Report

Parallel Computing Assignment 2

Names	BN	Sec
Abdelaziz Salah Mohammed	3	2
Ahmed Hosny Abdelrazik	2	1

Delivered to:

Eng/ Mohammed Abdullah

Abdelaziz Salah

Ahmed Hosny

Date: 26/2/2024

Introduction:

We have created many different test cases to be able to analyze the performance in a better way, so we used the provided test cases, and then we have created another 3 test cases each of sizes: 10x10, 100x100, 1000x1000 respectively, and then we ran them and got the results which will be shown in the following screenshots.

Also, we have 3 main kernels as required in the requirement, but we have a fixed window of 16 x 16 threads per block, and we used this because we have found out that the best window size should be 256 and it is a common choice also.

However, the number of used blocks varies depending on the number of rows and columns using the equation used in the lecture by dividing the rows / 16 then getting the ceiling, and same operation is done on the columns, to have 2 dimensions block.

Moreover, we have made 2 assumptions:

1. size of vector is number of columns to apply multiplication (Matrix (n,m) *vector (m,1) the result is (n,1))
2. input files have numbers only (no comments)

Analysis:

Here we will show the total time taken by executing the kernals only, then at the end of the report we will provide screenshots for time taken for allocating memory on the GPU, copying elements and so on.

	K1	**K2**	**K3**
10x10	3520 ns	4448 ns	7584 ns
100x100	3168 ns	22529 ns	42593 ns
1000x1000	3392 ns	16928 ns	37921 ns

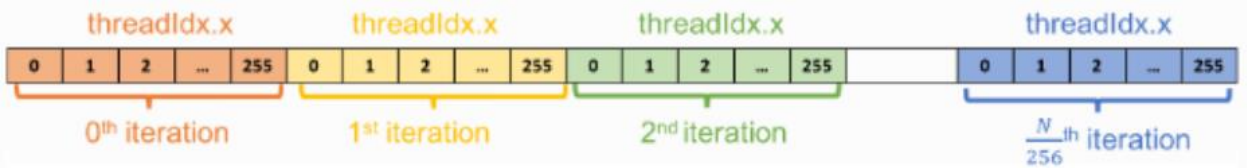
Abdelaziz Salah Ahmed Hosny

Date: 26/2/2024

As we expected, K1 is the fastest kernal, and this is obvious, because we have a lot of threads which are running concurrently, so all of them will calculate the addition of two corresponding cells in matrix A, and B, then evaluate the result in the matrix C.

What is interesting now is why does kernal 3 slower than kernal 2?

If we remember how elements are sorted in the memory, we will notice that the matrix is linearized in the memory in format similar to this:



So, this implies that elements in the same row are inserted next to each other, that indicates that threads could get the whole row inside its cache, and get the data closer to it, then apply the computation.

So, this allows it to reduce the number of IOs which is the bottle neck usually and decrease latency.

However, in kernel 3, we fix the column, and iterate over rows, so we stride with huge step in each iteration to get the col index in each row, so it does not use the benefit of having all elements next to each other.

Other Factors:

Here is an analysis of other factors such as time taken for copying data to GPU, and allocating space there and so on.

Usually transferring data, and allocating space, takes a huge amount of time relative to processing the data on the GPU.

Abdelaziz Salah Ahmed Hosny

Date: 26/2/2024

K1:

10x10:

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
99.2	136513003	33	4136757.7	2224.0	1974	136437511	23750311.4	cudaMalloc
0.6	759109	33	23003.3	9247.0	7003	300762	51260.3	cudaMemcpy
0.1	199310	33	6039.7	1773.0	1523	135688	23296.0	cudaFree
0.0	55556	1	55556.0	55556.0	55556	55556	0.0	cudaLaunchKernel
0.0	19867	1	19867.0	19867.0	19867	19867	0.0	cuModuleGetLoadingMode
0.0	14497	1	14497.0	14497.0	14497	14497	0.0	cuModuleUnload
0.0	5711	1	5711.0	5711.0	5711	5711	0.0	cuCtxSynchronize
0.0	511	1	511.0	511.0	511	511	0.0	cuDeviceGetLuid

100x100:

97.5	197406393	303	651506.2	2274.0	1954	196615390	11295108.8	cudaMalloc
2.1	4171370	303	13766.9	10590.0	6483	194099	12295.7	cudaMemcpy
0.4	880723	303	2906.7	1983.0	1573	244114	13936.9	cudaFree
0.0	62809	1	62809.0	62809.0	62809	62809	0.0	cudaLaunchKernel
0.0	28865	1	28865.0	28865.0	28865	28865	0.0	cuModuleUnload
0.0	13806	1	13806.0	13806.0	13806	13806	0.0	cuModuleGetLoadingMode
0.0	5069	1	5069.0	5069.0	5069	5069	0.0	cuCtxSynchronize
0.0	380	1	380.0	380.0	380	380	0.0	cuDeviceGetLuid

1000x1000:

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
95.7	124705228	303	411568.4	2234.0	1884	123925988	7119213.9	cudaMalloc
3.6	4633336	303	15291.5	11341.0	6953	175944	13225.7	cudaMemcpy
0.7	866437	303	2859.5	2014.0	1523	209709	12024.8	cudaFree
0.1	67558	1	67558.0	67558.0	67558	67558	0.0	cudaLaunchKernel
0.0	48302	1	48302.0	48302.0	48302	48302	0.0	cuModuleUnload
0.0	18565	1	18565.0	18565.0	18565	18565	0.0	cuModuleGetLoadingMode
0.0	6943	1	6943.0	6943.0	6943	6943	0.0	cuCtxSynchronize
0.0	732	1	732.0	732.0	732	732	0.0	cuDeviceGetLuid

