In [1]:
```
!/usr/local/cuda/bin/nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:33:58_PDT_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
```

In [2]:
```
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) http
s://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.pkg.de
v/colab-wheels/public/simple/)
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git (https://github.
com/andreinechaev/nvcc4jupyter.git) to /tmp/pip-req-build-xxi1lfcf
  Running command git clone --filter=blob:none --quiet https://github.com/an
dreinechaev/nvcc4jupyter.git (https://github.com/andreinechaev/nvcc4jupyter.
git) /tmp/pip-req-build-xxi1lfcf
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git (https://githu
b.com/andreinechaev/nvcc4jupyter.git) to commit aac710a35f52bb78ab34d2e52517
237941399eff
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) ... done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-py3-none-any.whl s
ize=4305 sha256=9ac0d74cfea2102bb7cc31936181f5e05a8109d6dd004495e636cb6b0f80
9d95
  Stored in directory: /tmp/pip-ephem-wheel-cache-x7nzram1/wheels/a8/b9/18/2
3f8ef71ceb0f63297dd1903aedd067e6243a68ea756d6feea
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
```

In [3]:
```
%load_ext nvcc_plugin
```

```
created output directory at /content/src
Out bin /content/result.out
```

In [4]:
```
%%cuda --name vector_add.cu
#include<iostream>
#include<bits/stdc++.h>
#include<cuda.h>

using namespace std;

void fill_array(int *arr,int size){
    for(int i = 0;i < size; i++){
        arr[i] = rand() % 100;
    }
}

void add_cpu(int *arr1, int *arr2, int *result, int size){
    for(int i = 0;i < size; i++){
        result[i] = arr1[i] + arr2[i];
    }
}

void print_matrix(int *arr, int size){
    for(int i = 0; i < size; i++){
        cout << arr[i] << " ";
    }
    cout << endl;
}

__global__ void add(int *arr1, int *arr2, int *arr3){
    int block_id = blockIdx.x;
    arr3[block_id] = arr1[block_id] + arr2[block_id];
}

int main(){
    int *arr1_cpu,*arr2_cpu,*result_cpu;
    int size;
    cout << "Enter size of vector: ";
    cin >> size;

    arr1_cpu = new int[size];
    arr2_cpu = new int[size];
    result_cpu = new int[size];

    fill_array(arr1_cpu,size);
    cout << "Array 1: ";
    print_matrix(arr1_cpu,size);
    fill_array(arr2_cpu,size);
    cout << "Array 2: ";
    print_matrix(arr2_cpu,size);

    int *arr1_gpu,*arr2_gpu,*result_gpu;

    cudaMallocManaged(&arr1_gpu, size * sizeof(int));
    cudaMallocManaged(&arr2_gpu, size * sizeof(int));
    cudaMallocManaged(&result_gpu, size * sizeof(int));

    cudaMemcpy(arr1_gpu,arr1_cpu,size * sizeof(int),cudaMemcpyHostToDevice);
    cudaMemcpy(arr2_gpu,arr2_cpu,size * sizeof(int),cudaMemcpyHostToDevice);
    cudaEvent_t start,stop;
```

```
    float elapsedTime;

    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    cudaEventRecord(start,0);

    add<<<size,1>>>(arr1_gpu,arr2_gpu,result_gpu);
    cudaEventRecord(stop,0);
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&elapsedTime,start,stop);
    cudaEventDestroy(start);
    cudaEventDestroy(stop);
    cudaMemcpy(result_cpu,result_gpu,size * sizeof(int),cudaMemcpyDeviceToHos
    cout << "GPU result:\n";
    print_matrix(result_cpu,size);
    cout<<"Elapsed Time = "<<elapsedTime<<" milliseconds" << endl;
    cudaFree(arr1_gpu);
    cudaFree(arr2_gpu);
    cudaFree(result_gpu);

    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    cudaEventRecord(start,0);

    add_cpu(arr1_cpu,arr2_cpu,result_cpu,size);
    cudaEventRecord(stop,0);
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&elapsedTime,start,stop);
    cudaEventDestroy(start);
    cudaEventDestroy(stop);
    cout << "CPU result:\n";
    print_matrix(result_cpu,size);
    cout<<"Elapsed Time = "<<elapsedTime<<" milliseconds" << endl;




    return 0;
}
```

Out[4]: 'File written in /content/src/vector_add.cu'

