OSCAR: Optimizing SCrAtchpad Reuse for Graph Processing

Shreyas G. Singapura [1], Ajitesh Srivastava[1], Rajgopal Kannan[2], and Viktor K. Prasanna[1]

[1]University of Southern California, Los Angeles, CA 90089
[2]Army Research Lab-West, Los Angeles, CA 90094
[1]{*singapur,ajiteshs,prasanna*}*@usc.edu*
[2]*rajgopal.kannan.civ@mail.mil*

***Abstract*—Recently, architectures with scratchpad memory are gaining popularity. These architectures consist of low bandwidth, large capacity DRAM and high bandwidth, user addressable small capacity scratchpad. Existing algorithms must be redesigned to take advantage of the high bandwidth while overcoming the constraint on capacity of scratchpad. In this paper, we propose an optimized edge-centric graph processing algorithm for scratchpad based architectures. Our key contribution is significant reduction in (slower) DRAM accesses through intelligent reuse of scratchpad data. We trade off reduction in DRAM accesses for slightly higher scratchpad accesses. However, due to the much higher bandwidth of scratchpad, the total memory access cost (DRAM + scratchpad) is significantly reduced. We validate our analysis with experiments on real world graphs using a simulator which mimics the scratchpad based architecture using Single Source Shortest Path (SSSP) and Breadth First Search (BFS). Our experimental results demonstrate $1.7\times$ to $2.7\times$ reduction in DRAM accesses leading to an improvement of $1.4\times$ to $2\times$ in total memory (DRAM + scratchpad) accesses.**

I. Introduction

Memory bandwidth has been lagging behind processors' ability to consume data. To reduce the gap between memory and processor performance, scratchpad based architectures [1], [2] have been proposed. Intel Knights Landing [1] is an example architecture using a DDR3/4 as DRAM and a 3D memory as scratchpad. Scratchpad provides higher bandwidth in comparison with DRAM while giving the control of data placement to the application developer. However, scratchpad is limited in terms of capacity and latency. Although it is much larger than cache, it is still smaller than DRAM and cannot replace it for large problem sizes. Further, the latency of accessing scratchpad is same as that of DRAM.

These new architectures are different from existing multi-level memory architectures. In existing architectures, processors can access only one level of memory hierarchy at a time. Scratchpad based architectures allow processors to access both DRAM and scratchpad in parallel. This hierarchy presents a new challenge for algorithm designers: scheduling data accesses to DRAM and scratchpad. Since the data brought into scratchpad will incur the overhead time of transfer between DRAM and scratchpad, a decision must be made whether to access the data directly from DRAM or bring it to scratchpad. If the data is brought into scratchpad, although it can be accessed at high bandwidth, it must be reused multiple times to make up for the additional transfer time between DRAM and scratchpad. If reuse of data is not possible, accessing data directly from DRAM would result in lower net access time than accessing from scratchpad. Therefore, the algorithm needs to be designed to decrease percentage of slow DRAM accesses and increase percentage of fast scratchpad accesses.

Graph processing is important for many real world applications [3]. There exists a large class of graph algorithms such as Breadth First Search (BFS), Single Source Shortest Path (SSSP) and Connected Components (CC) that are non-stationary in nature [4]. This implies that a different set of vertices are accessed in each iteration and the number of iterations vary with the input graph. These algorithms share several performance limiting characteristics such as little to no spatial and temporal locality and low ratio of computations to memory accesses. These properties result in frequent random accesses and low memory bandwidth causing memory access time to dominate the execution time [5], [6]. The high bandwidth of scratchpad makes it a good fit for these memory bound graph applications, while posing a challenge for handling large graph due to limited capacity. Therefore, existing algorithms must be redesigned to overcome the capacity limitation of scratchpad and achieve high performance for graph processing.

In this paper, we present an optimized edge-centric graph processing technique for scratchpad based architectures named OSCAR. Our technique trades off DRAM accesses for scratchpad accesses leading to a significant reduction in total memory access cost. It proceeds by bringing a subset of the graph from DRAM into scratchpad and applying edge-centric processing multiple times. By reusing scratchpad data, multiple DRAM accesses are replaced by one DRAM access and multiple scratchpad accesses, thus reducing the total memory cost. The main contributions of our work are:

- We prove that OSCAR reduces DRAM accesses compared to the state-of-the-art approach on traditional architectures.
- We identify performance bottlenecks as well as regions of high performance by varying the parameters of our technique.
- We show that OSCAR achieves $1.7\times$ to $2.7\times$ improve-

This material was supported by the NSF under Grant Number CNS-1643351.

