Processing Time in the GPU and CPU

If we were to test the processing time of the same operation over and over, say z = 2x + y, performed both by the CPU and GPU, there would be a noticeable difference if the size of the vectors to store this repeated operations is enormous (102,400 cells). Consider the image below:

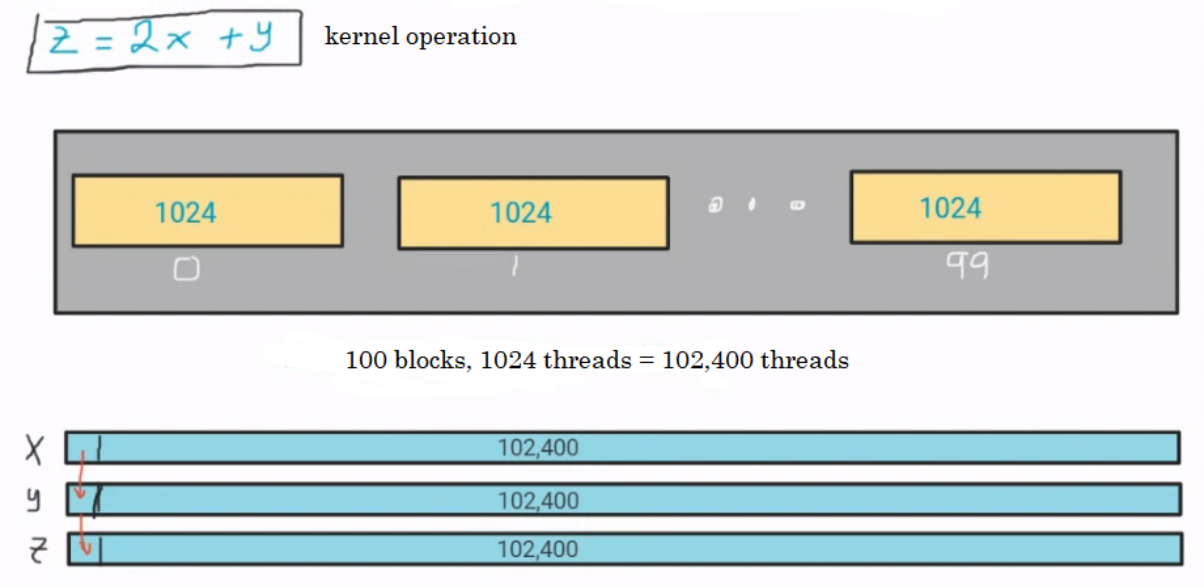

Figure 1: img

Implementation

```cpp
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <stdio.h>
#include <iostream>
#include <time.h>

using namespace std;

__global__ void GPU_fn(int* x, int* y, int* z) {
    int gId = blockIdx.x * blockDim.x + threadIdx.x;
    z[gId] = 2 * x[gId] + y[gId];
}


__host__ void CPU_fn(int* x, int* y, int* z, int vecSize) {
    for (int i = 0; i < vecSize; i++) {
        z[i] = 2 * x[i] + y[i];
    }
}
```

```
21
22 __host__ void checkCUDAError(const char* msg) {
23     cudaError_t error;
24     cudaDeviceSynchronize();
25     error = cudaGetLastError();
26     if (error != cudaSuccess) {
27         printf("ERROR %d: %s (%s)\n", error,
              cudaGetErrorString(error), msg);
28     }
29 }
30
31 __host__ void validate(int* result_CPU, int* result_GPU, int N
      ) {
32     for (int i = 0; i < N; i++) {
33         if (result_CPU[i] != result_GPU[i]) {
34             printf("The vectors are not equal\n");
35             return;
36         }
37     }
38     printf("Kernel validated successfully\n");
39     return;
40 }
41
42 int main()
43 {
44     cudaDeviceProp prop;
45     cudaGetDeviceProperties(&prop, 0);
46     printf("maxThreadsPerBlock: %d\n", prop.maxThreadsPerBlock
          ); // 1024 in all its block dimension
47     printf("maxThreadsDim[0]: %d\n", prop.maxThreadsDim[0]);
          // 1024 in a block's x dim
48     // 100 blocks x 1024 threads = 102 400 threads
49     // x,y,z vectors of size 1024
50
51     int numBlocks = 100000; // add one zero and you get ERROR
          2: run out of global memory
52     int numThreadsPerBlock = 1024;
53     int vecSize = numBlocks * numThreadsPerBlock;
54
55
56     int* hostx = (int*)malloc(vecSize * sizeof(int));
57     int* hosty = (int*)malloc(vecSize * sizeof(int));
58     int* hostzCPU = (int*)malloc(vecSize * sizeof(int));
59     int* hostzGPU = (int*)malloc(vecSize * sizeof(int));
60
61     int* devx, * devy, * devz;
62     cudaMalloc((void**)&devx, vecSize * sizeof(int));
63     checkCUDAError("Error at cudaMalloc: devx");
64     cudaMalloc((void**)&devy, vecSize * sizeof(int));
65     checkCUDAError("Error at cudaMalloc: devy");
66     cudaMalloc((void**)&devz, vecSize * sizeof(int));
```

```
67      checkCUDAError("Error at cudaMalloc: devz");
68
69      for (int i = 0; i < vecSize; i++) {
70          hostx[i] = 1;
71          hosty[i] = 2;
72      }
73
74      cudaMemcpy(devx, hostx, vecSize * sizeof(int),
           cudaMemcpyHostToDevice);
75      cudaMemcpy(devy, hosty, vecSize * sizeof(int),
           cudaMemcpyHostToDevice);
76      dim3 block(numThreadsPerBlock);
77      dim3 grid(numBlocks);
78
79      cudaEvent_t startGPU;
80      cudaEvent_t endGPU;
81      cudaEventCreate(&startGPU);
82      cudaEventCreate(&endGPU); // be able to mark the time
83      cudaEventRecord(startGPU); // save current time
84      GPU_fn << <grid, block >> > (devx, devy, devz);
85      cudaEventRecord(endGPU);
86      cudaEventSynchronize(endGPU); // so that cudaEventRecord(
           startGPU) and cudaEventSynchronize(endGPU) are not done
            at the same time
87      float elapsedTimeGPU;
88      cudaEventElapsedTime(&elapsedTimeGPU, startGPU, endGPU);
89      cudaMemcpy(hostzGPU, devz, vecSize * sizeof(int),
           cudaMemcpyDeviceToHost);
90
91      clock_t startCPU = clock(); // save current time
92      CPU_fn(hostx, hosty, hostzCPU, vecSize);
93      clock_t endCPU = clock();
94      float elapsedTimeCPU = endCPU - startCPU;
95      printf("Time elapsed CPU: %f miliseconds\n",
           elapsedTimeCPU);
96      printf("Time elapsed GPU: %f miliseconds\n",
           elapsedTimeGPU);
97
98      validate(hostzCPU, hostzGPU, vecSize);
99
100     free(hostx);
101     free(hosty);
102     free(hostzCPU);
103     free(hostzGPU);
104     cudaFree(devx);
105     cudaFree(devy);
106     cudaFree(devz);
107
108     cudaEventDestroy(startGPU);
109     cudaEventDestroy(endGPU);
110
```

```
111         return 0;
112     }
```

