Accelerating PageRank using Partition-Centric Processing

Kartik Lakhota, Rajgopal Kannan, Viktor Prasanna
Ming Hsieh Department of Electrical Engineering, University of Southern California
{klakhoti, rajgopak, prasanna}@usc.edu

Abstract—PageRank is a fundamental link analysis algorithm and a key representative of the performance of other graph algorithms and Sparse Matrix Vector (SpMV) multiplication. Calculating PageRank on sparse graphs generates large amount of random memory accesses resulting in low cache line utilization and poor use of memory bandwidth. In this paper, we present a novel Partition-Centric Processing Methodology (PCPM) that drastically reduces the amount of communication with DRAM and achieves high memory bandwidth. Similar to the state of the art Binning with Vertex-centric Gather-Apply-Scatter (BVGAS) method, PCPM performs partition wise scatter and gather of updates with both phases enjoying full cache line utilization. However, BVGAS suffers from random memory accesses and redundant read/write of update values from nodes to their neighbors. In contrast, PCPM propagates single update from source node to all destinations in a partition, thus decreasing redundancy effectively. We make use of this characteristic to develop a novel bipartite Partition-Node Graph (PNG) data layout for PCPM, that enables streaming memory accesses, with very little generation overhead.

We perform detailed analysis of PCPM and provide theoretical bounds on the amount of communication and random DRAM accesses. We experimentally evaluate our approach using 6 large graph datasets and demonstrate an average 2.7× speedup in execution time and 1.7× reduction in communication, compared to the state of the art. We also show that unlike the BVGAS implementation, PCPM is able to take advantage of intelligent node labeling that enhances locality in graphs, by further reducing the amount of communication with DRAM. Although we use PageRank as the target application in this paper, our approach can be applied to generic SpMV computation.

I. INTRODUCTION

Graphs are a preferred choice for representing objects and their relationships in many real world applications like social networks, genomics, world wide web etc [1]. Recent years have witnessed explosive growth in the size of these graphs leading to substantial research interest in high performance graph analytics. A large fraction of this research is focussed on shared memory platforms because of their low communication overhead compared to distributed memory systems and the high capacity of DRAM which enables a single server to fit large graphs on main memory [2], [3]. However, obtaining high performance on graph algorithms is challenging even on a single node because of their low communication to computation ratio and irregular memory access patterns. The growing disparity between CPU speed and external memory bandwidth, termed memory wall [4], has become a key issue in the performance of graph algorithms.

PageRank is a quintessential algorithm that exemplifies the performance challenges posed by graph computations. It is also a fundamental link analysis algorithm [5] that iteratively performs Sparse Matrix Vector (SpMV) multiplication over the adjacency matrix of the target graph and the dense rank value vector. The irregularity in adjacency matrices leads to random accesses to the rank vector which comprise majority of the main memory traffic as shown in Fig.1. This is the major performance bottleneck for PageRank computation.

Recent works have proposed the use of Binning with Vertex-centric Gather-Apply-Scatter (BVGAS) methodology to improve locality and reduce communication for PageRank [6], [7], [8]. BVGAS exploits the 2 phased (scatter and gather) computation in GAS model by dividing the graph into partitions and allocating bin space for incoming updates (PageRank values) to each partition. By binning the updates, BVGAS increases spatial locality and reduces main memory traffic. However, these gains come at the cost of increased redundancy caused by read and write of same update value on multiple edges. Moreover, the vertex-centric nature of BVGAS results in random memory accesses and poor bandwidth utilization. In this paper, we present a new parallel methodology for computing PageRank (or SpMV in general) that overcomes these drawbacks to reduce DRAM communication and random accesses. The major contributions of our work are:

1) A novel Partition-Centric Processing Methodology (PCPM) that propagates updates from nodes to partitions instead of neighbors, thus reducing the redundancy associated with BVGAS.

2) A low cost bipartite Partition-Node Graph (PNG) data layout that enables PCPM to stream all updates to one destination partition at a time and sustain high memory bandwidth. We also propose branch avoidance techniques to further improve the performance and bandwidth utilization of PCPM.

3) Extensive analytical and experimental evaluation of parallel PCPM implementation on a shared memory system.

Fig. 1: Rank Value accesses constitute large fraction of DRAM traffic except in case of high-locality graphs like web
Results on 6 large datasets show that PCPM achieves $2.1 \times -3.8 \times$ speedup and $1.3 \times -2.5 \times$ reduction in communication over state of the art BVGAS method.

4) We use a node relabeling algorithm GOrder [9], to show that unlike BVGAS, PCPM can take advantage of an intelligent node labelling by further reducing the amount of communication, making it suitable even for high locality graphs.

PCPM also exhibits a higher energy efficiency achieved by the use of PNG data layout and resulting reduction in random memory accesses. Although we demonstrate the advantages of our approach on PageRank algorithm in this paper, it is applicable to generic SpMV computation.

The rest of the paper is organized as follows: Section II provides motivation and related work; Section III discusses BVGAS based implementation; Section IV describes PCPM along with PNG data layout and bandwidth optimizations; Section V gives a detailed analytical evaluation of PCPM; Section VI reports the experimental results and Section VII concludes the paper.

II. BACKGROUND AND RELATED WORK

A. Pagerank

PageRank is an important benchmarking algorithm that delineates the challenges associated with graph processing and SpMV computation [7]. It is also a fundamental graph algorithm used to rank nodes in the order of their importance. PageRank is computed iteratively and in each iteration, rank of all vertices is updated by the new weighted sum of their in-neighbors’ rank. Given a directed graph $G(V, E)$, equation 1 gives a mathematical representation of $i^{th}$ iteration of PageRank, where $V$ is the vertex set of the graph, $E$ is the set of edges in the graph, $d$ is the damping factor (typically set to 0.85), and $N_i$ and $N_o$ represent the sets of in-neighbors and out-neighbors of a node, respectively.

$$PR_{i+1}(v) = \frac{1 - d}{|V|} + d \sum_{u \in N_i(v)} \frac{PR_i(u)}{|N_o(u)|}$$

To assist visualization of some techniques and optimizations, we also use the Linear Algebraic abstraction in which a graph is represented by an adjacency matrix $A$ with $A_{ij} = 1$ if $(i, j) \in E$. From a Linear Algebra perspective, equation 1 for PageRank can be re-written as follows:

$$\overrightarrow{PR}_{i+1} = \frac{1 - d}{|V|} \cdot 1_{|V| \times 1} + A^T \cdot \overrightarrow{SPR}_i$$

where $\overrightarrow{PR}$ is the PageRank vector, $1_{|V| \times 1}$ represents an all ones column vector of size $|V|$ and $\overrightarrow{SPR}_i$ is the scaled PageRank vector obtained by dividing each element of $\overrightarrow{PR}_i$ by the number of non-zeros in corresponding row of $A$ i.e. $\overrightarrow{SPR}_i(v) = \frac{PR_i(v)}{|N_o(v)|}$. The most computationally intensive term that dictates the performance of computing this equation is the Sparse Matrix-Vector (SpMV) multiplication $A^T \cdot \overrightarrow{SPR}_i$. Henceforth, we will focus on the SpMV term to improve the performance of PageRank algorithm.

