old idea. With the advent of big data computing and also recent advances in memory technologies, such as emerging nonvolatile memories, die stacking, and high-bandwidth memory interfacing, PIM has been revisited recently by both academia and industry.

Processing-in-memory proposes to move computational components to the memory units to alleviate the high cost of data movement in big data processing.

Hybrid memory cube (HMC) proposed and manufactured by Mircron company in 2011 trying to reach high memory bandwidth using PIM idea. They put several 3D-stacked DRAM dies on a logic layer to increase available bandwidth while providing high performance near memory processing. Using trough silicon Vias (TSVs) inside a memory cube, several DRAM layers are connected to the logic layer at the bottom of the cube.

A single memory cube consists of 32 vertical slices Based on Hybrid Memory Cube Specification 2.1 [11]. Each of these slices called a vault. Each vault benefits from 10GBps of memory bandwidth, therefore a single cube has total of 320GBps of bandwidth.

Motivation and Innovation

Graph processing suffers from random access patterns, poor locality and unbalanced workloads [1]. Therefore accessing each neighbour may lead to a cache miss. Upon a cache miss, processor should bring necessary data block from memory into each level of the cache and then use that data to continue the application process, which degrades system performance. This can happen to each and every neighbor of each node of graph without a proper prediction. Repeating cache misses makes the cache useless. Several techniques has been hired to solve this problem such as graph mapping and custom prefetchers. These methods try to solve proposed issue indirectly and could gain noticeable performance increase in some cases. But if one solution can resolve the problem directly, huge performance increase can be achieved. One solution that may come to mind is to use GPUs and many cores such as [12], to tackles these issues. GPUs are being used to accelerate various applications through parallelism such as Neural network algorithms. As said earlier, the main challenge of graph processing is their random memory accesses and irregularity of their algorithms.

One possible way to accelerate graph processing is to hire GPUs.

GPU is a highly structured SIMT architecture and it is not suitable for graph applications [1].

The performance of graph processing on GPU is still limited by memory latency despite of many efforts spent of accelerating graph applications using them [1]. The whole concept of processing in memory (PIM) is to overcome an important issue which is memory bandwidth wall. On one hand, GPU can't be used properly to generally accelerate graph processing due to memory latency.

On the other processing in memory is a solution to overcome memory latency and memory bandwidth wall. Therefore processing in memory is selected as a baseline technique to directly solve issues tied with graph processing.

Several other researches in the field of processing in memory has been done to use internal bandwidth of memory to help with processing graphs. Some of them added logic into or near conventional memory units such as GraphR [13] and which used ReRAM or Graphi-cionado [15], But others used HMC to achieve their desired goal. Graph processing acceleration can be done by moving computations into the logic layer of HMC to exploit High in-memory bandwidth. Among those who used HMC as their baseline, there are two main categories.

- Hiring Network of several memory cubes with a specific topology [14]-[16]. Tesseract [17] achieved a significant performance improvement. Several researches tried to improve performance in a Tesseract-based system such as GraphH [9], GraphP [8] and Enhanced Tesseract.
- Implementing on a single cube which is extendable to be used in any network of multiple cubes. For example HMC-MAC [18] tried to add a MAC operation to a HMC device.

In-memory graph processing have to address several issues such as random access patterns, poor locality and unbalanced workloads. This paper proposes a novel PIM accelerator called NodeFetch based on a single HMC to accelerate graph processing by reducing random accesses and poor locality.

Related Work

Sseveral recent related works and ideas in graph processing acceleration with he help of the HMC have been reviewed in this section. Various techniques have been proposed to accelerate graph processing such as Graphicionado [5], Tesseract [17], GraphH [9], GraphP [8], Enhanced Tesseract [7] and Centaur [19].

Graphicionado [5] accelerates graph processing by the use of parallelism and. They proposed a domain-specific hardware accelerator. HMC is not being used in their sub-system. They could achieve a better performance than a state-of-the-art software graph processing framework being executed on a 16-core Haswell Xeon processor. Tesseract [17] is a large-scale graph processing architecture which uses a network of modified HMCs towards graph processing acceleration. Altough They could gain a remarkable performance, but the main problem is very long waiting times in processors. Apparantly Tesseract spends 59% of execution time waiting for synchronization barriers [17]. Several researches tried to improve performance in a Tesseract-based system such as GraphH [9], GraphP [8] and Enhanced Tesseract [7].

