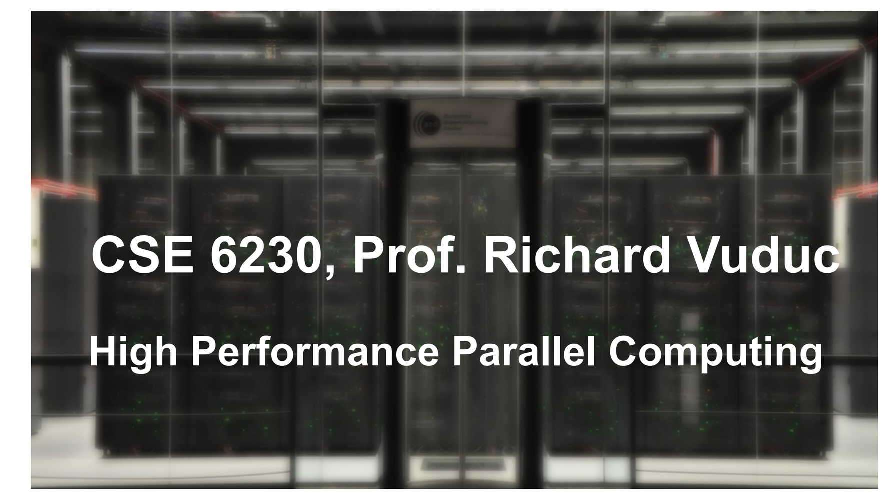
Optimizing Short Read Error Correction on Graphics Processing Unit (GPU)

Aman Mangal (amanmangal@gatech.edu)
Chirag Jain (cjain@gatech.edu)
Georgia Institute of Technology, Atlanta, GA



Abstract

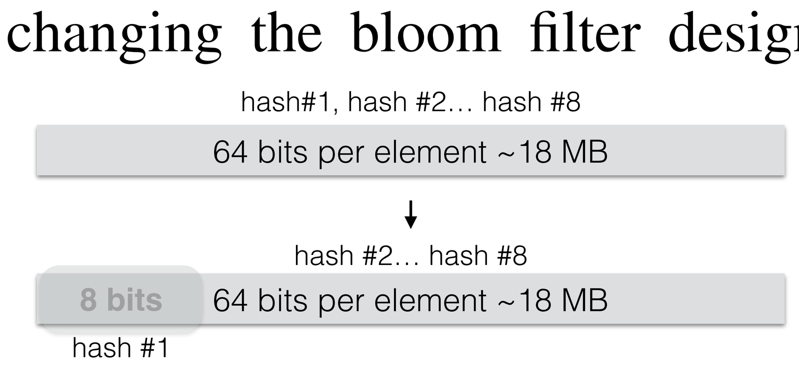
*We propose a modified parallel implementation for the state of the art error correction software CUDA-EC[1]. We reduce the kernel execution time by approximately **14.81%**. This was achieved by increasing warp efficiency from **3.5% to 67.5%** and some other code optimization*

CUDA-EC Profiling Results

1. **Single GPU thread corrects one read**, leads to high warp divergence (Warp Execution Efficiency 3.5%)
2. **No use of shared memory**. extensive use of local memory instead
3. **High register usage per block**. limits kernel's ability to fully utilize the GPU

Improving Bloom Filter's Access throughput

CUDA-EC utilizes bloom filter, a space-efficient probabilistic data structure to hash the frequently occurring k-mers. We tried changing the bloom filter design to make it more cache efficient.



Metric	Value (Before)	Value (After)
Time	5.73 sec	6.16 sec
Texture L2 hit rate	1.96%	14.45%
L2 texture throughput	2.76 GB/s	6.29 GB/s
Total texture accesses	68,381,701	91,551,531

Algorithm and Implementation Details

1. **Improving warp divergence:** Restructured the code by having single read processed by one warp
2. **Using shared memory:** Using selected underlying buffers in shared memory instead of local memory
3. **Other optimization**

(a) Loop unrolling
(b) Reducing register count by removing unnecessary variables as well as correcting the scope of some variables
(c) Removing unnecessary copying of reads, simplifying branches

```
Allocate vote[] ;                              // In local memory
while read chunk ≠ Empty do
    /* Access to read for this thread tid */
    current_read ← global_mem_all_reads[tid];
    while current_read_correction_ongoing do
        for Every tmpTuple k-mer in current_read do
            if tmpTuple is not solid i.e. rare then
                for Each character position i in tmpTuple do
                    for newtmpTuple ← mutate i with other
                    nucleotides m : (0 . . . 3) do
                        if newtmpTuple is solid then
                            Update vote[i][m]+ = 1
                        end
                    end
                end
            end
        end
        if all tmpTuple were solid then
            current_read_correction_ongoing ←FALSE
        end
        /* Sequential iteration over vote[][]
           to find maximum maxVote = vote[p][q]
           */
        /* Update current_read[p] =
           Nucleotide[q]                                */
    end
end
```