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

ment in DRAM accesses compared with the state-of-the-art technique for BFS and SSSP on real world graph datasets resulting in a reduction of $1.4\times$ to $2\times$ in total memory access cost.

The outline of the paper is as follows. Section II covers related work on scratchpad based architectures and graph processing. Section III describes the target architecture and applications. Section IV presents the Baseline technique and its analysis. Section V presents our proposed optimized technique. Section VI presents the experimental results for various algorithms and Section VII concludes the paper.

II. Background and Related Work

Edge-centric Graph Processing: In Edge-centric graph processing [7], [8], [9], the edges are traversed in a predetermined fashion which can be matched with the data layout to achieve high memory bandwidth. The processing is iterative with scatter and gather phases. In scatter phase, each edge produces a message based on the source vertex to be consumed by the destination vertex of the edge. In the gather phase, all the vertices are updated based on their respective messages. Edge-centric graph processing trades redundant edges being traversed for regular accesses to memory resulting in higher memory bandwidth. Edge centric graph processing is described in detail in Section IV.

High Bandwidth Memory and Graph Processing: With the advent of high bandwidth memory such as 3D memory, there have been many works focusing on 3D memory as the main memory. In [10], [11], authors propose data layout optimizations in 3D memory for applications such as FFT, Image processing and Matrix Multiplication. There have been works focusing on graph processing on 3D memory as well. In [12], authors focus on processing in memory with 3D memory integrated with simple processors to accelerate graph processing. In [13], authors focus on accelerating edge-centric graph processing using FPGA with high bandwidth memory. All these works assume that the entire input is stored in high bandwidth memory and consider an architecture with DDR3/4 replaced with 3D memory. Although this assumption is valid for small problem sizes, they do not apply to real world graphs that are much larger than the capacity of 3D memory [14], [15]. In contrast, we target graph processing on scratchpad based architectures where only a part of the graph needs to fit in the scratchpad.

Work on Scratchpad Architectures: There have been few works targeting Scratchpad based architectures. In [16], the authors analyze the performance improvement for sorting using scratchpad. In [17], authors demonstrate the effectiveness of scratchpad memory for k-means clustering. In contrast to these works, we target graph applications which do not have predetermined access patterns and have input dependent random accesses. Therefore, scheduling data accesses between DRAM and scratchpad is more challenging for graph processing. In this paper, we propose an Optimized edge-centric graph processing technique which reduces DRAM accesses while increasing reuse of scratchpad. To the best of our knowledge, there is no existing work which discusses the application of graph processing on Scratchpad based architectures.

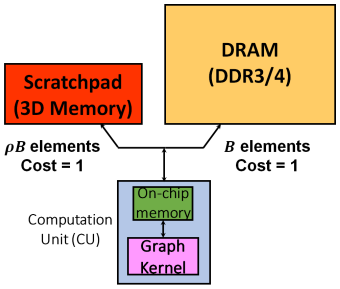


Fig. 1: Target Architecture

III. Target Architecture and Applications

A. Architecture

The target architecture consists of DRAM and scratchpad as multiple levels of main memory and a Computation Unit (CU) (Figure 1). The scratchpad model for a similar target architecture has been defined in [16], [17]. We describe the model and components of target architecture for completeness.

DRAM is connected to both scratchpad and CU. We assume the DRAM to be of size D, and a block of B units of data can be accessed in one time unit. **Scratchpad** is connected to DRAM and CU and has the same access time as DRAM for one block of data units. However, the granularity of access is a larger block of size $\rho B, (\rho > 1)$ and hence, results in a higher bandwidth. The parameter ρ is determined by the parameters of Scratchpad [14], [15]. The size of scratchpad (SP) is smaller than that of DRAM, i.e., $SP < D$. Therefore, data must be moved between scratchpad and DRAM in order to access the entire input data. The **Computation unit (CU)** in our target architecture consists of an application specific graph processing kernel and on-chip memory. As described in [7], [8], CPUs or FPGAs can be represented using the above model of CU. Although computation time varies with the platform, in edge-centric processing (Section IV) memory access time dominates the execution time. This is due to the fact that memory access pattern is regular in edge centric processing [7], [8] and graph applications are memory bound [5], [6]. Therefore, we focus on memory access time as the primary performance limiting factor and our analysis is agnostic towards the computing platforms.

B. Applications

In this paper, we target non-stationary graph algorithms [4] such as BFS, SSSP and CC, wherein, the number of active vertices vary with each iteration, and the total number of iterations is a function of the source and destination vertex of the algorithm. It is challenging to achieve high performance for these algorithms due to their random access patterns dependent on the active vertices of the iteration.