GraphP considers data organization as a first-order design consideration to improve Tesseract-base system. Therefore they could provide a better performance in comparison to Tesseract by designing a hardware/software co-designed graph processing. GraphH on the other hand is a PIM architecture for graph processing on the Hybrid Memory Cube array. It integrates SRAM-based on-chip vertex buffers to eliminate local bandwidth degradation.

Enhanced Tesseract [7] targets the main problem of Tesseract which is low utilization due to synchronization barriers. They modified each HMC device in a way to manage and accelerate message queues and could reduce execution time by 40% in average.

Centaur [19] tries to divide graph processes into two parts. One part that can be processed in off-chip memory and the other part which should be processed in on-chip memory to accelerate graph processing. Processes related to each vertex can be done in an on-chip of off-chip memory based on the intensity of process related to that particular vertex.

Architecture

To solve the irregular data access pattern while processing a graph, we are proposing NodeFetch. NodeFetch consists of a hardware and software co-design.

From processor perspective, NodeFetch is a new command which is supported by memory subsystem. Processor can use this command to bring neighbors of a given node from memory into host processor cache. Therefore reducing cache miss rate while access to irregular neighbors of that node during executing a graph application.

Presented hardware is able to collect neighbors of a given node, inside memory and send them back to host processor as a response.

The process of finding a node and its neighbors inside memory and putting them together as a block, happens inside memory.

Therefore this is a case of using processing in memory.

To avoid building from ground up, Presented hardware placed inside logic layer of a HMC device by providing a new command inside HMC device. Figure 1 shows the flow of data between software and hardware in presented architecture.

1. Processor sends a NodeFetch command to memory, requesting to fetch a node and its neighbours using NodeFetch hardware inside the memory.
2. NodeFetch hardware receives the request and starts to gather requested node itself, and neighbors of the node, inside a buffer. The buffer size equals to a normal memory response.
3. After collecting node neighbors and the node itself, memory returns node data and its neighbors to processor. As of now, software running on processor knows that all neighbors are in adjacent addresses in memory. Therefore Upon each software request to access any of neighbors, neighbor data is already inside cache.

NodeFetch only finds level one neighbors which means the response does not contain neighbors of neighbors and so on. Figure 2 shows logic layer of a HMC device. NodeFetch hardware placed between crossbar switch and vault controllers. NodeFetch hardware consists of several components. Figure 3 shows block diagram of a NodeFetch unit.

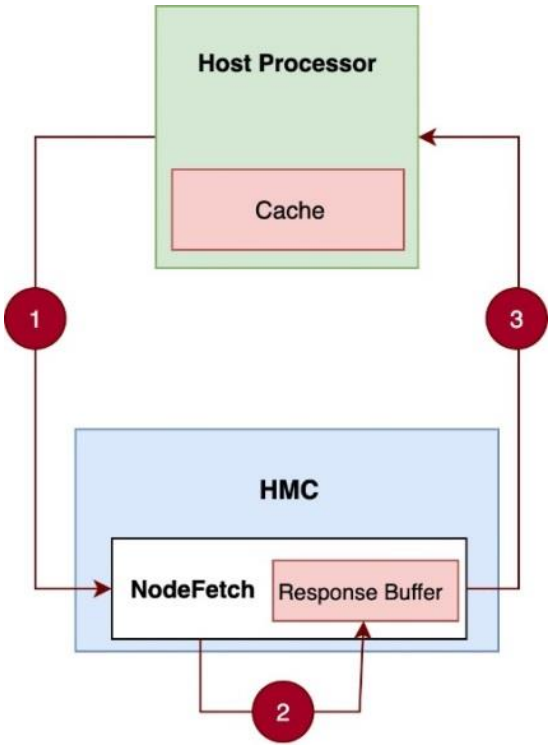


Fig. 1: Flow of data between software and hardware.

Activation Register is used to enable or disable the whole unit. If NodeFetch unit is not activated, then memory packets will pass around the unit.

Vertex Address Register keeps data address of requested node.

Offset Register holds the offset of neighbors to fetch them. Processor sends this variable with the memory request. When fetching neighbours of a node, there might not be enough space in a memory response to fit all neighbors. Therefore processor can request for the remaining neighbors of a node by properly setting the Offset Register.

Neighbor Prefetcher checks Activation register to determine whether to start the process or not. First, Finds address of node data and sets the Vertex Address Register with that address. Then fetch node neighbors and put them inside Block Buffer.

Neighbor Address Buffer keeps address of neighbors during the actual process.

