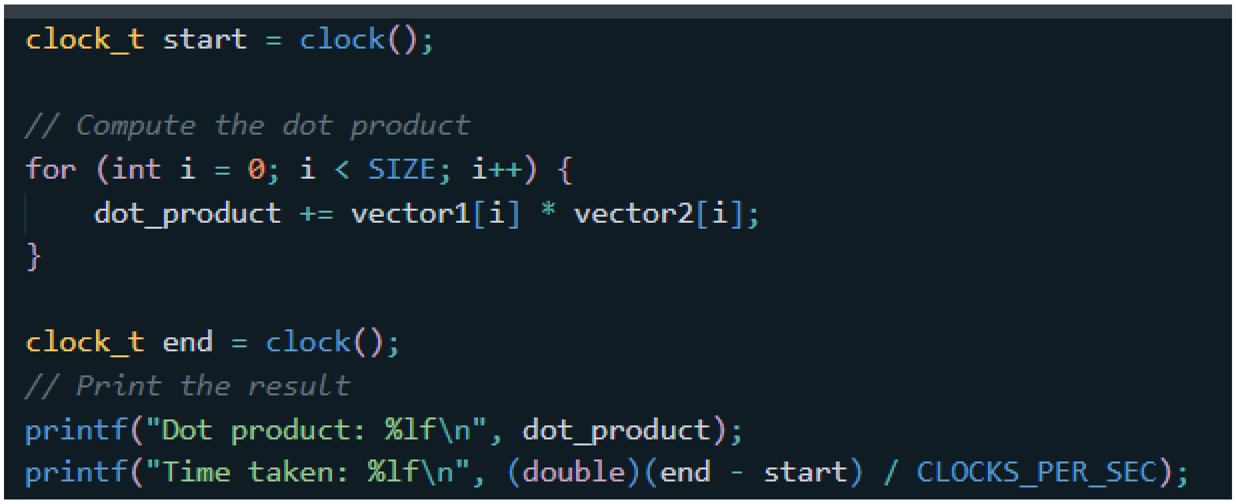
HPC Tutorial 10 Report
CS22B2012 K Aditya Sai

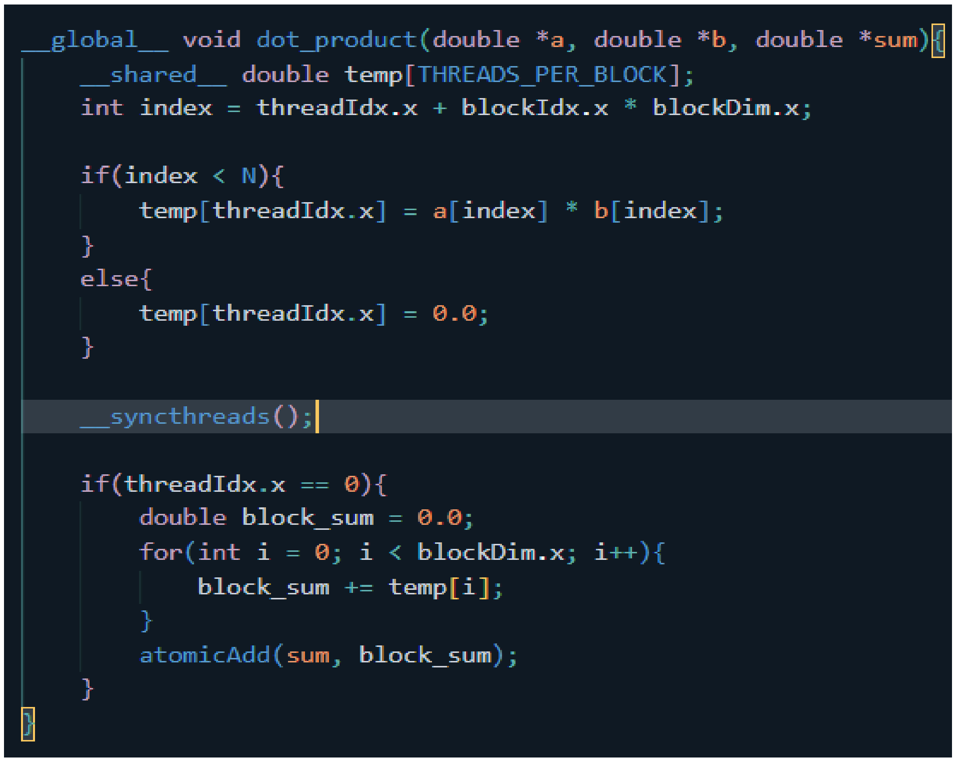
This report analyses the performance of CUDA parallel code on two 10 million double precision floating-point vectors while performing dot product. The values range from 8888.0 to 10000.0.