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

IV. Baseline Technique and Analysis

In this section, we develop performance upper bounds for edge-centric graph processing technique [7]. The parameters used in our analysis and their definitions are described in Table I. We assume that the given graph is directed. For and undirected graph, each edge (i, j) is replaced with two directed edges (i, j) and (j, i). This assumption does not affect our analysis.

TABLE I: Parameters for the Baseline Technique

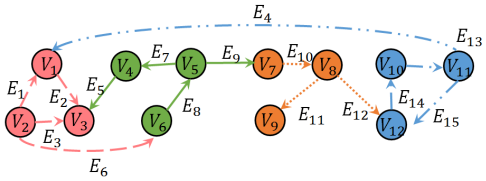
Notation	Definition
$G(V, E)$	Original graph of Vertices and Edges
$V_{P_i}, E_{P_i}, M_{P_i}$	Vertex, edge and message list of a partition P_i
d_1	# Iterations of accesses to DRAM
B	DRAM block size

We use edge-centric graph processing on architectures composed of a DRAM and a CU [7] as the Baseline technique. The input graph $G(V, E)$ is divided into partitions P_i and stored in DRAM. Each partition P_i has a vertex list (V_{P_i}), edge list (E_{P_i}) and a message list (M_{P_i}). Size of V_{P_i} is decided based on number of vertices that can fit in the on-chip memory of the CU. E_{P_i} consists of outgoing edges of vertices in V_{P_i} whereas, M_{P_i} consists of messages intended for incoming edges of vertices in V_{P_i}. Partition based edge-centric graph processing is illustrated in Technique 1.

Technique 1: Baseline Edge-centric Graph Processing

```
while ∃V_Pi ∈ G such that V_Pi is updated do
    //Scatter Phase:
    for each partition P_i of G do
1       Transfer V_Pi from DRAM to on-chip memory of CU
2       Stream E_Pi from DRAM
3       Process the V_Pi and E_Pi to produce M_Pj
4       Write M_Pj to P_j in DRAM based on destination vertex
    end
    //Gather Phase:
    for each partition P_i of G do
5       Transfer V_Pi from DRAM to on-chip memory of CU
6       Stream M_Pi from DRAM
7       Update V_Pi based on M_Pi
8       Write updated V_Pi to DRAM
    end
end
```

We illustrate the working of Technique 1 with the graph shown in Figure 2. First, V_{P_1} is transferred from the DRAM to on-chip memory of CU. Then, E_{P_1} is streamed from the DRAM and based on the target application, messages M_{1-3} and M_6 are produced on the respective edges. The messages are written to different partitions in DRAM based on their destination vertex; M_1, M_2, M_3 is written to P_1, whereas M_6 is written to P_2. This process is repeated for all the partitions ($P_1 - P_4$) of the graph to conclude the scatter phase. In the Gather phase, V_{P_1} is transferred from DRAM to on-chip memory of CU. Then, M_{P_1} is streamed from DRAM, and updated V_{P_1} is written to DRAM. The above steps are carried out for all the partitions to complete the gather phase. It



P_i	V_{P_i}	E_{P_i}	M_{P_i}
P_1	V_{1-3}	E_{1-3}, E_6	M_{1-5}
P_2	V_{4-6}	E_5, E_{7-8}	M_{6-8}
P_3	V_{7-9}	E_{10-12}	M_{9-11}
P_4	V_{10-12}	E_4, E_{13-15}	M_{12-15}

Fig. 2: Graph for Baseline Technique

should be noted that E_{P_i} does not change during the scatter and gather phases whereas, V_{P_i} is updated after each gather phase, and M_{P_i} is updated after each scatter phase. In this technique, scatter phase for all partitions must be performed before starting the gather phase of any partition. One scatter phase and one gather phase constitute one iteration of graph processing. The technique iterates until there are no updated messages. The number of iterations of accesses to DRAM is d_1.

A. Performance Analysis

We analyze the performance of Baseline technique using **Memory Access Cost** as the metric. It is defined as the total number of blocks accessed from DRAM.

