FPGA-Accelerated Ripples

Reece Neff, Marco Minutoli, Antonino Tumeo
{reece.neff,marco.minutoli,antonino.tumeo}@pnnl.gov
Pacific Northwest National Laboratory
Richland, Wa, USA

Michela Becchi
mbecchi@ncsu.edu
North Carolina State University
Raleigh, NC, USA

1 INTRODUCTION

Given a social network G, a budget k, and a model of diffusion M, the Influence Maximization problem (IM) is to find a cohort of vertices in G of size k that, when initially activated, maximizes the expected number of activations in G at the end of the diffusion process. IM has a wide application in social media analysis, networked epidemiology, cyber-security, and in the analysis of misinformation. The problem has been shown to be NP-hard and defining approximation schemes for IM has become a very active research area. While existing approximation schemes are theoretically efficient, their naïve, sequential implementation is often impractical on real-world networks. Minutoli et al. [4] proposed a parallel and distributed IM library that leverages CPUs and (multiple) GPUs to meet efficiency requirements of real-world applications. While multi-GPU acceleration brings massive improvements in performance, understanding if other accelerators can provide similar performance benefits at lower power profiles is still an open question. This work presents preliminary results in this direction by studying the performance and energy usage of FPGA based acceleration for IM.

In summary, we make the following contributions:

- We design an FPGA accelerated sampling step for the IM algorithm described in [4];
- We study the performance of our prototype and show promising preliminary results in terms of improved performance (up to 3.96×) and energy efficiency (up to 5.15×) when compared to a parallel CPU implementation.

2 INFLUENCE MAXIMIZATION ALGORITHM

Our study leverages the Ripples framework by Minutoli et al. [4], which implements a parallel version of the IMM algorithm by Tang et al. [6]. The IMM algorithm can be broken into two fundamental building blocks: *Sample* and *SelectSeeds*. Broadly, the *Sample* stage generates a θ number of graphs determined by an iterative estimation process that leverages the *SelectSeeds* algorithm to estimate the influence function and bound the quality of approximation. The *Sample* algorithm represents the majority of the work of the algorithm and resembles a randomized BFS where vertices are introduced in the next frontier with some probability that is determined by the diffusion process being optimized. The *SelectSeeds* stage finds the most influential seed by solving a maximum coverage problem over the results of the exploration performed during the *Sample* algorithm. Two traits make this algorithm particularly challenging to optimize. First, the *Sample* step requires many randomized BFSs starting from random places in the input graphs and that tend to terminate quickly. Second, the number of such explorations grows quickly by increasing the requirements on the quality of solutions, making any attempt at limiting the algorithm at using the sole accelerator memory ineffective in any practical instance.

3 FPGA ACCELERATION

Previous works in FPGA-accelerated graph algorithms have shown considerable speedup and have led to reduced energy consumption compared to CPU implementations [3]. This work focuses on the FPGA acceleration of the sampling stage of the IMM algorithm, as this step has proven to more reliably benefit from acceleration in cuRipples. We utilized the Xilinx Vitis Unified Software Platform, and used the OpenCL and Xilinx Runtime Environment (XRT) API framework for the host code and OpenCL kernels. The two biggest challenges when designing an accelerated FPGA kernel for the sampling stage are memory accesses and loop complexity, causing bottlenecks in a naïve FPGA port. Our study identifies these two challenges and proposes optimizations to circumvent them.

Memory Accesses: By relying on graph traversal, the Sampling stage performs irregular array accesses. Thus, the FPGA can not take advantage of contiguous, coalesced array accesses during data transfer. This results in multiple repeated accesses to the same array. Since the graphs are typically too large (>128KB) to fit the Xilinx FPGA's high bandwidth PLRAM, they have to be stored in lower bandwidth off-chip DRAM, causing large memory stalls due to frequent accesses to high latency DRAM. Due to the random and relatively short traversals, graph pre-processing to achieve more coalesced memory accesses such as Sabet et al. [5] would be ineffective.

