Energy-Efficient Graph Traversal on Integrated CPU-GPU Architectures

Heng Lin Jidong Zhai Wenguang Chen
linheng11@mails.tsinghua.edu.cn, {zhaijidong,cwg}@tsinghua.edu.cn
Department of Computer Science and Technology
Tsinghua University, Beijing, China

ABSTRACT

Recently, architecture designers tend to integrate CPUs and raphics Processing Units(GPUs) on the same chip to produce energy-efficient designs. On the other hand, graph applications are becoming increasingly important for big data analysis. Among the graph analysis algorithms, Breadth-First Search (BFS) is the most representative one and also an important building block for other algorithms. Despite previous efforts, it remains an important problem to get optimal performance for BFS on integrated architectures.

In this paper, we propose an adaptive algorithm to atomically find the optimal algorithm on the suitable devices, which can get 1.6X speedup compared with the state-of-the-art algorithms in an energy consumption index, namely TEPS/Watt(Traversed Edges Per Second every Watt).

1. INTRODUCTION

Recently, architecture designers tend to integrate CPUs and GPUs on the same chip to produce energy-efficient designs. For example, AMD launched its first integrated architectures in 2011, called Accelerated Processing Unit (APU). One main advantage of integrated architectures is that transmission overhead between CPUs and GPUs on conventional discrete architecture can be significantly reduced since both CPUs and GPUs shared the same physical memory.

On the other hand, graph applications are becoming increasingly important for big data analysis. Among the graph analysis algorithms, Breadth-First Search (BFS) [3] is the most representative one and also an important building block for other algorithms. Moreover, BFS is currently the main benchmark of Graph500 [1], which is a rating of supercomputer systems.

There have been some related studies for BFS on various computing platforms. Yasui et al. [5] proposed vertex rearrangement method through considering graph layout. Beamer et al. [2] proposed bottom-up direction search aside top-down direction to skip some edges, which can significantly reduce memory access for some graphs. Daga et al. [4] attempted to apply the Beamer's method on APU.

Despite previous efforts, it remains an important problem to get optimal performance for BFS on integrated architecture. Since BFS can dynamically adjust different search algorithms (top-down and bottom-up direction search) based on the edge number of each level to obtain optimal performance. At the same time, due to unified memory access on integrated architectures, we can flexibly switch different BFS algorithms between CPU and GPU without introducing large data transmission overhead. However, based on our experimental results, we find that different BFS algorithms demonstrate different performance on different devices of integrated architectures with the execution of BFS, as shown in Figure 1, which makes it very difficult to select an optimal combination between search algorithms and executing devices.

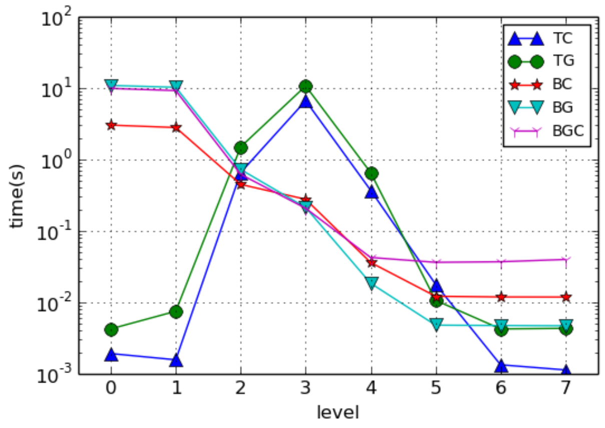

Figure 1: A comparison of different implementation. Methods description list in Table 1. In this example, best combination can outperform simple *TC* plus *BG* about 20%.

In this paper, we try to solve how to get optimal BFS performance for a given input graph through selecting optimal devices and algorithms on integrated architectures. To achieve this goal, we propose an adaptive algorithm to atomically find the optimal combination of search algorithm and executing device for each level of BFS and evaluate our method using synthesized and real world datasets in terms of time and energy consumption. Experimental results show that our algorithm has 1.6X speedup compared to previous state-of-the-art algorithms in TEPS/Watt.

2. METHODOLOGY

To achieve the best performance on integrated architectures, we choose four optimization methods. (1)Graph layout [5];

(2)Direction optimization [2]; (3)Software cache for auxiliary data structure; (4)Devices choice. It is known every optimization method is a trade off, which may even hurt the performance if not suitable. At last, four of the combinations(Table 1) are chosen to be the candidate of each BFS level.