We assume the graph applications are memory bound, i.e., computation time is significantly smaller in comparison with memory access time. In scatter phase, $|V| + |E|$ accesses to read vertices and edges, followed by $|E|$ accesses to write messages are made to the DRAM. In gather phase, $|V| + |E|$ accesses to read vertices and messages, followed by $|V|$ accesses to write back vertices are made to the DRAM. Algorithms such as BFS and SSSP require multiple such **iterations of accesses** to DRAM. Let d_1 be the number of iterations of accesses to DRAM. Assuming a DRAM block size of B data units, the total number of blocks accessed from memory in the Baseline technique is:

$$Baseline_{total\ blocks} = \left(\frac{3|V| + 3|E|}{B}\right) \cdot d_1$$

V. Optimized Technique: OSCAR

To utilize the high bandwidth of scratchpad, we propose to bring a set of partitions into scratchpad and reuse them to increase scratchpad accesses, thus, reducing DRAM accesses. Suppose the partitions of G used in Baseline technique are grouped into sets. We denote S_i as a set of partitions, which consists of a vertex list (V_{S_i}), an outgoing edge list (E_{S_i}), and an incoming message list (M_{S_i}) such that, $V_{S_i} = \cup_{P_j \in S_j}\{V_{P_j}\}$, $E_{S_i} = \cup_{P_j \in S_j}\{E_{P_j}\}$, and $M_{S_i} = \cup_{P_j \in S_j}\{M_{P_j}\}$. The number of partitions in S_i is determined by the accumulated size of V_{S_i}, E_{S_i}, and M_{S_i} that can fit in scratchpad. We assume the scratchpad memory can fit a large number of partitions, therefore, $V_{S_i} \gg V_{P_j}$. The list of local

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

edges whose source and destination are in V_{S_i} are denoted by $E^l_{S_i}$, whereas the list of remote edges with source in V_{S_i} and destination in V_{S_j} $(\forall j \neq i)$ are denoted by $E^r_{S_i}$. Therefore, $E_{S_i} = E^l_{S_i} \cup E^r_{S_i}$. The number of iterations of accesses to scratchpad for processing S_i is denoted by d_{S_i}. The additional parameters used in our analysis are listed in Table II.

Taking the same example described for the Baseline technique (Figure 2), we arbitrarily define the sets, $S_1 = (P_1, P_2)$, $S_2 = (P_3, P_4)$. For simplicity, we assume that the scratchpad is large enough to fit any of the sets. In the first step, we transfer S_1 with its (V_{S_1}, E_{S_1}) from DRAM to Scratchpad. Next, we process P_1 followed by P_2 using the edge-centric graph processing described in Technique 1. Partitions P_1 and P_2 are processed d_{S_1} times until there are no updates. It should be noted that while processing this set of partitions, the accesses are limited to the scratchpad and any messages to other sets will be processed by the respective sets later. Once the

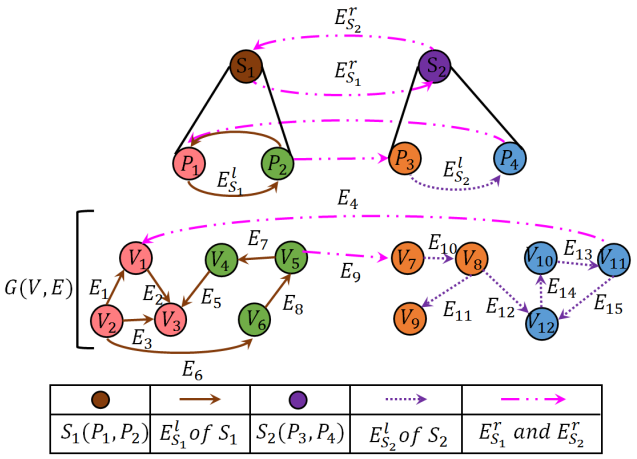


Fig. 3: Graph for Optimized Technique

partitions have no updates, updated vertices $V_1 - V_6$ are transferred back to DRAM. E_{P_1} and E_{P_2} need not be transferred and can be overwritten by the next set. Messages produced on local edges enclosed within S_1, i.e., (E_{1-3}, E_5, E_{7-8}), are already processed and can be overwritten by the next set. On the other hand, messages on remote edges (such as E_9 destined for S_2) are still not processed are moved to DRAM to be used when other sets are brought in. The above steps are repeated for all the other sets (in this case

Technique 2: Optimized Edge-centric Graph Processing

```
G(V, E): input graph with sorted edge list
while ∃V_Pi ∈ G such that V_Pi is updated do
    for each set S_i do
        Transfer S_i = (V_Si, E_Si) from DRAM to scratchpad
        while ∃V_Pi ∈ S_i such that V_Pi is updated do
            Run Edge-centric graph processing ∀P_i ∈ S_i
        end
        Transfer V_Si and E^r_Si to DRAM
    end
end
```

