MULTI-CORE PROGRAMMING

PROJECT – 4

GPU PROGRAMMING – MATRIX MULTIPLICATION

Team Members:

Praveen Erode Murugesan(pem150030)

Rahul Aravind Mehalingam(rxm151730)

Problem Statement:

The aim of the project is to implement the matrix multiplication using cuda programming and compare the performance measure of CPU execution with GPU execution.

Experimental Setup:

Matrix multiplication is a good example for parallel computation as we can efficiently parallelize the multiplication operations across the threads and synchronization is achieved for the summation and the results are stored in 2D kernel, which uses shared memory. Two flavors of design strategy followed here. The first implementation is the naïve implementation. GPUs assigns one thread to compute one element of Matrix C. Each thread loads one row of Matrix A and one column of Matrix B from global memory, do the inner product and store the result back to matrix C in the global memory. The second implementation is the efficient one where parallelism is leverage across multiple blocks of multiple threads per block. One thread in the thread block computes one element of the tile. The Number of blocks are dynamically selected with respect to the input and only one thread per block that corresponds to the architecture is defined.

Correctness strategy:

The A matrix and B matrix are filled with numbers in a random fashion depending on their dimensions specified in the command line. The correctness of the matrix multiplication routine is checked against the naïve cpu implementation. Each element of the resultant matrix (GPU) is compared with the result matrix (CPU). The experiment is carried out against the two implementations for different dimensions of the matrix and the performance is compared with respect to the two implementations for different size dimensions.

Note - Sample output included at the last

Results:

Rows	Columns	GPU - One Block Implementation (seconds)	GPU - Multiple Blocks Implementation (seconds)	CPU - Naïve Implementation (seconds)
16	**16**	**0**	**0**	**0**
64	**64**	**0**	**0**	**0**
256	**256**	**0**	**0**	**0**
512	**512**	**0**	**0**	**1**
1024	**1024**	**0**	**0**	**10**
2048	**2048**	**0**	**0**	**123**

Testing in TACC Machine:

The code is executed in an interactive session in stampede tacc machine. The instructions on how to execute the program is provided in the readme file.

Execution Results:

c557-103.stampede(40)$ ./a.out 512 512 512 512

Time taken for GPU Multiplication flavor 1: 0 seconds

Time taken for GPU Multiplication flavor 2: 0 seconds

Time taken for CPU Multiplication : 1 seconds

GPU multiplication flavor 1 is performed without any error

GPU multiplication flavor 2 is performed without any error

Inference:

It is seen that the second implementation (multiple blocks) performs better for invariably large inputs. This programming exercise helped us understand the core concepts of GPU architecture and thread organization, and also aided the understanding to apply in a practical purpose.