Block Buffer keeps final memory response. Figure 4 shows more details of a NodeFetch unit. After activation, NP checks the request type to find out whether it's a read request or a write request.

If the request type is of type read, Finds address of given node and write it into vertex address register (VAR).

Also write given offset from request into offset register (OR).

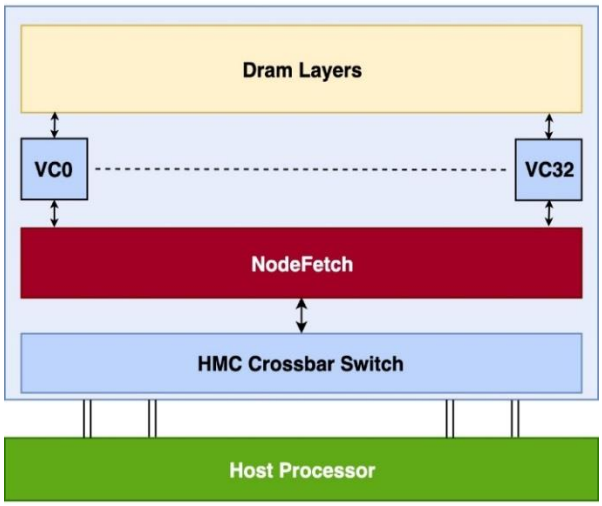


Fig. 2: HMC Logic layer including NodeFetch hardware.

After that, fetches address of neighbors from memory and writes them in neighbor address buffer (NAB). Offset Flag (OF) become activated if there isn't enough space to fit all neighbors inside NAB. Following that, iterates through NAB and fetch neighbor data into block buffer (BB).

After all a memory response emerges from BB and which goes to processor. This response contains node itself, neighbors and offset flag. On the other hand if the request is of type write, repeats previous steps, only instead of fetching neighbors data, update them inside memory.

NodeFetch is not a programmable unit and therefore only can work with a standard form graph storage inside memory. In this standard, there is an array for graph nodes.

Each key in array, refers to a node and the value for that key, contains a pointer to node data and a list of neighbors connected to that node. As a result finding address of node data and neighbors only takes up to O(1).

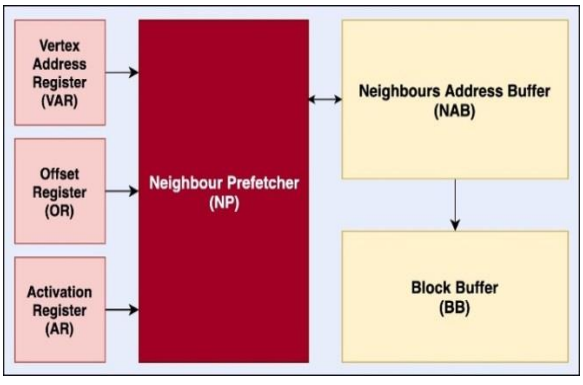


Fig. 3: NodeFetch block diagram.

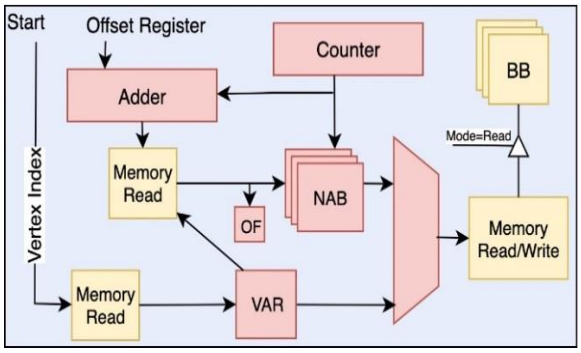


Fig. 4: Nodefetch detail.

Evaluation Methodology

A. Simulation Configuration

We evaluate the proposed system using an in-house cycle accurate simulator. Table 1 shows simulation configuration:

Table 1: Simulation configuration of Nodefetch

Memory	1 HMC module with 8GB of memory capacity and a NodeFetch unit
Processor	2 cores of 2GHz ARM Cortex A15 with 64KB of level 1 cache for data and instruction

Figure 5 illustrates block diagram of the simulator. Input graph and graph algorithm are inputs of the simulator. Dispatcher stores input graph of HMC memory.

Programmer puts graph algorithm in processor. Processor starts working on input graph based on given algorithm. After all, generates several reports such as timings and power consumption.

This simulator developed in a way to use the help of modified HMCSim.

