PageRank Acceleration for Large Graphs with Scalable Hardware and Two-Step SpMV

Fazle Sadi, Joe Sweeney, Scott McMillan, Tze Meng Low, James C. Hoe, Larry Pileggi and Franz Franchetti
Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
Email: {fsadi, joesweeney}@cmu.edu, smcmillan@sei.cmu.edu, {lowt, jhoe, pileggi, franzf}@cmu.edu

***Abstract*—PageRank is an important vertex ranking algorithm that suffers from poor performance and efficiency due to notorious memory access behavior. Furthermore, when graphs become bigger and sparser, PageRank applications are inhibited as most current solutions profoundly rely on large random access fast memory, which is not easily scalable. In this paper we present a 16nm ASIC based shared memory platform for PageRank implementation that fundamentally accelerates Sparse Matrix dense Vector multiplication (SpMV), the core kernel of PageRank. This accelerator is scalable, guarantees full DRAM streaming and reduces off-chip communication. More importantly, it is capable of handling very large graphs (~2 billion vertices) despite using significantly less fast random access memory than current solutions. Experimental results show that our proposed accelerator is able to yield order of magnitude improvement in both energy efficiency and performance over state of the art shared memory commercial off-the-shelf (COTS) solutions.**

I. INTRODUCTION

PageRank is an iterative algorithm that ranks the vertices of a graph according to their relative importances, which is the probability of reaching any given vertex. The PageRank vector holds numerical values that represent these importances for a set of vertices within a graph. For example, in the case of the most well known application of PageRank, it is used to rank web pages for a particular key word search. Here, each vertex of the graph represents a web page and the edges represent hyperlinks among the web pages. The higher the value of a vertex in the PageRank vector, the higher is the likelihood that anyone randomly browsing through the World Wide Web will land on that particular web page.

Mathematical representation of widely used Power Method [1] of an iteration of PageRank algorithm on static graph is given in Equation 1.

$$\boldsymbol{x}_{(i+1)}^T = \underbrace{\alpha \boldsymbol{x}_i^T \boldsymbol{A}}_{\text{SpMV}} + \underbrace{(1-\alpha)\boldsymbol{x}_i^T \frac{\boldsymbol{e}\boldsymbol{e}^T}{N}}_{\text{constant addition}} \qquad (1)$$

Here, $\boldsymbol{x}_{(i+1)}^T$ is the output PageRank vector at iteration i, $\boldsymbol{A}$ is the hyperlink sparse matrix of dimension $(N \times N)$, α is a constant damping factor. The term $\boldsymbol{e}\boldsymbol{e}^T/N$ is the teleportation matrix that models the random probability of a user to jump to any page with uniform distribution. The column vector $\boldsymbol{e}$ has constant 1 for each element.

In Equation 1, the second term essentially contributes only a constant addition in the update process of each element in the resultant PageRank vector. On the other hand, the first term is a Sparse Matrix dense Vector multiplication (SpMV) operation. As this is the core kernel for each iteration, all challenges related to SpMV kernel are inherited by PageRank.

Challenges. SpMV is a bandwidth bound operation with adverse memory access behavior. It requires random access to either the input vector ($\boldsymbol{x}_i$) or the resultant vector ($\boldsymbol{x}_{i+1}$). For numerous real world graphs, these vectors are much larger than fast memories that are affordable by current technologies, such as Static Random Access Memory (SRAM) and Embedded DRAM (eDRAM). Hence, the majority of the memory accesses of PageRank occur in random fashion to the main memory; i.e. DRAM. This translates to poor utilization of already scarce DRAM bandwidth. Furthermore, it causes redundant off-chip transfers due to granular access that is smaller than cache line, making this bandwidth bound kernel even more inefficient.

A major implication of these issues is that most state of the art PageRank solutions strongly depend on fast storages to achieve decent performance and efficiency. This dependency inhibits these solutions to scale effectively as the graphs get larger and sparser. It is mainly because - a) fast storages are not easily scalable in a shared memory scenario, and b) distributed systems have huge communication overhead [2]. Custom hardware solutions in the literature have reported to handle only a few million nodes, despite their significant advantage in design flexibility. For example, the FPGA accelerator in [3] reported maximum 2.3M nodes using 8.4MB SRAM and the Application Specific Integrated Circuit (ASIC) based architecture in [4] reported maximum 8M nodes in spite of using a huge 32MB eDRAM scratchpad. On the other hand, with large last level caches (LLCs) commercial off-the-shelf (COTS) solutions, such as [5], [6], tend to handle larger graphs, but with low efficiency and poor bandwidth utilization. Nonetheless, the graphs reported only have tens of millions nodes at maximum. Moreover, costly data pre-processing to extract locality in the data is prevalent in both custom and COTS architectures [6]–[11].

Goals and Contributions. For high performance and efficient PageRank application on very large (~billion nodes) and highly sparse (avg. degree <10) graphs, a number of goals are needed to be achieved: - a) streaming DRAM access, b) off-chip traffic reduction, c) full utilization of DRAM bandwidth, d) low requirement of fast memory to scale, and e) no dependence on data locality and costly pre-processing. Contributions of this work in achieving these goals are as follows.

1. We have designed a 16nm ASIC (currently under fabrication) based shared memory accelerator for PageRank that guarantees 100% streaming access to main memory.

2. Our proposed architecture incorporates state of the art High Bandwidth Memory (HBM) [12]. We have developed an optimization technique to reduce off-chip traffic of PageRank and fully utilize the extreme bandwidth delivered by 3D DRAM.