PageRank is typically computed in pull direction [3], [10], [11], [12] where each vertex accumulates the rank of its in-neighbors and computes its own rank as shown in alg. 1. This corresponds to traversing $A^T$ in a row-major fashion and computing dot product of each row vector with the scaled PageRank vector $SPR_i$. Since the iterative SpMV computation uses only $SPR_i$, we use the $PR[]$ array to store $SPR_i$ during the computation and $\overrightarrow{PR}$ while outputting final results.

Algorithm 1 Pull Direction PageRank (PDPR)

1: function PAGE-RANK($G(V, E)$, num_iterations)
2:     for $v \in V$ do
3:         $PR[v] = PR_{next}[v] = \frac{|V|^{-1}}{|N_o(v)|}$
4:     end for
5:     for num_iterations do
6:         for $v \in V$ do
7:             $temp = 0$
8:             for all $u \in N_i(v)$ do
9:                 $temp = temp + PR[u]$;
10:                $PR_{next}[v] = \frac{(1-d)\cdot|V|^{-1} + d \cdot temp}{|N_o(v)|}$
11:                swap($PR, PR_{next}$)
12:         end for
13:     end for

The pull direction methodology has an inherent advantage for parallel computing arising from the fact that each row completely owns the computation of its corresponding element in the output vector. This enables all rows of $A^T$ to be traversed asynchronously in parallel without the need of storing partial sums in main memory. On the contrary, in push direction, each node updates its out-neighbors by adding its own rank to them. This requires a column-major traversal of $A^T$, storage for partial sums since each column contributes partially to multiple elements in output vector and synchronization to ensure conflict-free processing of multiple columns that update the same output element.

Challenges in PageRank Performance: Sparse matrix layouts like Compressed Sparse Row (CSR) store all non-zero elements of a row sequentially in memory allowing fast row-major traversal of $A^T$ matrix. CSR also packs the rows tightly in a large array lending high spatial locality to consecutive row accesses. Thus, processing nodes in the order of their identifiers ensures efficient streaming access to the graph’s edges. However, as shown in fig. 2, the neighbors of a node (non-zero columns in adjacency matrix) can be scattered anywhere in the whole graph and reading their values can result in highly random accesses to the $SPR_i$ vector. Similarly, the push direction implementation suffers from random accesses to the partial sums vector. These low locality access patterns incur large number of cache misses and unused data within cache lines that results in poor bandwidth utilization and massive amount of DRAM traffic. In this paper, we propose a new implementation that overcomes the locality challenges and drastically reduces communication with main memory while sustaining high memory bandwidth.
Fig. 2: Access locations into scaled PageRank vector are derived from node adjacencies (non-zero columns in the corresponding row of $A^T$) and can be highly irregular.

Since nodes are stored sequentially in memory in the order of their IDs, node ordering has significant impact on access pattern locality and in turn, the performance of PageRank algorithm. We show that our approach can extract benefits from an intelligent node labelling by further reducing the amount of DRAM communication.

B. Related Work

The performance of PageRank (or SpMV) is bounded by the total DRAM communication which is largely determined by the locality in memory access pattern of the graph. Therefore, many prior works have investigated locality enhancement by the use of graph reordering or partitioning [13], [14], [15], [16]. Recently, a relabelling algorithm GOrder [9] was proposed to improve spatial locality. GOrder outperforms the well known partitioning and tree-based techniques. Such sophisticated algorithms provide significant speedup but also introduce substantial pre-processing overhead which limits their practicability. Other approaches like [17] and [2] propose the use of Hilbert curves to increase locality by rearranging the order of traversal of edges in the graph. However, space filling curves impose edge-centric programming and make it challenging to parallelize the application. In addition, scale-free graphs like social networks are less tractable by reordering transformations because of their skewed degree distributions [7].

Cache Blocking (CB) is another popular method used to induce locality by restricting random node accesses to an address range that can fit in cache [18], [19]. CB partitions the $A^T$ matrix along columns into multiple block matrices, each of which is stored using a CSR or COO representation. However, SpMV computation with CB requires the partial sums to be re-read for each block introducing extra work. In addition, the extremely sparse nature of these block matrices negatively impacts the locality of partial sum accesses [20].

Recently, [6] and [7] proposed the use of Binning with Vertex-centric Gather-Apply-Scatter (BVGAS) method for computing SpMV and PageRank, respectively. GAS is a popular model that has been widely implemented in many graph analytics frameworks [21], [22], [23]. It splits the graph analytic computation into 2 phases: scatter (propagate) updates and gather (accumulate) updates. The updates for PageRank algorithm correspond to scaled rank values and we will use the 2 terms interchangeably in the paper. By binning the updates in a semisorted manner, BVGAS enhances spatial locality in scatter phase and temporal locality during gather phase. BVGAS performs better than CB for sparse low-locality graphs and effectively reduces main memory traffic. However, it is inherently suboptimal because of multiple graph scans and redundant update reads/writes. Moreover, it does not optimize the data layout and incurs random DRAM accesses. We will discuss BVGAS in detail in section III.

III. BINNING WITH VERTEX-CENTRIC GAS (BVGAS)

The Vertex-centric GAS model computes PageRank in 2 phases: scatter and gather. In the scatter phase, each vertex transmits its scaled PageRank value (updates) along with the destination node identifiers (IDs) on all of its outgoing edges. In the gather phase, the propagated values for all edges are read and accumulated into the rank of corresponding destination node. Since the computation proceeds in push direction, it uses Compressed Sparse Row (CSR) storage format for the adjacency matrix $A$ for efficient row major traversal.

The binning technique works by dividing the graph into partitions of contiguous vertices and allocating bin space for each partition to store incoming updates. In the scatter phase, the updates are written at consecutive addresses in respective bins of the destination vertices as shown in fig. 3. When an update is inserted into a bin, the destination node ID is appended to it to inform the next phase about the target node. The number of partitions is kept small so that insertion points for all bins can fit in the cache, thus providing good spatial locality in the scatter phase.

Fig. 3: BVGAS scatters all PageRank values destined to nodes within a partition in its corresponding bin.
At the same time, the size of partition is selected in such a way that the range of vertices contained in it can be accommodated in cache. The gather phase that processes one bin at a time, thus enjoys good temporal locality with all random accesses to the partial sums being served by cache. Since the gather phase reads updates and destination IDs sequentially from a bin, it enjoys good spatial locality as well. Alg. 2 illustrates BVGAS based PageRank implementation.

