

```

1 // MATRIX_MULTIPLICATION.CU
2
3 #include <cmath>
4 #include <cstdlib>
5 #include <iostream>
6 using namespace std;
7
8 // Matrix multiplication Cuda
9 __global__ void matrixMultiplication(int *a, int *b, int *c, int n) {
10     int row = threadIdx.y + blockDim.y * blockIdx.y;
11     int col = threadIdx.x + blockDim.x * blockIdx.x;
12     int sum = 0;
13
14     if (row < n && col < n)
15         for (int j = 0; j < n; j++) {
16             sum = sum + a[row * n + j] * b[j * n + col];
17         }
18
19     c[n * row + col] = sum;
20 }
21 int main() {
22     int *a, *b, *c;
23     int *a_dev, *b_dev, *c_dev;
24     int n = 3;
25
26     a = new int[n * n];
27     b = new int[n * n];
28     c = new int[n * n];
29     int *d = new int[n * n];
30     int size = n * n * sizeof(int);
31     cudaMalloc(&a_dev, size);
32     cudaMalloc(&b_dev, size);
33     cudaMalloc(&c_dev, size);
34
35     // Array initialization
36     for (int i = 0; i < n * n; i++) {
37         a[i] = 2; // rand()%n;
38         b[i] = 1; // rand()%n;
39         // d[i]=a[i]+b[i];
40     }
41
42     cudaEvent_t start, end;
43
44     cudaEventCreate(&start);
45     cudaEventCreate(&end);
46
47     cudaMemcpy(a_dev, a, size, cudaMemcpyHostToDevice);
48     cudaMemcpy(b_dev, b, size, cudaMemcpyHostToDevice);
49
50     dim3 threadsPerBlock(n, n);
51     dim3 blocksPerGrid(1, 1);
52
53     if (n * n > 512) {
54         threadsPerBlock.x = 512;
55         threadsPerBlock.y = 512;
56         blocksPerGrid.x = ceil((double)n / (double)threadsPerBlock.x);
57         blocksPerGrid.y = ceil((double)n / (double)threadsPerBlock.y);
58     }
59     // GPU Multiplication
60     cudaEventRecord(start);
61     matrixMultiplication<<<blocksPerGrid, threadsPerBlock>>>(a_dev, b_dev, c_dev, n);
62
63     cudaEventRecord(end);
64     cudaEventSynchronize(end);
65
66     float time = 0.0;
67     cudaEventElapsedTime(&time, start, end);
68
69     cudaMemcpy(c, c_dev, size, cudaMemcpyDeviceToHost);
70
71     // CPU matrix multiplication
72     int sum = 0;
73     for (int row = 0; row < n; row++) {
74         for (int col = 0; col < n; col++) {

```

```
75         sum = 0;
76         for (int k = 0; k < n; k++) sum = sum + a[row * n + k] * b[k * n + col];
77         d[row * n + col] = sum;
78     }
79 }
80 int error = 0;
81 for (int i = 0; i < n * n; i++) {
82     error += d[i] - c[i];
83     // cout<<" gpu "<<c[i]<<" CPU "<<d[i]<<endl;
84 }
85
86 cout << "Error : " << error;
87 cout << "\nTime Elapsed: " << time;
88
89 return 0;
90 }
```

```

1 // MATRIX_VECTOR_MULTIPLICATION.CU
2
3 #include <time.h>
4
5 #include <cmath>
6 #include <cstdlib>
7 #include <iostream>
8 using namespace std;
9
10 __global__ void matrixVectorMultiplication(int *a, int *b, int *c, int n) {
11     int row = threadIdx.x + blockDim.x * blockIdx.x;
12     int sum = 0;
13
14     if (row < n)
15         for (int j = 0; j < n; j++) {
16             sum = sum + a[row * n + j] * b[j];
17         }
18
19     c[row] = sum;
20 }
21
22 int main() {
23     int *a, *b, *c;
24     int *a_dev, *b_dev, *c_dev;
25     int n = 32;
26
27     a = new int[n * n];
28     b = new int[n];
29     c = new int[n];
30     int *d = new int[n];
31     int size = n * sizeof(int);
32     cudaMalloc(&a_dev, size * size);
33     cudaMalloc(&b_dev, size);
34     cudaMalloc(&c_dev, size);
35
36     for (int i = 0; i < n; i++) {
37         for (int j = 0; j < n; j++) {
38             a[i * n + j] = i * n + j + 1; // rand()%n;
39         }
40
41         b[i] = i + 1; // rand()%n;
42         // cout<<a[i]<<" ";
43         // d[i]=a[i]+b[i];
44     }
45
46     cudaEvent_t start, end;
47
48     cudaEventCreate(&start);
49     cudaEventCreate(&end);
50
51     cudaMemcpy(a_dev, a, size * size, cudaMemcpyHostToDevice);
52     cudaMemcpy(b_dev, b, size, cudaMemcpyHostToDevice);
53
54     dim3 threadsPerBlock(n, n);
55     dim3 blocksPerGrid(1, 1);
56
57     if (n * n > 512) {
58         threadsPerBlock.x = 512;
59         threadsPerBlock.y = 512;
60         blocksPerGrid.x = ceil((double)n / (double)threadsPerBlock.x);
61         blocksPerGrid.y = ceil((double)n / (double)threadsPerBlock.y);
62     }
63
64     cudaEventRecord(start);
65     matrixVectorMultiplication<<<blocksPerGrid, threadsPerBlock>>>>(a_dev, b_dev, c_dev, n);
66
67     cudaEventRecord(end);
68     cudaEventSynchronize(end);
69
70     float time = 0.0;
71     cudaEventElapsedTime(&time, start, end);
72
73     cudaMemcpy(c, c_dev, size, cudaMemcpyDeviceToHost);
74     cout << "\nGPU Time Elapsed: " << time;

