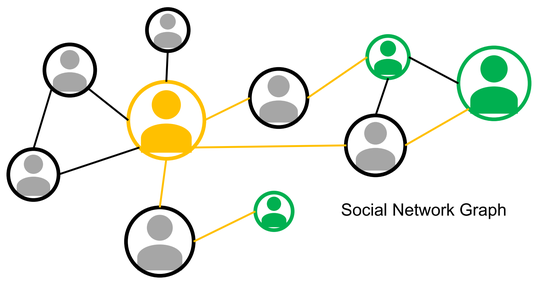
CornellEngineering
Electrical and Computer Engineering

GraphLily - Accelerating Graph Linear Algebra on HBM-Equipped FPGAs

Author: Matt Hales Advisor: Professor Zhang

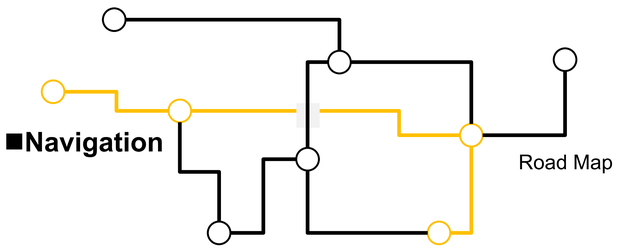
Sparse Processing for Graphs

Breadth-first Search (BFS)



- Recommend 2-hop neighbors as new friends

Single-source Shortest Path (SSSP)



PageRank

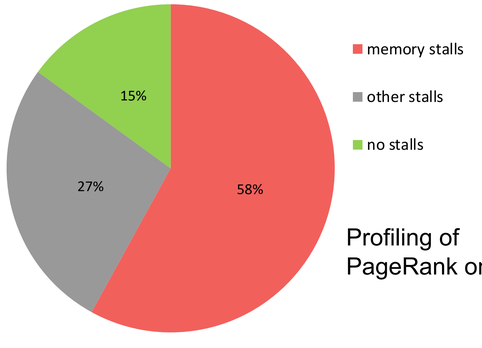
Rank	Website
1	www.google.com
2	www.youtube.com
3	www.facebook.com

Website Link Graph

- Used in search engines

Sparse Processing Challenges

Memory bandwidth bound



Profiling of PageRank on [1]

- The operational intensity (OI), operations per unit memory traffic, is low for sparse workloads.

Abstract

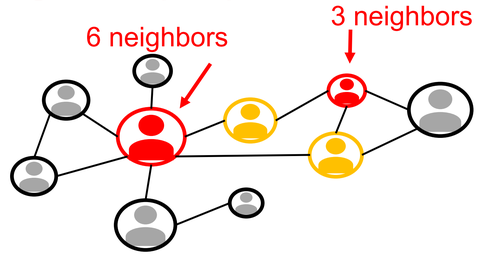
The purpose of GraphLily is to accelerate sparse matrix linear algebra computations on a Xilinix Field Programmable Gate Array (FPGA) by leveraging the benefits of High Bandwidth Memory (HBM). Most matrix multiplications can have long execution times due to their O(N2) complexity. To address this, we convert between sparse matrices, matrices filled with many zeroes, and dense matrices, matrices where all zeroes have been removed.

While kernels for sparse and dense matrix multiplication have already been implemented, my project aims to adapt these existing kernels with HBM and extend the design to incorporate a multi-tenancy system. This system allows two users to input their own matrices and perform computation on a new combined matrix to test if there is a performance speedup or deficit because of this combination.

The hardware accelerators can benefit from HBM due to the significantly higher data transfer rate over the standard Double Data Rate (DDR) memory onboard the FGPA, as well as their ability for parallel memory accessing, meaning multiple memory transactions can occur simultaneously. In the initial phase, we will incorporate 16 of the 32 HBM memory channels into our design. Once successful, we intend to include up to 26 of the 32 HBM memory channels as 6 of the channels are required for other processes that benefit overall system performance.

Ideally, with the added HBM channels, once we incorporate this design, we hope to increase our performance from 230 MHz to 275-285 MHz, which is almost as fast as the FPGA will allow: 300 MHz. This is important as data sets continue to increase in size, and without creating new ways to compute this data, users will be forced to wait for much longer times. With the multi-tenancy system and HBM implementation, we hope to optimize performance as much as possible, given the hardware specifications, and further enhance the scalability and efficiency of sparse matrix computations on FPGAs.

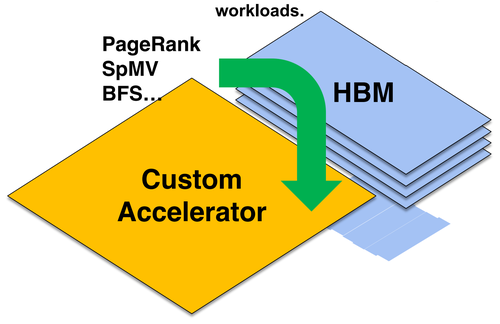
Irregular compute pattern



- The vertex degree varies dramatically
- Vertices may share neighbors

High-bandwidth Memory (HBM)

- HBM provides multiple memory channels that can be accessed concurrently, bringing the potential to boost the performance of sparse workloads.