TABLE II: Additional Parameters for Optimized Technique

Notation	Definition
S_i	A set of partitions
$V_{S_i}, E_{S_i}, M_{S_i}$	vertices, edges and messages of a set S_i
$E^l_{S_i}$	Edges with source and destination in V_{S_i}
$E^r_{S_i}$	Edges with source in V_{S_i} and destination in V_{S_j}
ρB	Scratchpad block size
d_{S_i}	# Iterations of accesses to scratchpad for S_i
d_2	# Iterations of accesses to DRAM

only S_2). This completes one iteration of accesses to DRAM. But, messages on remote edges between sets such as E_4, produced by $P_i \in S_2$, which are destined for $P_j \in S_1$ are yet to be processed. Hence, the above procedure is repeated for multiple iterations of accesses to DRAM until no such remote messages remain. The total number of such iterations of bringing sets from DRAM to scratchpad is denoted by d_2. Our proposed **O**ptimized **SCrA**tchpad **R**euse (OSCAR) technique is illustrated in Technique 2.

Theorem. *Iterations of accesses to DRAM can be reduced by a factor of $d_1/d_2 \geq 1$ compared to Baseline technique.*

Since a set S_i contains multiple partitions, by doing multiple iterations of scatter and gather phases on a set, updates on $E^l_{S_i}$ propagate to all the partitions in S_i. Therefore, by fetching partitions in S_i from DRAM once, and performing multiple accesses to scratchpad, all the vertices in S_i can be processed. On the other hand, in the Baseline technique, partitions in S_i would have to accessed multiple times from DRAM to ensure all the vertices in S_i are updated. Therefore, by reusing data brought into scratchpad, OSCAR reduces iterations of accesses to DRAM. Accessing a set from DRAM once will ensure updates on $E^l_{S_i}$ are processed. However, multiple iterations of accesses to DRAM needs to be made for each set to ensure updates between sets on $E^r_{S_j}$ are processed. Therefore, OSCAR reduces iterations of DRAM accesses by a factor of $d_1/d_2 \geq 1$. Although the number of iterations can be as high as that of Baseline, in our experiments on real world graphs, we observed that this factor is much larger than 1 (See Table VI).

A. Analysis

Memory Access Cost for scratchpad based architectures is defined as the total number of blocks accessed from DRAM and scratchpad. In the first step of OSCAR, a set of partitions (S_i) is transferred from DRAM to scratchpad. The total number of accesses is equal to $|V_{S_i}|+|E_{S_i}|$. Next, edge centric graph processing is performed on S_i in the scratchpad (scatter and gather phases) with accesses limited to scratchpad. The edge centric processing is performed d_{S_i} number of times for S_i. Once there are no updates within S_i, V_{S_i} and updates to S_j are written back to DRAM. The above steps are performed for all S_i to complete one iteration of processing the graph and there will be d_2 iterations of accesses to DRAM to ensure

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

the messages between sets are processed. The total memory accesses for Baseline and OSCAR are summarized in Table III.

$$\textit{Read } S_i = \textit{Write } S_i = \left(\frac{|V_{S_i}| + |E_{S_i}|}{B}\right)$$

$$\textit{Processing } S_i = \left(\frac{3|V_{S_i}| + 3|E_{S_i}|}{\rho B}\right) \cdot d_{S_i}$$

$$\textit{Optimized}_{S_i} = \left(\frac{2|V_{S_i}| + 2|E_{S_i}|}{B} + \left(\frac{3|V_{S_i}| + 3|E_{S_i}|}{\rho B}\right) \cdot d_{S_i}\right)$$

Total memory cost for the optimized technique is

$$\textit{Optimized}_{\textit{total blocks}} \leq \left(\frac{2|V| + 2|E|}{B}\right) d_2 + \left(\frac{3|V| + 3|E|}{\rho B}\right) d_{S_i}^{max} \cdot d_2$$

TABLE III: Memory Access Comparison

Technique	# DRAM Blocks	# Scratchpad Blocks
Baseline	$\left(\frac{3\|V\| + 3\|E\|}{B}\right) \cdot d_1$	–
Optimized	$\left(\frac{2\|V\| + 2\|E\|}{B}\right) \cdot d_2$	$\left(\frac{3\|V\| + 3\|E\|}{\rho B}\right) \cdot d_{S_i}^{max} \cdot d_2$

B. State-of-art Edge Centric Processing

In non-stationary algorithms such as BFS and SSSP only a subset of vertices are active in an iteration. State-of-the-art edge-centric technique [9] uses a bitmap to store the status of updated partitions and accesses the partitions updated in previous iteration. Our analysis did not include partition skipping since the actual number of accesses in each iteration varies with the input and algorithm. However, in our experiments, we ensure that only the partitions with updates during the last iteration are accessed in both Baseline and OSCAR techniques.