The BVGAS method also leverages a deterministic layout for PageRank computation by exploiting the fact that the destination node identifiers remain same across iterations if the graph is traversed in same order. This halves the amount of writes during scatter phase since the destination node IDs are written only in the first iteration. Another optimization that is used in BVGAS is to collect updates destined to the bins in cached buffers before writing them into the main memory which ensures full cache line utilization during the scatter phase. BVGAS is the state of the art implementation of SpMV kernel and is more effective in reducing DRAM communication for low locality graphs, as compared to Cache Blocking or naive pull direction implementation.

![Fig. 4: PCPM decouples PageRank values and destination node identifiers to reduce communication from partition 3 to partition 1. Applying an update to multiple destinations within same partition does not require data transfer with DRAM technique from BVGAS. However, it replaces the vertex-centric nature of BVGAS with a partition-centric approach that decreases redundancy and random DRAM accesses.](image)

### Algorithm 2 PageRank using BVGAS

Let $m \rightarrow$ no. of nodes in a partition, $P \rightarrow$ set of partitions

1. function PAGE-RANK($G(V,E)$, num_iterations)
2.   $PR[:]=1/|V|
3.   for num_iterations do
4.       for $v \in V$ do
5.         $PR[v] = PR[v]/N_o(v)$
6.         for all $u \in N_o(v)$ do
7.           insert $(PR[v],u)$ into bins[$u/m$]
8.     $PR[:] = 0$
9.   for all partitions $p \in P$ do
10.      for all $(update\_dest)$ in bins[$p$] do
11.        $PR[dest] = PR[dest] + update$
12.     for all $v \in V$ do
13.        $PR[v] = \lceil\frac{1+d}{|V|}\rceil + d \times PR[v]$
14.     for $v \in V$ do
15.        $PR[v] = PR[v] \times |N_o(v)|$

However, these advantages come at the expense of redundant reads and writes of the same update value to multiple neighbors. This redundancy manifests itself in the form of BVGAS’ incapability to utilize high locality in graphs with optimized node labelling. Moreover, even though the scatter phase enjoys full cache line utilization, the selection of destination bin to write into is data dependent which results in random DRAM accesses and poor bandwidth utilization. Assembling updates in cached buffers introduces additional data dependent branches that worsen sustained bandwidth of BVGAS [7].

### IV. PARTITION-CENTRIC PROCESSING METHODOLOGY

To overcome the drawbacks of BVGAS, we propose a Partition-Centric Processing Methodology (PCPM). PCPM borrows the 2-phased computation concept and the binning


distribution of vertices.

| Propagate Updates on all Edges | Updates | Dest. ID |
|-------------------------------|---------|----------|
| Bin 1                         |         |          |
| PR[7]                         | 3       |
| PR[8]                         | 1       |
| PR[8]                         | 2       |

| Non-redundant updates only    | Updates | Dest. ID |
|-------------------------------|---------|----------|
| Bin 1                         |         |          |
| PR[7]                         | 3       |
| PR[8]                         | 1       |

Fig. 4: PCPM decouples PageRank values and destination node identifiers to reduce communication from partition 3 to partition 1. Applying an update to multiple destinations within same partition does not require data transfer with DRAM technique from BVGAS. However, it replaces the vertex-centric nature of BVGAS with a partition-centric approach that decreases redundancy and random DRAM accesses.

#### A. Redundancy Reduction

PCPM decouples the bin space allocated to store updates and destination node identifiers into separate update\_bins and destID\_bins, respectively. While destination IDs are propagated for each edge, updates are only transmitted once from a node to all the destination nodes in a partition, thus reducing the redundant writes. Fig. 4 illustrates the difference between PCPM and BVGAS scatter process for the example graph shown in fig. 3a. PCPM also uses a deterministic layout so that the destination IDs are only propagated once (preferably in the first iteration). This drastically reduces the number of writes during scatter phase because number of updates to be written is much less than the total edges.

PCPM manipulates the Most Significant Bit (MSB) of IDs in the destID\_bins to indicate the range of destination nodes that use the same update value. All node IDs in the range are written consecutively and the last ID’s MSB is set to 1, to mark the end of neighborhood of a source node. This reduces the size of vertex set of supported graphs from 4 billion to 2 billion for 4 Byte node IDs. However, to the best of our knowledge, this is still enough to support most of the large publicly available datasets.

The gather phase reads destID\_bins and updateID\_bins in a disjoint manner. It checks the MSB of destination IDs to determine whether to apply the previously read update or
to read the next value. The MSB is then masked to generate the true identifier of destination node whose partial sum is updated. Alg. 3 describes the scatter and gather functions for PageRank computation using PCPM with the modified deterministic layout. Note that the destination IDs are written in first iteration, which is not shown in alg. 3.

Although this method reduces propagation of redundant update values, it does not address the issues of reading unused edges and random writes during the scatter phase. Moreover, the manipulation of MSB in node identifiers introduces additional data dependent branches in both scatter and gather phases which hurts the sustained memory bandwidth.

### B. Bipartite Partition-Node Graph (PNG) Data Layout

Until now, we have used a column-wise partitioned CSR format to represent the adjacency matrix A in PCPM. In this section, we describe a new Bipartite Partition-Node Graph (PNG) data layout that rids PCPM of the redundant edge reads and random DRAM accesses. PNG also brings out the true partition-centric nature of PCPM by grouping the edges on the basis of destination partition to ensure that all updates are streamed to one partition at a time.

PNG exploits the fact that once the deterministic destID_bins are written, the only required information is the connectivity between nodes and partitions. Therefore, all edges going from a source to multiple destinations in a partition are compressed into a single edge whose destination is the partition number of the neighboring node. This gives rise to a bipartite graph consisting of two disjoint vertex sets - V and P, where P is the set of partitions in original graph G, as shown in fig. 5. Such a bipartisan division has the following effects:

1) Eff1 → the redundant unused edges are removed  
2) Eff2 → the range of destination IDs reduces from |V| to |P|.

The advantages of Eff1 are obvious but those of Eff2 will only become clear when we discuss the storage format for PNG.

The compression step reduces memory traffic by removing redundant edges from the graph but the issue of random DRAM accesses still persists. To tackle this problem, we transpose the adjacency matrix of the bipartite PNG. The resulting CSC matrix stores edges sorted by destination partitions which enables streaming updates to 1 partition at a time.

PNG is constructed on a per-partition basis i.e. every partition creates a separate bipartite graph to ensure that random accesses to source nodes are restricted within the range of vertices contained in a single partition. Thus, the scatter phase enjoys the same temporal locality as the gather phase.

