Performance Comparison Between GPU and Hardware Accelerator for Bloom Filter

CSCI 338 Parallel Processing, Final Project (Prof. Kelly Shaw)

Josh Minwoo Kang and Andrew Thai

(Date: November 11, 2019)

PROJECT PROPOSAL

This project aims to accelerate the execution of membership queries using a Bloom filter (BF) on large text data, first using a GPU with CUDA programming, and then using a BF hardware accelerator. Performance gains from each approach will be compared.

For our GPU acceleration, we will explore different parallelization and optimization schemes, such as (1) varying the CUDA block dimensions, (2) optimizing the memory usage with shared memory and (3) pre-processing the input data. Similarly, we will try and test different implementations of the hardware accelerator with (1) a naive implementation of a BF on hardware and (2) an architecture to possibly support parallelized item insertions and queries.

I. THE BLOOM FILTER

A Bloom filter is a data structure that is used to represent a set [1]. Its main function is to test whether an element is part of a set in a space space and time efficient manner. However, it does not store data about the element (a separate data structure should be used to store the elements/data). Rather, it is a logical array (initialized with 0's) that keeps track of whether an element has been mapped to a position or not. That is, whenever an element is added, the BF will use its hash function(s) to and place a 1 where the element would be mapped to. Then, the search function of the bloom filter checks whether an element is not in the set, or potentially in the set, by checking the index of where the element would map to. Note that the search function cannot guarantee that an element is in the set since a 1 only indicates that *some* element(s) was mapped to that index.

Our implementation will be reading in text from input files, first placing words into the set. Then, we will have another input file with words to search. Thus, the key areas to focus on are the mapping and searching functions.

II. CUDA GPU IMPLEMENTATION

Each thread block will handle an input file and each thread within the block will process a word. We will implement two kernels. The first will generate the filter while the second will be used to check if a word is in the set. In the initial implementation, the Bloom filter will be stored on global memory; each thread will write its result onto this array when its computation is complete.

A. Organization of Parallelism

For our implementation in CUDA, we will explore different configurations of the thread block dimensions. Since the Bloom filter is a 1D array, our primary implementation will use 1D blocks. Another potential configuration is the use of 2D blocks where each thread is mapped to an index of a 1D array, similar to how we transformed a 2D array into a 1D array for the blurring assignment.

B. Non-parallelization Optimizations

We can use shared memory within each block to store local Bloom filters so that we do not repeatedly write to a BF stored on global memory. Each block will work on their own copy of the BF, placing their results into the global BF when all blocks complete their computations.

A second non-parallelization optimization we can perform is prepossessing the input text. We can remove all instances of stop words, which are words that have little significance in search queries, but high frequency. Thus, removing these words will reduce the size of the data we are working with.

III. BLOOM FILTER HARDWARE ACCELERATOR

Our implementation of a Bloom filter hardware accelerator will largely follow the model presented by Lyons and Brooks [2]. However, instead of synthesizing the accelerator as an ASIC chip, we will attach our unit as a co-processor to the RISC-V in-order, single-issue scalar Rocket core; hardware will further be simulated using Verilator, which is a cycle-accurate behavioral model written in C++.

The accelerator architecture will consist of a Bloom bit array, hash function units and hardware to support the interfacing with the host processor. We will further attempt to parallelize item insertion and querying so that the accelerator can work with larger chunks of data at once–however, the host-accelerator data communication is the major bottleneck. To cope with this limitation, we envision an vector of Bloom filters with a pre-processing unit to efficiently allocate workloads to each filter.

REFERENCES

[1] A. Iacob, L. M. Itu, L. Sasu, F. Moldoveanu, and C. Suciu, "GPU accelerated information retrieval using Bloom filters," pp. 872–876, 10 2015.

[2] M. J. Lyons and D. Brooks, "The Design of a Bloom Filter Hardware Accelerator for Ultra Low Power Systems," in *Proceedings of the 2009 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '09, (New York, NY, USA), pp. 371–376, ACM, 2009.