In [5]:
```
!nvcc /content/src/vector_add.cu -o /content/src/vector_add
```

In [6]:
```
!/content/src/vector_add
```

```
Enter size of vector: 25000
Array 1: 83 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 40 26 72 36 11 6
8 67 29 82 30 62 23 67 35 29 2 22 58 69 67 93 56 11 42 29 73 21 19 84 37
98 24 15 70 13 26 91 80 56 73 62 70 96 81 5 25 84 27 36 5 46 29 13 57 24
95 82 45 14 67 34 64 43 50 87 8 76 78 88 84 3 51 54 99 32 60 76 68 39 12
26 86 94 39 95 70 34 78 67 1 97 2 17 92 52 56 1 80 86 41 65 89 44 19 40 2
9 31 17 97 71 81 75 9 27 67 56 97 53 86 65 6 83 19 24 28 71 32 29 3 19 70
68 8 15 40 49 96 23 18 45 46 51 21 55 79 88 64 28 41 50 93 0 34 64 24 14
87 56 43 91 27 65 59 36 32 51 37 28 75 7 74 21 58 95 29 37 35 93 18 28 43
11 28 29 76 4 43 63 13 38 6 40 4 18 28 88 69 17 17 96 24 43 70 83 90 99 7
2 25 44 90 5 39 54 86 69 82 42 64 97 7 55 4 48 11 22 28 99 43 46 68 40 22
11 10 5 1 61 30 78 5 20 36 44 26 22 65 8 16 82 58 24 37 62 24 0 36 52 99
79 50 68 71 73 31 81 30 33 94 60 63 99 81 99 96 59 73 13 68 90 95 26 66 8
4 40 90 84 76 42 36 7 45 56 79 18 87 12 48 72 59 9 36 10 42 87 6 1 13 72
21 55 19 99 21 4 39 11 40 67 5 28 27 50 84 58 20 24 22 69 96 81 30 84 92
72 72 50 25 85 22 99 40 42 98 13 98 90 24 90 9 81 19 36 32 55 94 4 79 69
73 76 50 55 60 42 79 84 93 5 21 67 4 13 61 54 26 59 44 2 2 6 84 21 42 68
28 89 72 8 58 98 36 8 53 48 3 33 33 48 90 54 67 46 68 29 0 46 88 97 49 90
3 33 63 97 53 92 86 25 52 96 75 88 57 29 36 60 14 21 60 4 28 27 50 48 56
```

In [7]:
```
%%cuda --name matrix_multiply.cu
#include<iostream>
#include<bits/stdc++.h>
#include<cuda.h>

using namespace std;

void initialize_matrix(int *array, int rows, int cols){
    for(int i = 0 ; i < rows; i++){
        for(int j = 0; j < cols; j++){
            array[i*cols + j] = rand() % 10;
        }
    }
}

void print_matrix(int *array, int rows, int cols){
    for(int i = 0 ; i < rows; i++){
        for(int j = 0; j < cols; j++){
            cout << array[i*cols + j] << " ";
        }
        cout << endl;
    }
}

void matrix_multiplication_cpu(int *a, int *b, int *c, int common, int c_rows
    for(int i = 0; i < c_rows; i++){
        for(int j = 0; j < c_cols; j++){
            int sum = 0;
            for(int k = 0; k < common; k++){
                sum += a[i*common + k] * b[k*c_cols + j];
            }
            c[i*c_cols + j] = sum;
        }
    }
}


__global__
void matrix_multiply(int *a, int *b, int *c, int m, int n, int k)
{
    int row = blockIdx.y*blockDim.y + threadIdx.y;
    int col = blockIdx.x*blockDim.x + threadIdx.x;
    int sum=0;

    if(col<k && row<m) {
      for(int j=0;j<n;j++)
      {
          sum += a[row*n+j] * b[j*k+col];
      }
      c[k*row+col]=sum;
    }

}


int main(){
```

```
int A_rows, A_cols, B_rows, B_cols, C_rows, C_cols;
cout << "Dimensions of matrix 1:\n";
cout << "Rows: ";
cin >> A_rows;
cout << "Columns: ";
cin >> A_cols;
cout << "Dimensions of matrix 2:\n";
cout << "Rows: " << A_cols << endl << "Columns: ";
cin >> B_cols;
B_rows = A_cols;
C_rows = A_rows;
C_cols = B_cols;

int A_size = A_rows * A_cols;
int B_size = B_rows * B_cols;
int C_size = C_rows * C_cols;

int *A, *B, *C;
int *m1,*m2,*result;

A = new int[A_size];
B = new int[B_size];
C = new int[C_size];

initialize_matrix(A,A_rows,A_cols);
cout << "Matrix 1\n";
print_matrix(A,A_rows,A_cols);
initialize_matrix(B,B_rows,B_cols);
cout << "Matrix 2\n";
print_matrix(B,B_rows,B_cols);

cudaMallocManaged(&m1, A_size * sizeof(int));
cudaMallocManaged(&m2, B_size * sizeof(int));
cudaMallocManaged(&result, C_size * sizeof(int));

cudaMemcpy(m1,A,A_size * sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(m2,B,B_size * sizeof(int), cudaMemcpyHostToDevice);

dim3 dimGrid(2,2);
dim3 dimBlock(C_rows,C_cols);

float gpu_elapsed_time;
cudaEvent_t gpu_start,gpu_stop;

cudaEventCreate(&gpu_start);
cudaEventCreate(&gpu_stop);
cudaEventRecord(gpu_start);
matrix_multiply<<<dimGrid,dimBlock>>>(m1,m2,result,C_rows,A_cols,C_cols);
cudaEventRecord(gpu_stop);
cudaEventSynchronize(gpu_stop);
cudaEventElapsedTime(&gpu_elapsed_time, gpu_start, gpu_stop);
cudaEventDestroy(gpu_start);
cudaEventDestroy(gpu_stop);

cudaMemcpy(C,result,C_size*sizeof(int),cudaMemcpyDeviceToHost);
cout << "GPU result:\n";
print_matrix(C,C_rows,C_cols);
```

```
    cout<<"GPU Elapsed time is: "<<gpu_elapsed_time<<" milliseconds"<<endl;

    cudaEventCreate(&gpu_start);
    cudaEventCreate(&gpu_stop);
    cudaEventRecord(gpu_start);
    matrix_multiplication_cpu(A,B,C,A_cols,C_rows,C_cols);
    cudaEventRecord(gpu_stop);
    cudaEventSynchronize(gpu_stop);
    cudaEventElapsedTime(&gpu_elapsed_time, gpu_start, gpu_stop);
    cudaEventDestroy(gpu_start);
    cudaEventDestroy(gpu_stop);

    cout << "CPU result:\n";
    print_matrix(C,C_rows,C_cols);
    cout<<"CPU Elapsed time is: "<<gpu_elapsed_time<<" milliseconds"<<endl;

    cudaFree(m1);
    cudaFree(m2);
    cudaFree(result);

    return 0;
}
```