Our Approach

Near-HBM hardware acceleration

- Co-design the sparse format and the accelerator architecture to maximize HBM bandwidth utilization
- Build dynamically executing pipelines to handle the irregular computation
- Prototype an GraphBLAS-compatible overlay on FPGA for programmability

Format-architecture Co-design

Traditional CSR format

Matrix

PE 1*		a_{12}	a_{13}		HBM Channel 1
PE 2*	a_{21}			a_{24}	
PE 1				a_{34}	HBM Channel 2
PE 2	a_{41}				

Row pointer	p_1	p_2		p_3	p_4
Non-zero	a_{12}	a_{13}	a_{21}	a_{24}	a_{34}	a_{41}
Column index	2	3	1	4	4	1

- Contiguous row storage → contention
- Indirect addressing → non-streaming

HBM-friendly sparse format

Matrix

PE 1*		a_{12}	a_{13}		HBM Channel 1
PE 2*	a_{21}			a_{24}	
PE 1				a_{34}	HBM Channel 2
PE 2	a_{41}				

→: Next-row marker

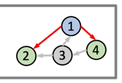
Non-zero	a_{12}	a_{13}	→	a_{34}	→	a_{21}	a_{24}	→	a_{41}
Column index	2	3		4		1	4		1

- Cyclic row storage → contention-free
- Next-row markers → streaming access

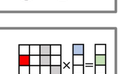
* According to [2], cyclically assigning rows to PEs is essential for load balancing.

Accelerator architecture

Graph algorithms (e.g. BFS, PageRank, SSSP) expressed as traversals

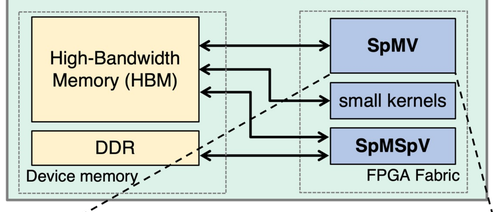


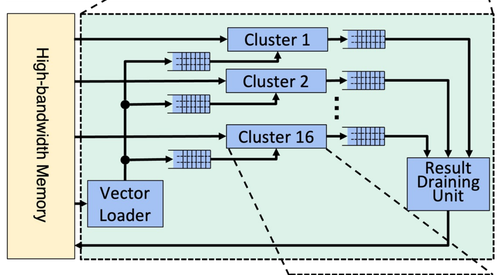
GraphBLAS: mapping graph traversals to Adj. matrix "×" frontier vector

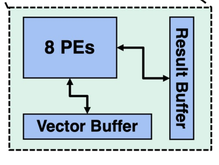


Middleware: kernel scheduling & module-level APIs

Hardware accelerator: direct, near-memory access





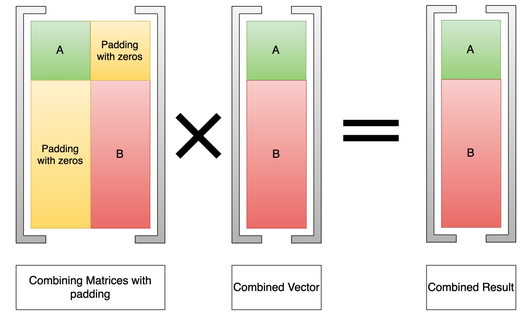


- Each cluster of 8 PEs accesses one HBM channel in a vectorized fashion
- The vector buffer is shared with pipelined arbitrated crossbar [4]
- PEs handle RAW hazards by data-forwarding [4]

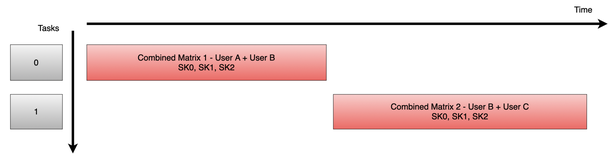
Multi-tenancy

What is Multi-tenancy:

The ability for multiple users to use the same device or hardware simultaneously



- Goal is to allow for multiple matrices to be computed at the same time
- Create a diagonal matrix with start of matrix B starting at the bottom right adjacent index of matrix A
- Combine test vectors together, created function that only performs matrix vector multiplication on each user's matrix depending on the number of columns in each User's matrix
- Returns both User A and User B results synchronously



Testing Strategy:

1. Calculate timing for performing matrix computations for each user individually and combine to find total time
2. Calculate timing for combine matrix computation
3. Depending on timing, determine if the combined matrix is worth the additional hardware resources and utilization based on the following example:

Independent Matrix Timing Analysis:
User A – 5 Seconds
User B – 10 Seconds
Total Time: 15 Seconds

Combined Matrix Timing Analysis – 11 Seconds

User A must wait an additional 6 seconds to receive results of the combined matrix, though User B would get their results back 4 seconds earlier as they no longer have to wait for User A to finish before they can start their task.

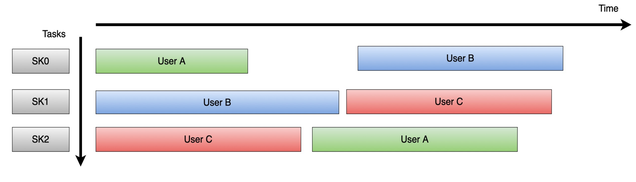
Future Implementations:

Use scheduling with OpenCL to split the matrix vector computations onto different sub-kernels to allow for multiple matrix vector operations to execute at the same time, allowing for asynchronous result output for each User.

Requirements:
- Command Queue

This Command Queue keeps track of which user is in line as well as which sub kernel is available: SK0, SK1, and SK2.

If successful, the new timing diagram would look like the following:



References

[1] Abanti Basak, et al. "Analysis and Optimization of the Memory Hierarchy for Graph Processing Workloads." HPCA 2019
[2] N. Srivastava, et al. "Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product," Int'l Symp. on Microarchitecture (MICRO), 2020.
[3] Y. Hu, et al. "GraphLily: Accelerating Graph Linear Algebra on HBM-equipped FPGAs" ICCAD 2021
[4] Y. Du, et al. "High-Performance Sparse Linear Algebra on HBM-equipped FPGAs Using HLS: A Case Study on SpMV" FPGA 2022
[5] X. Chen, et al. ThunderGP: Resource-Efficient Graph Processing Framework on FPGAs with HLS, ACM TRETS 2022