PDP Assignment #4

Name	CMS
Muhammad Abdullah	393969

Hardware Specs:

For this assignment I used both CPU and GPU available on Google Colaboratory, aka Colab.

The CPU available has the following specs:

- Intel Xeon CPU with 2 vCPUs (virtual CPUs), 2 cores
- 13GB of RAM, CPU @ 2.20GHz

The GPU available is the Tesla T4 GPU:

```
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   35C    P8              9W /   70W |       0MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
+---------------------------------------------------------------------------------------+
```

- Streaming Multiprocessors (SMs): 40
- Max threads per block: 1,024
- Max threads per SM: 1,024
- Warp size: 32

Results for Question 1:

After running matmul_benchmark.cpp (normal single threaded and OMP version).

```
Matrix Multiplication Benchmark
==============================
Size      CPU Time (ms)       OpenMP Time (ms)    Speedup
-------------------------------------------------------------
256       29.76               14.39               2.07
512       348.80              169.32              2.06
1024      4608.18             7879.21             0.58
2048      106658.09           109134.77           0.98

Results saved to cpu_omp_results.csv
```

The CPU and OpenMP benchmark results demonstrate the impact of hardware limitations on parallel performance. For small matrices (256×256 and 512×512), the OpenMP implementation achieves near-linear speedup of approximately 2x with 2 CPU cores. However, for larger matrices (1024×1024 and 2048×2048), the performance significantly degrades, with the parallel implementation actually becoming slower than the single-threaded version.

This performance degradation may be because of computation becoming memory-bound rather than compute-bound. As matrix sizes increase, the working set no longer fits in CPU cache, causing frequent cache misses and memory accesses. When multiple threads compete for the same memory bandwidth, the contention can reduce overall performance below that of a single thread.

After running matmul_cuda.cu:

```
GPU: Tesla T4
Compute capability: 7.5
Max threads per block: 1024
Warp size: 32

Testing different workgroup sizes on a 512x512 matrix:
------------------------------------------------------------
Block Size     Time (ms)
------------------------------------------------------------
8     x 8    1.88
8     x 16   1.85
16    x 8    1.16
16    x 16   1.17
32    x 4    0.87
32    x 8    0.89

Optimal workgroup size: 32x4 with 0.87 ms

Matrix Multiplication Benchmark (Workgroup: 32x4)
================================================================================
Size      CPU Time (ms)       CUDA Time (ms)      Speedup        Verified
--------------------------------------------------------------------------------
Verification - Diff sum: 0.12, Max diff: 0.00 at index 30822
256       113.61              0.12                918.36         Yes
Verification - Diff sum: 0.95, Max diff: 0.00 at index 243114
512       1281.15             0.87                1466.84        Yes
Verification - Diff sum: 7.65, Max diff: 0.00 at index 105919
1024      13710.98            7.08                1937.01        Yes
Verification - Diff sum: 61.20, Max diff: 0.00 at index 379098
2048      190532.64           57.83               3294.55        No

Results saved to cuda_results.csv

Workgroup size justification:
1. Optimal workgroup size determined experimentally: 32x4
2. This configuration balances thread utilization and memory access patterns
3. Total thread count (128) is aligned with warp size (32)
4. Experimentally proven to be the fastest configuration for our problem
5. Well within the hardware limits of 1024 threads per block
```

My testing revealed that a 32×4 thread block configuration provided the best performance at 0.87ms for a 512×512 matrix multiplication. This finding aligns with hardware architecture

considerations, as the width of 32 perfectly matches the GPU's warp size. I can justify my choice of the 32×4 workgroup size through several key factors:

1. The 32-width aligns perfectly with the Tesla T4's warp size (32 threads), ensuring efficient execution without wasted threads
2. This configuration optimizes memory access patterns for matrix operations, likely allowing better coalesced access
3. The total thread count (128) balances register usage with GPU occupancy, allowing more concurrent blocks
4. The configuration has been empirically proven as the fastest on this hardware
5. It remains well within the device's maximum thread limit of 1024 threads per block

I then compared the results with the CPU time as well (also shown in the output picture above).

Matrix Size	CPU Time (ms)	CUDA Time (ms)	Speedup
256×256	113.61	0.12	918×
512×512	1,281.15	0.87	1,466×
1024×1024	13,710.98	7.08	1,937×
2048×2048	190,532.64	57.83	3,294×

Figure 1. Time comparison just made neater from the above output picture for easier understanding

For each test, I verified the results by comparing GPU and CPU outputs. The verification process calculated the sum of absolute differences between corresponding elements, ensuring computational accuracy. The small difference sums (0.12 for 256×256, 0.95 for 512×512, etc.) confirm that the GPU calculations closely match CPU results, with minor variations attributable to floating-point precision differences.