B. Workloads

For evaluation, we implement five different graph algorithms, namely Page Rank (PR), Single-Source Shortest Paths (SSSP), Connected Components (CC), Triangle Counting (TC) and Betweenness Centrality (BC). PR computes importance nodes in graph. This is commonly used in search engines.

PR algorithm assign a number to each node of graph which indicates importance of that node. SSSP computes shortest path between two nodes of graph. SSSP has various applications in networks and also used to find critical path.

SP used in results as a shorter form for SSSP. CC finds connected components in a graph which has several applications in image processing. TC counts triangles in a graph. TC is used in social networks. BC finds the most important node between two given nodes.

BC has various applications in social networks and computer networks. There are a few benchmark suites such as GAP [20] or CRONO [21].

These suits are known to researchers and are being used to evaluate their architectures and ideas.

These benchmark suits include similar applications such as SSSP, BC and PR to process graphs.

We chose GAP [20] benchmark suite as a baseline for graph algorithms. We simulate well-known real-world workloads from Stanford large network dataset (SNAP) [22].

Results and Discussion

This section provides the results of simulations.

A. Execution Time

Figure 6 shows speedup of the chosen workloads normalized to the baseline HMC. Due to better management of poor graph locality, the proposed architecture could reach a better execution time for all

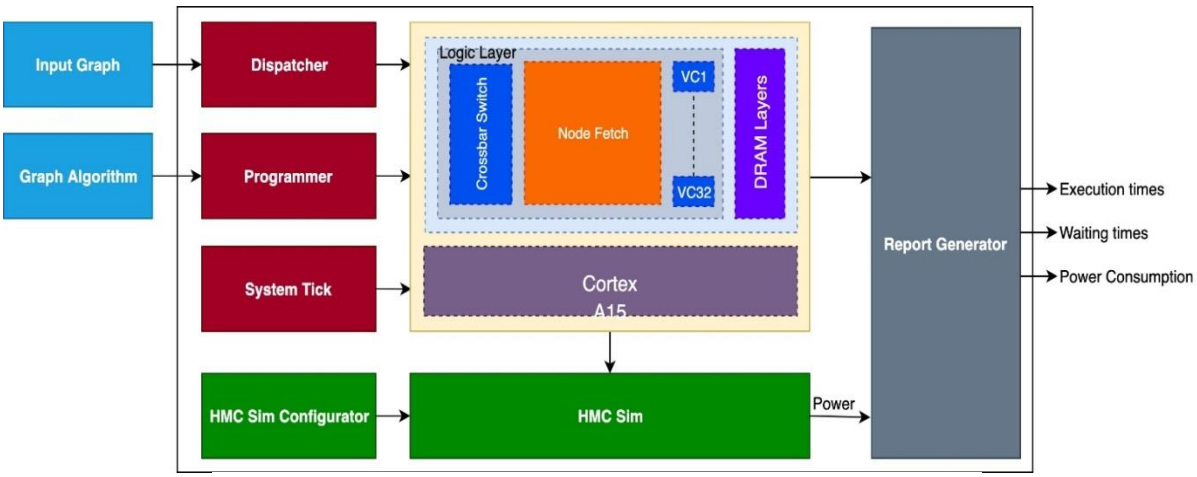


Fig. 5: Block diagram of the simulator.

of the benchmark graph applications. The simulation results indicate an average speed up of 3.3x in comparison to the baseline.

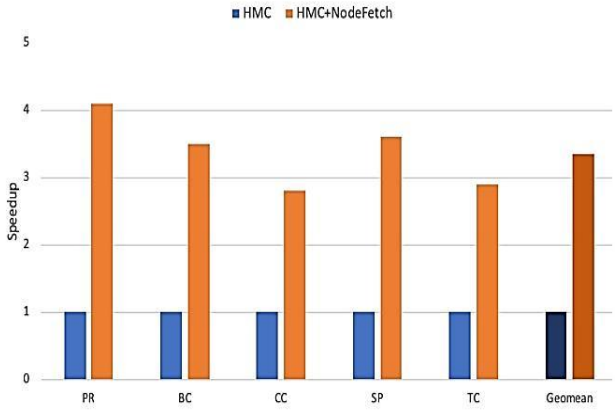


Fig. 6: NodeFetch Speedup in comparison to baseline.

B. Power and Energy Consumption

Improving the execution time and offloading parts of computation from the processor to HMC, results in reduction of energy consumption, indeed at the cost of energy overheads caused by the additional hardware. Simulation results shows that the system energy overheads are significantly less than energy savings. As a result, the overall system energy is decreased. Figure 7 shows the system energy of the NodeFetch normalized to the baseline HMC.