Loop Complexity: The loops within the base IMM algorithm have variable bounds, early conditional exits, subloops, and logic between nested loops, causing them to be classified by the Vitis HLS as imperfect loops and therefore unable to be flattened [1]. This prevents the Vitis HLS workflow from pipelining the largest loop in the kernel, leading to increased loop latency.

3.1 Sampling on FPGA

The Sampling stage is configured in a similar fashion to cuRipples [4] but with modifications in the kernel to better leverage the FPGA architecture. We used the linear threshold (LT) diffusion model as described in cuRipples for this preliminary study. The work scheduling is configured the same as cuRipples with each CPU core sending work to one FPGA via OpenMP. We extracted the pseudorandom number generation algorithms from Tina's Random Number Generator Library [2] used in the host code and performed pseudorandom number generation on the FPGA itself. The starting seed of the random number generation is performed by the host at the beginning, and the state of the pseudorandom numbers are stored in DRAM for persistence between kernel calls. The memory access bottlenecks are mitigated by performing burst accesses of the random number state data from DRAM and batching them in smaller amounts to fit in low latency block RAM (BRAM), allowing for faster memory accesses. These burst accesses are performed at the beginning before performing computations on the batched

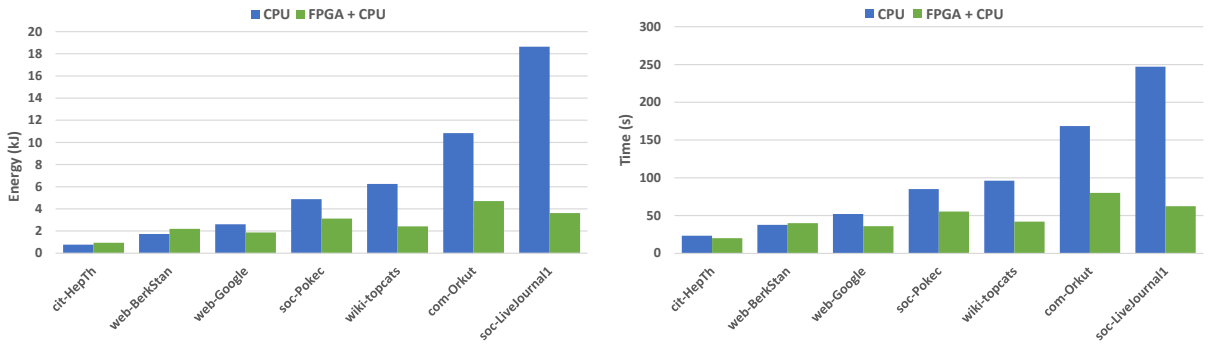

Figure 1: Energy and Performance Analysis: CPU (8 CPU Cores) and FPGA (7 CPU Cores, 1 FPGA). Ripples: ($\epsilon = 0.5, k = 100$)

memory, repeating the cycle once the current batch has been processed. Memory bursting ensures the only irregular non-coalesced DRAM accesses are from the graph itself. After the entire computation is complete, the batch of results are also written back to DRAM in another burst memory access, reducing the impact of DRAM's high latency. While some loops could be automatically pipelined by the Xilinx Vitis compiler, the largest loop could not. To help reduce the impact of this bottleneck, we synthesized four compute units to be run on one super logic region (SLR) in parallel, increasing DRAM bandwidth utilization and overall throughput. We also performed loop unrolling on known loop bounds to attempt to further increase throughput. An example of one known bound is the max BFS walk size of 8 used to ensure the burst memory access buffer size will fit within the BRAM resources.

4 EXPERIMENTAL SETUP

We wrote the kernels in OpenCL and synthesized them using the Vitis Unified Software Platform version 2020.2 while the parallel CPU implementation was written utilizing OpenMP. The system used in our experiments is equipped with an Intel Xeon E5-2637 CPU with 8x32GB 2400MHz DDR4 RAM and a Xilinx Alveo U250 FPGA with 4x16GB 2400MHz DDR4 RAM. Since all kernels were compiled on one Super Logic Region (SLR), only 1x16GB of DRAM was utilized on the FPGA. We tested two configurations: **1)** a full parallel CPU configuration using all eight cores, and **2)** a heterogeneous configuration with seven CPU cores running in parallel and one CPU core managing the Alveo U250 FPGA Data Center Accelerator Card. We utilized the same input graphs used by cuRipples from the SNAP data set containing real-world social networks.