The Eff2 is crucial for the transposition of bipartite graph in every partition. Note that the number of offsets required to store an adjacency matrix in CSC format is equal to the range of destination node IDs. If this range is |V|, a total of $\Omega(|V| \times |P|)$ memory is required to store PNG for all partitions. This is clearly prohibitive for graphs containing millions of vertices. On the other hand, Eff2 reduces this requirement to $\Omega(|P|^2)$ which is negligible as compared to number of edges even in large graphs.

Although PNG construction looks like a 2-step approach, the compression and transposition can actually be merged in the same step. We first scan the edges for each partition and individually compute the degree of all the destination partitions while discarding redundant edges. A prefix sum of these degrees is carried out to compute the offsets array for CSC matrix. In the next scan, the edge array in CSC format is filled with source vertices completing both compression and transposition. Furthermore, PNG construction can be easily parallelized over all partitions making the pre-processing very cost effective.

Algorithm 4 shows the modified pseudocode for scatter phase of PCPM using PNG layout. Mathematically, we denote PNG as a bipartite graph $G'(P, V, E')$. Note that unlike alg. 3 the scatter function in alg. 4 does not contain data dependent branches to check for redundant edges. The execution flow of alg. 4 is as simple as the naive Pull Direction PageRank (PDPR), which further boosts the sustained memory bandwidth. Overall, use of PNG with PCPM provides dramatic performance gains for the scatter phase with very little pre-

---

**Algorithm 3 PageRank using PCPM**

Let $m \rightarrow$ no. of nodes in a partition, $P \rightarrow$ set of partitions

1. function SCATTER($G(V, E), P, update\_bins$)
2.   for all $p \in P$
3.     for all $v \in p$
4.       $prev\_bin = \infty$
5.       for all $u \in N_o(v)$
6.         if $|u/m| \neq prev\_bin$ then
7.           insert $\{PR[v]\}$ into update\_bins$[u/m]$
8.       $prev\_bin = \lfloor u/m \rfloor$
9. function GATHER($G(V, E), P, update\_bins, destID\_bins$)
10. $PR[;] = 0$
11. for all partitions $p \in P$
12.   while $update\_Bins[p] \neq \emptyset$
13.     pop update\_Bins[p]
14.     $id$ from destID\_bins$[p]$
15.       while $MSB(id) \neq 1$
16.         $PR[id] = PR[id] + update$
17.     $id = id & bitmask$
18.     $PR[id] = PR[id] + update$
19.   for all $v \in V$
20.     $PR[v] = \frac{(1-d)/|V| + d \times PR[v]}{|N_o(v)|}$

---

Fig. 5: Bipartite PNG layout $G'(P, V, E')$ has lesser edges than the original graph (fig. 3a)
processing overhead.

Algorithm 4 PCPM scatter using PNG layout

Let \( G'(P, V, E') \rightarrow \) PNG, \( N^p_r(p') \rightarrow \) in-neighbors of partition \( p' \) in bipartite graph of partition \( p \)

1: function SCATTER(PNG(V, P, E'), update_bins)
2:   for all \( p \in P \) do
3:       for all \( p' \in P \) do
4:           for all \( u \in N^p_r(p') \) do
5:               insert \( (PR[u]) \) into update_bins[\( p' \)]

C. Branch Avoidance

Data dependent branches have been shown to have significant impact on performance of graph algorithms [24] and PNG largely reduces such branches in scatter phase of PCPM. In this subsection, we propose a branch avoidance mechanism for the gather phase that makes use of the modified identifiers in the deterministic layout of \( destID_{bins} \). At this point, it is worth mentioning that the pop operations shown in alg. [5] are implemented using pointers that increment after reading an entry from the respective bin. Let \( destID_{ptr} \) and \( update_{ptr} \) be the pointers to \( destID_{bins}[p] \) and \( update_{bins}[p] \), respectively. Note that the \( destID_{ptr} \) is always incremented whereas the \( update_{ptr} \) is only incremented if \( MSB[id] \neq 0 \).

To implement the branch avoiding gather function, we first change the parent loop to iterate till \( destID_{bins}[p] \neq 0 \) instead of \( update_{bins}[p] \neq 0 \) as shown in alg. [5]. Therefore, \( destID_{ptr} \) is incremented by 1 in every iteration. However, instead of using a condition check over \( MSB(id) \), we add the \( MSB(id) \) itself to \( update_{ptr} \). When \( MSB(id) \) is 0, the pointer is not incremented and the same update value is read in the next iteration; when \( MSB(id) \) is 1, the pointer is incremented executing the pop operation on \( update_{bins}[p] \). The modified pseudocode for gather phase is shown in alg. [5]

Algorithm 5 PCPM gather with Branch Avoidance

1: function GATHER(G(V, E), P, update_bins, destID_bins)
2:   \( PR[i] = 0 \)
3:   for all partitions \( p \in P \) do
4:       \( destID_{ptr} = update_{ptr} = 0 \)
5:       while \( destID_{ptr} < size(destID_{bins}[p]) \) do
6:           \( id = destID_{bins}[p][destID_{ptr} + ] \)
7:           \( update = update_{bins}[p][update_{ptr}] \)
8:           \( update_{ptr} = update_{ptr} + MSB(id) \)
9:           \( PR[id & bitmask] = PR[id & bitmask] + update \)
10:      for all \( v \in V \) do
11:          \( PR[v] = \frac{(1-d)\Delta v + d \times PR[v]}{N_v(v)} \)

Branch avoidance does not affect the amount of communication but increases the sustained memory bandwidth and in turn, the performance of PCPM based PageRank implementation.

V. ANALYTICAL EVALUATION

In this section, we present analytical models to compare the performance of conventional Pull Direction PageRank (PDPR) and BVGAS and PCPM based PageRank implementations. Our models provide valuable insights into the behavior of different methodologies with respect to varying graph structure and locality. Table I describes the parameters used for the analysis. We assume the graph to be stored in CSR format for PDPR and BVGAS and use a synthetic kronecker graph \( G \) of scale 25 \((kron)\) as an example for illustration purposes.

A. Communication Model

Communication is the major limiting factor for performance of PageRank. In this subsection, we compare different programming methodologies on the basis of amount of DRAM communication (in Bytes) performed per iteration of PageRank as a function of the parameters listed in Table I. We assume the graph to be in CSR format for DRAM access. We use a deterministic layout for BVGAS and PCPM so that destination indices are written only in the first iteration and hence, not accounted for in our model. We also assume full cache line utilization in scatter phase of BVGAS.