An average of 69% energy reduction is obtained for the evaluated workloads.

Table 2 shows area and power consumption overheads of NodeFetch, Tesseract [17] and Enhanced Tesseract [7] relative to one HMC device. It demonstrates the proposed idea leads to a very low power and area overhead.

Table 2: Area and power overhead NodeFetch, Tesseract and Enhanced Tesseract relative to HMC

Relative to Logic layer of One HMC	Tesseract [17]	Enhanced Tesseract [7]	NodeFetch
Area Overhead	9.6%	9.73%	0.1%
Power Overhead	40%	42%	4.5%

Table 3 shows area power density of NodeFetch, Tesseract and Enhanced Tesseract.

The highest power density of the logic die across all workloads in our design is 14mW/mm2 which is by far below the maximum power density that does not require faster DRAM refresh using a passive heat sink (i.e. 133mW/mm2 [23]).

Table 3: Power density and area comparison

	Tesseract [17]	Enhanced Tesseract [7]	NodeFetch
Max Power Density	94 mW/mm^2	96 mW/mm^2	14 mW/mm^2
Area Overhead per HMC	21.75 mm^2	22 mm^2	0.07 mm^2

The total area of a NodeFetch unit is 0.07mm^2 which solely account for 0.1% area overhead. Our approach increases the average power consumption by 4.5% in comparison to HMC, which may lead a negative impact on device temperature. However according to recent measurements in industrial research on thermal feasibility of 3D-stacked PIM [23], the power consumption should be within the power budget. Therefore, proposed idea is thermally feasible.

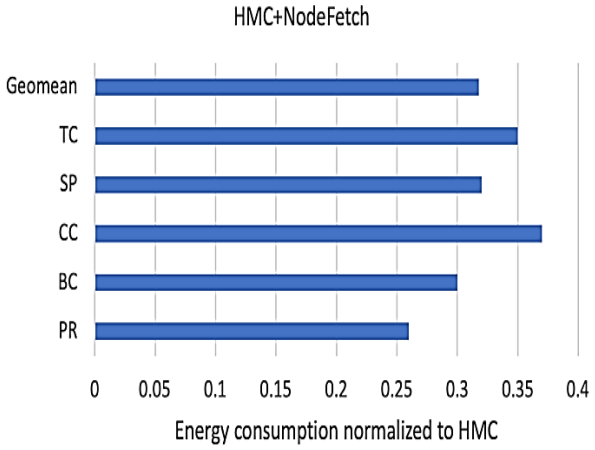


Fig. 7: Relative system energy consumption of the proposed architecture normalized to the baseline architecture.

Conclusion

This paper proposed an optimization to PIM-based graph processing with the help of HMC. Most of the techniques in the field of processing-in-memory, hire methods to reduce movement of data between processor and memory.

This paper proposed a method to reduce graph processing execution time and energy consumption by reducing cache misses while processing a graph. Proposed idea which named NodeFetch, adds a command to HMC for that purpose.

NodeFetch helps graph processing to have a better performance by increasing locality and decreasing irregularity.

Simulation results shows that NodeFetch in average is 3.3x faster than HMC itself, and reduces energy consumption by 69% in average.

Author Contributions

M.A. Mosayebi, and M. Dehyadegari contributed to the design and implementation of the research, to the analysis of the results and to the writing of the manuscript

Acknowledgment

We thank the editor and all anonymous reviewers.

Conflict of Interest

The authors declare no potential conflict of interest regarding the publication of this work. In addition, the ethical issues including plagiarism, informed consent, misconduct, data fabrication and, or falsification, double publication and, or submission, and redundancy have been completely witnessed by the authors.

References

[1] X. Chen, "GraphCage: Cache Aware Graph Processing on GPUs," arXiv preprint arXiv:1904.02241, 2019.

[2] J.E. Gonzalez, Y. Low, H. Gu, D. Bickson, C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in Proc. the 10th Symposium on Operating Systems Design and Implementation (OSDI): 17-30, 2012.

[3] A. Fidel, N.M. Amato, L. Rauchwerger, "Kla: A new algorithmic paradigm for parallel graph computations," in Proc. 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT): 27-38, 2014.

[4] S. Hong, H. Chafi, E. Sedlar, K. Olukotun, "Green-Marl: a DSL for easy and efficient graph analysis," in Proc. Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems: 349-362, 2012.