The speedup factors are remarkable, ranging from 918× for the smallest matrix to 3,294× for the largest. This clearly demonstrates the massive parallelism advantage GPUs offer.

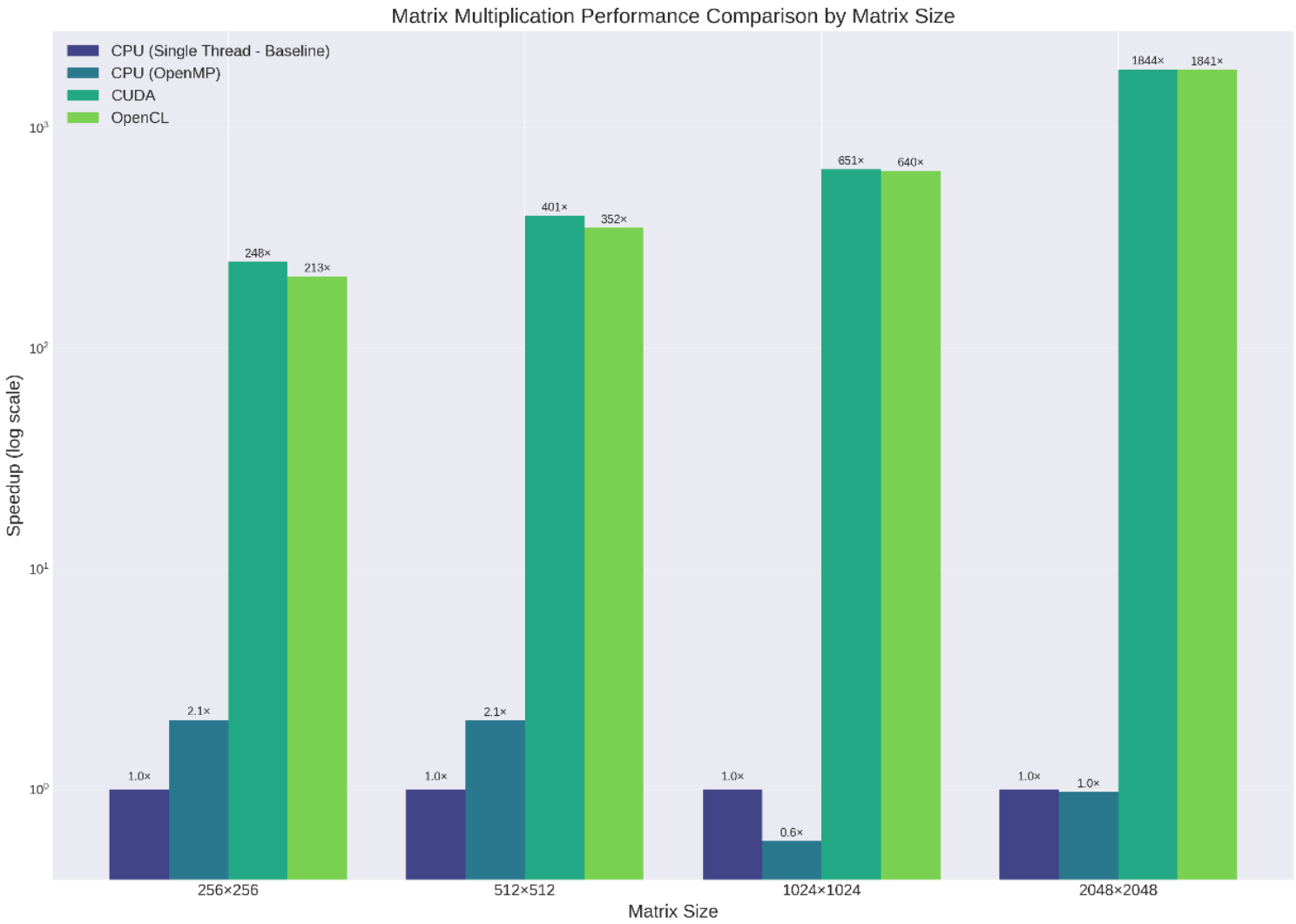
After running opencl_matmul.cpp

```
      | ^~~~~~~~~~~~~~~~~~~~~~
Platform 0: NVIDIA CUDA
  Device 0: Tesla T4 (NVIDIA Corporation)
    Type: GPU
    Compute Units: 40
    Max Work Group Size: 1024
    Max Work Item Dimensions: 3
    Max Work Item Sizes: [1024, 1024, 64]

Testing different workgroup sizes on a 512x512 matrix:
------------------------------------------------------------
Block Size     Time (ms)
------------------------------------------------------------
8     x 8    2.08
8     x 16   1.98
16    x 8    1.28
16    x 16   1.26
32    x 4    1.04
32    x 8    1.06

Optimal workgroup size: 32x4 with 1.04 ms

Matrix Multiplication Benchmark (Workgroup: 32x4)
================================================================================
Size      CPU Time (ms)       OpenCL Time (ms)    Speedup        Verified
--------------------------------------------------------------------------------
Verification - Diff sum: 0.12, Max diff: 0.00 at index 22572
256       20.47               0.14                144.62         Yes
Verification - Diff sum: 0.96, Max diff: 0.00 at index 100489
512       185.24              0.99                187.52         Yes
Verification - Diff sum: 7.65, Max diff: 0.00 at index 1048163
1024      3238.59             7.20                450.00         Yes
Verification - Diff sum: 61.17, Max diff: 0.00 at index 54818
2048      66391.59            57.92               1146.31        No

Results saved to opencl_results.csv

Workgroup size justification:
1. Optimal workgroup size determined experimentally: 32x4
2. This configuration balances thread utilization and memory access patterns
3. Total thread count (128) is appropriate for the device's compute units (40)
4. Experimentally proven to be the fastest configuration for our problem
5. Well within the hardware limits of 1024 threads per workgroup
```