PDPR: As given in alg. [1] the pull technique scans all edges in the graph once. For a CSR representation, this requires reading edge offsets and source node indices generating \((n+m)\) Bytes of data transfer. For each vertex, the updated values are outputted using \( n_d \) Bytes of writes to DRAM. Summing all individual contributions, we get the total memory traffic for PDPR as:

\[
PDPR_{comm} = m(d_i + c_{m,r}) + n(d_i + d_v)
\]

BVGAS: The pseudocode given in alg. [2] scans the graph and writes updates for outgoing edges of every source node, thus communicating \((n+m)d_i + (n+m)d_v\) Bytes of data. The gather phase loads updates and destination node IDs on all the edges and accumulates them into the new rank values generating \(m(d_i + d_v)\) Bytes of read traffic. After a bin is processed, it writes \( n_d \) Bytes of new rank values into the DRAM. We ignore the apply phase communication since it can be merged with gather phase by completing computation of \( PR[v] \) \(\forall v \in p\) before writing it into main memory. Total DRAM communication for BVGAS is therefore, given by:

\[
BVGAS_{comm} = 2m(d_i + d_v) + n(d_i + 2d_v)
\]

PCPM with PNG: PNG consists of individual bipartite graphs

| Original Graph \( G(V, E) \) | PNG layout \( G'(P, V, E') \) |
|-----------------------------|-------------------------------|
| \( n \) no. of vertices \( |V| \) | \( k \) no. of partitions \( |P'| \) |
| \( m \) no. of edges \( |E| \) | \( r \) compression ratio \( |E'|/|E| \) |

| Architecture | Software |
|--------------|----------|
| \( c_{m,r} \) cache miss ratio for source value reads in PDPR | \( d_v \) sizeof (rank value element) |
| l sizeof (cache line) | d_i sizeof (node or edge index) |
for every partition, each of which is a CSC matrix with \( k \) edge offsets and partition indices for outgoing edges. Number of edges in PNG( \( |E'| \) ) is given by \( m/r \). Therefore, one scan of PNG reads \( (k^2 + m/r)d_i \) Bytes from main memory. In scatter phase(alg. 4), updates are propagated from \( V \) on \( E' \) resulting in \( nd_v \) Bytes of read and \( md_v/r \) Byte writes. The gather phase(alg. 5) reads \( m \) destination IDs and \( m/r \) updates followed by writing of \( n \) new PageRank values. Overall, the net amount of data that PCPM communicates is:

\[
PCPM_{comm} = m \left( d_i \left( 1 + \frac{1}{r} \right) + \frac{2d_v}{r} \right) + k^2d_i + 2nd_v \tag{5}
\]

**Comparison:** The performance of pull technique depends heavily on \( c_{mr} \). In worst case, all accesses are cache misses i.e. \( c_{mr} = 1 \) and in best case, only cold misses are encountered to load the PageRank values in cache i.e. \( c_{mr} = nd_v/mr \). Therefore, we can say that \( PDPR_{comm} \in [md_i, m(d_i + l)] \), considering \( n \ll m \). On the other hand, communication for BVGAS stays relatively constant. Analyzing equations [5] and [4] we see that BVGAS reduces communication when:

\[
c_{mr} > \frac{d_i + 2d_v}{l} \tag{6}
\]

BVGAS is inherently suboptimal since it performs \( O(m) \) extra loads and stores and can never reach the lower bound of \( PDPR_{comm} \). Comparatively, PCPM offers a lower bound on \( c_{mr} \), becoming advantageous when:

\[
c_{mr} > \frac{d_i + 2d_v}{r \cdot l} \tag{7}
\]

We ignore the \( k^2 \) term since \( k^2 \ll n \ll m \). In the worst case when there is no compression i.e. \( r = 1 \), PCPM is still as good as BVGAS. PCPM reaches optimality when outgoing edges for every vertex can be compressed into a single edge i.e. \( r = m/n \). Plugging these values in equation [5] we get \( PCPM_{comm} \in [md_i, m(2d_i + 2d_v)] \). By reaching the same lower bound as pull technique, PCPM gets rid of the inherent suboptimality of BVGAS.

At first glance, equations [6] and [7] give an impression that both BVGAS and PCPM are beneficial only for low locality graphs. With optimized node labelling, we can improve \( c_{mr} \) and outperform both the techniques. However, this is true only for BVGAS and not for PCPM. This is because the compression ratio \( r \) is also a function of locality. With an intelligent node labeling, \( r \) increases, further tightening the inequality constraint on \( c_{mr} \). As a result, unlike BVGAS, PCPM is advantageous even for high locality graphs.

Fig. 6 shows the effect of \( r \) on predicted DRAM communication for the \( kron \) graph with partition size 256 KB (64K nodes). Obtaining an optimal node labeling that makes \( r = m/n \) might be very difficult or even impossible for some graphs. However, as can be observed from fig. [6], DRAM traffic decreases rapidly for \( r \leq 5 \) and converges slowly for \( r > 5 \). Therefore, a node reordering that can achieve \( r \approx 5 \) is good enough to optimize communication.

Furthermore, in equation [5] we observe that the advantages of BVGAS reduce dramatically if \( d_v \) is increased, say, by using 8 Byte double instead of 4 Byte floating point numbers for PageRank values. However, in equation [7] this penalty is scaled by a factor of \( 1/r \) which allows PCPM to maintain large performance gains over pull technique even in the case of 8 Byte long PageRank values. Nevertheless, for the sake of comparison with prior works, we will use 4 Byte floating point representation for rank values in the rest of the paper.

**B. Random Memory Accesses**

We define a random access as a non-sequential jump in the address of memory location being read from or written to DRAM. Random accesses can incur latency penalties and negatively impact the sustained memory bandwidth. In this subsection, we model the amount of random accesses performed by different methodologies in 1 PageRank iteration.

**PDPR:** Reading edge offsets and source node IDs in pull technique is completely sequential because of CSR format. However, all accesses to source node rank values that are served by DRAM can potentially be random resulting in:

\[
PDPR_{ra} = O(m c_{mr}) \tag{8}
\]

**BVGAS:** In scatter phase of alg. [2] updates are inserted into bins chosen on the basis of destination node index which can be arbitrary. Therefore, BVGAS generates a total of \( O(m) \) random writes. However, it collects these updates in cached buffers of size \( l \) before writing them into the main memory. Therefore, for every \( l \) Bytes written, there is at most 1 random memory access performed. The gather phase sequentially reads the bins generating only 1 random access per bin. Hence, total random DRAM accesses for BVGAS are given by:

\[
BVGAS_{ra} = O \left( \frac{md_i}{l} + k \right) = O \left( \frac{md_i}{l} \right) \tag{9}
\]

**PCPM:** For the PNG layout, since updates destined to same partition are produced consecutively, scattering updates from 1 partition generates a maximum of \( k \) random accesses. Accesses to nodes within cached partition are not served by main memory and hence, do not add to random DRAM accesses. Scanning bipartite graphs in CSC format and gathering updates (similar to BVGAS) generates 1 random access per partition. Since there are \( k \) such partitions, overall number of
random accesses in PCPM is bound by:

\[ PCPM_{ra} = O(k^2 + k) = O(k^2) \]  

\[ (10) \]

**Comparison:** Similar to communication model, random DRAM accesses for pull technique are heavily dependent on \( c_{mr} \). For low locality graphs, BVGAS_{ra} is lesser than PDPR_{ra} and is constant. However, PCPM_{ra} is much smaller than BVGAS_{ra} given \( k^2 \ll m \), for example, in the kron dataset with \( \delta_v = 4 \) Bytes, \( l = 64 \) Bytes and partition size 256 KB, BVGAS_{ra} \approx 60.9 million whereas PCPM_{ra} \approx 0.26 million. Moreover, the number of random accesses in BVGAS is proportional to \( \delta_v \) and hence, using double precision numbers can further worsen the sustained memory bandwidth. On the other hand, PCPM_{ra} is independent of the size of data type used to represent rank values.