5 PRELIMINARY RESULTS

Performance Comparison: Figure 1 (right) shows the preliminary execution time results of the two configurations across the considered graphs. There is a positive correlation between the number of nodes and the speedup achieved by the FPGA implementation, while the FPGA implementation lags behind for the smaller graphs. This happens because the static overhead of configuring the FPGA with the binary and managing data transfers becomes more prominent on smaller graphs due to the lower batch size and execution time, diminishing the speedup from FPGA acceleration.

Energy Consumption: We collected the preliminary host energy consumption using perf_events with RAPL, and FPGA energy consumption with the profiler tools available within the Vitis Unified Software Platform. Figure 1 (left) shows there is a similar positive correlation between energy savings and node count, with energy savings beginning after the two smallest graphs (e.g., cit-HepTh and web-BerkStan).

Resource Usage Analysis: We collected preliminary resource utilization results from the Vitis Unified Software Platform profiler, and table 1 shows individual kernel utilization. A single linear threshold kernel uses few resources on the FPGA, leaving room for more parallelization until the work becomes memory bandwidth bound by the high latency DRAM. The current implementation has four linear threshold kernels synthesized on the device while still showing low resource usage.

FPGA Kernel	FFs	LUTs	BRAMS
Graph Init	5,763 (0.17%)	4,544 (0.28%)	4 (0.16%)
LT Kernel	43,345 (1.32%)	32,323 (1.99%)	24 (0.95%)
All Combined	179,143 (5.44%)	133,836 (8.25%)	100 (3.96%)

Table 1: Utilization of flip-flops (FF), look-up tables (LUT), and block RAMs (BRAM) with percentages of total resources.

6 CONCLUSION AND FUTURE WORK

We developed a heterogeneous CPU + FPGA implementation of the Sampling step within the IMM Algorithm and found up to 3.96× speedup and up to 5.15× lower energy consumption in preliminary tests on graphs compared to CPU-only implementation. We identified bottlenecks in creating the FPGA linear threshold kernel in OpenCL with Vitis and introduced optimization techniques to lower their impacts. We intend to accelerate the seed selection stage on FPGA and expand the parallelized sampling kernels to all four SLRs. Expanding to multiple SLRs will improve overall FPGA utilization and could raise the memory bandwidth ceiling since each SLR has their own DRAM. Because these preliminary results were tested on a single FPGA, we hope to see future work exploring multi-FPGA implementations on a single node and across several nodes with the resulting performance and energy savings benefits. Another interesting exploration direction is comparing the OpenCL implementation with a hand-tuned FPGA implementation to see the performance impact of using high level synthesis on this algorithm. This work lays the groundwork for more FPGA acceleration exploration on not only simple graph kernels, but more complex graph analytics and workflows such as influence maximization.

REFERENCES

[1] 2021. HLS Pragmas. https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/hls_pragmas.html

[2] Heiko Bauke. 2021. *Tina's Random Number Generator Library.* https://www.numbercrunch.de/trng/trng.pdf

[3] Xiaoyu Ma, Dan Zhang, and Derek Chiou. 2017. FPGA-Accelerated Transactional Execution of Graph Workloads. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '17)*. ACM.

[4] Marco Minutoli, Maurizio Drocco, Mahantesh Halappanavar, Antonino Tumeo, and Ananth Kalyanaraman. 2020. CuRipples: Influence Maximization on Multi-GPU Systems. In *Proceedings of the 34th ACM International Conference on Supercomputing* (Barcelona, Spain) *(ICS '20)*. ACM.

[5] Amir Hossein Nodehi Sabet, Junqiao Qiu, and Zhijia Zhao. 2018. Tigr. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM.

[6] Youze Tang, Yanchen Shi, and Xiaokui Xiao. 2015. Influence Maximization in Near-Linear Time: A Martingale Approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. ACM.