SCIENCE CHINA
Information Sciences

• PERSPECTIVE •

December 2017, Vol. 60 126102:1–126102:3
doi: 10.1007/s11432-017-9226-8

Towards dataflow based graph processing

Hai JIN*, Pengcheng YAO & Xiaofei LIAO

Service Computing Technology and System Lab, Cluster and Grid Computing Lab, Big Data Technology and System Lab School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

Received May 18, 2017; accepted August 21, 2017; published online November 8, 2017

Citation Jin H, Yao P C, Liao X F, et al. Towards dataflow based graph processing. Sci China Inf Sci, 2017, 60(12): 126102, doi: 10.1007/s11432-017-9226-8

Modern graph processing is widely used for solving a vast variety of real-world problems, e.g., web sites ranking [1] and community detection [2]. To better adapt and express the procedure of graph iteration, a wide spectrum of research is proposed with highly concurrent programming models and smart strategies of graph partition [1, 3].

While existing researches greatly improve the performance of memory subsystem [4], they are still subject to the underlying modern processor. We divide the process of graph applications into three pieces of slots, which indicates hardware resources needed to process micro-ops (uops), and present the result in Figure 1(a). Although previous work has made great progress to significantly improve the performance of graph applications by optimizing 41% stalls resulting from the memory subsystem [5, 6], a vast body of inefficiencies (35% slots wasted) inside the underlying processor is still unknown and seldom studied in existing work.

In this article, we conduct an in-depth micro-architectural study on a variety of graph algorithms using four mainstream graph processing frameworks (GraphChi, Ligra, Galois, GAPBS), to study potential inefficiencies inside the underlying processor. In addition, four non-graph benchmarks (433.milc, 445.gobmk, 454.h264ref, 471.omnetpp) from SPEC CPU2006 are selected to identify the inherent characteristics unique to the graph processing. All experiments are conducted on an Intel Ivy-bridge server with an E5-2680 v2 processor and 64 GB of DDR3-1600 memory.

Inefficient instruction-level-parallelism. Modern processors run in an Out-of-Order (OOO) manner to simultaneously execute multiple independent instructions. However, our experimental results in Figure 1(a) show that only about 23% pipeline slots are eventually retired in graph processing. It means only one fourth of the pipeline's abilities are used, resulting in an extremely low instructions per cycle (IPC) of graph processing.

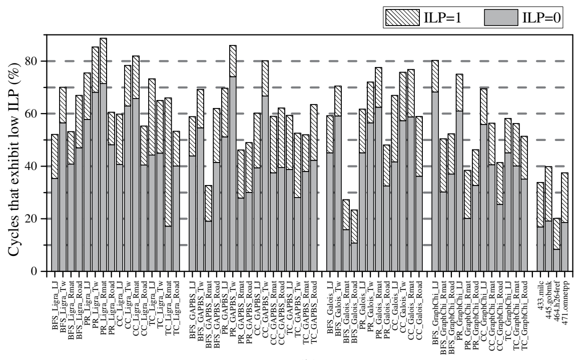
To get a deeper understanding on the inefficiencies inside the core, we further evaluate ILP of graph processing. As shown in Figure 1(b), about 62% execution cycles exhibit low ILP (0 or 1 in a 4-wide core) in graph processing. Specifically, about 41% execution cycles execute 0 uop in graph processing, which means that the core is just idle and waits for the end of former uops in these cycles. In contrast, this value in SPEC is about 13%, which is significantly smaller.