Out[7]: 'File written in /content/src/matrix_multiply.cu'

In [8]:
```
!nvcc /content/src/matrix_multiply.cu -o /content/src/matrix_multiply
```

In [10]:
```
!/content/src/matrix_multiply
```

```
Dimensions of matrix 1:
Rows: 10
Columns: 8
Dimensions of matrix 2:
Rows: 8
Columns: 12
Matrix 1
3 6 7 5 3 5 6 2
9 1 2 7 0 9 3 6
0 6 2 6 1 8 7 9
2 0 2 3 7 5 9 2
2 8 9 7 3 6 1 2
9 3 1 9 4 7 8 4
5 0 3 6 1 0 6 3
2 0 6 1 5 5 4 7
6 5 6 9 3 7 4 5
2 5 4 7 4 4 3 0
Matrix 2
7 8 6 8 8 4 3 1 4 9 2 0
6 8 9 2 6 6 4 9 5 0 4 8
7 1 7 2 7 2 2 6 1 0 6 1
5 9 4 9 0 9 1 7 7 1 1 5
9 7 7 6 7 3 6 5 6 3 9 4
8 1 2 9 3 9 0 8 8 5 0 9
6 3 8 5 6 1 1 5 9 8 4 8
1 0 3 0 4 4 4 4 7 6 3 1
GPU result:
236 168 226 188 189 175 84 227 210 126 134 187
214 163 165 237 161 217 69 190 233 193 71 156
204 140 198 183 159 212 83 248 270 159 106 221
202 126 175 179 156 122 72 163 208 151 124 164
243 182 222 190 180 209 90 253 195 84 134 184
277 237 242 288 210 236 98 237 295 225 125 217
134 122 139 136 116 101 51 112 146 120 76 88
177 83 156 132 160 121 81 158 176 133 123 115
271 215 241 252 206 243 101 261 263 169 135 203
193 164 173 172 132 160 68 187 169 81 103 155
GPU Elapsed time is: 0.01104 milliseconds
CPU result:
236 168 226 188 189 175 84 227 210 126 134 187
214 163 165 237 161 217 69 190 233 193 71 156
204 140 198 183 159 212 83 248 270 159 106 221
202 126 175 179 156 122 72 163 208 151 124 164
243 182 222 190 180 209 90 253 195 84 134 184
277 237 242 288 210 236 98 237 295 225 125 217
134 122 139 136 116 101 51 112 146 120 76 88
177 83 156 132 160 121 81 158 176 133 123 115
271 215 241 252 206 243 101 261 263 169 135 203
193 164 173 172 132 160 68 187 169 81 103 155
CPU Elapsed time is: 0.007648 milliseconds
```

In [9]: