GPU Computing Assignemnt-1 Report

Aditya Rana

Roll.No.:2019405, aditya19405@iiitd.ac.in

1 DOCUMENTATION

Allocated “int pdf” array, “int cdf” array, “unsigned char mapping” array, “unsigned char img_in” array and “unsigned char img_out” array in both host and device. Used malloc for host variables and cudaMalloc for device variables.

Broke Step-1 of “ahe_cpu.cpp” into two parts. Part-1, iterate over every pixel to calculate PDFs corresponding to every tile. Part-2, iterate over every tile to calculate CDFs from corresponding PDFs.

- Calc_pdf_kernel : Kernel to parallelize part-1 of step-1. To calculate histogram of pixel values in each tile. For each pixel index find out the tile index in which that pixel lies. Block dimension = 32x32. Grid dimension = (width/32) x (heightx32).
- Calc_mapping_kernel : Kernel to parallelize part-2 of step-1. To iterative to histogram and calculate CDF and mapping for each tile. Number of blocks = number of tiles in image. Number of threads per block = 256.

Step-2, perform adaptive equalization. Reduced two nested loops to iterate over pixel.

- Adap_eq_kernel : Calculate resultant pixel value for every pixel parallelly

Computation using constant memory. Allocated “unsigned char MAP[]”, constant memory to “unsigned char mapping” array. Created new kernels – “const_calc_mapping_kernel”, “const_adap_eq_kernel”, for “calc_mapping_kernel” and “adap_eq_kernel” respectively. Changed implementation simply by using constant array in place of global mapping array.

Created device function “__device__ unsigned char interp()” as helper function for “adap_eq_kernel”.

.

2 ANALYSIS

Analysis of CPU and GPU runtime with varying parameters such as Tile size and change in approach. Executed CPU and GPU code taking "earth.png" as input image.

2.1 Tables

Below table shows the tile size and the corresponding time taken by CPU and GPU in milliseconds for adaptive equalization. Last column indicates the RMS error values of GPU output with respect to CPU output.

Table 1: CPU & GPU runtime

TILE SIZE	CPU Time	GPU Time	RMS Error
256	1658.53	54.8207	1.91908
512	1660.73	56.0001	0.00029901
1024	1660.7	60.8787	0.000559396
2048	1662.91	73.25	0.000622439
4096	1666.13	80.0928	66.2174

[a] This is example of table footnote.

According to observations there is a speedup by factor of 26.08 on average on given tile sizes.

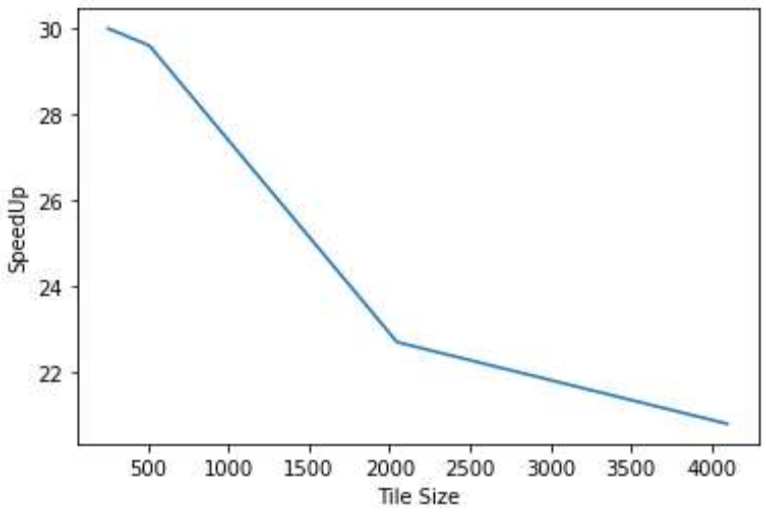

Figure: Speedup-vs-TileSize plot

Varying TILE_SIZE_X and TILE_SIZE_Y independently we observe the below GPU runtimes (in milliseconds). Along x-axis is varying TILE_SIZE_X and along y-axis is varying TILE_SIZE_Y.

For larger TILE_SIZE_Y (i.e., 2048, 4096) there is speedup in runtime by an average factor of 22.5.

	256	512	1024	2048	4096
256	54.8207	52.1444	51.8514	51.5438	51.7053
512	56.2559	56.0001	55.5702	55.7849	55.8879
1024	60.8154	60.7834	60.8787	60.8491	61.3688
2048	72.9439	73.1067	73.4572	73.4151	73.658
4096	78.5955	78.8155	79.1215	79.215	80.0928

2.2 Only two kernels

On implementing the step-2 (mentioned in section-1) using nested loops instead of corresponding kernel observed an RMS error of zero. Though this approach gave a speedup of average 10.7.