Serial Code:



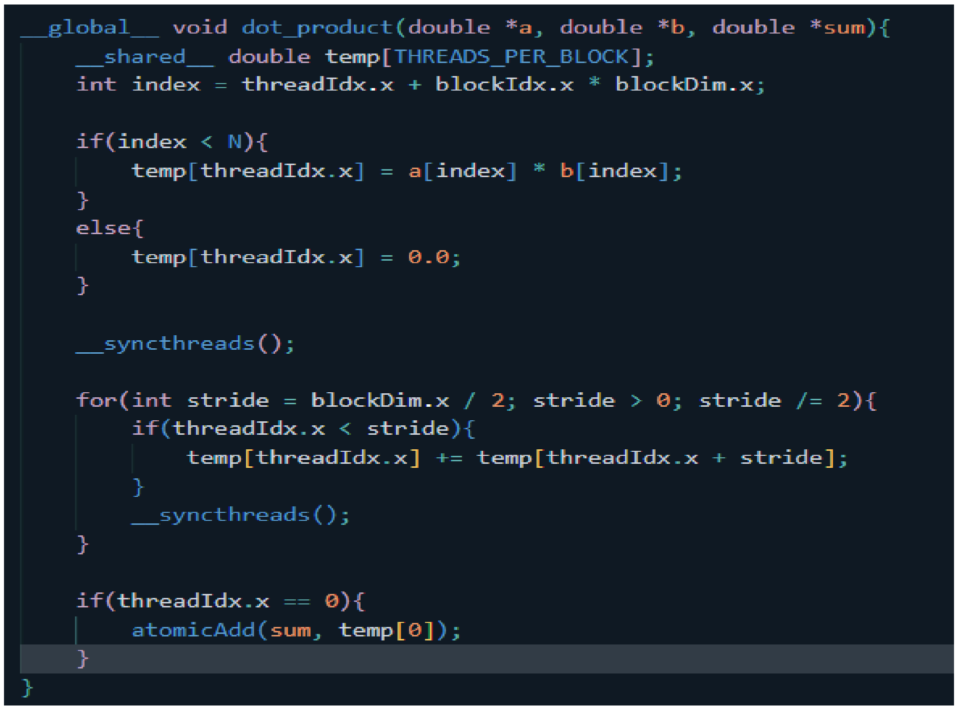
```c
clock_t start = clock();

// Compute the dot product
for (int i = 0; i < SIZE; i++) {
    dot_product += vector1[i] * vector2[i];
}

clock_t end = clock();
// Print the result
printf("Dot product: %lf\n", dot_product);
printf("Time taken: %lf\n", (double)(end - start) / CLOCKS_PER_SEC);
```

Parallel Code (Atomic):



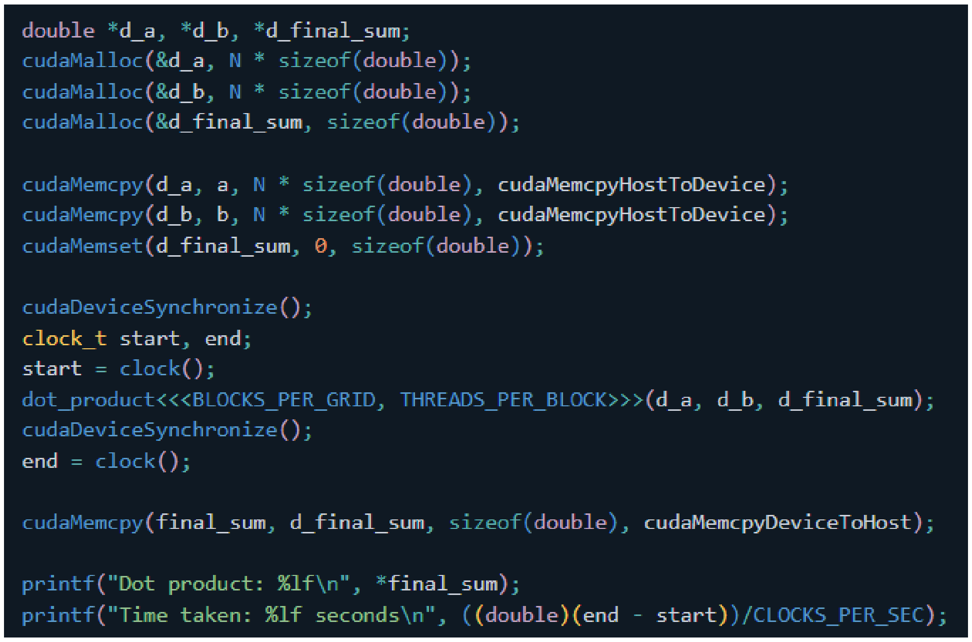
```c
__global__ void dot_product(double *a, double *b, double *sum){
    __shared__ double temp[THREADS_PER_BLOCK];
    int index = threadIdx.x + blockIdx.x * blockDim.x;

    if(index < N){
        temp[threadIdx.x] = a[index] * b[index];
    }
    else{
        temp[threadIdx.x] = 0.0;
    }

    __syncthreads();

    if(threadIdx.x == 0){
        double block_sum = 0.0;
        for(int i = 0; i < blockDim.x; i++){
            block_sum += temp[i];
        }
        atomicAdd(sum, block_sum);
    }
}
```