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

[6] S. Ghose, K. Hsieh, A. Boroumand, R. Ausavarungnirun, O. Mutlu, "Enabling the adoption of processing-in-memory: Challenges, mechanisms, future research directions," arXiv preprint arXiv:1802.00320, 2018.

[7] M.A. Mosayebi, A.M. Hasani, M. Dehyadegari, "Enhanced graph processing in PIM accelerators with improved queue management," Microelectron. J., 94: 104637, 2019.

[8] M. Zhang et al., "GraphP: Reducing communication for PIM-based graph processing with efficient data partition," in Proc. 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA): 544-557, 2018.

[9] G. Dai et al., "Graphh: A processing-in-memory architecture for large-scale graph processing," IEEE Trans. Comput. Aided Des. Integr. Circuits Syst., 38(4): 640-653, 2018.

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

[11] H.M.C. Specification, "2.1, Nov. 2015, Hybrid Memory Cube Consortium," Tech. Rep.

[12] B. Soltani Farani, H. Dorosti, M. Salehi, S. M. Fakhraie, "Ultra-low-energy dsp processor design for many-core parallel applications," JECEI," J. Electr. Comput. Eng. Innovations (JECEI), 8(1): 71-84, 2019.

[13] L. Song, Y. Zhuo, X. Qian, H. Li, Y. Chen, "GraphR: Accelerating graph processing using ReRAM," in Proc. 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA): 531-543, 2018.

[14] G. Kim, J. Kim, J. H. Ahn, J. Kim, "Memory-centric system interconnect design with hybrid memory cubes," in Proc. the 22nd international conference on Parallel architectures and compilation techniques: 145-155, 2013.

[15] J. Kim, W. Dally, S. Scott, D. Abts, "Cost-efficient dragonfly topology for large-scale systems," IEEE micro, 29(1): 33-40, 2009.

[16] J. Kim, W. J. Dally, D. Abts, "Flattened butterfly: a cost-efficient topology for high-radix networks," in Proc. 34th Annual International Symposium on Computer Architecture: 126-137, 2007.

[17] J. Ahn, S. Hong, S. Yoo, O. Mutlu, K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in Proc. 42nd Annual International Symposium on Computer Architecture: 105-117, 2015.

[18] D.-I. Jeon, K.-B. Park, K.-S. Chung, "HMC-MAC: Processing-in memory architecture for multiply-accumulate operations with hybrid memory cube," IEEE Comput. Archit. Lett., 17(1): 5-8, 2017.

[19] A. Addisie, V. Bertacco, "Centaur: Hybrid processing in on/off-chip memory architecture for graph analytics," in Proc. 57th ACM/IEEE Design Automation Conference (DAC): 1-6, 2020.

[20] S. Beamer, K. Asanović, D. Patterson, "The GAP benchmark suite," arXiv preprint arXiv:1508.03619, 2015.

[21] M. Ahmad, F. Hijaz, Q. Shi, O. Khan, "Crono: A benchmark suite for multithreaded graph algorithms executing on futuristic multicores," in Proc. EEE International Symposium on Workload Characterization: 44-55, 2015.

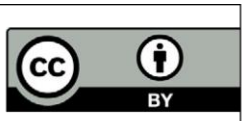
[22] J. Leskovec, A. Krevl, "SNAP Datasets: Stanford large network dataset collection," ed, 2014.

[23] Y. Eckert, N. Jayasena, and G. H. Loh, "Thermal feasibility of die-stacked processing in memory," 2014.

Biographies

Mohammad Amin Mosayebi received his MSc. Degree from K. N. Toosi University in 2019 in computer engineering. He is currently a Ph.D. student at Shahid Beheshti University researching on bio inspired hardware designs and bio-medical signal processing.

Masoud Dehyadegari received his Ph.D. degree from University of Tehran, Tehran, IRAN, in 2013 in computer engineering. He is currently an Assistant Professor of school of computer engineering with the K. N. Toosi University of Technology. His research interests include Low-power system design, Network-on-chips, and Multi-Processor System-on-chip.



How to cite this paper:
M.A. Mosayebi, M. Dehyadegari, "NodeFetch: High performance graph processing using processing in memory," J. Electr. Comput. Eng. Innovations, 9(1): 67-74, 2021.

DOI: 10.22061/JECEI.2020.7453.393

URL: http://jecei.sru.ac.ir/article_1486.html