The low ILP is attributed to the heavy dependency in graph processing, where the execution of an instruction is usually related to the results of previous instructions. Moreover, when adding an active vertex to the active list, the operations must be serialized to avoid data race, which brings in extra dependency. Another reason of low ILP is the serial semantic of instruction stream. For instance, if a load instruction causes an L3 miss, the

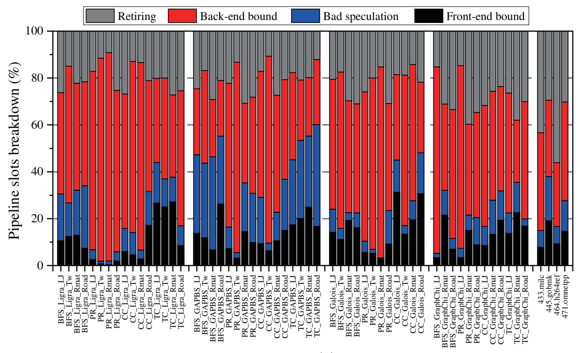
* Corresponding author (email: hjin@hust.edu.cn)
The authors declare that they have no conflict of interest.

		Previous work	Our work
Graph algorithm	Not stalled	Stalled by memory subsystem	Stalled by core inefficiencies
Breadth-First Search (BFS)	25.486	38.022	36.492
PageRank (PR)	23.704	46.683	29.613
Connected Components (CC)	20.691	43.751	35.558
Triangle Counting (TC)	23.674	37.750	38.576

(a)



(b)



(c)

Figure 1 (Color online) (a) Breakdown of experiment on pipeline slots; (b) cycles of ILP of different graph applications; (c) time breakdown of pipeline slots

OOO buffers might soon be clogged by succeeding instructions that are already executed. For general applications, the influence of serial semantic could be ignored since the cache miss rates are low. However, the high cache miss rates of graph processing make serial semantic a severe problem.

Inefficient branch prediction. While the core could issue and execute several uops per cycle, not all these uops could eventually retire. If a predicted instruction is incorrect, the corresponding slot would be flushed and wasted.

To have a better understanding of inefficient branch prediction, we evaluate the time breakdown of pipeline slots. Specifically, bad speculation denotes the slots wasted due to incorrect speculations. As shown in Figure 1(c), branch misprediction leads to almost 15% of total pipeline slot waste, which means that the predictors can hardly provide satisfied performance for graph applications. In graph applications, the branches are more frequent and complex than traditional applications, thus stressing more on predict units.

Introduction of dataflow model. To cope with the significant inefficiencies of modern processor in graph processing, we propose to leverage the dataflow model to mitigate and even break through the underlying limitation, particularly in terms of ILP and branch prediction.

Unlike the traditional control flow based paradigm that enforces sequential retirement, the dataflow model provides a relax and powerful paradigm for describing parallel computation. In dataflow model, a program is generally described by a directed graph. During the process of dataflow graph, a given node can be active if and only if all inputs are available.

Highlight of dataflow in ILP. As discussed above, complex dependencies and serial semantics severely limit the ILP for graph processing. Specifically, the traditional control flow model might enforce a number of false dependencies that limit the potential parallelism. This phenomenon is particularly serious in graph processing.

Fortunately, dataflow only focuses on the data dependency, which is the unique indicator used to determine the scheduling order of instructions. As a consequence, it can be leveraged to totally eliminate false dependencies in graph processing and further provide a substantial amount of optimized parallelism. More importantly, since there is no implicit restrictions between independent instructions, the sequential operations of instruction retirement in traditional control flow model can be greatly relaxed in parallel with only the availability of operands taken into consideration. The instructions would not be interrupted by the execution of other independent instructions.

Highlight of dataflow in branch prediction. High rate of branch misprediction would lead to a mass number of stalls, which is particularly severe for the graph algorithms. Improving the performance of branch prediction is another important factor for performance enhancement of graph processing. The conditions in graph processing can be divided into two broad categories: loop conditions and computation conditions.

Loop conditions represent conditional instructions that controls the loop, e.g., the while/for loop that controls the traversal of vertex list. In control flow model, although the results of loop conditions are easy to be predicted, the frequent predictions might largely increase the fetch latency in graph processing. In contrast, in dataflow graph, the logic of loop controlling is generally hidden behind the flow of data, which indicates that loop conditions do not have to be performed actually.