VI. Experiments

A. Evaluation Methodology

3D memories have become popular recently, and there are several simulators for 3D memory [18], [19] as a stand alone memory. But, the concept of scratchpad based architecture is relatively new, and to the best of our knowledge, open source simulators are not available for such architectures. Therefore, to test the performance of OSCAR, we developed an in-house simulator to evaluate the performance of Baseline and OSCAR techniques. The scratchpad and DRAM memory is modeled based on the target architecture described in Section III. Our C++ based functional simulator emulates the target application for the input graph and enumerates the set of memory accesses. It works at the interface of the CU and DRAM/scratchpad. Since the on-chip memory in CU is used as a buffer in edge-centric processing [7], [8], we assume a single set of memory accesses from the CU. The simulator divides the set of memory accesses into blocks of size B for the DRAM and ρB for the scratchpad. Based on this list of block accesses, our simulator outputs various performance parameters including number of DRAM and scratchpad accesses, number of blocks accessed etc. Considering the sequential access pattern of edge-centric processing, consecutive blocks in DRAM and scratchpad can be prefetched to ensure the latency of accesses is negligible [7]. Therefore, the simulator ignores the latency of memory accesses.

1) Graph Representation: Each vertex consists of $\{attribute, ID\}$ fields to specify the application specific attribute and the vertex ID. Each edge $\{src, dst, weight\}$ fields to specify the source, destination and weight of the edge. The messages consist of $\{dst, value\}$ referring to the destination vertex and the update to the attribute of the vertex. Each field in a vertex/edge/message is defined as an integer of 4 bytes. We use real world graph datasets [20] listed in Table IV for our experiments. We compare the performance of OSCAR against Baseline technique for Breadth First Search (BFS) and Single Source Shortest Path (SSSP).

TABLE IV: Graph Datasets

Dataset	$\|V\|$	$\|E\|$	Category
Baidu	2.1 M	17.8 M	Hyperlink
LiveJ	4.8 M	68.5 M	Social
Wikipedia	18.3 M	172.2 M	Hyperlink

2) Architecture Parameters:

a) Partition Size: Assuming that on-chip memory of CU can fit m vertices, the graph is partitioned into $\lceil \frac{|V|}{m} \rceil$ partitions. The i^{th} partition maintains a vertex set with indices from $[i \times m]$ to $[(i+1) \times m - 1]$, $(0 < i < \lceil \frac{|V|}{m} \rceil)$. We choose $m = 2K$ in our experiments based on edge-centric processing on different CU platforms such as CPU and FPGAs [7], [8].

b) Set Size: Based on the number of active partitions and scratchpad size, we vary the number of partitions in the set such that each set can fit in the scratchpad.

c) DRAM and Scratchpad Size: The DRAM is assumed to be large enough to fit all the partitions in the input graph, whereas the scratchpad is assumed to be $(1/4)^{th}$ the size of DRAM based on the current sizes of DDR3/4 [21] and 3D memory [22]. We use a 64B DRAM block and 1KB scratchpad block based on DDR3 [21] and 3D memory [22].

B. Results

We present the results of our experiments in terms of performance gain and study the effect of choices made in our technique. Some of the results have been omitted due to brevity, as they produced similar trends compared to those presented here.

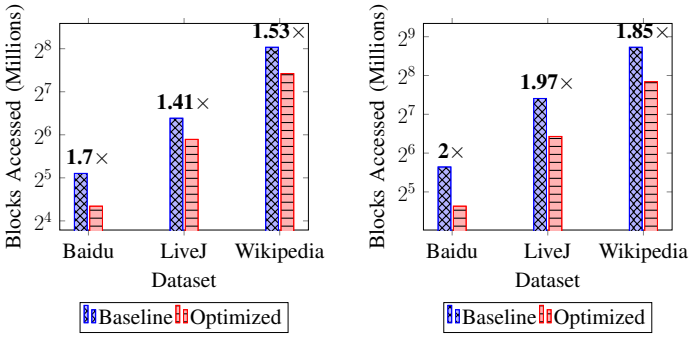
*1) **Total Memory Accesses (DRAM + Scratchpad)**:* We compare the total number of accesses made to DRAM and scratchpad by the Baseline and OSCAR techniques in Table V. We observe that OSCAR consistently outperforms the Baseline technique in terms of number of DRAM accesses with $1.7\times$ to $2.7\times$ improvement in number of DRAM memory accesses.

