OBFS: OpenCL Based BFS Optimization on Software Programmable FPGAs

***Abstract*—Breadth First Search (BFS) is a key building block of graph processing and there have been considerable efforts devoted to accelerating BFS on FPGAs for the sake of both performance and energy efficiency. Prior work typically built the BFS accelerator through handcrafted circuit design using hardware description language (HDL). Despite the relatively good performance, the HDL based design leads to extremely low design productivity, and incurs high portability and maintenance cost. While the evolving high level synthesis (HLS) tools make it convenient to create a functional correct BFS accelerator, the performance of the baseline design remains much lower.**

To obtain both the near handcrafted design performance and the software-like features, we propose OBFS, an OpenCL based BFS accelerator on software programmable FPGAs, and explore a series of high-level optimizations to the OpenCL design. With the observation that OpenCL based FPGA design is rather inefficient on irregular memory access, we propose an accelerator-specific graph reordering in combination with a distributed bitmap buffer to enable parallel traverse and reduce external memory access as well. In addition, we shift the random BFS level update from the main BFS processing and hide it with overlapped execution of different BFS processing. According to the experiments on a set of representative graphs, OBFS achieves up to 12X performance speedup compared to the reference design in Spector benchmark on Intel Harp-v2. When compared to prior handcrafted design on similar FPGA cards, it achieves comparable performance or even better on some R-MAT graphs.

I. Introduction

Breadth-first search (BFS) is the basic building component of many graph algorithms and is thus of vital importance to high-performance graph processing. Nevertheless, it is notoriously difficult for accelerating on FPGAs because of the irregular memory access and the low computation-to-memory ratio. At the same time, BFS on large graphs also involves tremendous parallelisms which indicate great potential for acceleration. With both the challenge and the parallelization potential, BFS has attracted a number of researchers exploring its acceleration on FPGAs [1], [2], [3], [4], [5], [6], [7], [8].

Previous work have shown that BFS accelerators on FPGAs can provide competitive performance and superior energy efficiency when given comparable memory bandwidth. However, these work typically optimize BFS or relatively general graph processing with dedicated circuit design using hardware description language (HDL). The HDL based customized designs are beneficial to the resulting performance and save resource consumption, but it usually takes long time for development, upgrade, maintenance and porting to a different FPGA device, which are all important concerns from the perspective of the accelerator developers.

Because of the limitation of the conventional HDL design method, high level synthesis (HLS) tools that advance rapidly in recent years become attractive. HLS tools are increasingly adopted in both industry and academia for rapid FPGA prototyping and application acceleration [9], [10]. Nevertheless, the current HLS based design tools are mostly used for applications with relatively regular memory access patterns and data paths. It remains challenging for the HLS tools to accelerate BFS with irregular memory access patterns and complex data paths.

First, there are considerable irregular memory access including random and short burst in BFS and the irregular memory access bandwidth utilization is orders of magnitude lower than that of the sequential memory access. In addition, the number of a frontier vertex's neighbor is only determined at run-time, so reading the unaligned neighboring vertices with OpenCL typically has narrow data width. The read is not efficient as pointed out in [11]. Worse still, the neighboring vertices may have their visiting status stored randomly either in main memory or on-chip buffer. Although they can be processed in parallel, read/write conflicts happen frequently and affects the BFS data path parallelization dramatically.

To solve this problem, we reorder each vertex's neighboring vertices i.e. edges in compressed sparse row (CSR) graph and split them into batches such that the neighboring vertices in each batch have their status stored in independent memory blocks. With the batching, the random memory access gets coalesced. Meanwhile, the data path can be simplified and implemented in parallel efficiently. We also observe that updating the BFS level information of the active vertices is essentially random memory access and time-consuming in BFS, while the updating can be done anytime when the active vertices of BFS are determined. Thus, we can separate the random memory operations from the conventional BFS and have the update overlapped with the next BFS processing. With the hyper pipelining, the random memory operations can be completely hidden, which is beneficial to the BFS processing.

According to the experiments on a set of big graphs, the proposed OpenCL based BFS accelerator achieves up to 12X performance speedup when compared to the design in Spector benchmark and it is on par with many handcrafted design on similar FPGA boards. The major contributions of this work are summarized as follows.

