Graph Partitioning for Near Memory Processing

Chenfeng Zhao
Roger D. Chamberlain
Xuan Zhang

McKelvey School of Engineering
Washington University in St. Louis

Graph Partitioning for Near Memory Processing

Chenfeng Zhao
McKelvey School of Engineering
Washington Univ. in St. Louis
St. Louis, MO, USA
chenfeng.zhao@wustl.edu

Roger D. Chamberlain
McKelvey School of Engineering
Washington Univ. in St. Louis
St. Louis, MO, USA
roger@wustl.edu

Xuan Zhang
McKelvey School of Engineering
Washington Univ. in St. Louis
St. Louis, MO, USA
xuan.zhang@wustl.edu

I. Motivation

The 3Vs of Big Data (volume, velocity and variety) often defy traditional mechanisms for data collection, management and processing. Due to their ability to capture complex relationships among individual data elements, graphs are an important data structure that have been widely used to represent social networks, citation networks, road networks, genome sequences, etc. The proliferation of graph processing applications, including machine learning, recommendation systems, social network analysis, bioinformatics, and others, has heightened the need for efficiently processing graphs, both in terms of performance and energy consumption. Hence, a number of graph analytic works for parallel computing on various systems have been proposed to efficiently process large-scale graphs [1], [2]. Since higher memory bandwidth is an important part of improving the performance of large-scale graph processing, near memory processing (NMP) is receiving increased attention to accelerate these tasks.

The emergence of 3-D stacked memory technology, in which multiple DRAM chips are stacked on top of a single logic chip, has opened the door for the deployment of computation units near the physical DRAM [3]. Instead of a single memory stack (often referred to as a *cube*), recent NMP architectures for large-graph parallel processing utilize multiple memory cubes [4]–[7], which are able to provide (1) higher memory capacity to store large-scale graphs and (2) memory-capacity-proportional bandwidth which provides higher intra-cube bandwidth as the number of cubes increases. Ahn et al. [4] proposed Tesseract, an NMP architecture for parallel graph processing with 16 cubes.

While providing substantial performance gains over conventional DRAM-based architectures, Dai et al. [8] and Zhang et al. [7] indicate that Tesseract's overall memory bandwidth utilization is less than 40%, implying that there are additional performance gains to be had. The reason given for this bandwidth utilization limit is cross-cube memory accesses, which Tesseract did not try to optimize. Therefore, it introduces a more fundamental challenge: *how effectively can graph partitioning algorithms reduce communications overheads while maintaining computational balance*?

Supported by NSF grants CNS-1739643 and CNS-1763503.

II. Limitations of the State of the Art

The state-of-the-art for static partitioning (targeting NMP) is GraphP [7] which proposed "source-cut" partitioning to reduce excess cross-cube communications when executing graph processing applications on a multi-cube NMP architecture combined with a vertex replication mechanism as a graph preprocessing technique. With source-cut, if two or more cross-edges share the same source vertex but have different destination vertices that are in a common cube, then a replica of the source vertex will be generated and placed in the destination cube. Therefore, the data of the source vertex need only be transferred once to the destination cube. As a result, less cross-cube communications volume is required.

However, despite the promising results from GraphP, cross-cube memory accesses still take a signification portion of the execution time (31%-74%) and energy consumption (54%-73%) when executing graph processing applications in a multi-cube NMP system.

There are two inherent properties of source-cut partitioning that limit the potential of GraphP:

- Since only one cross-cube edge pattern is considered in source-cut, it only processes a fraction of cross-cube edges that might potentially be eliminated, which limits its ability to reduce cross-cube communication volume.
- Although graph partitioning methods like source-cut are sensitive to the initial vertex distribution among memory cubes, source-cut limits itself to a single initial vertex distribution, which also limits its ability to reduce cross-cube communication overhead.

III. SuperCut Framework

To address the above limitations, we introduce an end-to-end software framework for multi-cube NMP systems, called *SuperCut*, to effectively reduce cross-cube communication overheads while maintaining workload balance. In SuperCut we consider both the diversity of cross-cube edge patterns and the initial vertex distribution across memory cubes.

SuperCut is comprised of the following elements: (1) a set of graph partitioning algorithms to yield lower cross-cube communications volume and a balanced computational load; (2) a three-phase programming model to express general vertex programs on the basis of user-defined functions that is consistent with our graph partitioning algorithms; (3) an accelerator generator with which near-memory accelerators are generated via high-level synthesis and mapped to an FPGA fabric on the logic layer of the memory cubes; (4) a custom graph representation that implements the resulting graph applications on the hardware NMP system while diminishing the irregularity of vertex traversal and data transfer.