3. This PageRank implementation is able to operate on

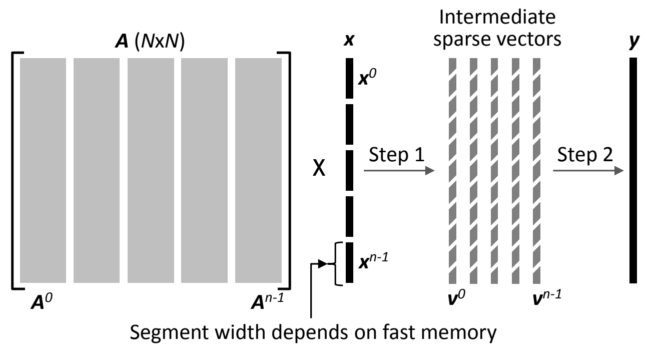


Fig. 1: Two-Step SpMV proposed in [13].

very large graphs (~2 billion nodes) while using significantly less fast memory (11MB). With significant room for possible expansion of fast memory, the proposed solution is scalable to handle even larger graphs. This ASIC design is also portable to FPGA due to reasonable hardware resource requirements.

4. Our proposed solution is independent of data (nonzero) locality and only requires basic matrix partitioning.

The remainder of the paper is organized as follows. Sec. II details the SpMV algorithm and basic PageRank implementation using SpMV. In Sec. III, we demonstrate an optimization technique to reduce off-chip communication and increase computation's streaming speed. Sec. IV describes the ASIC developed for PageRank acceleration. In Sec. V, we evaluate the performance and efficiency of our proposed methods against recent benchmarks. Lastly, Sec. VI concludes this work.

II. Two-Step SpMV Driven PageRank

SpMV Algorithm. In this work, we have implemented a SpMV algorithm, namely Two-Step that is presented in [13]. The main reason for using this algorithm for PageRank is that it guarantees full DRAM streaming. Furthermore, for large graphs with high sparsity, Two-Step produces less off-chip traffic than most conventional SpMV algorithms. This algorithm is depicted in Figure 1. Before computation, the matrix $\boldsymbol{A}$ is partitioned into 1D column blocks and the source vector ($\boldsymbol{x}$) is partitioned into smaller segments. The segment width of $\boldsymbol{x}$ is dictated by the available fast random access memory. The width of the column blocks of $\boldsymbol{A}$ is same as the source vector segment width. The column blocks of the sparse matrix $\boldsymbol{A}$ is stored in a row-major sparse format [14].

As the name suggests, the operation is conducted in two separate steps. In the first step, a single segment of $\boldsymbol{x}$ is streamed from DRAM and stored in the fast memory. Afterwards, a single column block of $\boldsymbol{A}$ is streamed to the computation core from DRAM and partial SpMV is conducted between that column block and vector segment. All the required random access to $\boldsymbol{x}$ is confined in the address space present in the fast memory, which has small and fixed latency. Hence, this operation can be easily pipelined and implemented in a fully streaming fashion. A sparse intermediate vector ($\boldsymbol{v}^k$) is generated as a result of this operation, which is streamed back to main memory. As the matrix block is stored and accessed in row-major direction, elements of $\boldsymbol{v}^k$ is naturally sorted according to their position indices. This partial SpMV is conducted sequentially for all the matrix blocks and, after the first step, we end up with n intermediate sparse vectors residing in the main memory.

In the second step, the intermediate sparse vectors are streamed back from DRAM and merged to form the dense resultant vector $\boldsymbol{y}$. To ensure full main memory streaming, page (row buffer in DRAM) size blocks are prefetched whenever an element of $\boldsymbol{v}^k$ is transferred from DRAM. Step 2 is essentially a large n-way merge operation on very long and sorted lists. It is difficult to implement such a large Multi-way Merge kernel in COTS architectures, such as CPUs and GPUs. In fact, this is the main reason SpMV algorithms similar to Two-Step have not been proposed in the literature despite its full streaming DRAM access pattern. However, recently it has been shown in [13] that this large merge network can be implemented efficiently with custom hardware. This work implements such hardware in an ASIC platform that is detailed in Sec. IV.

PageRank using SpMV. Our proposed basic implementation of PageRank using Two-Step SpMV is depicted in Figure 2. As this is an iterative application, the entire Two-Step SpMV operation is conducted once independently for each iteration. The resultant dense PageRank vector ($\boldsymbol{y}_i = \boldsymbol{x}_{i+1}$) of iteration i works as the source vector for SpMV in iteration $i+1$. As the entire resultant vector is too large to be stored in on-chip fast memory, it is streamed out to DRAM at the end of each iteration. Then in the next iteration, segments of $\boldsymbol{x}_{i+1}$ are streamed back to computation core for Step 1 of the SpMV operation.

Off-chip Communication. It is evident that during each iteration and during the transition between iterations, Two-Step driven PageRank algorithm is guaranteed to only require streaming DRAM accesses, which is imperative for high performance and energy efficiency. Another important factor in this regard is the off-chip traffic that is transferred between the computation core and main memory. As there is no DRAM random access involved in our proposed Two-Step driven PageRank (*PR_TS*), we can exactly calculate the off-chip data traffic for this implementation. We compare this with the baseline data traffic of PageRank driven by dot product based SpMV algorithm (*PR_Base*), where each element of the resultant vector is computed directly from the dot product of matrix row and source vector [15]. For *PR_Base*, we assume traditional eviction policy of LLC in CPU and 64B cache line transfer between DRAM and LLC. We select this baseline for comparison because in the literature it is difficult to find shared memory PageRank implementation for very large graphs with moderate fast storage. For example, [3] uses moderate on-chip BRAM (8MB), but only handles matrix with of maximum dimension of 2.3M×2.3M. On the other hand, [4] developed an ASIC for SpMV using large fast random access eDRAM (32MB). Nonetheless, this work reported to only handle 8M×8M matrix at maximum. In this work, our goal is to handle matrices with dimensions in the order of hundreds of millions to billions.