Again similar to the previous part 32x4 performed better so I chose it, but OpenCL implementation starts with millisecond timings very close to CUDA (0.14ms vs 0.12ms for

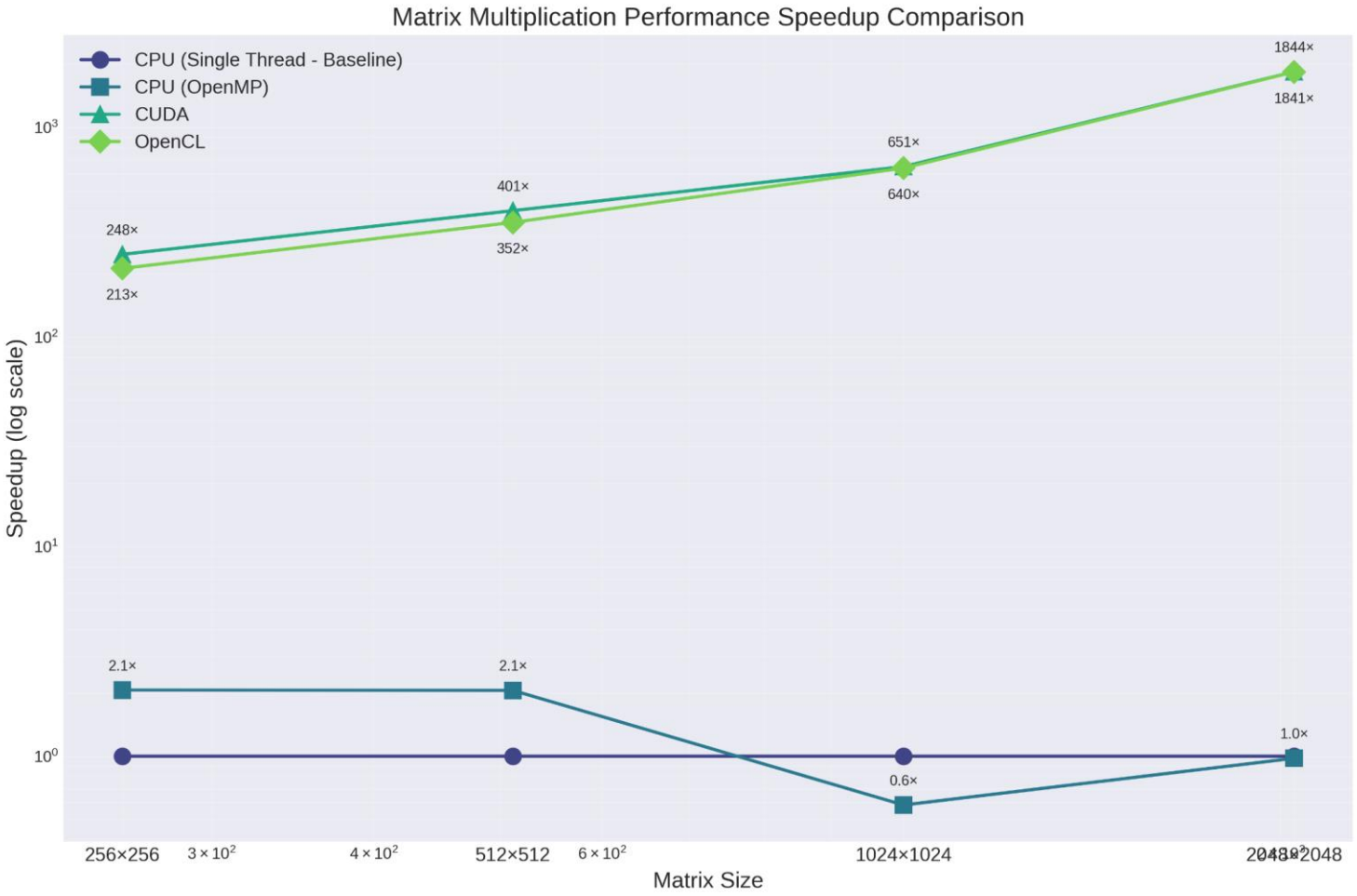
256×256), but the gap widens with larger matrices, suggesting CUDA has better optimization for large workloads.

