Error correction in high-throughput short-read data on GPU (Mid-Term Progress)

Aman Mangal, Chirag Jain

November 12, 2014

1 Outline

Our goal for this checkpoint was to verify our hypothesis that there is possibility of improvement in the error correction time of the reads if we improve cache hit rate for the bloom filter accesses on GPU. We have compiled and executed the CUDA-EC software implemented by Shi et. al. [1] on a Kepler GPU using CUDA 6.5. We have collected the statistics by profiling the code execution using couple of real datasets in order to support our argument.

Link to the public bitbucket repository: https://bitbucket.org/cjain7/cse6230-project

2 Bloom Filter

Bloom filter is a highly space efficient probabilistic data structure for group membership query. It is used in the CUDA-EC algorithm to query the presence of a given k-mer in high multiplicity k-mer set (i.e. solid k-mers). The algorithm uses 8 hash functions to compute the bit indices corresponding to a given k-mer and sets them in the bloom filter. While inquiring the presence of a k-mer, bit indexes are computed using the same hash functions and queried in the bloom filter. In the standard implementation of bloom filter,

checking the presence of an element requires random memory accesses which leads to frequent cache misses. Some research papers such as [2, 3, 4] propose solutions to overcome this issue. While profiling the code execution on the dataset SRR006331 (Table 1), we observed that 98% of all the queries made to the bloom filter got a negative response from the filter.

3 Cache Hit Rate

3.1 Using texture memory

Once the bloom filter is constructed during the pre-processing stage, CUDA-EC[1] saves the hash table inside the texture memory of the GPU. Authors claim that the memory throughput for the bloom filter queries is crucial for the overall software performance.

Texture memory is a type of read-only memory available on the GPUs. Since it is cached on the chip, it can be used to derive much better memory throughput than the global memory, especially if memory accesses by a warp exhibit some spatial locality pattern.

We have utilised the Kepler GK110 GPU processor to benchmark the CUDA-EC code. The GK110 processor is equipped with a 48 KB read-only cache per SM, which is an extension of the texture cache available in the older generations of GPUs[5]. The texture requests are also cached on 1.5 MB L2 cache, which is shared among all the SMs. Texture memory throughput depends upon how good the hit rate is on both of these cache levels.

3.2 Profiling results

We have used the *nvprof* CUDA profiler to compute the cache hit rates for the texture requests. Table 1 shows the datasets we have used for the experiments. Table 2 shows the timings of the GPU kernel and hash table size while Table 3 shows the cache hit-rates achieved on these datasets.

From the above numbers, we infer that the cache hit rates are low for the texture cache and very low at the L2 level. This leaves us with enough scope

Table 1: Datasets

Accession number	Platform	No. of reads	Read length (bp)
SRR006331	Illumina	1,693,848	36
SRR016146	Illumina	4,443,913	51

Table 2: Execution details

Parameter	SRR006331	SRR016146
GPU kernel time	7.42 sec	12.83 sec
Hash table size	18.69 MB	24.87 MB

Table 3: Cache hit rates

Metric	SRR006331	SRR016146
Texture Cache Hit Rate	29.25%	22.64%
Texture Cache Throughput	407.56 MB/s	321.31 MB/s
L2 Hit Rate (Texture Reads)	3.26%	2.22%
L2 Throughput (Texture Reads)	298.37 MB/s	249.04 MB/s
Texture Cache Transactions	82,681,342	118,097,327

for optimising the texture memory throughput.

4 Future Work

We plan to replace the existing naive bloom filter implementation in the CUDA-EC code with aforementioned algorithms to improve the cache hit rate of Texture Memory. Testing the code on more datasets will also strengthen our conclusions. If time persists, we also plan to look into optimization opportunities in the code to further reduce the error correction timings on GPU.

References

[1] H. Shi, B. Schmidt, W. Liu, and W. Müller-Wittig, "A parallel algorithm for error correction in high-throughput short-read data on cuda-enabled graphics hardware," *Journal of Computational Biology*, vol. 17, no. 4, pp. 603–615, 2010.

[2] Y. Chen, A. Kumar, and J. Xu, "A new design of bloom filter for packet inspection speedup," in *Global Telecommunications Conference, 2007. GLOBECOM'07. IEEE*, pp. 1–5, IEEE, 2007.

[3] F. Putze, P. Sanders, and J. Singler, "Cache-, hash-, and space-efficient bloom filters," *J. Exp. Algorithmics*, vol. 14, pp. 4:4.4–4:4.18, Jan. 2010.

[4] E. Krimer and M. Erez, "The power of $1+\alpha$ for memory-efficient bloom filters," *Internet Mathematics*, vol. 7, no. 1, pp. 28–44, 2011.

[5] NVIDIA, "Nvidia kepler gk110 architecture whitepaper," Link: `http://www.nvidia.com/content/PDF/kepler/NVIDIA-kepler-GK110-Architecture-Whitepaper.pdf`.