It is worth mentioning that the number of data dependent unpredictable branches in BVGAS is also \( O(m) \), since for every update written, the scatter function has to check if the corresponding cached buffer is full. In contrast, the number of branch mispredictions for PCPM (using branch avoidance) is \( O(k^2) \) with 1 misprediction for every destination partition \( (p') \) switch in alg. 4. This enables PCPM to utilize available bandwidth much more efficiently than BVGAS. The derivations are similar to random access model and for the sake of brevity, we will not provide a detailed deduction.

### VI. Evaluation

#### A. Experimental Setup

We conducted our experiments on a dual-socket Sandy Bridge server equipped with Intel Xeon E5-2650 v2 processors@2.6 GHz running Ubuntu 14.04 operating system. Table II lists some important properties of our machine. Memory bandwidth is measured using STREAM benchmark [25].

All codes are written in C++ and compiled using G++ 4.7.1 with the highest optimization -O3 flag. The programs are parallelized using OpenMP and executed on all 16 cores. The memory statistics are collected using Intel Performance Counter Monitor [26].

#### B. Datasets

For performance evaluation, we use 6 large real world and synthetic graph datasets coming from different applications. Table III summarizes the size and sparsity characteristics of these graphs. Gplus and twitter are follower graphs on social networks; pld, web and sd1 are hyperlink graphs obtained by web crawlers; and kron is a scale 25 graph generated using Graph500 Kronecker generator. The web is a very sparse graph but has high locality and the kron has higher edge density as compared to other datasets.

| Dataset | Description | # Nodes (M) | # Edges (M) | Degree |
|---------|-------------|-------------|-------------|--------|
| gplus [27] | Google Plus | 28.94 | 462.99 | 16 |
| pld [28] | Pay-Level-Domain | 42.89 | 823.06 | 14.53 |
| web [25] | Webbase-2001 | 118.14 | 992.84 | 8.4 |
| kron [11] | Synthetic graph | 33.5 | 1047.93 | 31.28 |
| twitter [29] | Follower network | 61.58 | 1468.36 | 23.84 |
| sd1 [25] | Subdomain graph | 94.95 | 1937.49 | 20.4 |

#### C. Implementation Details

We use a simple hand-coded implementation of alg. 1 for pull direction computation and parallelize it over all vertices using the OpenMP pragma **parallel for**. Such an implementation does not have any overheads typically associated with graph analytics frameworks [3], [12], [11] and hence, is faster than framework based programs [7].

For BVGAS implementation, we use a number of optimizations specified in [7]. To parallelize BVGAS, we give each thread its own set of bins for different partitions and assign each thread a fix range of nodes with static load balancing. We use the Intel AVX non-temporal store instructions [30] to bypass the cache while writing updates and use cache line aligned buffers to accumulate the updates for streaming stores. In gather phase, each thread processes all bins destined to a partition at a time to avoid the need of atomics, and load balancing is done using OpenMP dynamic scheduling. The optimal partition size is empirically determined as 256 KB (64K nodes) and buffer size in scatter phase is set to 2 cache lines for each destination partition.

The PCPM scatter function is parallelized over partitions since each partition processes its own bipartite graph in the PNG layout. The gather function parallelization is similar to BVGAS. Load balancing in both the phases is done dynamically using OpenMP. Partition size is empirically optimized and set to 256 KB. A detailed design space exploration of PCPM will be discussed later in section VI-D2.

A pre-processing step is used to allocate bin spaces for each thread in BVGAS and each partition in PCPM to avoid atomicty issues. All the programs mentioned in this section use 16 concurrent threads and execute 20 iterations of PageRank. For the accuracy of information, we repeat these algorithms 5 times and report the average values.

#### D. Results

1) **Comparison with Baselines:** In this section we provide the outcomes of extensive evaluation of PCPM against the Pull Direction PageRank (PDPR) and BVGAS baselines, based on various criterion.

**Overall Performance:** We use GTEPS as the main metric for performance evaluation. GTEPS is computed as the ratio of giga edges in the graph to the runtime of single PageRank iteration. We observe that PCPM is \( 2.1 - 3.8 \times \) faster than the state of the art BVGAS implementation and \( 1 - 4.1 \times \) faster than the pull technique (fig. 7). BVGAS achieves constant throughput irrespective of the graph structure and is able to accelerate computation on low locality graphs. However, it is worse than the pull direction baseline for high locality (web)
As an additional consequence, PCPM memory traffic per edge reduces the communication in both scatter and gather phases. This is because unlike BVGAS, PCPM propagation updates only on non-redundant edges which drastically reduces the communication in both scatter and gather phases. As an additional consequence, PCPM memory traffic per edge for web and kron is lower than other graphs because of their high compression ratio (table V). The normalized communication for BVGAS is almost constant and therefore, its utility depends on how efficient the pull direction baseline utilizes very few instructions and therefore, has better bandwidth utilization.

The reduced communication in PCPM also enhances its energy efficiency resulting in lower $\mu J/edge$ consumption as compared to BVGAS and the pull technique, as shown in fig. 10. Energy efficiency is particularly important from eco-friendly computing perspective as highlighted by the Green Graph500 benchmark [31]. We observe that the improvement in energy efficiency exceeds the communication reduction over baselines. This is because, just like the execution time, DRAM and BVGAS, respectively. For large graphs like sd1, PCPM achieves 45.4 GBps which is 77% of the peak read bandwidth (table I) of our system. Although both PDPR and BVGAS suffer from random memory accesses, the former executes very few instructions and therefore, has better bandwidth utilization.

### TABLE IV: Execution time per iteration of PDPR, BVGAS and PCPM averaged over 20 iterations

| Dataset | PDPR | BVGAS | PCPM |
|---------|------|-------|------|
| gplus   | 0.64 | 0.66  | 0.67 |
| pld     | 0.68 | 0.68  | 0.68 |
| web     | 0.21 | 0.21  | 0.21 |
| kron    | 0.65 | 0.65  | 0.65 |
| twitter | 1.83 | 1.83  | 1.83 |
| sd1     | 1.97 | 1.97  | 1.97 |

### TABLE V: Locality vs compression ratio

| Dataset | #Edges in Graph | #Edges in PNG | $r$ | #Edges in PNG | $r$ |
|---------|----------------|---------------|-----|---------------|-----|
| gplus   | 463            | 243.8         | 1.9 | 157.4         | 2.94|
| pld     | 623.1          | 347.7         | 1.79| 166.7         | 3.73|
| web     | 992.8          | 118.1         | 8.4 | 126.8         | 7.83|
| kron    | 104.8          | 342.7         | 3.06| 109.7         | 6.17|
| twitter | 1468.4         | 722.4         | 2.03| 386.2         | 3.8 |
| sd1     | 1937.5         | 976.9         | 1.98| 366.2         | 5.29|

Note that the speedup obtained by PCPM is larger than the reduction in communication. This is because by reducing random memory accesses and data dependent branches, PCPM is able to efficiently utilize the available DRAM bandwidth. As shown in fig. 9, PCPM can sustain an average 42.4 GBps bandwidth compared to 33.1 GBps and 26 GBps of PDPR.
power consumption also benefits from the streaming memory access patterns obtained by using PNG layout.

**Impact of Locality on Communication:** To assess the effect of locality on different methodologies, we relabel the nodes in our graph datasets using a recently proposed GOrder algorithm. GOrder increases spatial locality by placing nodes with common in-neighbors closer together in the memory. As a result, outgoing edges of the nodes tend to be concentrated in few partitions in PCPM which significantly enhances the compression ratio $r$ as shown in table [V]. We refer to the original node labeling in graph as $\text{Orig}$ and GOrder computed labeling as simply $\text{GOrder}$.

Because of the power law degree distribution of the datasets, even $\text{Orig}$ achieves a high average value for $r$ (2.2 excluding the web graph) as the high degree nodes tend to have many neighbors in each destination partition. By improving the spatial locality, $\text{GOrder}$ significantly improves $r$ (table [V]) which further reduces DRAM traffic for most of the datasets. However, the web graph exhibits large compression ($r = 8.4$) with $\text{Orig}$ and does not show any improvement with $\text{GOrder}$.

Table [VI] shows the impact of $\text{GOrder}$ on DRAM communication. Note that BVGAS communicates a constant amount of data for a given graph irrespective of the labeling scheme used. On the contrary, memory traffic generated by PDPR and PCPM decreases because of reduced $c_{narr}$ and increased $r$, respectively. These observations are in complete accordance with the performance models discussed in section [V-A]. The communication reduction for PCPM is not as dramatic as PDPR because when $r$ becomes greater than a threshold, PCPM enters the slow convergence region shown in fig. [6]. Nevertheless, for almost all of the datasets, the net data transferred in PCPM is remarkably lesser than both PDPR and BVGAS for either of the vertex labelings. This makes PCPM a preferred choice for implementing PageRank, irrespective of the graph locality.

2) **PCPM Design Space Exploration:** Fig. [11] shows the individual effects of various inbuilt optimizations in PCPM on the execution time of PageRank normalized against the BVGAS baseline. The Redundancy reduction (section [V-A]) step provides significant speedup averaging $1.2 \times$ (excluding the web graph), over the BVGAS method. The web graph has a very high compression ratio which results in $2.4 \times$ acceleration just by avoiding redundant update transfers. This also shows that redundancy reduction is extremely crucial for graphs with high locality access patterns.

For the other graphs, a major contribution to the performance comes from the use of PNG data layout (section [IV-B]) which gives an average $1.8 \times$ speedup. This is because use of PNG further reduces communication by avoiding redundant edge traversals and also improves bandwidth utilization to dramatically accelerate the scatter phase of PCPM. The branch avoidance mechanism (section [IV-C]) improves sustained memory bandwidth in the gather phase of PCPM to provide another $15\%-20\%$ improvement in execution time. Overall, computing PageRank using PCPM is $\sim 2.75 \times$ faster than the state of the art BVGAS method.

We also evaluate the impact of partition size on the performance of PCPM by varying the width parameter from 32 KB (8K nodes) to 8 MB (2M nodes). As we increase the size of partition, the neighbors of each node are forced to fit in fewer partitions resulting in better compression as shown in fig. [12]. The web graph is an exception for which the $r$ value remains almost constant because its node labeling provides high spatial locality and close to optimal compression even for small partition sizes. The kron dataset exhibits larger compression than other graphs because of high edge density.

A direct consequence of variation in $r$ is observed in the amount of memory communication which reduces as we increase the partition size (fig. [13]). However, if the partition size grows beyond a threshold, cache is unable to accommodate all the nodes of a partition resulting in cache misses. This drastically increases the main memory traffic since for each
Fig. 12: Compression ratio increases with partition size.

The execution time (fig. 14) also benefits from communication reduction but is penalized by cache misses for large partitions. Note that the execution time starts increasing for partition size > 256 KB, whereas amount of DRAM communication decreases till partition size is < 1 MB. This is because for partitions > 256 KB, many requests are served from the larger shared L3 cache which is slower than the private L1 and L2 caches. Therefore, it decelerates the computation but does not add to DRAM traffic. For partition size > 1 MB, even the L3 cache is not sufficient and the random accesses are served from DRAM rapidly increasing the execution time.

Fig. 13: Impact of partition size on DRAM communication. Large partitions result in cache misses and increased traffic.

Fig. 15: Time consumption of scatter and gather phases for sd1 graph. Both the phases achieve minimum time for 256 KB partition size.

Partition size represents an important trade off in PCPM. Large partitions result in better compression but poor locality for random accesses to nodes within the partition. Fig. 15 shows the effect of altering partition size separately on scatter and gather phases of computation for the sd1 dataset. Both scatter and gather phase initially benefit from higher compression. However, gather phase performance saturates early because its memory accesses are proportional to $1 + \frac{1}{r}$, as compared to scatter phase where accesses are proportional to $\frac{1}{r}$. However, in both the phases, nodes within a partition are randomly accessed and hence, the performance declines if partition size grows beyond what fits in cache. Based on our observations, we chose partition size 256 KB to compare PCPM against the baselines.

3) Pre-processing Time: We assume that a CSR representation of adjacency matrix is available and hence, the pull direction baseline does not incur any pre-processing overhead. Both BVGAS and PCPM however, require a beforehand computation of bin space and offsets for each partition. PCPM needs additional pre-processing to construct the PNG layout. Fortunately, the bin space and offset computation can be merged with first graph scan in PNG construction that computes the degree of destination partitions (section IV-B).