It can be observed that the total number of accesses from DRAM + scratchpad by OSCAR is higher than that of DRAM accesses by the Baseline. However, since the block size of scratchpad is higher than that of DRAM, in the same time span to access B units of data from DRAM, ρB units of data are available from scratchpad. Therefore, our approach reduces the total number of memory blocks accessed. Figure 4 shows

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

TABLE V: Elements Accessed

Algorithm	Dataset	DRAM		Scratchpad
		Baseline	OSCAR	OSCAR
BFS	Baidu	549.44 M	252.97 M	1.14 B
	LiveJ	1.34 B	0.78 B	2.63 B
	Wikipedia	4.2 B	2.21 B	8.47 B
SSSP	Baidu	798.72 M	297.51 M	1.58 B
	LiveJ	2.72 B	1.06 B	4.98 B
	Wikipedia	6.78 B	2.82 B	13.51 B



(a) BFS (b) SSSP

Fig. 4: Performance Comparison (# Blocks Accessed)

that OSCAR outperforms Baseline technique on all the three datasets for both BFS and SSSP.

*2) **Algorithm Iterations**:* The total number of iterations in the algorithm determines the number of times the graph or parts of the graph are accessed from the memory (DRAM + scratchpad). As shown in Table VI, the number of iterations of accesses to DRAM in OSCAR (d_2) is much smaller than that of Baseline (d_1) and results in a significant reduction in number of DRAM accesses. We believe this has the highest impact on performance improvement even though the sum of d_2 and maximum number of iterations of accesses to scratchpad ($d_{S_i}^{max}$) can be higher than that of DRAM in the Baseline technique. As we described earlier, we tradeoff accesses to DRAM for higher number of accesses to scratchpad since the bandwidth of scratchpad is ρ times that of DRAM, $\rho > 1$.

TABLE VI: Iterations for BFS

Dataset	DRAM		Scratchpad
	Baseline (d_1)	Optimized (d_2)	Optimized ($d_{S_i}^{max}$)
Baidu	21	10	12
LiveJ	15	8	9
Wikipedia	48	8	41

*3) **Scratchpad Size**:* In our experimental results, we assumed that scratchpad is $(1/4)^{th}$ the size of the DRAM. Here, we present the results of varying the scratchpad size relative to the DRAM size for a given dataset.

TABLE VII: Effect of Scratchpad Size on SSSP Performance

	Wikipedia			
Scratchpad Size	6.25%	12.5%	25%	50%
Improvement	1.45×	1.65×	1.85×	1.96×

As shown in Table VII, the performance improvement does not change significantly beyond a certain scratchpad size. This is due to the fact that we access only the active partitions in an iteration and if the total size of active partitions is smaller than the scratchpad size, increasing scratchpad size does not lead to performance improvement.

*4) **DRAM and Scratchpad Block Size (Bandwidth)**:* We analyzed the relationship between the DRAM, scratchpad bandwidth and performance. Scratchpad is $(1/4)^{th}$ of DRAM size. DRAM block size was varied from 64B to 256B while keeping the scratchpad block size at 1024B. Later, we fixed B = 64B and varied the value of ρ from 8 to 32. From

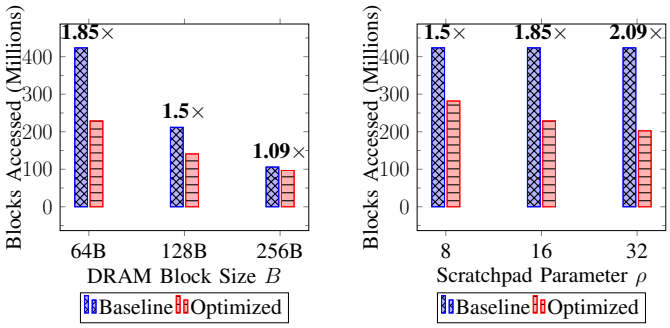


Fig. 5: Effect of B and ρ on SSSP Performance for Wikipedia

Figure 5, we observe that increasing B while maintaining a constant ρB improves the performance of both Baseline and Optimized techniques, but the improvement is marginal for large B. This is due to the fact that DRAM accesses are the bottleneck and for large B, both Baseline and Optimized technique will have similar performance. Similar trend can be seen by increasing ρB with B = 64B. However, we observe that performance of $(B, \rho B) = (128, 1024)$ for baseline technique is approximately equal to that of $(B, \rho B) = (64, 2048)$ for the optimized technique. Therefore, to achieve high performance, block sizes of both DRAM and scratchpad have to be increased. Increasing just one of them will not lead to significant performance improvement.