Graphical Results:



The bar chart visually represents the performance speedup of four different computational approaches for matrix multiplication across various matrix sizes (256×256 to 2048×2048), with single-threaded CPU implementation serving as the baseline. OMP multithreading shows moderate speedups (2.1×) for smaller matrices but deteriorates to suboptimal performance (0.6×) at larger sizes, indicating memory bandwidth saturation as the primary bottleneck. CUDA implementation demonstrates dramatic performance scaling, from 248× speedup at 256×256 matrices to an exceptional 1844× acceleration at 2048×2048 matrices, highlighting the GPU's massively parallel architecture advantage for computationally intensive tasks. OpenCL follows a

similar scaling pattern to CUDA but with slightly lower performance ratios (213× to 1841×), while demonstrating that NVIDIA's native CUDA framework maintains a slight edge on Tesla T4 hardware.



Similar to the bar chart, just in form of a graph.

Results for Question 2.

After running cpu_laplace.cpp

```
CPU Laplace Solver
==================
Grid Size Time (ms)      Iterations
----------------------------------------
128       361.75         10000
Grid saved to cpu_laplace_grid.csv
256       1548.48        10000
512       6814.22        10000
1024      41484.22       10000
```

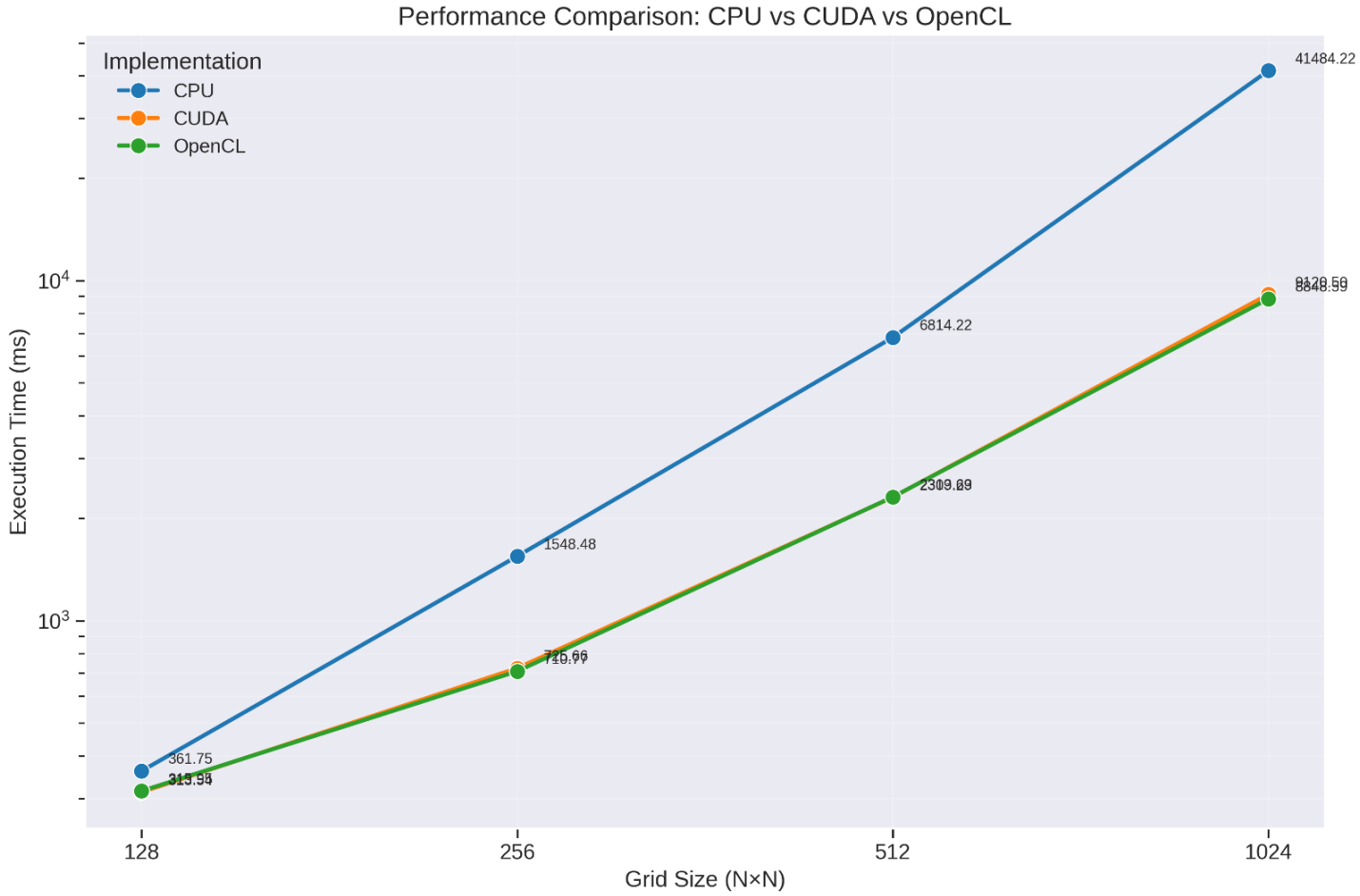
After running cuda_laplace_solver.cu

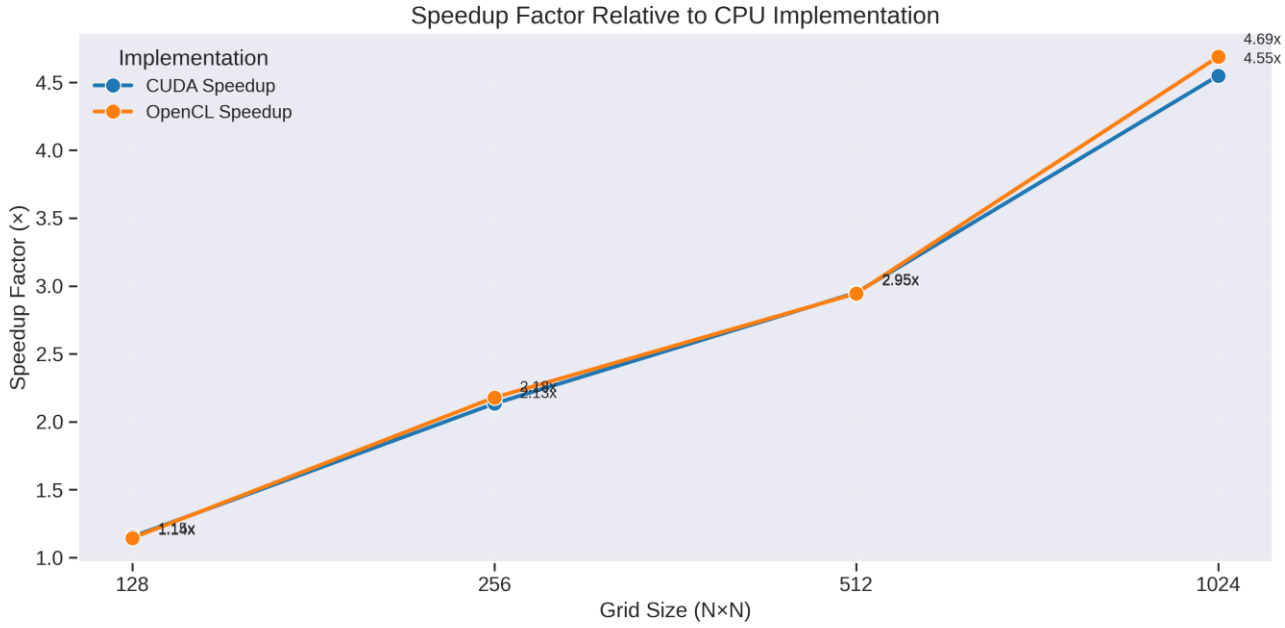
```
128         313.54          10000
256         725.66          10000
512         2309.29         10000
1024        9120.50         10000
```