Algorithm 1: Original CUDA-EC algorithm

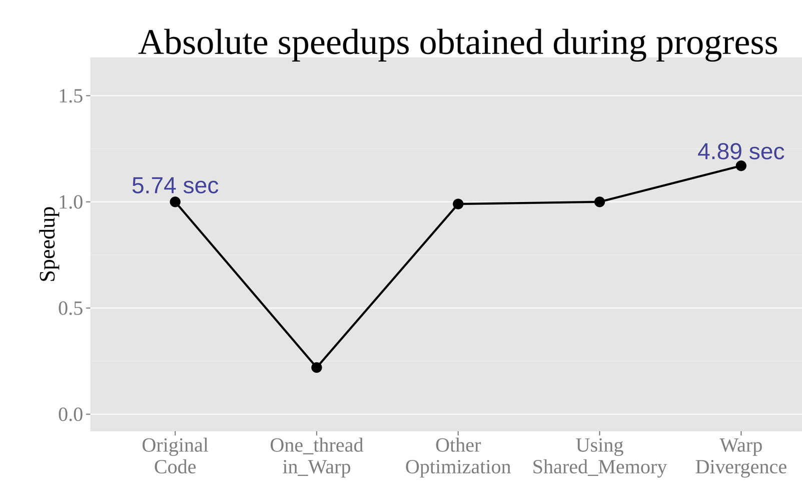
```
Allocate vote[] ;                              // In shared memory
while read chunk ≠ Empty do
    /* Access to read for this warp tid                */
    /* Threads in warp load read to shared
       mem from global                                 */
    current_read ← shared_mem_all_reads[warp_id];
    while current_read_correction_ongoing do
        for Every tmpTuple k-mer in current_read do
            if tmpTuple is not solid i.e. rare then
                for Each character position i = 0 in
                tmpTuple; i+=WARPSIZE do
                    for newtmpTuple ← mutate i with other
                    nucleotides m : (0 . . . 3) do
                        if newtmpTuple is solid then
                            Update vote[i][m]+ = 1;
                            tmpTupleUnChanged = false;
                        end
                    end
                end
            end
        end
        if __all(tmpTupleUnChanged) then
            current_read_correction_ongoing ←FALSE
        end
        /* REDUCTION within warp for vote[][]
           to find maximum maxVote = vote[p][q]
           */
        /* Update current_read[p] =
           Nucleotide[q]                                */
    end
end
```

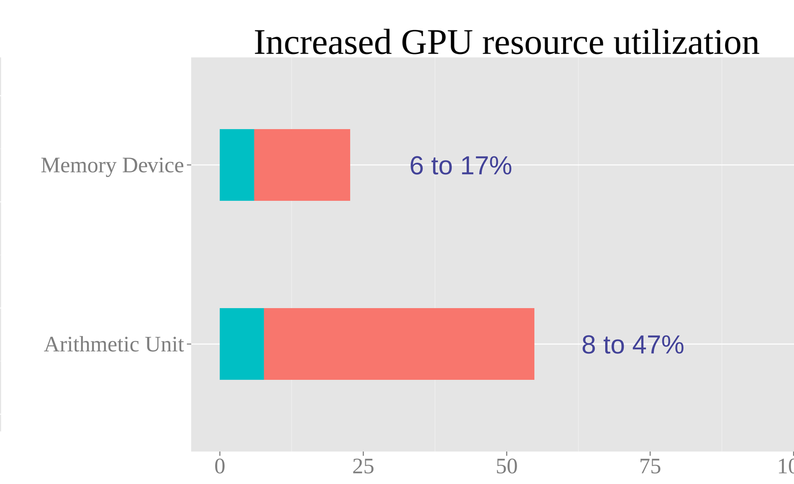
Algorithm 2: New CUDA-EC execution flow

Results

Metric	Value (OLD)	Value (NEW)
Warp execution efficiency	3.5%	67.5%
L1/Shared bandwidth	57.6 GB/s	146.9 GB/s
L2 bandwidth	7.5 GB/s	40.6 GB/s
Device memory bandwidth	2.0 GB/s (Limit: 208 GB/s)	37.75 GB/s
Shared memory usage	0 bytes	1.95 KB
Register usage	85 per thread	56 per thread
Occupancy	24.6%	47.1%

Table 1: Profiling results of original code computed on Kepler K20m GPU





Conclusions

- Improved runtime by **approximately 15%** without changing output
- **Compute and latency bound** algorithm as it performs heavy arithmetic with the available data
- Showed that **multiple-threads-one-read** model has enough instruction parallelism to exploit on GPU (classical model is one-thread one-read)

Future work

- Using newly introduced instruction for instance *shuffle* to carry out reduction operations and communication within a warp
- Making the software work on large reads and data sets (CUDA-EC itself is not scalable for large reads)
- Implementing the algorithm from scratch using latest version of CUDA without constraints such as getting precisely same output data as the original code

References

[1] Haixiang Shi, Bertil Schmidt, Weiguo Liu, and Wolfgang Muller Wittig. A parallel algorithm for error correction in high-throughput short-read data on cuda-enabled graphics hardware. *Journal of Computational Biology, 17(4):603615, 2010.*

Acknowledgements

We thank Nagakishore Jammula, senior PhD student at Georgia Tech for his inputs on this problem and allowing us to use CyEnce cluster.