Output

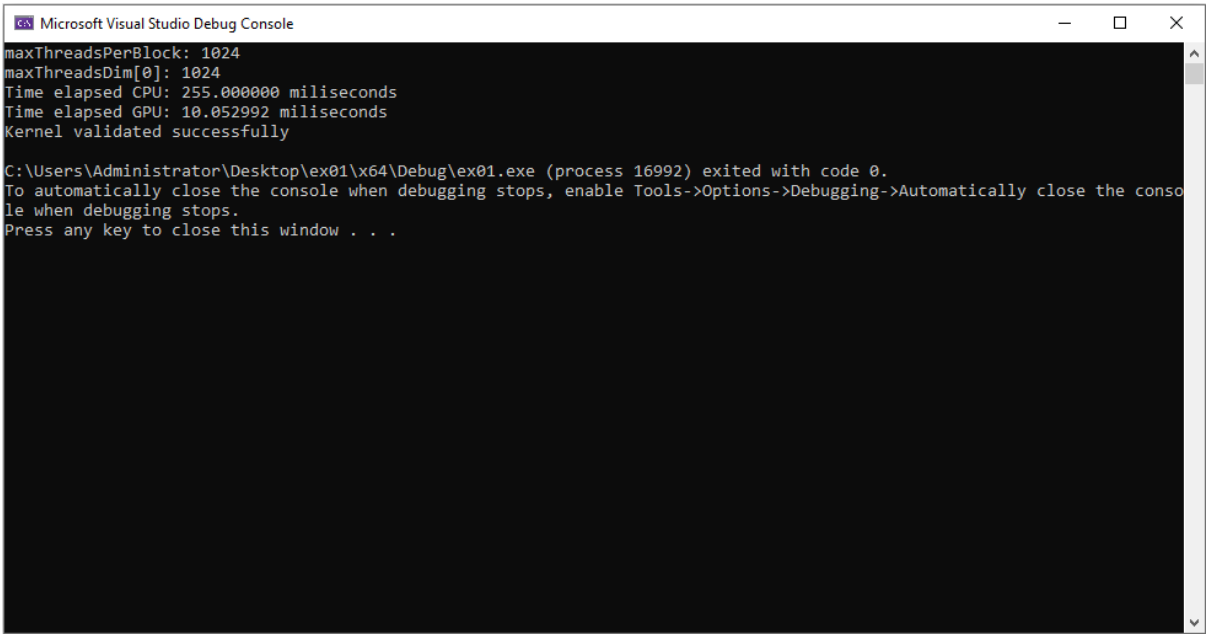

Figure 2: img