The apriori computation required for BVGAS and PCPM can be easily parallelized over partitions resulting in low pre-processing time as shown in table VII. Furthermore, the pre-processing time gets amortized over multiple iterations of PageRank providing substantial time savings overall. By drawing a comparison from table IV and VII we see that even a single iteration of PCPM along with the associated overheads outperforms PDPR and BVGAS (including pre-processing) for almost all of the datasets.

VII. CONCLUSION

In this paper, we presented a novel Partition-Centric Processing Methodology (PCPM) that perceives a graph as a set of links between nodes and partitions instead of nodes and their individual neighbors. This abstraction coupled with a deterministic data layout for PCPM deliver dramatic reduction in execution time and DRAM communication. Although we
TABLE VII: Pre-processing time of PCPM is higher than BVGAS because of PNG construction

demonstrate the benefits of PCPM on PageRank, it can be applied to generic SpMV computation. PCPM can also be extended to weighted graphs by reading edge weights with the destination IDs and applying them to the source node values.

PCPM avoids data transfer on redundant edges to reduce main memory traffic. Consequently, it is also able to take advantage of locality optimized node labeling unlike the state of the art Gather-Apply-Scatter (GAS) based technique that is oblivious to it. The built-in redundancy reduction feature of PCPM can be easily generalized to distributed graph algorithms and even frontier based graph algorithms like BFS.

We also developed a novel Partition Node Graph (PNG) layout for PCPM. The idea for a new layout originated when we were trying to relax the vertex-centric programming constraint that all outgoing edges of a vertex should be traversed consecutively. The additional freedom arising from treating edges individually led to the development of PNG which intelligently groups the edges to avoid random memory accesses and sustains > 75% of peak memory bandwidth. We apply optimizations to reduce the time of computing PNG and show that its benefits far exceed the pre-processing overhead. The streaming memory access patterns of PNG enabled PCPM make it highly suitable for Accelerators and High Bandwidth Memory (HBM) based architectures.

We conducted extensive analytical and experimental evaluation of PCPM and observed 2.7× speedup and 1.7× reduction in DRAM communication over state of the art. Moreover, irrespective of the graph locality and density, PCPM remains the preferred choice of PageRank implementation as it communicates the least amount of data at higher bandwidth compared to other methodologies.

REFERENCES

[1] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, “Introducing the graph 500,” Cray Users Group (CUG), 2010.

[2] F. McSherry, M. Isard, and D. G. Murray, “Scalability! but at what COST?” in 15th Workshop on Hot Topics in Operating Systems. USENIX Association, 2015.

[3] J. Shun and G. E. Blelloch, “Ligra: a lightweight graph processing framework for shared memory,” in ACM Sigplan Notices, 2013.

[4] A. W. Wulf and S. A. McKee, “Hitting the memory wall: implications of the obvious,” ACM SIGARCH computer architecture news, 1995.

[5] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web.” Stanford InfoLab, Tech. Rep., 1999.

[6] D. Buono, F. Petrini, F. Checconi, X. Liu, X. Que, C. Long, and T.-C. Tuan, “Optimizing sparse matrix-vector multiplication for large-scale data analytics,” in Proceedings of the 2016 International Conference on Supercomputing. ACM.

[7] S. Beamer, K. Asanović, and D. Patterson, “Reducing pagerank communication via propagation blocking,” in Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International.

[8] M. Kumar, M. Serrano, J. Moreira, P. Pattanaik, W. P. Horn, J. Jann, and G. Tanase, “Efficient implementation of scatter-gather operations for large scale graph analytics,” in High Performance Extreme Computing Conference (HPEC), 2016 IEEE.

[9] H. Wei, J. X. Yu, C. Lu, and X. Lin, “Speedup graph processing by graph ordering,” in Proceedings of the 2016 International Conference on Management of Data. ACM.

[10] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, “Gunnrock: A high-performance graph processing library on the gpu,” in Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2016.

[11] N. Sundaram, N. Satish, M. M. A. Patwary, S. R. Dulloor, M. J. Anderson, S. G. Vadlamudi, D. Das, and P. Dubey, “Graphmat: High performance graph analytics made productive,” Proceedings of the VLDB Endowment, 2015.

[12] D. Nguyen, A. Lenharth, and K. Pingali, “A lightweight infrastructure for graph analytics,” in Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, 2013.

[13] P. Boldi, M. Santini, and S. Vigna, “Permuting web and social graphs,” Internet Mathematics, 2009.

[14] W.-H. Liu and A. H. Sherman, “Comparative analysis of the cuhlill–mckee and the reverse cuhlill–mckee ordering algorithms for sparse matrices,” SIAM Journal on Numerical Analysis, 1976.

[15] 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.

[16] A. Abou-Rjeli and G. Karypis, “Multilevel algorithms for partitioning power-law graphs,” in Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International. IEEE.

[17] A.-J. W. Yezieman and R. H. Bisseling, “A cache-oblivious sparse matrix–vector multiplication scheme based on the hilbert curve,” Progress in Industrial Mathematics at ECMI 2010, 2012.

[18] S. Williams, L. Olier, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, “Optimization of sparse matrix–vector multiplication on emerging multicore platforms.” Parallel Computing, 2009.

[19] M. Penner and V. K. Prasanna, “Cache-friendly implementations of transitive closure,” Journal of Experimental Algorithms (JEA), 2007.

[20] R. Nishitai, R. W. Vuduc, J. W. Demmel, and K. A. Yelick, “When cache blocking of sparse matrix vector multiply works and why,” Applicable Algebra in Engineering, Communication and Computing, 2007.

[21] A. Roy, I. Mihailovic, and W. Zwaenepoel, “X-stream: Edge-centric graph processing using streaming partitions,” in Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, 2013.

[22] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, “Pregel: a system for large-scale graph processing,” in Proceedings of the 2010 ACM SIGMOD International Conference on Management of data.

[23] M. Han and K. Daudjee, “Giraph unchained: Barrierless asynchronous parallel execution in pregl–like graph processing systems,” Proceedings of the VLDB Endowment, 2015.

[24] O. Green, M. Dukhan, and R. Vuduc, “Branch-avoiding graph algorithms,” in Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures, 2015.

[25] J. D. McCalpin, “Stream benchmark,” Link: www.cs.virginia.edu/stream/ref.html, 1995.

[26] T. Willhalm, R. Dementiev, and P. Fay, “Intel performance counter monitor—a better way to measure cpu utilization. 2012,” URL: http://software.intel.com/en-us/articles/intel-performance-counter-monitor-a-better-way-to-measure-cpuutilization, 2016.

[27] N. Z. Gong, W. Xu, L. Huang, P. Mittal, E. Stefanov, V. Sekar, and D. Song, “Evolution of social-attribute networks: measurements, modeling, and implications using google+,” in Proceedings of the 2012 ACM conference on Internet measurement conference.

[28] R. Meusel, S. Vigna, O. Lehmberg, and C. Bizer, “The graph structure of the obvious,” in Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International.