VII. Conclusion

We have presented an optimized technique called OSCAR for edge-centric graph processing on scratchpad based architectures. Our technique is applicable to non-stationary algorithms such as BFS and SSSP where a different set of vertices are accessed in each iteration and the number of iterations vary with the input graph. OSCAR reduces the total memory access cost by trading off DRAM accesses for higher scratchpad accesses. To utilize the high bandwidth of scratchpad, OSCAR brings a set of partitions of the graph into scratchpad and reuses them to increase scratchpad accesses, thus, reducing DRAM accesses. Experimental results on BFS and SSSP using real world graphs demonstrate 1.7× to 2.7× reduction in DRAM accesses and 1.4× to 2× reduction in total memory accesses against the state-of-the-art edge-centric graph processing technique.

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

REFERENCES

[1] Intel Knights Landing. https://www.hpcwire.com/2014/06/24/micron-intel-reveal-memory-slice-knights-landing/.

[2] Trinity. https://nnsa.energy.gov/mediaroom/pressreleases/trinity.

[3] Graph 500. http://www.graph500.org/.

[4] Zuhair Khayyat, Karim Awara, Amani Alonazi, Hani Jamjoom, Dan Williams, and Panos Kalnis. Mizan: a system for dynamic load balancing in large-scale graph processing. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 169–182. ACM, 2013.

[5] Andrew Lumsdaine, Douglas Gregor, Bruce Hendrickson, and Jonathan Berry. Challenges in parallel graph processing. *Parallel Processing Letters*, 17(01):5–20, 2007.

[6] Andrew Lenharth, Donald Nguyen, and Keshav Pingali. Parallel graph analytics. *Communications of the ACM*, 59(5):78–87, 2016.

[7] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. X-stream: edge-centric graph processing using streaming partitions. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 472–488. ACM, 2013.

[8] Shijie Zhou, Charalampos Chelmis, and Viktor K Prasanna. High-throughput and energy-efficient graph processing on fpga. In *Field-Programmable Custom Computing Machines (FCCM), 2016 IEEE 24th Annual International Symposium on*, pages 103–110. IEEE, 2016.

[9] Guohao Dai, Tianhao Huang, Yuze Chi, Ningyi Xu, Yu Wang, and Huazhong 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*, FPGA '17, pages 217–226, New York, NY, USA, 2017. ACM.

[10] Qiuling Zhu, Bilal Akin, H Ekin Sumbul, Fazle Sadi, James C Hoe, Larry Pileggi, and Franz Franchetti. A 3D-Stacked Logic-in-Memory Accelerator for Application-Specific Data Intensive Computing. In *3D Systems Integration Conference (3DIC), 2013 IEEE International*, pages 1–7. IEEE, 2013.

[11] Shreyas G Singapura, Rajgopal Kannan, and Viktor K Prasanna. On-chip memory efficient data layout for 2d fft on 3d memory integrated fpga. In *High Performance Extreme Computing Conference (HPEC), 2016 IEEE*, pages 1–7. IEEE, 2016.

[12] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 105–117. ACM, 2015.

[13] Jialiang Zhang, Soroosh Khoram, and Jing 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*, pages 207–216. ACM, 2017.

[14] Micron HMC. http://www.hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.1_20151105.pdf.

[15] HBM. https://www.jedec.org/sites/default/files/docs/JESD235A.pdf.

[16] Michael A Bender, Jonathan W Berry, Simon D Hammond, K Scott Hemmert, Samuel McCauley, Branden Moore, Benjamin Moseley, Cynthia A Phillips, David Resnick, and Arun Rodrigues. Two-level main memory co-design: Multi-threaded algorithmic primitives, analysis, and simulation. *Journal of Parallel and Distributed Computing*, 2017.

[17] Michael A Bender, Jonathan Berry, Simon D Hammond, Branden Moore, Benjamin Moseley, and Cynthia A Phillips. k-means clustering on two-level memory systems. In *Proceedings of the 2015 International Symposium on Memory Systems*, pages 197–205. ACM, 2015.

[18] John D Leidel and Yong Chen. HMC-Sim: A Simulation Framework for Hybrid Memory Cube Devices. *Parallel Processing Letters*, 24(04):1442002, 2014.

[19] Ke Chen, Sheng Li, Naveen Muralimanohar, Jung Ho Ahn, Jay B Brockman, and Norman P Jouppi. CACTI-3DD: Architecture-Level Modeling for 3D Die-Stacked DRAM Main Memory. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 33–38. EDA Consortium, 2012.

[20] Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM, 2013.

[21] Micron DDR3 Datasheet. https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/2gb_ddr3_sdram.pdf.

[22] Dong Hyuk Woo, Nak Hee Seong, Dean L Lewis, and Hsien-Hsin S Lee. An optimized 3d-stacked memory architecture by exploiting excessive, high-density tsv bandwidth. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12. IEEE, 2010.

978-1-5386-3472-1/17/$31.00 ©2017 IEEE