Report

1. Description

In this assignment, we need to improve the performance of Matrix normalization and take advantage of the power of GPUs to perform it as fast as possible.

Following is my design decisions:

We know that in the sequential version, we first need to calculate the mean of each column. Secondly we use the mean of that column to get the standard deviation. Finally, calculate the normalized value for each element with equation: B[row][col]=(A[row][col]-mean)/standard_deviation. My basic idea of my CUDA version Matrix normalization is that. for each column, assigned to a CUDA thread, then each CUDA thread execute the task independently at the same time, which will decrease the total running time and improve the efficiency.

At the beginning of the assignment, I tried to use the sum-reduction method to improve the performance, but in fact, this sum-reduction cannot do better than my way.

In my code, I allocate a big block, which I use N CUDA threads to execute N column at the same time. In this way, the performance and efficiency can be improved a lot.

2. Performance Evaluation

I tested both the CUDA version and sequential of this algorithm on my local machine, with the GPU version GTX980M, and on Jarvis node, gpu-compute-0-2, with the GPU version GTX650.

The data size varies from 1000 to 8000.

① Following is the result data on Jarvis:

Data_Size	1000	2000	3000	4000	5000	6000	7000	8000
SEQUENTIAL	14.236	60.352	135.115	242.171	422.611	670.021	894.72	1328.48
CUDA	379.239	408.495	434.039	472.454	517.41	565.059	625.501	681.578

Table 1. the running time of Sequential version and CUDA version on Jarvis

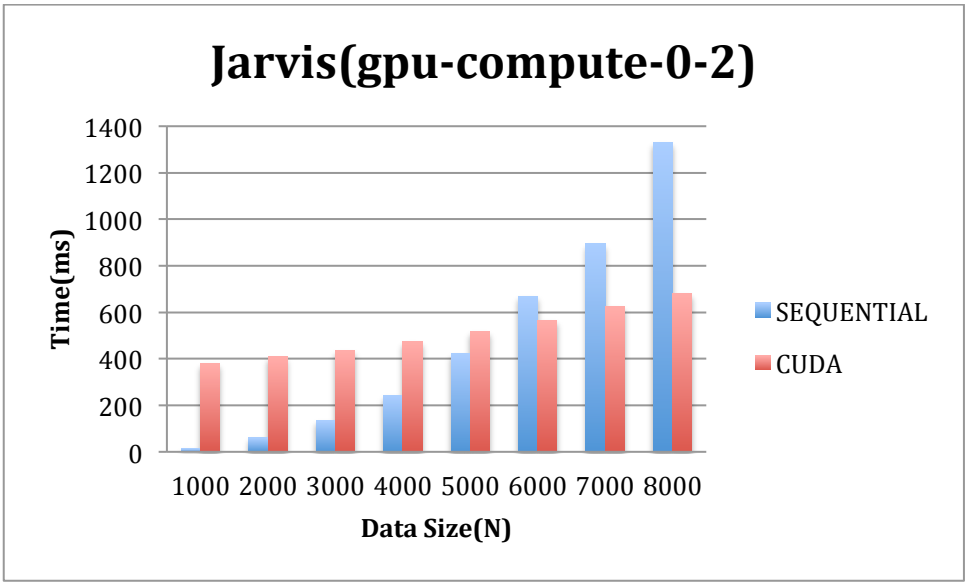

Figure 1. the running time of Sequential version and CUDA version on Jarvis

② Following table is the result data on local machine:

Data_Size	1000	2000	3000	4000	5000	6000	7000	8000
SEQUENTIAL	37.152	135.256	319.643	600.517	983.274	1346.39	2019.52	2824.04
CUDA	106.681	116.313	135.861	160.847	188.367	220.388	256.276	294.051

Table 2. the running time of Sequential version and CUDA version on local machine

