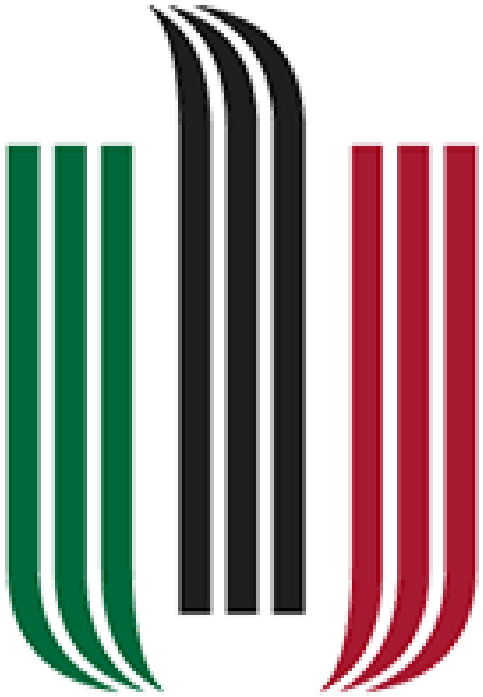


AGH

AGH University of Science and Technology

Introduction to CUDA and Open CL

Lab Reduction

Michał Kunkel
Wiktor Żychowicz

1. Reduction Algorithms

There is a very common set of algorithms which reduce blocks of data into smaller ones(typically single value). We call them reduction algorithms. To name a few: Image Segmentation, Maximum Bipartite Matching, Sum. We can find in them parallel potential. Of course it depends on how individual chunks of input data affect one another. Ideally we can divide input into small(proceeded by one thread), independent blocks on which we can perform in any order.

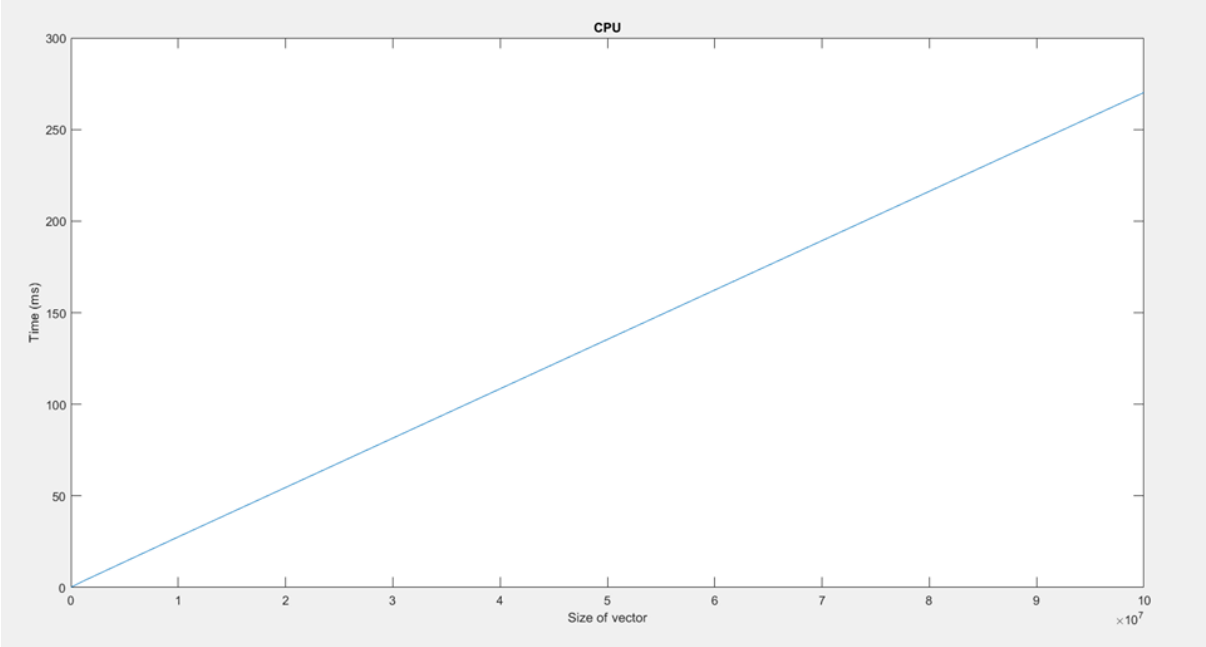
Good example of such a task is addition of vector elements. In next part we will discuss how using CUDA we can achieve O(log n) complexity (however fetching data is still O(n)) in comparison to one thread algorithm with O(n) complexity.