- As far as we know, this is the first highly optimized OpenCL based BFS accelerator on FPGAs targeting portability and ease of use on top of performance. We

will make it open-sourced. The URL is omitted due to the double-blinded review.
- We proposed a set of methods to optimize the irregular memory accesses and data path parallelization of BFS using OpenCL. This may shed light on similar irregular application acceleration on FPGAs using OpenCL.
- The resulting accelerator shows significant performance speedup over a baseline OpenCL design and gets close to state-of-art handcrafted design on a set of representative graphs.

The rest part of the paper is organized as follows. In Section II, we show the challenges of BFS accelerator design with HLS and motivate this work. In Section III, we detail the major OpenCL based BFS optimization methods. In Section IV, we present comprehensive experiments of the BFS accelerator. In Section V, we present a brief survey of the graph processing accelerators particularly on FPGAs. Finally, we conclude this work in Section VI.

II. Motivation

BFS is a widely used graph traversal algorithm and it is the basic building component of many other graph processing algorithms. A typical frontier based BFS algorithm implementation is named as level synchronous BFS [1], [2], [12]. It has a frontier queue to store the vertices to be inspected. Initially, the source vertex is only vertex in the frontier. By inspecting the frontier vertices, new vertices are traversed and become the frontier. Iteratively process the graph until the frontier is empty. Graphs are usually sparse and stored as compressed sparse row (CSR) format. CSR includes a row pointer array (RPA) and a column index array (CIA). RPA contains the edge index starting position of each vertex while CIA consists of the incoming/outgoing neighboring vertex indices.

High level synthesis tools greatly alleviate the efforts of building BFS accelerators on FPGAs. Nevertheless, the performance of the resulting accelerators can be far from satisfying especially for irregular applications like BFS. In this section, we take our own early experience on optimizing BFS on Alpha-Data as an example and show the challenge of optimizing BFS with HLS tools.

We started from basic sequential BFS algorithm for the HLS design. First of all, we separated the nested loop into different pipeline stages connected via FIFO using data flow model supported by Xilinx HLS. We explored all the possible pipeline combinations ranging from single-stage pipeline to seven-stage pipeline. With the optimized pipeline, we further introduced prefetch buffer to improve memory access efficiency, added customized cache structure for the vertex property to reduce the external memory accesses, hash table based redundancy removal logic to reduce the repeated outgoing neighboring vertices of the frontier vertices. Finally, we also tried to duplicate the pipeline stages for parallel processing and tuned the design parameters of the accelerators such as cache and prefetch buffer size. We believed we had tried the major design optimization methods that could be applied to the baseline HLS design.

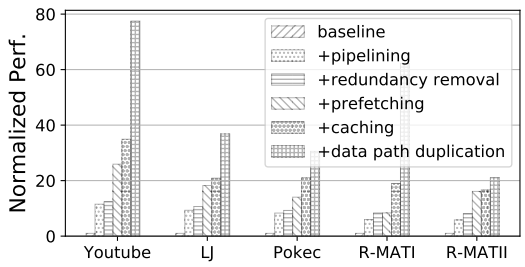

Fig. 1. The normalized performance of an HLS based BFS accelerator with various optimizations on Alpha-Data

We measured the performance of the accelerator with a set of representative graphs including Youtube (YT), Live Journal(LJ), Pokec(PK) and two R-MAT graphs. Details about the graph benchmark can be found in Table I in the experiment section. Figure 1 presents the normalized performance when different optimization techniques are gradually applied to the naive baseline design. It can be found that the resulting accelerator achieves up to 70X speedup compared to the baseline design. In spite of the significant performance improvement, the HLS implementation is still far from ideal compared to the state-of-art handcrafted design on FPGAs [2], [1], [12], [13], [14]. Therefore, the current BFS needs a careful redesign to achieve competitive performance with HLS.

Another challenge of the BFS design with HLS is the optimization productivity. The optimization took an experienced hardware designer with basic high level synthesis knowledge up to months, though most of time was spent on implementing the large amount of combined optimization methods and cumbersome debugging when software simulation passed but the accelerator got stalled during hardware execution. This motivates us to focus on the high-level BFS accelerator optimization and offer open-sourced BFS design with both the near handcrafted performance and the software-like flexibility.

III. HLS based BFS optimization

As there are large amount of irregular memory accesses in BFS and they lead to rather low memory bandwidth utilization [11], irregular memory access is a key factor that affects the BFS performance. Centering this problem, we propose a set of combined high-level optimizations including graph reordering, on-chip buffer partition and hyper BFS pipelining for efficient implementation with OpenCL on Intel Harp-v2. The optimizations are detailed in the rest of this section.

