EO 243:HIGH PERFORMANCE COMPUTER ARCHITECTURE

Programming Assignment part B

Submitted By:

Name: Chetan Pant

SR no : 18071

Device Specification:

To run our program ,We use the High end GPU server provided to us.

Cuda version used : CUDA 11.0

Implementation Details:

We have implemented the DMM program using CUDA programming .For this ,we have taken the number of threads in the program equal to the number of elements in the output array.So Each thread will calculate one element in the output array.So there is no problem of synchronization among threads and all the thread will work in parallel,Therefore we will get a significant speedup.

Steps that we follows:

1. Allocating Space in Device Memory : Using cudaMalloc() function we have allocated space for matrix A ,B and output vector in device memory.

2. Transfer of data from host memory to device memory: Using cudamemcpy() ,we transferred matrix A and B from host memory to device memory .

3.Implementation of Kernel code:DMM function with prefix __global__ contains the kernel code that will run by all threads in GPU.

4.Invoking GPU Kernel: using call DMM<<<blockpergrid,threadperblock>>() ,we launched the kernel with a suitable number of threads in a block and number of blocks.

5.Transfer output vector from device memory to Host memory Using cudaMemcpy() function ,we transfer back the result of computation to the host.

Optimization :

To find the optimum value of the number of threads per thread block, we have implemented the program using different values of thread per block and also on different values of N.

We got the following data

Number of thread per block	Execution time with N=128	Execution time with N=4096	Execution time with N=16384
64	328.08	426.97	800.49
128	357.87	477.30	851.91
256	359.02	510.2	1143.91

From above data we can clearly see that if we use a number of thread per thread block equal to 64 ,we are getting best results.This is due to the fact that one thread block is assigned to a SM and a SM can only have some number of active warps . If we increase the number of thread per block ,then the parallelization between threads will reduce.

And also decreasing threads per block will increase the number of blocks ,so the GPU can run as many blocks at a time as it is capable of .

At last we finally choose the value of number of thread per thread block =64

Further Optimizations possible:

1.We can use the shared memory to speedup the program because access to shared memory is faster than access to global memory.

2.We can also modify the program such that the adjacent threads will use consecutive elements in the memory.

3. We can modify the program such that we can remove branch from the Kernel Code .This will speed up the program because we know that branches reduced the SIMD core utilization .(Some threads are active in one part of the branch and some are active in other part .Therefore SIMD utilization will be lower.)

Results and Analysis:

We have executed the program using threads per block equal to 64.

And we got the following results.

Size of Matrix(N)	Reference execution time	Execution time using GPU (Thread Per Block=64)	Speedup
N=128	0.26	328.08	<1
N=4096	550.38	426.97	1.3
N=16384	8076.49	800.49	10.1

Analysis : In above data , when the matrix size (N) is small, the speed up we get due to GPU is very low (or even less than 1 in some cases) .The reason for this is due to the overhead of transferring memory from host to device and vice versa and also in launching the kernel is greater than the execution time of reference program. If the size of the matrix is large ,then we can get a significant speed up in the program because the overhead will be very less as compared to the work we want to parallelise.

References:

[1].”GPU Architecture and programming”, referenced from https://nptel.ac.in/courses/106/105/106105220/

[2].”CUDA programming”, referenced from https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html