Abdelaziz Salah Ahmed Hosny

Date: 26/2/2024

K2:

10x10:

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
98.8	176991151	33	5363368.2	2125.0	1913	176902020	30794204.2	cudaMalloc
1.0	1871347	33	56707.5	46829.0	38533	304218	45824.1	cudaMemcpy
0.1	226742	33	6871.0	1844.0	1573	152881	26288.7	cudaFree
0.0	85903	1	85903.0	85903.0	85903	85903	0.0	cudaLaunchKernel
0.0	20839	1	20839.0	20839.0	20839	20839	0.0	cuModuleGetLoadingMode
0.0	15810	1	15810.0	15810.0	15810	15810	0.0	cuModuleUnload
0.0	5761	1	5761.0	5761.0	5761	5761	0.0	cuCtxSynchronize
0.0	471	1	471.0	471.0	471	471	0.0	cuDeviceGetLuid

[6/8] Executing 'cuda_gpu_kern_sum' stats report

100x100:

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
96.0	129499072	303	427389.7	2174.0	1873	128738928	7395713.9	cudaMalloc
3.3	4388088	303	14482.1	10981.0	7204	192826	12963.0	cudaMemcpy
0.6	788617	303	2602.7	1903.0	1493	176285	10057.0	cudaFree
0.0	65965	1	65965.0	65965.0	65965	65965	0.0	cuModuleGetLoadingMode
0.0	60405	1	60405.0	60405.0	60405	60405	0.0	cudaLaunchKernel
0.0	27372	1	27372.0	27372.0	27372	27372	0.0	cuModuleUnload
0.0	6913	1	6913.0	6913.0	6913	6913	0.0	cuCtxSynchronize
0.0	811	1	811.0	811.0	811	811	0.0	cuDeviceGetLuid

[6/8] Executing 'cuda_gpu_kern_sum' stats report

1000x1000:

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
96.0	129499072	303	427389.7	2174.0	1873	128738928	7395713.9	cudaMalloc
3.3	4388088	303	14482.1	10981.0	7204	192826	12963.0	cudaMemcpy
0.6	788617	303	2602.7	1903.0	1493	176285	10057.0	cudaFree
0.0	65965	1	65965.0	65965.0	65965	65965	0.0	cuModuleGetLoadingMode
0.0	60405	1	60405.0	60405.0	60405	60405	0.0	cudaLaunchKernel
0.0	27372	1	27372.0	27372.0	27372	27372	0.0	cuModuleUnload
0.0	6913	1	6913.0	6913.0	6913	6913	0.0	cuCtxSynchronize
0.0	811	1	811.0	811.0	811	811	0.0	cuDeviceGetLuid

[6/8] Executing 'cuda_gpu_kern_sum' stats report

Abdelaziz Salah Ahmed Hosny

Date: 26/2/2024

K3:

10x10:

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
99.3	191055520	33	5789561.2	2745.0	2414	190961883	33241687.6	cudaMalloc
0.5	912706	33	27657.8	14698.0	8356	260525	44869.0	cudaMemcpy
0.1	250417	33	7588.4	2245.0	1883	163531	28077.6	cudaFree
0.0	69663	1	69663.0	69663.0	69663	69663	0.0	cudaLaunchKernel
0.0	19737	1	19737.0	19737.0	19737	19737	0.0	cuModuleGetLoadingMode
0.0	16121	1	16121.0	16121.0	16121	16121	0.0	cuModuleUnload
0.0	4969	1	4969.0	4969.0	4969	4969	0.0	cuCtxSynchronize
0.0	441	1	441.0	441.0	441	441	0.0	cuDeviceGetLuid

100x100:

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
99.3	191055520	33	5789561.2	2745.0	2414	190961883	33241687.6	cudaMalloc
0.5	912706	33	27657.8	14698.0	8356	260525	44869.0	cudaMemcpy
0.1	250417	33	7588.4	2245.0	1883	163531	28077.6	cudaFree
0.0	69663	1	69663.0	69663.0	69663	69663	0.0	cudaLaunchKernel
0.0	19737	1	19737.0	19737.0	19737	19737	0.0	cuModuleGetLoadingMode
0.0	16121	1	16121.0	16121.0	16121	16121	0.0	cuModuleUnload
0.0	4969	1	4969.0	4969.0	4969	4969	0.0	cuCtxSynchronize
0.0	441	1	441.0	441.0	441	441	0.0	cuDeviceGetLuid

1000x1000:

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
97.2	192659746	303	635840.7	2255.0	1853	191886741	11023458.4	cudaMalloc
2.3	4598365	303	15176.1	10440.0	7064	202254	14791.5	cudaMemcpy
0.4	810645	303	2675.4	1914.0	1493	184851	10543.9	cudaFree
0.0	68490	1	68490.0	68490.0	68490	68490	0.0	cudaLaunchKernel
0.0	38163	1	38163.0	38163.0	38163	38163	0.0	cuModuleUnload
0.0	15519	1	15519.0	15519.0	15519	15519	0.0	cuModuleGetLoadingMode
0.0	6652	1	6652.0	6652.0	6652	6652	0.0	cuCtxSynchronize
0.0	511	1	511.0	511.0	511	511	0.0	cuDeviceGetLuid

Abdelaziz Salah Ahmed Hosny

Date: 26/2/2024