A. Graph reordering

To alleviate the irregular memory access in BFS, we propose a graph reordering scheme for the target accelerator. The basic idea is to optimize the edge layout for more efficient memory access or processing without changing the graph structure. It needs to be done once for each graph and can be considered as graph pre-processing. Note that graph pre-processing covers many distinct methods such as graph partition and graph reordering. It is a common practice in many graph processing systems [3] [15] [16] [17].

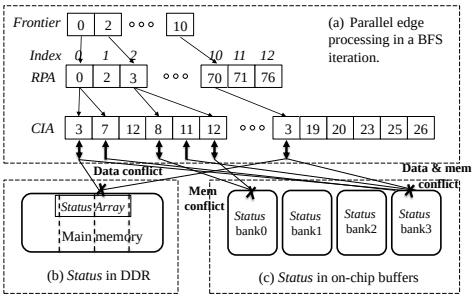

Fig. 2. Conflicts among the parallel BFS data paths

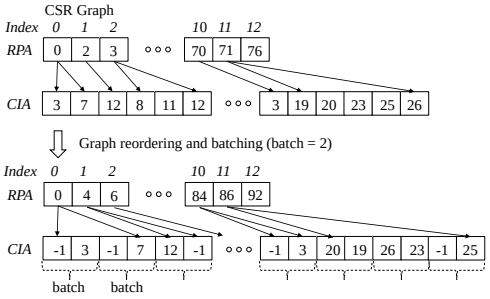

Fig. 3. CSR layout after the graph reordering and batching

As each frontier vertex may have multiple neighboring vertices or associated edges to be processed, sequential edge processing will soon become the bottleneck of the BFS. Parallel processing alleviates the bottleneck, but it is usually constrained by the memory access conflicts. Figure 2 shows the conflicts when inspecting the frontier neighbors in parallel. Both vertex 8 and Vertex 12 are neighbors of vertex 2 and they are supposed to be processed in parallel. However, the status or property of the two vertices may stay in the same DDR bank or on-chip buffer bank. As a result, they can only be written sequentially. The conflict is essentially caused by the limited memory level parallelism, so we call it memory conflict. There is also data dependency conflict. When the frontiers are processed in parallel, vertex 0 and 10 that have the same neighbor i.e. Vertex 3 and we can not perform status write of vertex 3 in parallel. We take it as data conflict. The data dependency remains despite the different data layouts, so we focus on resolving the memory conflict in this subsection.

To address the memory conflict in BFS, we split the outgoing edges i.e. CIA array into batches and edges in each batch can be processed independently. With the reordering and batching, the parallelization of the edge processing becomes straightforward and friendly to the high-level compilation tools. Meanwhile, the batching enforces a coalesced CIA read and it is also beneficial to the efficiency of the memory accesses according to the experiments in Section II.

This reordering can be performed offline. A typical example of the reordering is shown in Figure 3. The original CSR graph is placed in the top part of the figure. To divide the edges into batches for parallel processing, we reorder the edges i.e. CIA of each vertex. In this example, batch is set to be 2. For Vertex 0, it has as 2 neighbors i.e. Vertex 3 and 7 according to the RPA. We expect the vertices in each batch to be divided into distinct memory banks either in DDR banks or on-chip buffer banks so that there is no access conflict during processing. Here we just use the simple modular operation to divide the data, so Vertex 3 and 7 must be put into two batches. The empty slot in each batch will be filled with -1 to notify the accelerator to ignore it during processing. Since the CIA is expanded after the reordering and batching, the RPA gets updated accordingly. With the reordering and batching, more memory space is required to store the CSR graph. The advantage is the aligned memory access with higher data width and conflict-free edge processing, both of which are beneficial to the hardware implementation using Intel OpenCL.

B. On-chip buffer partition

To avoid frequent main memory access, we utilize bitmap to store the visiting status during BFS. Each vertex needs only one bit storage and the latest FPGAs with dozens of mega bits can accommodate graphs with dozens of millions of vertices such as twitter2010 graph which has around 42M vertices [18]. In line with the graph reordering and batching, we split the visiting status bitmap into multiple banks that can be accessed in parallel. Suppose the batch size is W, we set the bitmap buffer partitions to be W as well. The vertex visiting status will be evenly distributed across the bitmap buffer banks. The visiting status of vertex id will be put into the ith memory bank where $i = mod(id, W)$. As the vertex status read/write is mostly random and parallel in BFS, accessing the status via the on-chip buffer banks guarantees determined and low latency. It is much more efficient compared to the DDR accessing.

Moreover, the parallel bitmap buffer banks further facilitate conflict-free processing as presented in Figure 4. In this example, $W = 8$ and it indicates that 8 edges can be processed in parallel. When a processing path gets -1 from CIA stream, it will skip the rest of the processing as shown in the pseudo code in Figure 4. Meanwhile, with the simplified and regular processing kernel, the initiation interval (II) of the kernel is 2 when implemented with Intel OpenCL. In contrast, the II of the crossbar based parallel processing kernels [15] goes up to 6 and it further requires the expensive crossbars and buffers. This is also a key factor that motivates us to adopt the reordering and batching approach despite the additional data padding.

C. Hyper pipelining

As mentioned in the above sub section, updating the level information of the active BFS frontier vertices are mostly random and time-consuming. While the classical BFS algorithm follows the bulk synchronous processing (BSP) model and the level update affects each BFS iteration, BFS performance is influenced. We notice that the level update can actually be postponed as long as the frontier vertices are kept. With this observation, we propose a hyper pipelining approach as shown in Figure 5. Basically, we separate the level update from the baseline BFS and overlap it within continuous BFS processing.

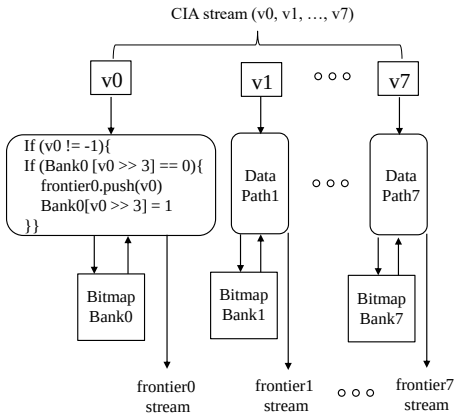

Fig. 4. Parallel visiting status bitmap buffer banks and conflict-free processing data paths

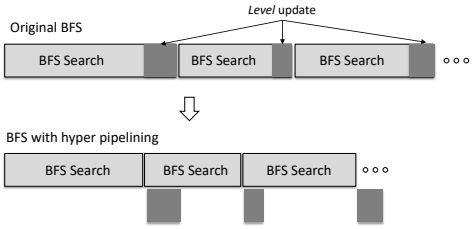

Fig. 5. Hyper pipelining

As the level update is fast compared to the overall BFS, it can either be processed on CPU or FPGA with negligible influence on the main BFS processing. Currently, we just put it on host processor to ensure that the main BFS accelerator implementation frequency is not affected.

D. Optimized BFS structure

With the above optimizations, the BFS structure is presented in Figure 6. The frontier inspection can be processed in parallel with the level update of previous BFS processing. The first two pipeline stages remain unchanged as they are not the performance bottleneck. In the third pipeline stage, random vertex status read and update over DDR is replaced with parallel on-chip buffer operations. In the last pipeline stage, level update is offloaded and hidden. Also we adopt the batch write to avoid random frontier vertices write to DDR. The irregular memory access in BFS is reduced dramatically.

IV. Experiments

In this section, we measure the overall performance of the optimized OpenCL based BFS accelerator on Intel Xeon-FPGA, a closely coupled CPU-FPGA architecture, over a set of representative graphs. Then we have an end-to-end comparison to the reference BFS design in Spector [19] which is the only available BFS accelerator that can be ported to Harp-v2 with reasonable design efforts to the best of our knowledge. Finally, we also analyze the graph reordering and batching comprehensively.

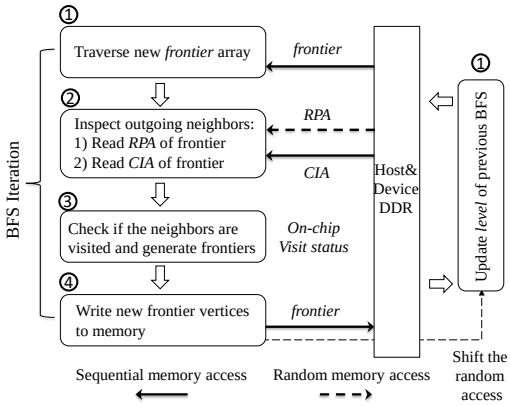

Fig. 6. Optimized BFS pipeline

TABLE I
Graph Benchmark

Name	# of vertex	# of edge	Type
YT [21]	1157828	2987624	Undirected
LJ [22]	4847571	68993773	Directed
PK [23]	1632804	30622564	Directed
R-MATI	524288	16777216	Directed
R-MATII	2097152	67108864	Directed

