HPC Tutorial 11 Report
CS22B2012 K Aditya Sai

This report analyses the performance of CUDA parallel code on the addition of two matrices of size 5000. The values range from 88.0 to 888.0.

Serial Code:

```
auto start = std::chrono::high_resolution_clock::now();

// Perform matrix addition
for (int i = 0; i < SIZE; ++i) {
    for (int j = 0; j < SIZE
        C[i][j] = A[i][j] + B[i][j];
    }
}

// Stop measuring time
auto end = std::chrono::high_resolution_clock::now();
```

double **B

Parallel Code :

```
__global__ void matrixAdd(const double* A, const double* B, double* C, int width) {
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int row = blockIdx.y * blockDim.y + threadIdx.y;

    if (row < width && col < width) {
        int index = row * width + col;
        C[index] = A[index] + B[index];
    }
}
```

This parallel Code uses the inbuilt dim construct.
It uses the dim construct to make a 2d block of threads (16, 16).

Memory Management :

```
cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

dim3 threadsPerBlock(16, 16);
dim3 blocksPerGrid((N + threadsPerBlock.x - 1) / threadsPerBlock.x,
                   (N + threadsPerBlock.y - 1) / threadsPerBlock.y);

cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);

cudaEventRecord(start);
matrixAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);
cudaEventRecord(stop);

cudaEventSynchronize(stop);

float milliseconds = 0;
cudaEventElapsedTime(&milliseconds, start, stop);

std::cout << "Execution time: " << milliseconds / 1000.0 << " seconds" << std::endl;

cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

cudaFree(d_A);
cudaFree(d_B);
cudaFree(d_C);
free(h_A);
free(h_B);
free(h_C);
```

Number of Threads per block = (16, 16)
Number of Blocks = (N + threads_per_block_x - 1) / (threads_per_block_x), (N + threads_per_block_y - 1) / (threads_per_block_y)

Outputs :

Serial :

```
PS C:\Users\adity\OneDrive\Documents\HPC\Tut11> g++ add.cpp -o add
PS C:\Users\adity\OneDrive\Documents\HPC\Tut11> .\add
Reading matrices from files...
Matrices successfully read.
Matrix addition completed.
Time taken: 0.140521 seconds
```

Parallel :

```
C:\Users\adity\OneDrive\Documents\HPC\Tut11>nvcc parallel_add.cu -o cuda_add
parallel_add.cu
tmpxft_00005c34_00000000-10_parallel_add.cudafe1.cpp
   Creating library cuda_add.lib and object cuda_add.exp

C:\Users\adity\OneDrive\Documents\HPC\Tut11>.\cuda_add
Execution time: 0.00233155 seconds
```

Speedup:

S= T(1) / T(p) = 0.14 / 0.002 = 70 (approximately)

Observations :

- If the memory loading time is also considered for the calculation of execution time, then, loading the memory to the GPU is comparatively taking longer because the host has to load the device.
- The **Device** can allocate larger data for example say 10000 x 10000 matrices and perform the addition while the **Host** prevented me from allocating a matrix of even 6000 x 6000
- If the same code was run in a Linux based OS, the speedup is much greater on the exact same hardware.