Figure 3 depicts the total off-chip traffic comparison for different fast memory sizes between *PR_Base* (1st bar) and *PR_TS* (2nd bar) for PageRank with 20 iterations. For this comparison, we have used a 1B×1B synthetic and uniformly random sparse matrix with an average degree of 3. It is evident that the biggest contributing factor in the off-chip communication for *PR_Base* is the redundant data for the source vector $\boldsymbol{x}$ (striped gray region). This redundancy is due to the cache line level block transfers for random accesses to $\boldsymbol{x}$, most of which never take part in actual computation.

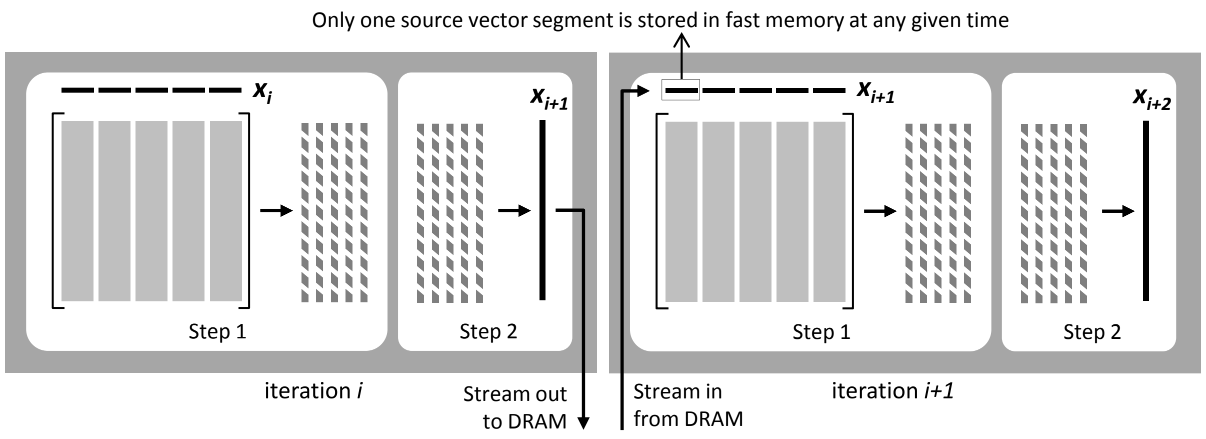


Fig. 2: Two-Step SpMV driven PageRank with independent iterations (*PR_TS*).

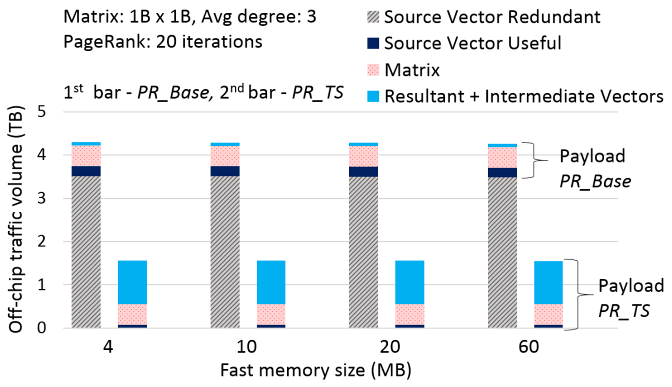


Fig. 3: Off-chip communication comparison for PageRank.

The transferred source vector data that actually takes part in computation (deep blue region), is also larger for *PR_Base* due to regular evictions. The matrix data (red dotted region) is same for both as matrix is streamed only once. However, the overall payload (data that actually takes part in computation) is larger in *PR_TS*. This is due to round trip of the intermediate vectors (light blue region that also includes resultant vector) in Two-Step algorithm. This is the cost of streaming for Two-Step SpMV. Nonetheless, the overall off-chip traffic is significantly less for *PR_TS*. Additionally, the entire off-chip traffic in *PR_TS* is transfered at DRAM streaming bandwidth, whereas the most traffic for *PR_Base* is transfered at DRAM random access speed, which is orders of magnitude slower than streaming. This comparison demonstrates the advantages of fully streaming *PR_TS* over non-streaming algorithms for highly sparse, non-structured and large graphs.

It is noteworthy that none of the algorithms is significantly benefited from the increase in fast memory. For *PR_Base*, even the largest fast memory is incapable to hold a sizable portion of the source vector to render any meaningful reuse of data. In the case of *PR_TS*, the high sparsity in data causes reduction operation in Step 1 to be rare. Hence, increase in fast memory actually has a negligible effect in regard to off-chip communication for *PR_TS*. However, a larger fast memory enables *PR_TS* to handle bigger graphs (more nodes).

III. Traffic Optimization by Iteration Overlap

In Figure 2, we have seen that PageRank iterations in *PR_TS* are sequential and completely independent, where the resultant dense vector ($\boldsymbol{x}_{i+1}$) is streamed out to dram and streamed back into the computation core as the source vector in the next iteration as it is too big to be stored in the fast memory. However, we can parallelize the SpMV steps of consecutive iterations and eliminate the off-chip communication for resultant and source vectors, as depicted in Figure 4. The idea is that even though we cannot store the entire $\boldsymbol{x}_{i+1}$, we can store a segment in the fast memory. For example, during Step 2 of iteration i, instead of sending the computed elements of $\boldsymbol{x}_{i+1}$ to DRAM, it is written in fast memory until a full segment is stored. When the first segment of source vector for iteration $i+1$ is completely

Pseudocode 1: Two-Step SpMV driven PageRank with off-chip communication optimization by iteration overlap.

```
$T$ = Total number of iterations
for $i = 0$ to $T-1$ do
    STEP 1
    for $k = 0$ to $n-1$ do
        Stream in Matrix Column Block $A^k$
        $\boldsymbol{u} \leftarrow 0$
        for All rows $A^k_{p,:}$ with $nnz > 0$ do
            for Each non-zero $A^k_{p,q}$ in $A^k_{p,:}$ do
                Random access to vector segment $\boldsymbol{x}^k_{[i+1]}$
                $u_p \leftarrow \alpha * A^k_{p,q} * x^k_{q[i+1]} + u_p$
            end
        end
        Sparsify $\boldsymbol{u}$ to $\boldsymbol{v}^k_{[i+1]}$
        Stream out $\boldsymbol{v}^k_{[i+1]}$ to main memory
    end
    STEP 2
    for $p = 0$ to $N-1$ do
        for $k = 0$ to $n-1$ do
            Stream in $\boldsymbol{v}^k_{[i]}$
            $x_{p[i+1]} \leftarrow x_{p[i+1]} + v^k_{p[i]}$ [Multiway merge]
        end
        $c_{[i+1]} \leftarrow c_{[i+1]} + x_{p[i+1]}$
        $x_{p[i+1]} \leftarrow x_{p[i+1]} + \frac{1-\alpha}{N} c_{[i]}$ [Constant addition]
    end
    Buffer $\boldsymbol{x}_{[i+1]}$ on chip
end
```

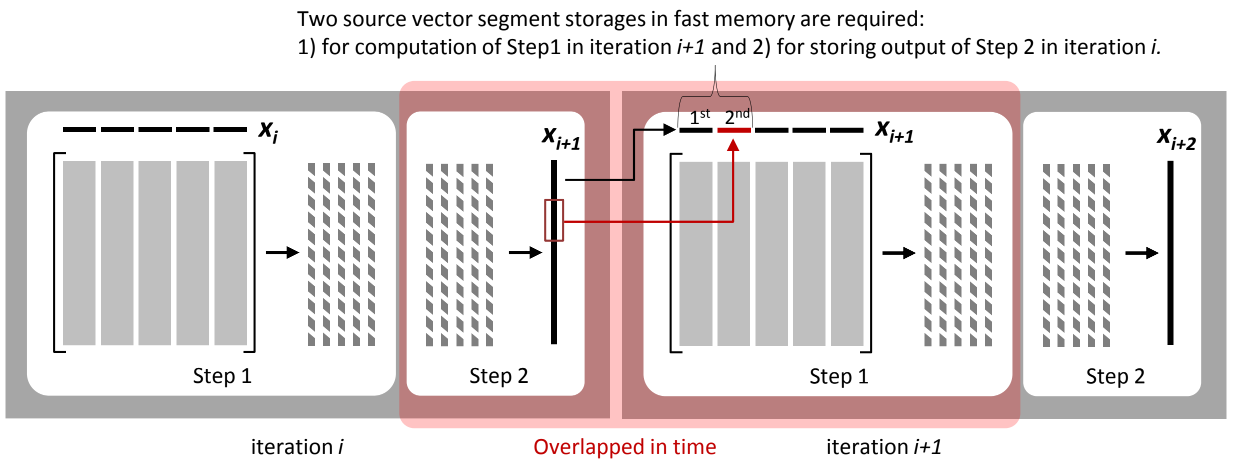

Two source vector segment storages in fast memory are required:
1) for computation of Step1 in iteration i+1 and 2) for storing output of Step 2 in iteration i.
x_i
x_{i+1}
x_{i+2}
1st 2nd
Step 1
Step 2
iteration i
Overlapped in time
iteration i+1

Fig. 4: Off-chip traffic optimized PageRank with iteration overlap (*PR_TS_Opt*).

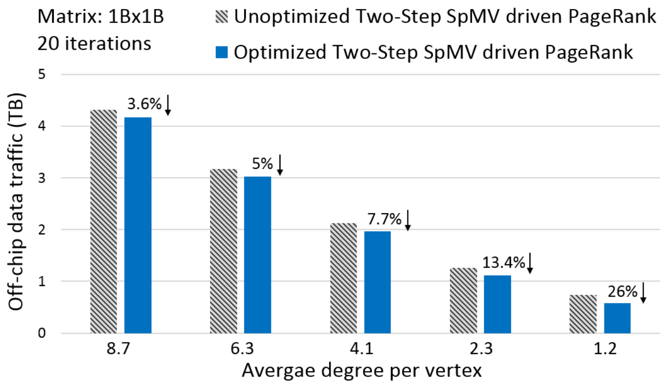


Fig. 5: Off-chip communication of *PR_TS_Opt* vs *PR_TS*.

available in fast memory, Step 1 of the Two-Step SpMV for this iteration starts computation with the first column block of the matrix. Concurrently, Step 2 of iteration i continues and stores the second segment of the source vector (for iteration $i+1$) in another fast memory buffer. Thus, Step 2 of iteration i and Step 1 of iteration $i+1$ are overlapped in time and run parallelly. A pseudocode of this entire operation, i.e. Two-Step SpMV driven PageRank with off-chip traffic optimization by iteration overlap (*PR_TS_Opt*), is shown in Pseudocode 1.