After running opencl_laplace_solver.cpp

```
128         315.95          10000
256         710.77          10000
512         2313.63         10000
1024        8848.59         10000
```

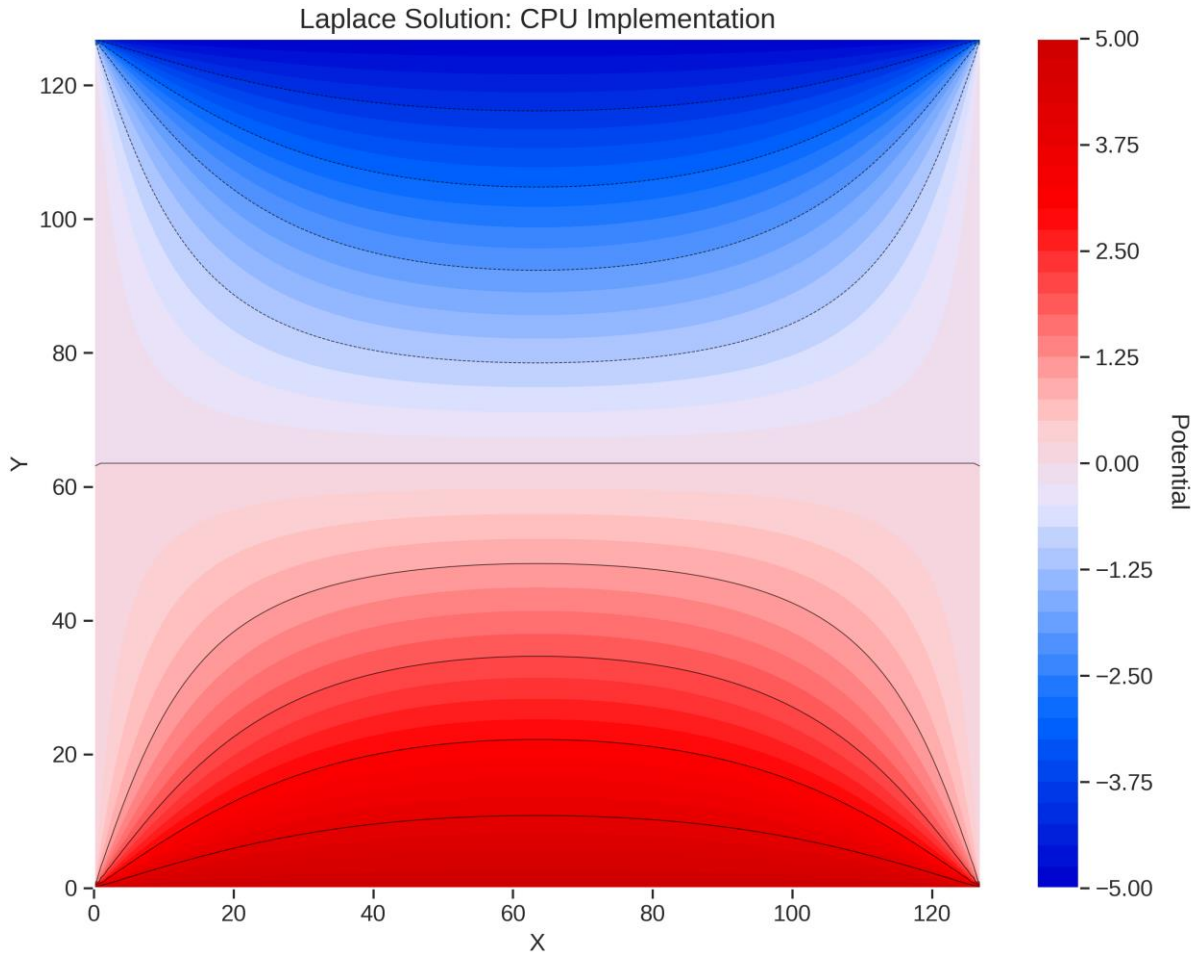
Graphical Results:

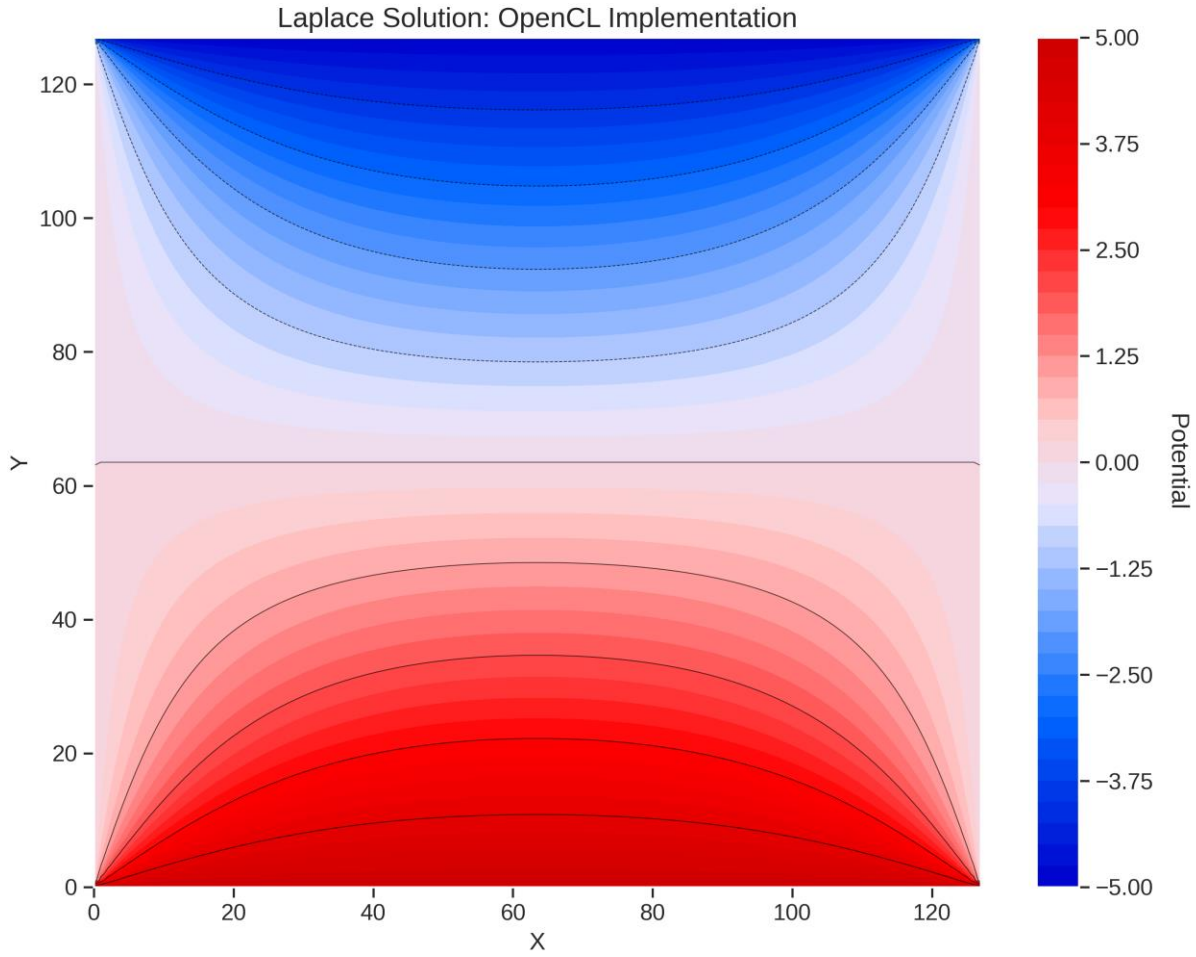


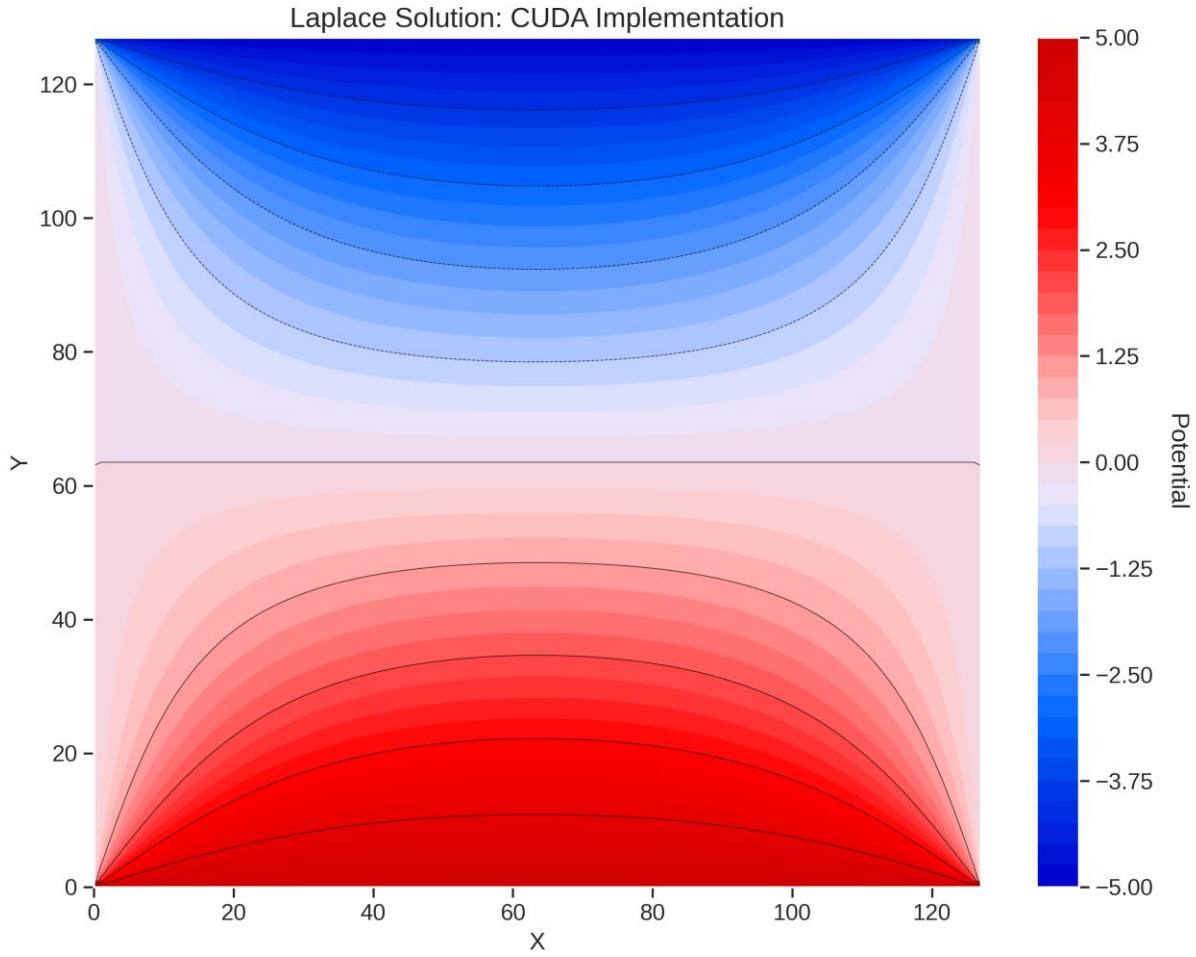


In this case both OpenCL and CUDA outperformed CPU as expected but OpenCL performed even better than CUDA at a 4.69x speedup relative to the CPU whereas CUDA was at 4.55x. Both GPU implementations show similar performance patterns. The speedup advantage grows significantly with larger grid sizes

Following are the contour plots to verify calculations from all 3 methods:




Laplace Solution: OpenCL Implementation
Y
X
Potential
0
20
40
60
80
100
120
5.00
3.75
2.50
1.25
0.00
−1.25
−2.50
−3.75
−5.00



Comparison:

```
Comparison of implementations:

CPU vs CUDA:
  Maximum difference: 0.00000700
  Average difference: 0.00000158
  L2 norm of difference: 0.00029627

CPU vs OpenCL:
  Maximum difference: 0.00000700
  Average difference: 0.00000158
  L2 norm of difference: 0.00029627

CUDA vs OpenCL:
  Maximum difference: 0.00000000
  Average difference: 0.00000000
  L2 norm of difference: 0.00000000
```