Parallel Code (Reduction) :



```
__global__ void dot_product(double *a, double *b, double *sum){
    __shared__ double temp[THREADS_PER_BLOCK];
    int index = threadIdx.x + blockIdx.x * blockDim.x;

    if(index < N){
        temp[threadIdx.x] = a[index] * b[index];
    }
    else{
        temp[threadIdx.x] = 0.0;
    }

    __syncthreads();

    for(int stride = blockDim.x / 2; stride > 0; stride /= 2){
        if(threadIdx.x < stride){
            temp[threadIdx.x] += temp[threadIdx.x + stride];
        }
        __syncthreads();
    }

    if(threadIdx.x == 0){
        atomicAdd(sum, temp[0]);
    }
}
```

Memory Management :

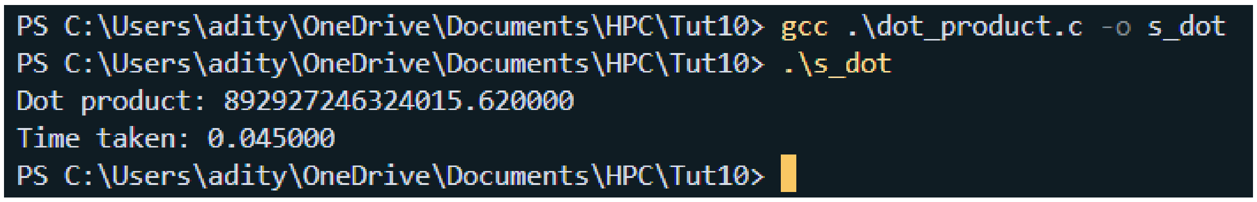


```
double *d_a, *d_b, *d_final_sum;
cudaMalloc(&d_a, N * sizeof(double));
cudaMalloc(&d_b, N * sizeof(double));
cudaMalloc(&d_final_sum, sizeof(double));

cudaMemcpy(d_a, a, N * sizeof(double), cudaMemcpyHostToDevice);
cudaMemcpy(d_b, b, N * sizeof(double), cudaMemcpyHostToDevice);
cudaMemset(d_final_sum, 0, sizeof(double));

cudaDeviceSynchronize();
clock_t start, end;
start = clock();
dot_product<<<BLOCKS_PER_GRID, THREADS_PER_BLOCK>>>(d_a, d_b, d_final_sum);
cudaDeviceSynchronize();
end = clock();

cudaMemcpy(final_sum, d_final_sum, sizeof(double), cudaMemcpyDeviceToHost);

printf("Dot product: %lf\n", *final_sum);
printf("Time taken: %lf seconds\n", ((double)(end - start))/CLOCKS_PER_SEC);
```

