HPC Tutorial 9 Report
CS22B2012 K Aditya Sai

This report analyses the performance of CUDA parallel code on two 8 million double precision floating-point vectors while performing vector addition and vector multiplication. The values range from 88.0 to 8888.0.

Serial Code (Addition) :

```
void vector_add_serial(double *a, double *b, double *c){
    for (int i = 0; i < N; i++){
        c[i] = a[i] + b[i];
    }
}
```

Serial Code (Multiplication) :

```
void vector_mul_serial(double *a, double *b, double *c){
    for (int i = 0; i < N; i++){
        c[i] = a[i] * b[i];
    }
}
```

Parallel Code(Addition):

```
__global__ void vector_add(double *a, double *b, double *c){
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    if (index < N){
        c[index] = a[index] + b[index];
    }
}
```

Parallel Code (Multiplication):

```
__global__ void vector_mul(double *a, double *b, double *c){
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    if (index < N){
        c[index] = a[index] * b[index];
    }
}
```

Memory Management for Addition

```
double *d_a, *d_b, *d_c;
cudaMalloc((void **)&d_a, N * sizeof(double));
cudaMalloc((void **)&d_b, N * sizeof(double));
cudaMalloc((void **)&d_c, N * sizeof(double));

cudaMemcpy(d_a, a, N * sizeof(double), cudaMemcpyHostToDevice);
cudaMemcpy(d_b, b, N * sizeof(double), cudaMemcpyHostToDevice);

cudaDeviceSynchronize();
clock_t start = clock();
int blocks = (N + THREADS_PER_BLOCK - 1) / THREADS_PER_BLOCK;
vector_add<<<blocks, THREADS_PER_BLOCK>>>(d_a, d_b, d_c);
cudaDeviceSynchronize();
clock_t end = clock();

cudaMemcpy(c_add, d_c, N * sizeof(double), cudaMemcpyDeviceToHost);

double time_taken = (double)(end - start) / CLOCKS_PER_SEC;
printf("Time taken by parallel addition code: %f\n", time_taken);
```

Memory Management for Multiplication :

```
double *d_a, *d_b, *d_c;
cudaMalloc((void **)&d_a, N * sizeof(double));
cudaMalloc((void **)&d_b, N * sizeof(double));
cudaMalloc((void **)&d_c, N * sizeof(double));

cudaMemcpy(d_a, a, N * sizeof(double), cudaMemcpyHostToDevice);
cudaMemcpy(d_b, b, N * sizeof(double), cudaMemcpyHostToDevice);

cudaDeviceSynchronize();
clock_t start = clock();
int blocks = (N + THREADS_PER_BLOCK - 1) / THREADS_PER_BLOCK;
vector_mul<<<blocks, THREADS_PER_BLOCK>>>(d_a, d_b, d_c);
cudaDeviceSynchronize();
clock_t end = clock();

cudaMemcpy(c_mul, d_c, N * sizeof(double), cudaMemcpyDeviceToHost);

double time_taken = (double)(end - start) / CLOCKS_PER_SEC;
printf("Time taken by parallel multiplication code: %f\n", time_taken);
```

Number of Threads per block = 1024
Number of Blocks = (N + threads_per_block - 1) / (threads_per_block)

Output :

Serial Add:

```
PS C:\Users\adity\OneDrive\Documents\HPC\Tut9> gcc .\serial_add.c -o s_add
PS C:\Users\adity\OneDrive\Documents\HPC\Tut9> .\s_add
Time taken for serial addition: 0.034000
First 10 elements of the result:
12957.986673
16285.972676
10636.105194
9246.305189
17338.627019
11819.466154
9480.283475
14053.477183
1480.172095
5981.360967
```

Parallel Addition :

```
C:\Users\adity\OneDrive\Documents\HPC\Tut9>nvcc -arch=sm_86 parallel_add.cu -o cuda_add
parallel_add.cu
tmpxft_00004fc8_00000000-10_parallel_add.cudafe1.cpp
   Creating library cuda_add.lib and object cuda_add.exp

C:\Users\adity\OneDrive\Documents\HPC\Tut9>.\cuda_add
Time taken by parallel addition code: 0.000000
First 10 elements of the result:
12957.986673
16285.972676
10636.105194
9246.305189
17338.627019
11819.466154
9480.283475
14053.477183
1480.172095
5981.360967
```

Addition Speedup :

S = T(1) / T(p) = 0.034000 / 0.000000 = **inf**

Serial Multiplication :

```
PS C:\Users\adity\OneDrive\Documents\HPC\Tut9> gcc .\serial_mul.c -o s_mul
PS C:\Users\adity\OneDrive\Documents\HPC\Tut9> .\s_mul
Time taken for serial multiplication: 0.034000
c[0] = 41977354.655966
c[1] = 66308226.502483
c[2] = 28281683.423395
c[3] = 21373539.913604
c[4] = 75156996.727733
c[5] = 34924945.039615
c[6] = 22468943.691305
c[7] = 49375055.236508
c[8] = 547727.357867
c[9] = 8944169.754507
```

Parallel Multiplication :

```
C:\Users\adity\OneDrive\Documents\HPC\Tut9>.\cuda_mul
Time taken by parallel multiplication code: 0.000000
First 10 values of the result:
41977354.655966
66308226.502483
28281683.423395
21373539.913604
75156996.727733
34924945.039615
22468943.691305
49375055.236508
547727.357867
8944169.754507
```

Multiplication Speedup :

S = T(1) / T(p) = 0.034 / 0.00000

NOTE : These execution times are in seconds.

Observations :

- The speedup is calculated as infinity by using the same time data-type(**clock_t**) which was used for the serial code.
- Upon further increase of precision it is observed that the execution time shown as 0 is not actually 0 but a very small value which was rounded off.
- In both cases we get a final speedup of around 50.