Item	Description
TC	Top-down direction + CPU
BC	Bottom-up direction + CPU
BG	Bottom-up direction + GPU
BGC	Bottom-up direction + GPU + Software Cache

Table 1: Selected implementation

At each level of BFS, we choose the best method followed by three principle:

1. Using TC when frontier(the active vertex in current level) is small, or $\gamma * m_f < m_a$ (m_f denotes the number of edges in frontier, m_a denotes the total edge number)
2. Turn TC to bottom-up when frontier gets bigger, or $\alpha * m_f < m_l$ (m_l denotes the number of edges left to visited)
3. When decides bottom-up, choose a implementation according to frontier size. If $m_l/m_f > \alpha_1$, use BC, $\alpha_1 > m_l/m_f > \alpha_2$, use BGC, $m_l/m_f < \alpha_2$, use BG.
4. If the frontier gets smaller enough, turn bottom-up to TC, or $\beta * n_f < n_a$ (n_f denotes the number of vertexes in frontier, n_a denotes the total vertexes number)

We choose $\alpha = 100, \alpha_1 = 40, \alpha_2 = 5, \beta = 100, \gamma = 1000$.

For implementation, we choose OpenMP for CPU and OpenCL for GPU to reduce the affect of compiler. OpenCL has memory size allocation limitation up to 2GB, we divide the data manually and process each partition streaming when the data size exceed the limitation.

If not explicitly declare, in this paper the graph generator is kronerker and edge_factor is set to 16.

3. EVALUATION

Item	Detail
OS	Ubuntu 14.04.1 LTS
Compiler	GCC 4.8.2 OpenCL 2.0
APU	AMD A10-7850K Radeon R7 4 CPU cores, 3.7GHz 8 GPU cores, 0.7GHz
Memory	32GB

Table 2: Experiment Environment

The experiment environment has been listed in Table 2, we are trying to make use of the whole 32GB memory.

We compare our algorithm with some other state-of-the-art ones in Figure 2, only energy data is listed and more data

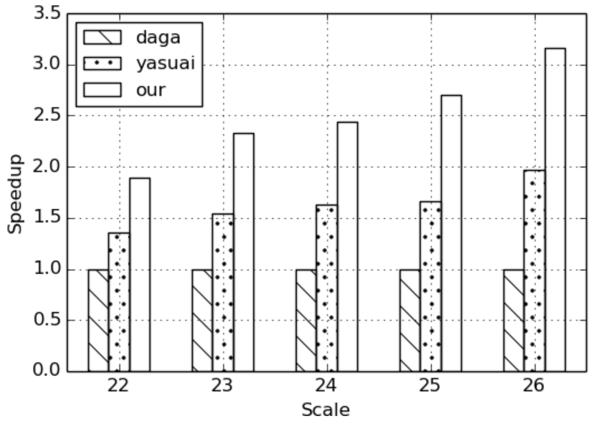

Figure 2: Power comparison (TEPS/Watt) among Daga [4], Yasuai [5] and our algorithm

can be found in poster. Our algorithm yields a speedup of 3.1X compare to Daga's and 1.6X compare to Yasuai's.

To make the result more reliable, we run our algorithm in multi synthesized and real world graph dataset, the result can be found in our poster.

4. CONCLUSION

The appearance of APU gives us an opportunity to use CPU and GPU concurrently without data transfer overhead. This paper discusses various BFS optimization method and implements it efficiently in APU. We raise a switch policy to automatically choose the best implementation at runtime. At last, the algorithm is tested both in synthesized and real world datasets to show the efficiency in time and energy.

5. ACKNOWLEDGMENTS

The authors sincerely thank the anonymous reviewers for their valuable comments and suggestions. This work has been partly supported by NSFC project 61472201 and Chinese National High-Tech Research and Development Plan (863 project) 2012AA010901.

6. REFERENCES

[1] Graph500. http://www.graph500.org/.
[2] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2012.
[3] T. H. Cormen. *Introduction to Algorithms*. MIT Press, 2009.
[4] M. Daga, M. Nutter, and M. Meswani. Efficient breadth-first search on a heterogeneous processor. In *2014 IEEE International Conference on Big Data (Big Data)*, 2014.
[5] Y. Yasui, K. Fujisawa, and Y. Sato. Fast and energy-efficient breadth-first search on a single numa system. In *Proceedings of the 29th International Conference on Supercomputing (ISC)*, 2014.