A. Experiment Setup

The graph benchmark used in this work includes three real-world graphs and two synthetic graphs generated using R-MAT model [20] as listed in Table I. The real-world graphs are from social network [21], [22], [23] while the R-MAT graphs are generated using the Graph 500 benchmark parameters ($A = 0.59, B = 0.19, C = 0.19$). To make the presentation easier, the five benchmark graphs are shorted as YT, LJ, Pokec, R-MATI, R-MATII respectively. We refer to an R-MAT graph with scale S (2^S nodes) and edge factor E ($E \times 2^S$). In the experiments, we averaged 64 BFS with random starting points. Note that trivial BFS will be removed from the test.

B. Performance evaluation

1) Overall performance: We use million traverse per second (MTEPS) as the performance metric. The performance of the proposed BFS accelerator i.e. OBFS on the graph benchmark is presented in Table II. It achieves up to 934.2 MTEPS on the R-MATII and 557.8 MTEPS on average. When compared to the reference BFS implementation in Spector, the proposed design shows 6.6X and 12.3X speedup on YT graph and R-MATI graph respectively while Spector fails on the other three graphs with larger data sets because of bugs in the code. In general, the proposed design performs better on R-MAT graphs with higher average degree. On the contrast, YT with low average degree suffers higher reordering and batching cost. In addition, YT is an undirected graph while the accelerator takes each edge as bidirected ones. Therefore, this also leads to relatively lower MTEPS in the experiments.

2) Performance comparison: On top of the comparison to the reference OpenCL design, we also compare this work to

TABLE II
PERFORMANCE SUMMARY

Benchmark	YT	LJ	Pokec	R-MATI	R-MATII
OBFS(MTEPS)	79.4	475.7	557.8	787.7	934.2
Spector(MTEPS)	12.01	-	-	64	-
Speedup	6.6X	-	-	12.3X	-

prior published BFS accelerators on FPGAs. As the platforms and graph benchmark used in these work are mostly different, it is difficult to make a completely fair end-to-end comparison. A summary of prior FPGA based BFS acceleration work is presented in Table III. The last column of the table shows the performance measurement and hardware design method. 'S' indicates that the performance is obtained by simulation. 'M' means that the performance is obtained by measuring execution on hardware. 'RTL' means the accelerator is implemented with RTL while 'OCL' means the accelerator is designed with OpenCL. SN in the table represents social network. fMRI stands for functional magnetic resonance imaging graphs.

Most prior work used RTL for the graph processing accelerator design despite the increasing popularity of the HLS design tools. In this work, we demonstrate that the OpenCL based BFS accelerator can achieve comparable performance to many prior handcrafted design and even outperforms some of them particularly on R-MAT graphs.

Compared to the state-of-art work in [24], OBFS remains slower for a few reasons. First, current OpenCL has many implementation constraints which limit the adoption of some optimization strategies. For instance, OpenCL does not allow on-chip buffer sharing between different kernels. As a result, the on-chip bitmap buffer can not be used by multiple parallel processing pipelines, which is not a problem in RTL design. Degree-aware optimization proposed in [25] is an effective BFS optimization used in [24]. This optimization is also orthogonal to this work, but sorting in the OpenCL based BFS pipeline leads to inefficient hardware implementation. Second, RTL design in [24] runs at 250 MHz, but OBFS ranges from 160 MHz to 200 MHz which will be illustrated in the rest of the experiments. Last but not least, DRAM simulator is used in [24] and it may also cause difference in terms of memory access efficiency. In summary, the gap can be alleviated with the continuous advancements of the OpenCL tools. Based on existing OpenCL tools, OBFS achieves good performance and gains the software-like features.

3) Optimization approach analysis: As mentioned, we mainly apply three different approaches including graph reordering, on-chip buffer partition, and hype pipelining to optimize the OpenCL based BFS accelerator. While the graph reordering and on-chip buffer partition are naturally combined, we have the two optimizations evaluated together. The performance improvement by gradually applying the optimizations to the BFS accelerator is presented in Figure 7. Note that the accelerators with different optimizations have 8Mb bitmap implemented. Batch size is set to be 16. It can be found that the graph reordering and on-chip buffer partition contributes most