```

```

75 // CPU matrixVector multiplication
76 clock_t t = clock();
77 int sum = 0;
78 for (int row = 0; row < n; row++) {
79     sum = 0;
80     for (int col = 0; col < n; col++) {
81         sum = sum + a[row * n + col] * b[col];
82     }
83     d[row] = sum;
84 }
85 t = clock() - t;
86 cout << "\nCPU Time Elapsed: " << ((double)t); //((double)t)/CLOCKS_PER_SEC;
87
88 int error = 0;
89 for (int i = 0; i < n; i++) {
90     error += d[i] - c[i];
91     // cout<<" gpu "<<c[i]<<" CPU "<<d[i]<<endl;
92 }
93
94 cout << "Error : " << error;
95
96 return 0;
97 }

```

```

1 // VECTOR_ADDITION.CU
2
3 #include <cstdlib>
4 #include <iostream>
5
6 using namespace std;
7
8 // VectorAdd parallel function
9 __global__ void vectorAdd(int *a, int *b, int *result, int n) {
10     int tid = threadIdx.x + blockIdx.x * blockDim.x;
11     if (tid < n) {
12         result[tid] = a[tid] + b[tid];
13     }
14 }
15 int main() {
16     int *a, *b, *c;
17     int *a_dev, *b_dev, *c_dev;
18     int n = 1 << 24;
19
20     a = new int[n];
21     b = new int[n];
22     c = new int[n];
23     int *d = new int[n];
24     int size = n * sizeof(int);
25     cudaMalloc(&a_dev, size);
26     cudaMalloc(&b_dev, size);
27     cudaMalloc(&c_dev, size);
28
29     // Array initialization..You can use Randon function to assign values
30     for (int i = 0; i < n; i++) {
31         a[i] = 1;
32         b[i] = 2;
33         d[i] = a[i] + b[i]; // calculating serial addition
34     }
35
36     cudaEvent_t start, end;
37
38     cudaEventCreate(&start);
39     cudaEventCreate(&end);
40
41     cudaMemcpy(a_dev, a, size, cudaMemcpyHostToDevice);
42     cudaMemcpy(b_dev, b, size, cudaMemcpyHostToDevice);
43     int threads = 1024;
44     int blocks = (n + threads - 1) / threads;
45     cudaEventRecord(start);
46
47     // Parallel addition program
48     vectorAdd<<<blocks, threads>>>(a_dev, b_dev, c_dev, n);
49
50     cudaEventRecord(end);
51     cudaEventSynchronize(end);
52
53     float time = 0.0;
54     cudaEventElapsedTime(&time, start, end);
55
56     cudaMemcpy(c, c_dev, size, cudaMemcpyDeviceToHost);
57
58     // Calculate the error term.
59     int error = 0;
60     for (int i = 0; i < n; i++) {
61         error += d[i] - c[i];
62         // cout<<" gpu "<<c[i]<<" CPU "<<d[i];
63     }
64
65     cout << "Error : " << error;
66     cout << "\nTime Elapsed: " << time;
67
68     return 0;
69 }

```