Another important benefit of *PR_TS_Opt* is that it achieves significantly higher streaming speed than *PR_TS*. This is because none of the computation cores for Step 1 and 2 remains idle in steady state, whereas for *PR_TS* computation logic for either step 1 or step 2 remains idle at any given moment. Hence, *PR_TS_Opt* enables the entire silicon area to be active for all the iterations (except the very first and very last one) that helps in fully utilizing extreme off-chip bandwidth offered by modern technologies such as 3D stacked DRAM. We will demonstrate practical example of this in Sec. V.

The cost of achieving off-chip traffic optimization is that we have to buffer two source vector segments in the fast memory instead of one in *PR_TS*. As a result, for any given amount of fast storage, the maximum matrix dimension that *PR_TS_Opt* can handle is roughly half of the maximum matrix dimension of *PR_TS*. Therefore, for *PR_TS_Opt*, there is a trade-off between sparse matrix dimension vs performance and efficiency.

The communication reduction with *PR_TS_Opt* is depicted in Figure 5. We generated five uniformly random graphs of dimension 1B×1B with different sparsity, which is labeled on the x-axis. The striped gray and solid blue bars represent the total off-chip traffic for PageRank with 20 iterations using *PR_TS* and *PR_TS_Opt* accordingly. We can see that when the graph becomes sparser (i.e. less average degree per vertex), the ratio of reduction in data transfer with iteration overlap gets larger. For example, with a very sparse graph of average degree 1.2 per vertex, 26% more off-chip DRAM traffic would incur if optimization is not applied. This is because with sparser matrix, the data transfer due to intermediate vectors in Two-Step SpMV operation gets less significant relative to the data transfer due to source and resultant vectors.

IV. ASIC for PageRank

In this section we demonstrate the custom ASIC to accelerate PageRank, which is designed using Verilog and currently being fabricated in 16nm FinFET technology. The block diagram of the overall accelerator is shown in Figure 6. The ASIC chip implements the computation logic required for the Two-Step SpMV algorithm. To conduct Step 1 of Two-Step algorithm, sixteen parallel single precision floating point multiplier and adder chains are implemented. For Step 2, sixteen Multi-way Merge cores are designed, which are able to parallelly merge 2048 lists (intermediate vectors) and have a overall throughput of 16 resultant vector elements per cycle. A radix-sort based data parallelization technique is used to distribute loads among the merge cores. However, implementation details of logic for data distribution, load balancing and synchronization is beyond the scope of this paper and, hence, skipped.

An actual image of the ASIC and key specifications are given in Figure 7. As the chip is currently being fabricated, these specifications are from post physical synthesis (after place and route) layout of the design. *Cadence® Innovus™* is used for area and frequency measurement and *Cadence® Voltus™* is used for power measurement. One key aspect of this chip is that it uses synthesized SRAM blocks, also known as Logic in Memory (LiM) technology [16]–[18], distributed all over the chip to facilitate fine grain data access during computation.

The two other parts of the accelerator, i.e. HBM main memory and the eDRAM scratchpad, are emulated using Cacti

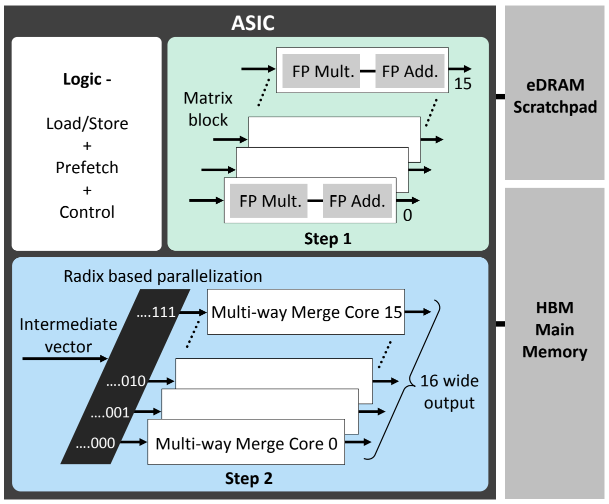
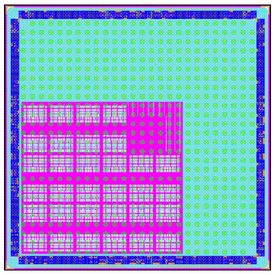


Fig. 6: Block diagram of 16nm FinFET ASIC based architecture for PageRank acceleration with Two-Step SpMV.



ASIC specifications
Frequency: 1.4 GHz
Occupied area: 7.5 mm^2
Leakage power: 0.10 W
Dynamic power: 3.01 W
Total power: 3.11 W

Fig. 7: Image of the actual 16nm ASIC and specifications.

[19] and Destiny [20] tools. This ASIC is designed to work with two 3D-stacked 2nd generation High Bandwidth Memories (HBM2s) [12], [21] as main memory, which are connected through interposer [22]. A single HBM2 provides 256GB/s aggregated bandwidth. Hence, this chip is designed to saturate the extreme off-chip bandwidth of total 512GB/s offered by state of the art 3D stacked DRAM technology. On the other hand, we have used eDRAM scratchpad as fast memory for the accelerator. This scratchpad buffers source vector segments and prefetched intermediate vector data.

