Lab2: GPU Vector-Vector Addition

Introduction

In today's world, nearly every emerging technology relies on specially designed GPUs, whether in AI, gaming, finance, construction, or other fields. GPUs have become an essential component of modern laptops and desktop computers. While we may have a general understanding of what a GPU is, truly grasping its functionality requires exploring its architecture and operation.

In this assignment, we will utilize Nvidia GPUs and the CUDA library to examine GPU processing capabilities. Specifically, we will perform vector-vector addition on randomly generated vectors of varying sizes, comparing execution times between the CPU and GPU.

Machine Specifications

Before getting on with the implementation of vector-vector addition, first, the GPU specifications of the device must be known:

Manufacturer	HP
Operating System	Windows 11 (24H2)
Processor	Intel(R) Core (TM) i7-14650HX
GPU	Nvidia GeForce RTX 4060
GPU Architecture	Ada Lovelace[1]
Total GPU Memory	15 GB
Dedicated GPU Memory	8 GB
Shared GPU Memory	7 GB
GPU Clock Speed	16 Gbps[1]
IDE	Visual Studio 2022
Programming Language	C++

Program

Pseudocode

A naïve approach was followed for vector-vector addition in both CPU and GPU implementations:

```
for (i=0; i<sizeOfVector; i++)
        c[i] = a[i] + b[i]
```

Residual Calculation

To check correctness of the GPU calculation of vector-vector addition, the following formula was used:

$$Residual = \frac{\sum_{1}^{sizeOfVector} \|Result_{GPU}[i] - Result_{CPU}[i]\|}{sizeOfVector}$$

Implementation of Correct Cases

Some of the screenshots for results for GPU vector-vector additions for different block and thread sizes are given below:

```
Block Size: 32
Thread Size: 1024
Size Of Vector: 32768
CPU Sequential Execution Time: 28 microseconds
GPU Time with Memory Copy Calls: 419 microseconds
GPU Time (only function calls): 70 microseconds

Residual: 0
**************************************************
```

```
Block Size: 4
Thread Size: 512
Size Of Vector: 2048
CPU Sequential Execution Time: 1 microseconds
GPU Time with Memory Copy Calls: 297 microseconds
GPU Time (only function calls): 12 microseconds

Residual: 0
**************************************************
```

```
Block Size: 1
Thread Size: 32
Size Of Vector: 32
CPU Sequential Execution Time: 0 microseconds
GPU Time with Memory Copy Calls: 276 microseconds
GPU Time (only function calls): 12 microseconds

Residual: 0
**************************************************
```

```
Block Size: 512
Thread Size: 1024
Size Of Vector: 524288
CPU Sequential Execution Time: 615 microseconds
GPU Time with Memory Copy Calls: 1886 microseconds
GPU Time (only function calls): 303 microseconds

Residual: 0
**************************************************
```

Implementation of Odd Cases

The following screenshots show the edge/wrong cases that were tried using the GPU:

1. In the case where the size of the vectors is larger than the calculated size of the vectors from the block size and the thread size, the GPU only computes on the data that it can (in terms of the block size and thread size given) and the rest of the vector remains 0.

```
Block Size: 1
Thread Size: 5
Size Of Vector: 10
CPU Sequential Execution Time: 0 microseconds
GPU Time with Memory Copy Calls: 366 microseconds
GPU Time (only function calls): 12 microseconds

Residual: 2.66642

Vector1: [ 6.76365, 3.16738, 0.399039, 6.69512, -6.93796, 0.911567, -3.88722, -8.56191, 7.60601, 1.54908, ]
Vector2: [ -3.62472, -4.94909, -2.2985, 9.72626, 7.53741, 4.32151, 9.39501, 6.08487, -9.70591, 9.79726, ]
Result: [ 3.13893, -1.7817, -1.89947, 16.4214, 0.599449, 0, 0, 0, 0, 0, ]
**************************************************
```

2. In the opposite direction from above, when the thread size is larger than the actual size of the vector, the GPU does not give any error. Internally though, it may be performing phantom calculations.

```
Block Size: 1
Thread Size: 10
Size Of Vector: 5
CPU Sequential Execution Time: 0 microseconds
GPU Time with Memory Copy Calls: 345 microseconds
GPU Time (only function calls): 12 microseconds

Residual: 0

Vector1: [ 2.89419, 8.918, -1.99587, -0.217339, 2.10066, ]
Vector2: [ 1.39083, 5.8949, 3.1899, 4.60266, 6.24079, ]
Result: [ 4.28502, 14.8129, 1.19403, 4.38533, 8.34145, ]
**************************************************
```

3. Similar to the first case, when the block size and the thread size is 1, only the first element of the vector is computed on, and the rest of the vector is left as is.

```
Block Size: 1
Thread Size: 1
Size Of Vector: 10
CPU Sequential Execution Time: 0 microseconds
GPU Time with Memory Copy Calls: 397 microseconds
GPU Time (only function calls): 29 microseconds

Residual: 8.55725

Vector1: [ -6.15513, -9.2147, 7.86493, 2.01893, 9.58354, -6.63752, 3.19214, -5.34346, 9.60647, -3.68691, ]
Vector2: [ -4.71621, -4.30025, 0.431476, 9.99866, 0.662453, 3.02379, 6.4656, -9.37224, -5.40707, -5.62407, ]
Result: [ -10.8713, 0, 0, 0, 0, 0, 0, 0, 0, 0, ]
**************************************************
```

4. In the case of exceeding the thread size greater than 1024, CUDA throws an error as shown – “invalid configuration argument”, and does not compute anything, leaving the result bare.

```
Block Size: 1
Thread Size: 2048
Size Of Vector: 2048
CUDA error: invalid configuration argument

CPU Sequential Execution Time: 2 microseconds
GPU Time with Memory Copy Calls: 291 microseconds
GPU Time (only function calls): 3 microseconds

Residual: 6.59473
```

```
Result: [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

5. Finally, for block size 1 and thread size 33 (1 more than the warp size), we see that although the residual is 0, and there is no issue with the result, the total execution time, even after ignoring the memory calls is much higher than that computed in the case where block size was 1 and thread size was 32 (the screenshot for that execution can be seen above).

```
Block Size: 1
Thread Size: 33
Size Of Vector: 33
CPU Sequential Execution Time: 0 microseconds
GPU Time with Memory Copy Calls: 731 microseconds
GPU Time (only function calls): 463 microseconds

Residual: 0

Vector1: [ 4.88364, 5.03728, -6.84683, -3.95817, 3.46915, -2.47977, 2.17363, -0.818906, -8.66245, -9.78902, -9.4701, -7.
17341, -6.34099, -3.84045, -7.33814, 4.14124, -0.301981, 1.43201, 9.04695, 2.09129, -6.30813, -4.88051, 8.64032, -9.3237
8, 5.66115, -7.5707, -2.80471, 7.37469, -9.26291, 9.06546, -7.70767, -7.42986, -0.799817, ]
Vector2: [ 2.23242, 9.28512, -7.09626, -9.40286, -0.755653, -1.78733, -7.40956, 4.61549, -6.90418, 2.46151, -6.61693, -6
.93932, -9.02033, -4.34922, 8.38309, -0.171758, 0.355013, -3.21714, -8.46774, 7.45327, -4.67899, -1.63961, 3.80291, -7.6
1327, 8.15502, 5.99344, 9.70442, -9.88627, -2.94168, 3.59083, 9.9239, 5.82386, 1.55035, ]
Result: [ 7.11606, 14.3224, -13.9431, -13.361, 2.7135, -4.2671, -5.23594, 3.79658, -15.5666, -7.32751, -16.087, -14.1127
, -15.3613, -8.18967, 1.04495, 3.96948, 0.0530323, -1.78513, 0.579214, 9.54456, -10.9871, -6.52011, 12.4432, -16.9371, 1
3.8162, -1.57726, 6.89971, -2.51158, -12.2046, 12.6563, 2.21623, -1.606, 0.750532, ]
******************************************************
```

Analysis

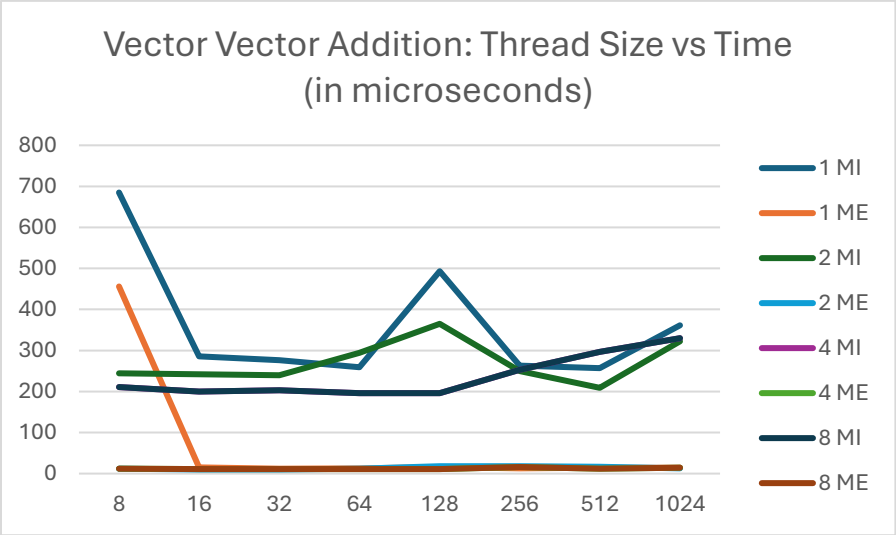
Sequential vs GPU calculations

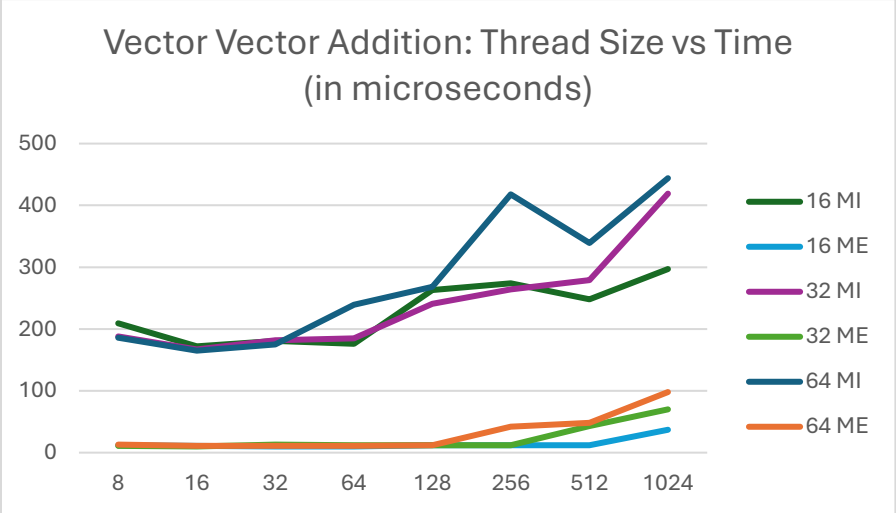
From the data attached in the zip file, we can clearly see that the sequential execution for vector-vector addition is much less than the GPU execution (even when comparing with memory exclusive times). This may be attributed to the fact that GPU calculations are more suited in case of very large datasets, where the memory overhead can be ignored for speed of parallelism offered by the GPU.

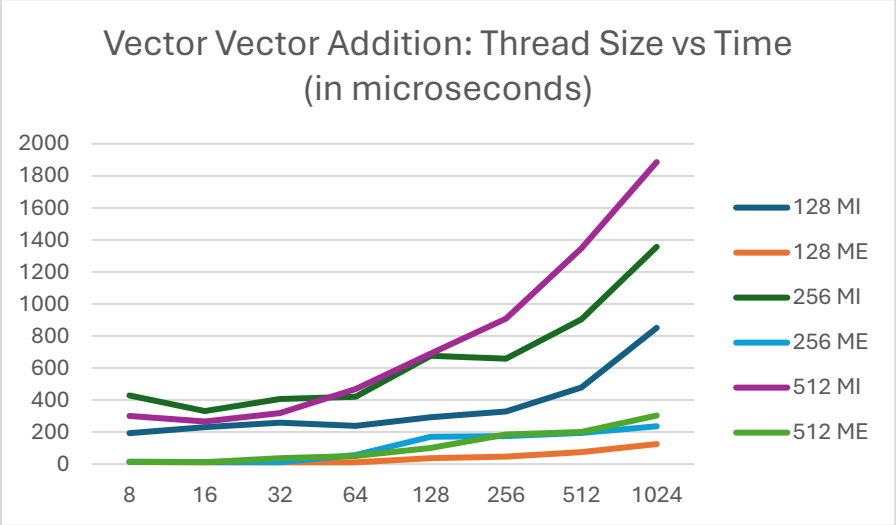
GPU Time Analysis

In the following analyses, MI refers to Memory Inclusive time (time including memcpy calls) while ME refers to Memory Exclusive Time (only function calls)

Thread Size vs Time





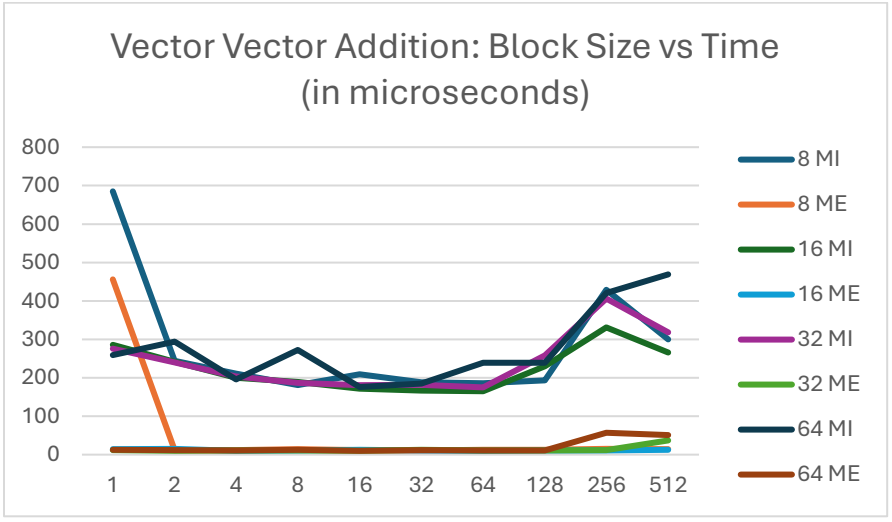


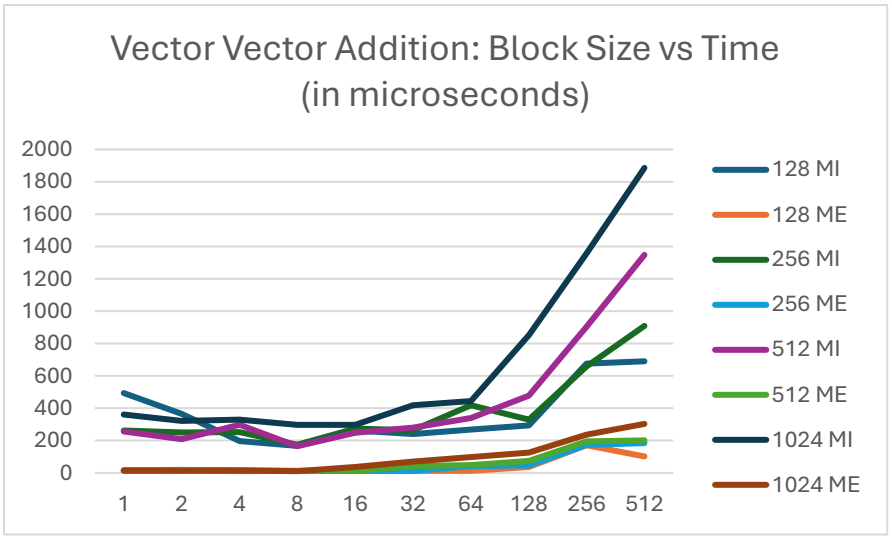
For each legend, the first number indicates the block size, while the second word indicates whether the time is memory inclusive or exclusive.

Some interesting things to note from the above graphs are:

1. Memory inclusive time is always greater than memory exclusive time as expected.
2. The time for block size 1 and thread size 8 is exceptionally high. This may be due to the fact that this was the first execution, and hence, a lot more overhead was encountered due to thrashing ad setup of GPU.
3. As both the thread size and block size increased, the execution time for vector-vector addition also increased.
4. Memory exclusive time remains the same for most cases, no matter the block or thread size.

Block Size vs Time





For each legend, the first number indicates the thread size, while the second word indicates whether the time is memory inclusive or exclusive.

We see a similar pattern as the previous analysis here as well. One interesting thing to note, especially in the first graph, is that between block sizes 4 and 128 for thread sizes between 8 and 64, the memory inclusive time stabilizes. This could mean that for my device, the most optimal range of block size and thread size is this.

References

[1] “NVIDIA GeForce RTX 4060 Laptop GPU - Benchmarks and Specs - NotebookCheck.net Tech.” Accessed: Feb. 14, 2025. [Online]. Available: https://www.notebookcheck.net/NVIDIA-GeForce-RTX-4060-Laptop-GPU-Benchmarks-and-Specs.675692.0.html