TABLE III
PERFORMANCE OF BFS ACCELERATORS ON FPGA-DRAM PLATFORMS

System	Dataset Type	Bandwidth (GB/s)	Hardware Platform	MTEPS /FPGA	Design Method
Work[26]	SN	0.1	Virtex-5	160-790	S&RTL
Work[2]	fMRI	80	HC-2	62.5-650	M&RTL
CyGraph[1]	R-MAT	80	HC-2	420-550	M&RTL
Work[5]	R-MAT	3.2	Zedboard	90-255	M&RTL
FPGP[14]	SN	12.8	VC707	122	M&RTL
ForeGraph[3]	SN	19.2	VCU110	364-1069	S&RTL
Work[27]	R-MAT	12.8	Harp-v1	330-670	M&RTL
Work[24]	SN	19.2	Ultrascale+	1500-3500	S&RTL
Spector [19]	R-MAT	16	Harp-v2	64	M&OCL
Spector [19]	SN	16	Harp-v2	12	M&OCL
OBFS	R-MAT	16	Harp-v2	861	M&OCL
OBFS	SN	16	Harp-v2	371	M&OCL

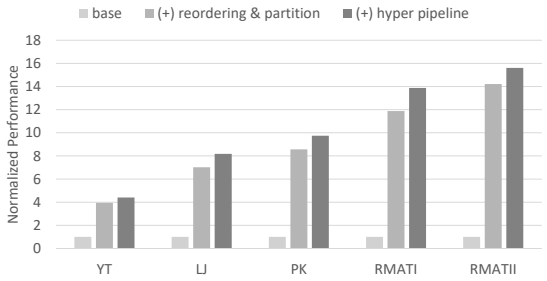

Fig. 7. Performance improvement with different high-level optimizations

to the overall performance improvement. 4X - 14X speedup over the base design is achieved on the graph benchmark. Hyper pipelining is also demonstrated to be beneficial. It improves the performance by 9.8% to 16.7%.

When different optimizations are applied to the OpenCL based BFS accelerator, the resulting FPGA implementation frequency is also different. Particularly, LJ has more than 4 million vertices and we set the bitmap buffer to be 8Mb. For the rest of the graphs with less than 4 million vertices, we set the bitmap to be 4Mb. Table IV shows the resulting accelerator frequency. It can be found that the proposed OBFS produces higher hardware implementation frequency which also contributes to the resulting performance improvement.

C. Batch trade-offs

Graph reordering and batching contributes most to the BFS performance improvement and will be analyzed in this subsection. Figure 8 shows the BFS performance of different batch sizes. In general, larger batch size induces higher performance and batch 16 achieves the best performance on all the benchmark graphs. BFS performance starts to drop when the batch size is larger.

TABLE IV
IMPLEMENTATION FREQUENCY OF THE BFS WITH DIFFERENT HIGH-LEVEL OPTIMIZATIONS

bitmap	base	(+) reordering&partition	(+)hyper pipelining
4Mb	123 MHz	155 MHz	202 MHz
8Mb	110 MHz	142 MHz	165 MHz

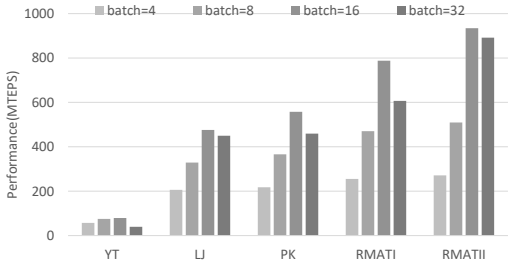

Fig. 8. BFS performance of different batch sizes

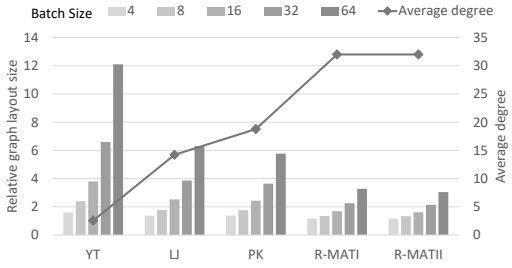

Fig. 9. Relative memory overhead of different batch sizes

The proposed graph reordering and batching approach expands the graph layout and it will induce more memory overhead. The batch overhead on the benchmark graphs is illustrated in Figure 9. When the batch size increases, the CSR data increases accordingly. Particularly, it can be seen that data replication overhead increases dramatically when the batch size reaches 32. As a result, the performance benefit is compensated by the additional data replication and memory access. While YT has the lowest average degree, it has the least performance speedup with larger batch size and drops dramatically at batch 32.