Fast Memory Requirement. For better scalability, one of the key goals of our proposed solution is to handle very large graphs while not requiring large amount of fast storage (such as SRAM or eDRAM based cache, scratchpad, etc.) for random access. In our proposed accelerator, the ASIC's computation core requires 0.5MB of synthesized SRAM. For source vector segment storage in Step 1, it requires 8MB of eDRAM scratchpad. Additionally, for proper streaming in Step 2, DRAM page (row buffer) size blocks have to be prefetched while accessing the 2048 intermediate vectors. The page size of HBM2 is 1KB and we allocate 1.25KB to store prefetched data for each list (instead 1KB) to hide loading latency. Hence, we require $2048 \times 1.25\text{KB} = 2.5\text{MB}$ of eDRAM buffer for prefetched data. Therefore, the fast memory requirement of our proposed solution is (0.5MB + 8MB + 2.5MB) = 11MB. To put this into perspective, Table I lists other shared memory solutions for PageRank and SpMV against our proposed Two-Step SpMV driven PageRank with (*PR_TS_Opt*) and without (*PR_TS*) optimization by iteration overlap. We see that our

TABLE I: Fast memory requirement and largest graph dimension comparison of current and proposed solutions.

Solution	Fast memory size (MB)	Max. vertices reported
FPGA [3]	8.4	2.3M
ASIC [4]	32	8M
CPU (single socket) [5]	20	95M
CPU (dual socket) [6]	50	118M
PR_TS_Opt (proposed)	11	2B
PR_TS (proposed)	11	4B

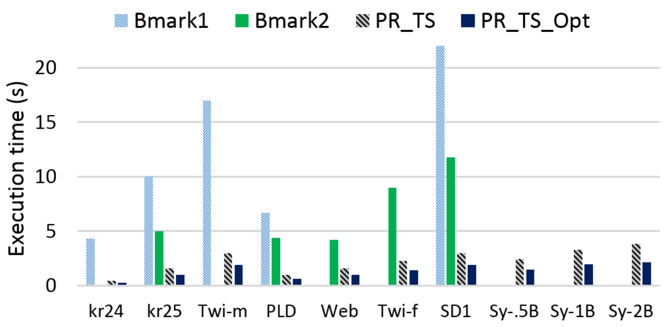


Fig. 8: Execution time comparison for 20 iterations.

proposed solutions can operate on much larger graphs despite having significantly less fast memory. This makes our solution easier to scale as requirement of fast memory in bulk hinders graph dimension to scale in many current solutions, such as [4]. For example, if we increase the source vector buffer to 16MB from 8MB, we will be able to handle graphs with twice more vertices with this ASIC, i.e graphs of 4B and 8B vertices with *PR_TS_Opt* and *PR_TS* accordingly. It should be noted that the total number of edges only dictates the requirement for main memory storage and has negligible impact on the computation core design for our developed accelerator.

V. Experimental Results

We ran PageRank with 20 iterations on our proposed accelerator with a number of graphs mentioned in Table II. Graph with prefix 'kr' are generated using the Kronecker graph generator in [1]. The ones with prefix 'Sy' are uniformly distributed random graphs representing worst case scenario for that dimension and sparsity. Rest are real word graphs from the cited sources.

TABLE II: Graph data sets used for experiments.

Graph	# Nodes (M)	Avg. Degree	# Edges (M)
kr24 [1]	16.7	16.1	268
kr25 [1]	33.5	31.3	1047
Twi-m [23]	52.5	37.4	1963
PLD [24]	42.9	14.5	623
Web [25]	118	8.6	1014
Twi-f [26]	61.6	23.8	1468
SD1 [24]	94.9	20.4	1936
Sy-.5B	500	3	1500
Sy-1B	1000	2	2000
Sy-2B	2000	1.1	2200

The 3D memory sub-system is emulated with CACTI3D [19] tool assuming 4 channels (each HBM2 has 2 pseudo-channels with 64B I/O width). As the chip is currently in

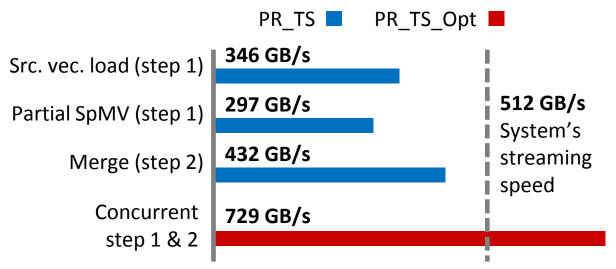


Fig. 9: Sustained streaming speed of proposed methods.

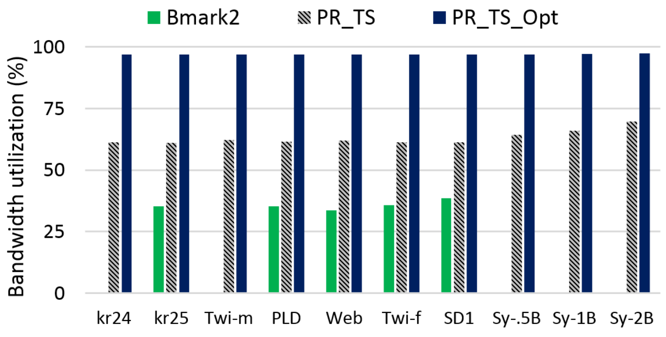


Fig. 10: Main memory bandwidth utilization comparison.

fabrication facility, we use the specs given in Figure 7 and conducted cycle accurate Verilog simulation. Column blocks of the matrix is stored in row-major COO format as hyper-sparsity causes CSR to be wasteful. Single precision is used for floating point values and 32 bits are used for all indices. We compared both our unoptimized (*PR_TS*) and optimized (*PR_TS_Opt*) implementations against two benchmarks. These recent works are *Bmark1* [5] (single socket, 20MB LLC) and *Bmark2* [6] (dual socket, 50MB LLC) CPU implementations of PageRank that reported comparably large graphs. *Bmark1* also took part in HPEC Graph Challenge 2017.