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

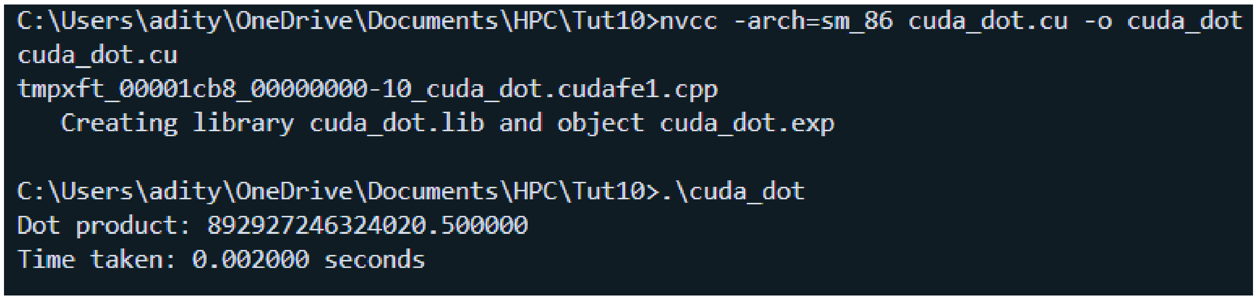
Outputs :

Serial :



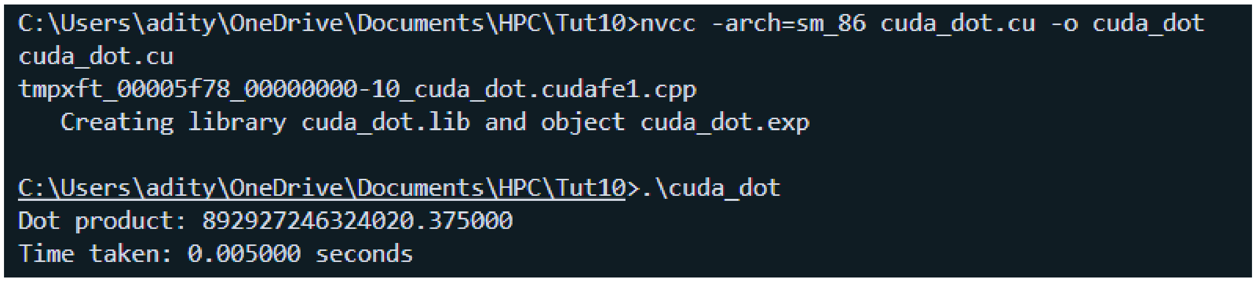
```
PS C:\Users\adity\OneDrive\Documents\HPC\Tut10> gcc .\dot_product.c -o s_dot
PS C:\Users\adity\OneDrive\Documents\HPC\Tut10> .\s_dot
Dot product: 892927246324015.620000
Time taken: 0.045000
PS C:\Users\adity\OneDrive\Documents\HPC\Tut10>
```

Parallel (Reduction) :



```
C:\Users\adity\OneDrive\Documents\HPC\Tut10>nvcc -arch=sm_86 cuda_dot.cu -o cuda_dot
cuda_dot.cu
tmpxft_00001cb8_00000000-10_cuda_dot.cudafe1.cpp
   Creating library cuda_dot.lib and object cuda_dot.exp

C:\Users\adity\OneDrive\Documents\HPC\Tut10>.\cuda_dot
Dot product: 892927246324020.500000
Time taken: 0.002000 seconds
```

Parallel (Atomic) :



```
C:\Users\adity\OneDrive\Documents\HPC\Tut10>nvcc -arch=sm_86 cuda_dot.cu -o cuda_dot
cuda_dot.cu
tmpxft_00005f78_00000000-10_cuda_dot.cudafe1.cpp
   Creating library cuda_dot.lib and object cuda_dot.exp

C:\Users\adity\OneDrive\Documents\HPC\Tut10>.\cuda_dot
Dot product: 892927246324020.375000
Time taken: 0.005000 seconds
```

Speedup (Reduction):

S= T(1) / T(p) = 0.045 / 0.002 = 22.5

Speedup (Atomic) :

S = T(1) / T(p) = 0.045 / 0.005 = 9

Observations :

- There is a small discrepancy between the dot product of the serial code and parallel code. This is happening due to the rounding off of the double precision values in the **Device.**
- The precision of the values in the **Host** is around 10 decimal places
- Due to the immense size of the vectors, there is a greater effect of the rounding of the decimal values.
- As we reduce the size of the vectors, the accuracy of the estimation increases and the discrepancy of the result also diminishes.
- For example, when vectors of size 2 million were considered, the error was prevalent in the last 3 decimal places.
- There is also a 2.5 times increase in the speedup of the program by using Reduction based sum instead of atomic.

Time Complexity :

- Reduction - O(log THREADS_PER_BLOCK) + O(n / THREADS_PER_BLOCK)
- Atomic - O(THREADS_PER_BLOCK) + O(n / THREADS_PER_BLOCK)