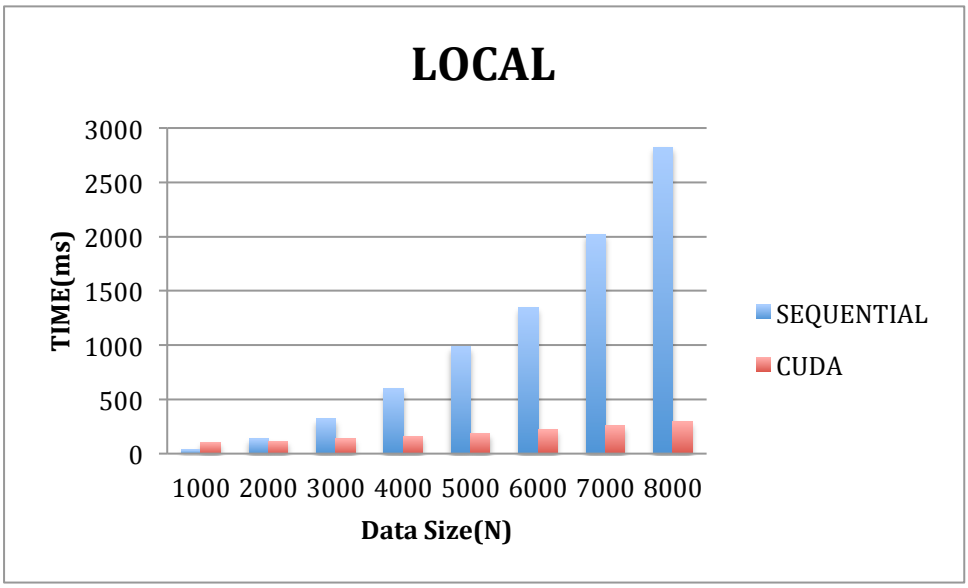

Figure 2. the running time of Sequential version and CUDA version on local machine

From the two figures above, we can see that, in sequential version, with the data size increasing, the running time improve a lot, almost an exponential growth. However, in CUDA version, there is a time of CUDAmemcpy(), memory copy at the beginning , so the cost time is more than sequential on small data size. With the data size become bigger, the proportion of time of the copy data becomes smaller and also the actual time of calculating is less,

so the total running is less and the efficiency improved.

Also, following table and figure will show the performance of CUDA on different GPU.

Data_Size	1000	2000	3000	4000	5000	6000	7000	8000
GTX650	379.239	408.495	434.039	472.454	517.41	565.059	625.501	681.578
GTX980M	106.681	116.313	135.861	160.847	188.367	220.388	256.276	294.051

Table 3. the running time CUDA version on Jarvis and local machine.

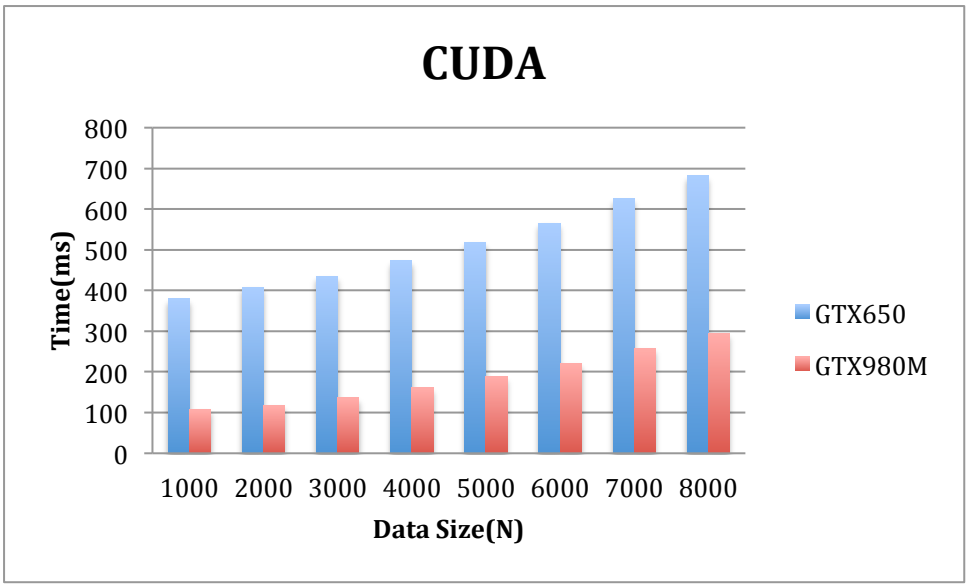

Figure 3. the running time CUDA version on Jarvis and local machine.

We can see that when the GPU is better, the performance is better.