**Assignment 3**

**Question 1:**

This program is designed to find the minimum distance between two points among a set of points scattered in a 2D space. Using CUDA and a GPU we distribute calculating the minimum distance in a very large number of threads. Without using a GPU kernel if the program is executed on 1 process on a CPU it will take a very long time as it will exhaustively check all the combination of particle pairs and the time complexity will be O(n2). As GPU kernels parallelise this computation using a very large number of threads, the time complexity is effectively reduced.

**The efficiency of approaches.**

The serial part running on CPU took 30 seconds for 98304 particles. Whereas for the same number of particles GPU kernel with one thread per particle took 183.46662 ms and GPU kernel with one thread per pair of particles took 351.84 ms.

Even for N = 1048576 GPU Kernel 2 took 26 seconds while GPU Kernel 1 took only 3.8 seconds. Beyond this number the operation to calculate number of blocks will cause integer overflow and the program will not run as desired.

Execution times for CPU and GPU with different number of threads.

|  |  |  |  |
| --- | --- | --- | --- |
| **No. of particles** | **CPU (ms)** | **GPU k1 (ms)** | **GPU k2 (ms)** |
| 16384 | 29350 | 183.467 | 351.843 |
| 32768 | 13010 | 92.124 | 329.875 |
| 65536 | 3210 | 13.475 | 103.584 |
| 98304 | 820 | 15.136 | 34.444 |

The following graph visualizes the relationship between number of particles and time taken by CPU and GPU with different number of threads.

A graph with a line going up

Description automatically generated

**Screenshots:A black rectangle with white text

Description automatically generatedA black screen with blue and white text

Description automatically generatedA black rectangle with white text

Description automatically generatedA black rectangle with white text

Description automatically generated**

**Question 2:**

**In this question we are dividing a large matrix and distributing it over different processes. Given that matrix size is divisible by the number of processes.**

Let the matrix size = N

Let the no. of process = P

To send the matrix block by block to all processes and to keep memory usage efficient I am only reading the part of the matrix which I am sending to all processes. Once the part of the matrix is sent, I read another part of the matrix.

The lines of the matrix to read at a time = N/P = step\_size

So, the size of the first Row\_block to read = step\_size \* N

Size of the block sent to other processes = step\_size \* step\_size = block\_size

Now the number of steps of size ‘steps\_size’ will be equal to P because it’s a square matrix. So, perform the above logic in a loop that runs P times. Inside this loop create a buffer to read the matrix step by step with ‘step\_size’ lines a time. Divide that buffer into blocks of size ‘block\_size’ and send those to corresponding procceses. Now invert the above process for write function as its just a reverses process of reading and distributing the matrix. So, the logic is almost the same.

Time taken for different processes for matrix size 1000.

|  |  |
| --- | --- |
| **No. of processes** | **Execution Time (seconds)** |
| **200** | **1.1** |
| **100** | **0.44** |
| **50** | **0.68** |
| **25** | **0.39** |

Screenshots:

A black background with white text

Description automatically generated

A screenshot of a computer

Description automatically generated