**HOMEWORK -4**

**README File**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Files Used:**

matrix\_cuda\_norm.cu

matrix\_cuda\_norm.c

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Commands to run parallel file on Jarvis:**

To run the file matrix\_cuda\_norm.cu

1. Go into the directory using the command cd hw4 on the Jarvis
2. vim matrix\_cuda\_norm.cu
3. qlogin –q interactive.q
4. cd hw4
5. nvcc matrix\_cuda\_norm.cu –o matrix\_cuda\_norm.
6. ./ matrix\_cuda\_norm matrix\_dimension number\_of\_threads

Now for Sequential File matrix\_cuda\_norm.c. Run it with following commands:

1. gcc matrix\_cuda\_norm.c
2. ./a.out 2000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Methods Used:**

**Note: Given that Column normalization must be done.**

main(): this is the entry point of the program which declares the clock time and call the following inside main() function.

1. **\_\_global\_\_ void mean\_of\_matrix ():**Here the input is taken from the host and copied in to the device in an array form. Mean of the matrix is calculated using a total variable for each block. In which total is initialzed from 0 and the end result of the mean that is the total/N is stored at the 0th position of every block, according to the block size mentioned.  The name global here refers to scope, as it can be accessed by both the host and the device. Global memory allocations can persist for the lifetime of the application. Depending on the [compute capability](http://devblogs.nvidia.com/parallelforall/how-query-device-properties-and-handle-errors-cuda-cc/) of the device, global memory may or may not be cached on the chip.
2. **\_\_global\_\_ void matrixSD():** To find the matrix standard deviation the formula is provided: sdata1[threadId] = powf(deviceInput[j] - deviceMean[blockIdx.x], 2.0) which calculates the standard deviation and stores the final result divided N which is the block size in the 0th position of each position . For every function we have defined one condition that if the no of threads provided are less than the number of elements in the block then, the first slot of thread works on the elements and then again performs till the number of elements gets over.
3. **\_\_global\_\_ void norm\_of\_matrix():** Here the actual normalization of matrix is done. Here the output matrix is formatted which will then be served as an input for host device. The normalization of matrix is also done using the formula.
4. **Void initialize\_matrix():**Here the matrix is initialized with the rand() function.
5. **Void print\_matrix():** After initialization, the matrix is printed for the user to see what is the input matrix.
6. **cudaMemcpy():**Copies count bytes from the memory area pointed to by source to the memory area pointed to by destination, where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy.
7. **cudaMalloc():**Linear memory is typically allocated usingcudaMalloc(). It passes two parameters one is the pointer element that is the data element and other parameter is the size that needs to be allocated.
8. **cudaDeviceSynchronize():**A kernel launch is asynchronous. This means it returns control to the CPU thread immediately after starting up the GPU process, before the kernel has finished executing.
9. **cudaEventSynchronize():**Waits for an event to complete. Wait until the completion of all device work preceding the most recent call to [cudaEventRecord()](http://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EVENT.html#group__CUDART__EVENT_1gf4fcb74343aa689f4159791967868446) (in the appropriate compute streams, as specified by the arguments to [cudaEventRecord()](http://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EVENT.html#group__CUDART__EVENT_1gf4fcb74343aa689f4159791967868446)).
10. **cudaFree():**A synchronization call is called internally when a cudaFree() is called from Host. It frees the pointer allocated in cudaMalloc() function. So cudaFree() has only one parameter i.e. the pointer parameter. Also, malloc and free work inside a kernel (2.x), but memory allocated in a kernel must be deallocated in a kernel (not the host). It can be freed in a different kernel, though.
11. **free():** Free all the allocated memory in the host machine.

**Sequential Time in Cuda:**While running the matrix\_cuda\_norm.c on Jarvis it takes 78.65 ms.While in Parallel Programming using cuda has been shown in the following:

**Figure:Parallel Performance in Cuda**

|  |  |
| --- | --- |
| Number of Threads | CPU Time(In ms) |
| 1024 | 80.398 |
| 512 | 25.553 |
| 256 | 13.215 |
| 128 | 13.891 (here we can see that the performance time of the thread is increasing). |
| 64 | 20.678 |
| 32 | 38.031 |
| 16 | 56.155 |
| 8 | 93.676 |
| 4 | 168.859 |
| 2 | 315.716 |
| 1 | 597.229 |

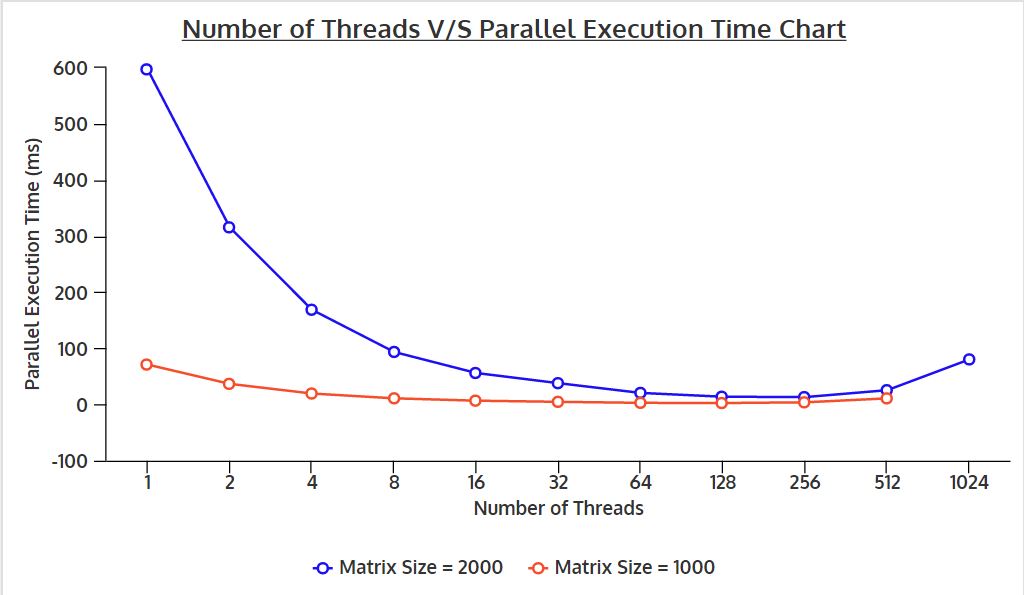
**For 2000 size matrix**

|  |  |
| --- | --- |
| Number of Threads | CPU Time(In ms) |
| 512 | 10.955 |
| 256 | 3.923 |
| 128 | 2.631 |
| 64 | 2.998 (here we can see that the performance time of the thread is increasing). |
| 32 | 4.794 |
| 16 | 6.842 |
| 8 | 11.032 |
| 4 | 19.59 |
| 2 | 36.804 |
| 1 | 71.173 |

**For 1000 Matrix size**

As we can see in the graph that upto thread 256 the performance of the thread timing is decreased as we increase thread.

**Comparison of Performance of threads in Parallel Cuda Code:**

****

**Conclusion**

* As per the graph and data find for different threads we can see that while we use above 256 thread it will increase parallel time execution increase.
* It is because the primary limitation here is the maximum of 256 threads per block, primarily imposed by the small number of registers that can be allocated across all the threads running in all the blocks assigned to an MP.
* The thread limit constrains the amount of cooperation between threads because only threads within the same block can synchronize with each other and exchange data through the fast shared memory in an MP.