Figure 8 depicts the comparison of execution time for PageRank with 20 iterations among the proposed implementations and the benchmarks (for the graphs where data is reported). While being able to handle much larger graphs, our optimized proposed solution (*PR_TS_Opt*) is 12x and 7x faster than *Bmark1* and *Bmark2* accordingly. It can be noticed that our unoptimized solution (*PR_TS*) is relatively slower than *PR_TS_Opt*. Besides more off-chip traffic, the main reason is that the ASIC is provisioned to saturate 100% 3D DRAM bandwidth of the system when Step 1 and 2 of SpMV is running in parallel. Figure 9 depicts the maximum streaming speed of different parts of the chip for matrix ‘Sy-1B’. For *PR_TS*, the source vector load, partial SpMV and multi-way merge are conducted sequentially. The streaming speed of the logic cores for these tasks are below what the system can provide. On the other hand, for *PR_TS_Opt* all the tasks in step 1, i.e. source vector load and partial SpMV, runs in parallel with the merging task in step 2. Thus, with the same amount of silicon real estate, we can attain much higher streaming speed. As shown in Figure 9, the maximum sustained streaming speed of *PR_TS_Opt* is well over the system's 512GB/s and actually can saturate almost three HBM2s (768GB/s).

The bandwidth utilization is given in Figure 10. Due to full streaming algorithm our proposed implementations achieve

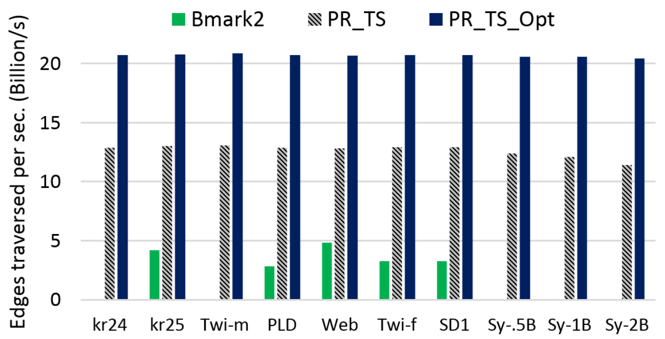


Fig. 11: Comparison of number edges traversed per second.

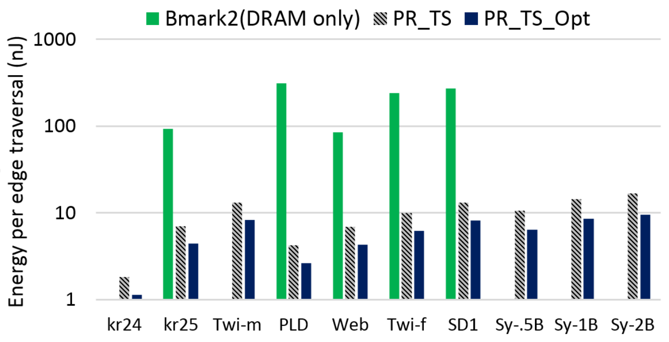


Fig. 12: Comparison of entire system energy per edge traversed. Note: Bmark2 reported DRAM energy only.

significantly higher bandwidth utilization than *Bmark2*. As explained previously, *PR_TS_Opt* achieves ~ 97% utilization for all graphs due to overlap of step 1 & 2 across iterations and having a streaming speed more than of two HBM2s.

Another metric of performance we used is the number of edges traversed per second. We avoided using GFLOP/s metric as this is data dependent and not representative of system's capability to process sparse data. As shown in Figure 11, *PR_TS_Opt* provides 7x faster edge traversal rate than *Bmark2*. Furthermore, we have compared the energy efficiency in Figure 12. Despite using entire system's energy for *PR_TS* and *PR_TS_Opt* against only the DRAM energy for *Bmark2*, it is evident that our proposed system is up to two orders of magnitude more efficient. This is due to less execution time, small fast memory, less off-chip traffic and efficient 3D DRAM.

VI. Conclusion

In this work we have developed a custom ASIC hardware accelerator with 3D DRAM for PageRank that can operate on very large graphs (~billion nodes), while providing high performance and energy efficiency. This solution guarantees full DRAM streaming access and proper utilization of off-chip bandwidth. Moreover, it is readily scalable as it requires less fast random access memory than most current architectures in literature. The key to these achievements is the use of Two-Step SpMV algorithm. As COTS architectures are not suitable for this algorithm, we have developed custom ASIC for PageRank implementation. Additionally, we have proposed an optimization technique that reduces off-chip traffic and increases the streaming speed of the computation core. Due to reasonable requirements of hardware resources, this ASIC accelerator design can also be ported to FPGA based platforms.