The advantages of the batch are multi-folded. In particular, batch mainly has the memory access coalesced and improves the memory bandwidth utilization, In combination with the bitmap, the irregular memory access reduces by more than 90% compared to the base design. Batched memory access has much lower average latency. To trade-off the batch overhead and the performance benefit, larger batch size is recommended for graphs with higher average degree and smaller batch size is better for graphs with lower degree such as YT.

V. Related work

The growing importance of efficient BFS traverse on large graphs have attracted attentions of many researchers. Different BFS optimization algorithms and accelerators have been proposed in literatures. We will particularly focus on the FPGA based BFS acceleration in this work.

To alleviate the memory bandwidth bottleneck of the BFS accelerators, the authors in [12], [28] explored the new Hybrid Memory Cube (HMC) which provides much higher memory bandwidth as well as flexibility for BFS acceleration, while the authors in [1] proposed to change the compressed sparse row (CSR) format slightly. Different from the first two work, the authors in [5] choosed to perform some redundant but sequential memory access for higher memory bandwidth utilization based on a spare matrix-vector multiplication model.

Most of the BFS accelerators are built on a vertex-centeric processing model, while the authors in [8] explored the edge-centeric graph processing and demonstrated significant throughput improvement. On top of the single FPGA board acceleration, the authors in [1], [2] also explored BFS acceleration on a FPGA based high performance computing system with multiple FPGAs and memory instances. There are also work exploring customized soft processors for graph processing and building a distributed solution on top of a group of embedded FPGA boards [29], [26].

The researchers opted to develop more general graph processing accelerator framework or library recently [7], [30], [24], [6], [3], [14]. They can also be utilized for BFS acceleration despite the lack of specialized optimization for BFS. Meanwhile, this is also a way to improve the ease of use FPGAs for graph processing acceleration.

Prior BFS acceleration work have demonstrated the potential benefits of accelerating BFS on FPGAs, but the accelerators were all handcrafted designs. Developing the accelerators with RTL takes long time and applying these accelerators on high level applications for software designers still requires a lot of efforts especially when the target computing platforms are different. On the contrast, OpenCL based BFS accelerators can be easily ported to diverse FPGA devices and easily utilized in high level applications by a *software designer*.

VI. Limitations and Conclusion

A. Limitations

In this work, we assume that the bitmap can be fully buffered on the FPGA on-chip RAM blocks. Basically, we limit the number of vertices in the graph that can be processed using this work. While the latest FPGAs has over 500Mb on-chip memory, we are supposed to handle graphs with hundreds of millions of vertices in theory on a single FPGA card. And we do not constrain the size of the edges in the graph. To handle larger graphs, we need to further explore the graph partitioning, while this work exhibits the efficiency of the BFS processing of a single graph partition.

B. Conclusion

Handcrafted BFS accelerators with HDL usually suffer high portability and maintenance cost as well as ease of use problem despite the relatively good performance. OpenCL based BFS accelerator can greatly alleviate these problems, but it is difficult to achieve satisfactory performance due to the inherent irregular memory access. In this work, we center the irregular memory access in BFS with a series of high-level optimizations. The resulting BFS can be implemented using OpenCL efficiently. Compared to a reference design in Spector benchmark, it achieves up to 12X performance speedup. When compared to the prior HDL based BFS accelerators on similar FPGA cards, the proposed OpenCL based BFS accelerator achieves competitive performance.

REFERENCES

[1] O. G. Attia, T. Johnson, K. Townsend, P. Jones, and J. Zambreno, "Cygraph: A reconfigurable architecture for parallel breadth-first search," in *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*. IEEE, 2014, pp. 228–235.

[2] B. Betkaoui, Y. Wang, D. B. Thomas, and W. Luk, "A reconfigurable computing approach for efficient and scalable parallel graph exploration," in *Application-Specific Systems, Architectures and Processors (ASAP), 2012 IEEE 23rd International Conference on*. IEEE, 2012, pp. 8–15.

[3] G. Dai, T. Huang, Y. Chi, N. Xu, Y. Wang, and H. Yang, "Foregraph: Exploring large-scale graph processing on multi-fpga architecture," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 217–226. [Online]. Available: http://doi.acm.org.libproxy1.nus.edu.sg/10.1145/3020078.3021739

[4] X. Ma, D. Zhang, and D. Chiou, "Fpga-accelerated transactional execution of graph workloads," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 227–236. [Online]. Available: http://doi.acm.org.libproxy1.nus.edu.sg/10.1145/3020078.3021743

[5] Y. Umuroglu, D. Morrison, and M. Jahre, "Hybrid breadth-first search on a single-chip fpga-cpu heterogeneous platform," in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*. IEEE, 2015, pp. 1–8.