Computation conditions represent the conditions inside the process of an original graph edge. Since these conditions in graph processing are not relevant to history branch results but dependent on the graph structure, the common prediction strategies based on history states might cause large number of mispredictions. In contrast, because of the strong determinacy in dataflow model, we could process every branch of a conditional instruction in parallel for only once, therefore greatly reducing the cost of branch misprediction.

Challenges. Despite that dataflow has a great opportunity in improving the performance of graph processing, it is still a tedious task to leverage it into solving graph processing problems. Generally, dataflow suffers from low instruction efficiency. For instance, detecting enable instructions and constructing result tokens would bring in additional costs. Consistent with the above conclusion, we find that the number of instruction of graph algorithms in recent dataflow systems is tens of that of specially-designed systems. Moreover, the dataflow graphs are huge and highly related to graph algorithms, making it hard to design a general graph processing accelerator architecture.

Design philosophy. To cope with these challenges, we make an attempt to present some potential methodologies for designing graph processing accelerator based on dataflow model. The preliminary method is to use Field-Programmable-Gate Array (FPGA) to increase instruction efficiency and flexibly adapt accelerator to different algorithms. We could map every operation in dataflow graph to a hardware module, and connect them by a wire if there exists data dependency. The overhead of fetching and scheduling instructions could be reduced since all instructions are connected by wire and triggered by the arrival of data.

The widely used spatial dataflow architecture [7] (e.g., systolic array) could also be applied to further reduce scheduling overheads. Although originally designed for dense matrix multiplication, it is also a good match for graph algorithms expressed by Gather-Scatter model. The similarities between their execution flows make it possible to map vertices in original graphs to nodes in spatial dataflow architecture. Moreover, scheduling overheads are extremely low since all nodes regularly exchange data with their spatial neighbors.

While dataflow graphs are huge, we could simplify the graph processing abstraction to make it easier to map dataflow graphs to hardware modules. We abstract the common vertex-centric model into some key operations (e.g., Read Active Source List) and reconstruct the dataflow graph based on these common operations. Since graph algorithms could always be expressed by iterative models, the newly constructed dataflow graph only includes operations in one iteration. To further deal with the hardware resources limitation, we just map the operations that are shared by the process of every vertex, instead of mapping the whole dataflow graph.

Conclusion. It is observed in our study that the performance of graph processing is still limited to the underlying modern processors, leading to the significant inefficiencies of ILP and branch prediction. In order to cope with these inefficiencies, we propose to leverage the dataflow paradigm, which can provide a considerable parallelism and reduce the cost of branch misprediction.

Acknowledgements This work was supported by National High Technology Research and Development Program of China (863 Program) (Grant No. 2015AA015303).

References

1 Malewicz G, Austern M H, Bik A J, et al. Pregel: a system for large-scale graph processing. In: Proceedings of ACM SIGMOD International Conference on Management of Data. Indiana: ACM, 2010. 135–146

2 Zhang Z Y. Community structure detection in social networks based on dictionary learning. Sci China Inf Sci, 2013, 56: 078103

3 Gonzalez J E, Low Y, Gu H, et al. Powergraph: distributed graph-parallel computation on natural graphs. In: Proceedings of USENIX Symposium on Operating Systems Design and Implementation. Hollywood: USENIX, 2012. 17–30

4 Beamer S, Asanovic K, Patterson D. Locality exists in graph processing: workload characterization on an Ivy bridge server. In: Proceedings of IEEE International Symposium on Workload Characterization, Atlanta, 2015. 56–65

5 Ham T J, Wu L, Sundaram N, et al. Graphicionado: a high-performance and energy-efficient accelerator for graph analytics. In: Proceedings of International Symposium on Microarchitecture, Taipei, 2016. 1–13

6 Ozdal M M, Yesil S, Kim T, et al. Energy efficient architecture for graph analytics accelerators. In: Proceedings of Annual International Symposium on Computer Architecture, Seoul, 2016. 166–177

7 Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit. In: Proceedings of Annual International Symposium on Computer Architecture. Toronto: ACM, 2017. 1–12