SuperCut uses a number of partitioning algorithms. The first partitioning algorithm is called *mixed-cut*, which is a sequential combination of source-cut partitioning from GraphP [7] and destination-cut partitioning, a run-time adaption of MessageFusion [5], proposed to optimize cross-cube edges exhibiting a different pattern in which there are multiple cross-cube edges that have the same destination vertex and distinct source vertices, all in the same cube. Therefore, by performing both partitioning methods in order, mixed-cut optimizes more cross-cube edges by recognizing more edge patterns, and compared with MessageFusion, the alterations to the graph structure are static rather than dynamic.

The next partitioning algorithm is a stochastic, greedy optimization algorithm in which the cost function is a weighted sum of the maximum and the total number of cross-cube edges across all cube pairs, with adjustable parameters representing the impact of performance and energy consumption under different contexts. In order to avoid introducing workload imbalance, we implement a vertex-swapping strategy as the perturbation function in the greedy algorithm. By swapping a pair of vertices in different cubes instead of moving a single vertex cross cubes, the greedy algorithm can maintain the workload balance to a reasonable degree.

The iterative greedy algorithm with mixed-cut can be slow, especially for large-scale graphs. Based on an observation that the vertex-swapping strategy can only influence a portion of the graph after exchanging a pair of vertices, we propose a new method, called partial graph repartitioning. Its key idea is to find the influence scope of the swapping operation and only process the influence scope instead of the whole graph in each iteration. Since the influence scope is much smaller, the speed of the greedy algorithm can be drastically improved.

In order to implement SuperCut, we propose a three-phased programming model along with a customized graph representation. The programming model divides the graph algorithm into three phases to match our partitioning algorithm while maintaining compatibility with prior work. In the first two phases, vertices originally existing in the graph and vertices generated later by the partitioning methods are processed. In the third phase, data are transferred between memory cubes.

In our accelerator design, we develop two modules to process the vertices for the first two phases, respectively, in parallel. After gathering data from input neighbors, some of them update themselves while others transfer aggregated data to custom DMAs so that these updates can be transferred between cubes during the third phase. Co-designed with the software system and hardware system proposed in this work, we also describe a customized data structure in which vertices are classified and then deployed within separate, disjoint address ranges.

We evaluate SuperCut on a simulated, bare-metal NMP architecture based on reconfigurable logic by extending `gem5-SALAM` [9], using four representative graph applications and five real-world graphs.

IV. Key Results and Contributions

Evaulation results show that SuperCut can achieve $2.51\times$ speedup and $2.54\times$ reduction in total energy consumption relative to Tesseract. It also provides $1.71\times$ speedup and $1.65\times$ total energy consumption reduction against GraphP with a 56% lower extra memory footprint.

In this work, we make several specific contributions:

- We propose the *mixed-cut* partitioning method, combining the principles of source-cut and destination-cut to optimize for both shared-source vertices and shared-destination vertices.
- We propose a *vertex-swapping-based greedy* algorithm to further reduce communications volume, including a partial graph evaluation method to improve its performance.
- We propose a *three-phase programming model* to match our graph partitioning method which avoids data races, and design a customized graph representation to minimize memory access overheads.
- We build a bare-metal multi-cube near memory processing simulation platform with reconfigurable logic kernels as computing units and use it to evaluate the framework.

The SuperCut framework results in significant benefits to near memory processing of graph algorithms.

References

[1] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed graph-parallel computation on natural graphs," in *Proc. of USENIX Symp. on OS Design and Impl.*, 2012, pp. 17–30.

[2] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proc. of ACM Int'l Conf. on Management of Data*, 2010, pp. 135–146.

[3] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra, "Near-memory computing: Past, present, and future," *Microprocessors and Microsystems*, vol. 71, 2019.

[4] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proc. of 42nd Int'l Symp. on Computer Architecture*, 2015, pp. 105–117.

[5] L. Belayneh, A. Addisie, and V. Bertacco, "MessageFusion: On-path message coalescing for energy efficient and scalable graph analytics," in *Proc. Int'l Symp. on Low Power Electronics and Design*. IEEE, 2019.

[6] L. Belayneh and V. Bertacco, "GraphVine: Exploiting multicast for scalable graph analytics," in *Proc. of Design, Automation & Test in Europe Conf. & Exhibition*. IEEE, 2020, pp. 762–767.

[7] 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 *Proc. of Int'l Symp. on High Performance Computer Architecture*. IEEE, 2018, pp. 544–557.

[8] G. Dai, T. Huang, Y. Chi, J. Zhao, G. Sun, Y. Liu, Y. Wang, Y. Xie, and H. Yang, "GraphH: A processing-in-memory architecture for large-scale graph processing," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 640–653, 2018.

[9] S. Rogers, J. Slycord, M. Baharani, and H. Tabkhi, "gem5-SALAM: A system architecture for LLVM-based accelerator modeling," in *Proc. of Int'l Symp. on Microarchitecture*. IEEE, 2020, pp. 471–482.