[6] T. Oguntebi and K. Olukotun, "Graphops: A dataflow library for graph analytics acceleration," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 111–117.

[7] N. Engelhardt and H. K.-H. So, "Gravf: A vertex-centric distributed graph processing framework on fpgas," in *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*. IEEE, 2016, pp. 1–4.

[8] S. Zhou, C. Chelmis, and V. K. Prasanna, "High-throughput and energy-efficient graph processing on fpga," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, May 2016, pp. 103–110.

[9] D. Koch, F. Hannig, and D. Ziener, *FPGAs for Software Programmers*. Springer, 2016.

[10] Xilinx, "SDAccel," https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html, 2017, [Online; accessed 1-Sep-2017].

[11] Z. Wang, J. Paul, B. He, and W. Zhang, "Multikernel data partitioning with channel on opencl-based fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 6, pp. 1906–1918, 2017.

[12] J. Zhang, S. Khoram, and J. Li, "Boosting the performance of fpga-based graph processor using hybrid memory cube: A case for breadth first search," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2017, Monterey, CA, USA, February 22-24, 2017*, 2017, pp. 207–216. [Online]. Available: http://dl.acm.org/citation.cfm?id=3021737

[13] E. Nurvitadhi, G. Weisz, Y. Wang, S. Hurkat, M. Nguyen, J. C. Hoe, J. F. Martnez, and C. Guestrin, "Graphgen: An fpga framework for vertex-centric graph computation," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2014.

[14] G. Dai, Y. Chi, Y. Wang, and H. Yang, "FPGP: graph processing framework on FPGA A case study of breadth-first search," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, February 21-23, 2016*, 2016, pp. 105–110. [Online]. Available: http://doi.acm.org/10.1145/2847263.2847339

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

[16] C. Gui, L. Zheng, B. He, C. Liu, X. Chen, X. Liao, and H. Jin, "A survey on graph processing accelerators: Challenges and opportunities," *arXiv preprint arXiv:1902.10130*, 2019.

[17] X. Shi, Z. Zheng, Y. Zhou, H. Jin, L. He, B. Liu, and Q.-S. Hua, "Graph processing on gpus: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 81, 2018.

[18] P. Boldi, M. Rosa, M. Santini, and S. Vigna, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," in *Proceedings of the 20th international conference on World wide web*. ACM, 2011, pp. 587–596.

[19] Q. Gautier, A. Althoff, P. Meng, and R. Kastner, "Spector: An opencl fpga benchmark suite," in *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE, 2016, pp. 141–148.

[20] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining," in *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM, 2004, pp. 442–446.

[21] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *2012 IEEE 12th International Conference on Data Mining*, Dec 2012, pp. 745–754.

[22] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[23] L. Takac and M. Zabovsky, "Data analysis in public social networks," in *International Scientific Conference and International Workshop Present Day Trends of Innovations*, vol. 1, no. 6, 2012.

[24] P. Yao, L. Zheng, X. Liao, H. Jin, and B. He, "An efficient graph accelerator with parallel data conflict management," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. ACM, 2018, p. 8.

[25] J. Zhang and J. Li, "Degree-aware hybrid graph traversal on fpga-hmc platform," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2018, pp. 229–238.

[26] Q. Wang, W. Jiang, Y. Xia, and V. Prasanna, "A message-passing multi-softcore architecture on fpga for breadth-first search," in *Field-Programmable Technology (FPT), 2010 International Conference on*. IEEE, 2010, pp. 70–77.

[27] S. Zhou and V. K. Prasanna, "Accelerating graph analytics on cpu-fpga heterogeneous platform," in *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 2017, pp. 137–144.

[28] S. Khoram, J. Zhang, M. Strange, and J. Li, "Accelerating graph analytics by co-optimizing storage and access on an fpga-hmc platform," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2018, pp. 239–248.

[29] N. Kapre, "Custom fpga-based soft-processors for sparse graph acceleration," in *Application-specific Systems, Architectures and Processors (ASAP), 2015 IEEE 26th International Conference on*. IEEE, 2015, pp. 9–16.

[30] S.-W. Jun, A. Wright, S. Zhang, S. Xu *et al.*, "Grafboost: Using accelerated flash storage for external graph analytics," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 411–424.