Acknowledgment

This work was supported in part by Defense Advanced Research Projects Agency (DARPA) contract HR0011-16-C-0038, "Circuit Realization At Faster Timescales (CRAFT)". This material is also based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center [DM18-0845]. Additionally, this work was sponsored by DARPA contract HR0011-13-2-0007, "Power Efficiency Revolution for Embedded Computing Technologies (PERFECT)". The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References

[1] P. Dreher, C. Byun, C. Hill, V. Gadepally, B. C. Kuszmaul, and J. Kepner, "Pagerank pipeline benchmark: Proposal for a holistic system benchmark for big-data platforms," *CoRR*, vol. abs/1603.01876, 2016. [Online]. Available: http://arxiv.org/abs/1603.01876

[2] F. McSherry, M. Isard, and D. G. Murray, "Scalability! but at what cost?" in *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems*, ser. HOTOS'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 14–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=2831090.2831104

[3] S. Zhou, C. Chelmis, and V. K. Prasanna, "Optimizing memory performance for fpga implementation of pagerank," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2015, pp. 1–6.

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

[5] S. Zhou, K. Lakhotia, S. G. Singapura, H. Zeng, R. Kannan, V. K. Prasanna, J. Fox, E. Kim, O. Green, and D. A. Bader, "Design and implementation of parallel pagerank on multicore platforms," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, Sept 2017, pp. 1–6.

[6] K. Lakhotia, R. Kannan, and V. K. Prasanna, "Accelerating pagerank using partition-centric processing," *CoRR*, vol. abs/1709.07122, 2017. [Online]. Available: http://arxiv.org/abs/1709.07122

[7] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrix–vector multiplication on emerging multicore platforms," *Parallel Computing*, vol. 35, no. 3, pp. 178–194, 2009.

[8] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 2009, p. 18.

[9] J. W. Choi, A. Singh, and R. W. Vuduc, "Model-driven autotuning of sparse matrix-vector multiply on gpus," *SIGPLAN Not.*, vol. 45, no. 5, pp. 115–126, Jan. 2010. [Online]. Available: http://doi.acm.org/10.1145/1837853.1693471

[10] X. Yang, S. Parthasarathy, and P. Sadayappan, "Fast sparse matrix-vector multiplication on GPUs: Implications for graph mining," *Proceedings of the VLDB Endowment*, vol. 4, no. 4, pp. 231–242, 2011.

[11] J. B. White III and P. Sadayappan, "On improving the performance of sparse matrix-vector multiplication," in *High-Performance Computing, 1997. Proceedings. Fourth International Conference on*. IEEE, 1997, pp. 66–71.

[12] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, and K. Kim, "Hbm (high bandwidth memory) dram technology and architecture," *2017 IEEE International Memory Workshop (IMW)*, pp. 1–4, 2017.

[13] F. Sadi, L. Fileggi, and F. Franchetti, "Algorithm and hardware co-optimized solution for large spmv problems," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, Sept 2017, pp. 1–7.

[14] A. Buluc and J. R. Gilbert, "On the representation and multiplication of hypersparse matrices," in *2008 IEEE International Symposium on Parallel and Distributed Processing*, April 2008, pp. 1–11.

[15] S. Toledo, "Improving memory-system performance of sparse matrix-vector multiplication," *IBM Journal of Research and Development*, vol. 41, pp. 711–726, 1997.

[16] D. Morris, K. Vaidyanathan, N. Lafferty, K. Lai, L. Liebmann, and L. Pileggi, "Design of embedded memory and logic based on pattern constructs," in *Symposium on VLSI Technology (VLSIT)*, 2011, pp. 104–105.

[17] Q. Zhu, K. Vaidyanathan, O. Shachamy, M. Horowitzy, L. Pileggi, and F. Franchetti, "Design automation framework for application-specific logic-in-memory blocks," in *IEEE 23rd International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, July 2012, pp. 125–132.

[18] D. Morris, V. Rovner, L. Pileggi, A. Strojwas, and K. Vaidyanathan, "Enabling application-specific integrated circuits on limited pattern constructs," in *Symposium on VLSI Technology (VLSIT)*, 2010, pp. 139–140.

[19] K. Chen, S. Li, N. Muralimanohar, J.-H. Ahn, J. Brockman, and N. Jouppi, "CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory," in *Design, Automation Test in Europe (DATE)*, 2012, pp. 33–38.

[20] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "Destiny: A tool for modeling emerging 3d nvm and edram caches," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, March 2015, pp. 1543–1546.

[21] J. Standard, "High bandwidth memory (hbm) dram," *JESD235*, 2013.

[22] K. Cho, H. Lee, H. Kim, S. Choi, Y. Kim, J. Lim, J. Kim, H. Kim, Y. Kim, and Y. Kim, "Design optimization of high bandwidth memory (hbm) interposer considering signal integrity," in *2015 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS)*, Dec 2015, pp. 15–18.

[23] J. Kunegis, "Konect: The koblenz network collection," in *Proceedings of the 22Nd International Conference on World Wide Web*, ser. WWW '13 Companion. New York, NY, USA: ACM, 2013, pp. 1343–1350. [Online]. Available: http://doi.acm.org/10.1145/2487788.2488173

[24] R. Meusel, S. Vigna, O. Lehmberg, and C. Bizer, "The graph structure in the web analyzed on different aggregation levels," *The Journal of Web Science*, vol. 1, no. 1, pp. 33–47, 2015. [Online]. Available: http://dx.doi.org/10.1561/106.00000003

[25] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2049662.2049663

[26] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 591–600. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772751