Code of implementation: https://github.com/Sightster/CUDA_Gr_5.

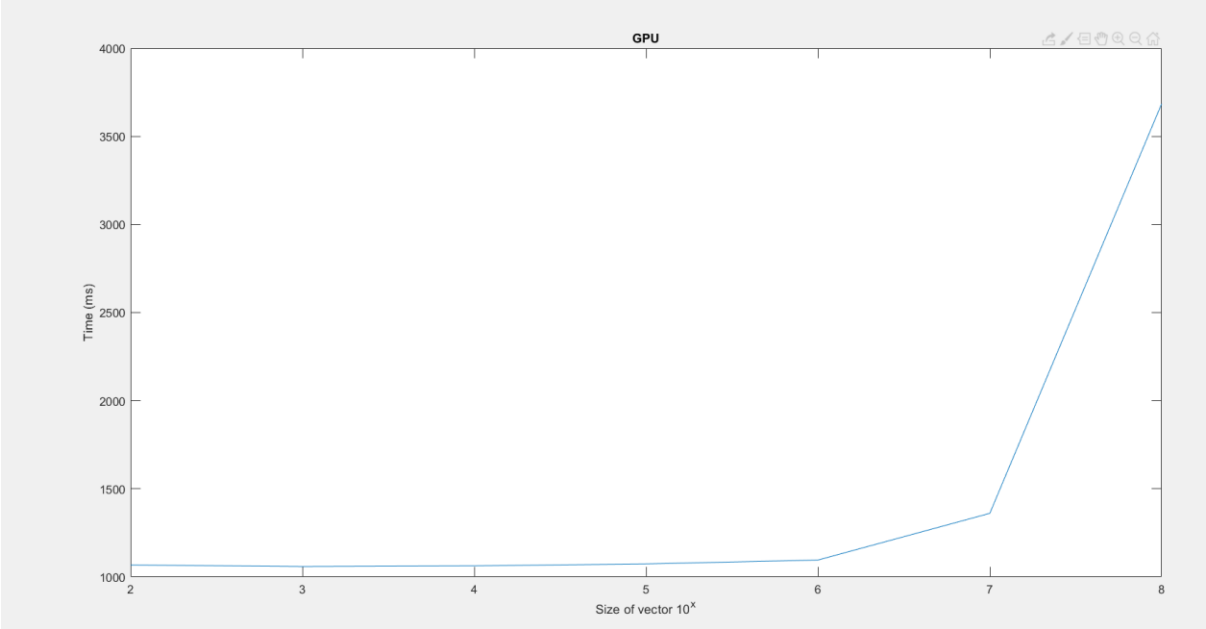
2. Sum of the vector.

In our implementation of one we make use of the shared memory and also we prefetched vector on GPU(previous version which divide input on CPU and fetch them independently for each kernel was slower and scaled in a worse way). Each block of threads loads 2 times more vector's elements than number of thread it consist to the shared memory than each of threads sum two elements. In the next step we "kill" every second thread and repeat addition. We repeat until vector within shared memory is reduced to a single value, then we move it into global memory. Our result vector is number of threads in block smaller than input one. Now we can reduce result one and repeat the circle until we get single value result (unfortunately recursion implementation couldn't handle big size of input vector so we ended up with code which is available on GitHub in ReductionFinal directory).

Performance time from size of input data:



Picture 1: CPU perfect O(n).



a

Picture 2: GPU witch spikes form data fetching.

3. Summary.

We did not achieve GPU supremacy however we clearly show trend which implies that reduction with CUDA scale better than simple CPU implementation. Also huge time of GPU performance is assimilated with data movement from CPU to GPU. Signification of this task drop down when we perform pipelined algorithms on GPU(without fetching data back to host). It is just a happy coincidence that reduction is often part of the bigger algorithms. There is also a big room for improvement in implementation (we can e.g make use of idling threads). So there is definitely advantage in GPU reductions when conditions are right.