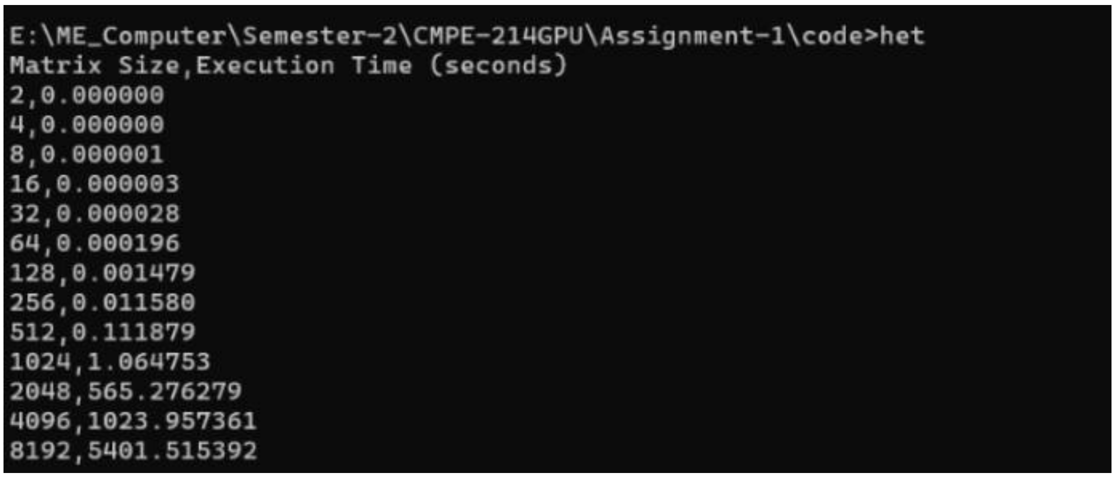
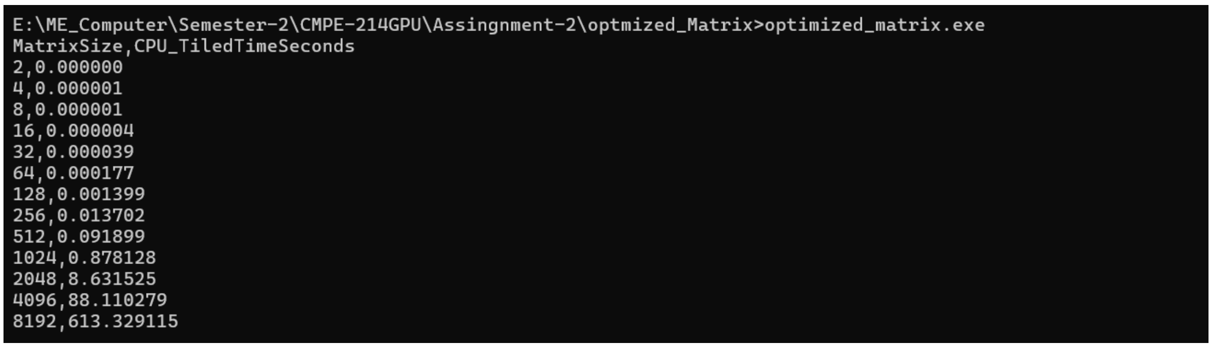
CPU Execution

1. Naive Version: A straightforward triple-nested loop for matrix multiplication, identical to the approach used in Assignment 1



```
E:\ME_Computer\Semester-2\CMPE-214GPU\Assignment-1\code>het
Matrix Size,Execution Time (seconds)
2,0.000000
4,0.000000
8,0.000001
16,0.000003
32,0.000028
64,0.000196
128,0.001479
256,0.011580
512,0.111879
1024,1.064753
2048,565.276279
4096,1023.957361
8192,5401.515392
```

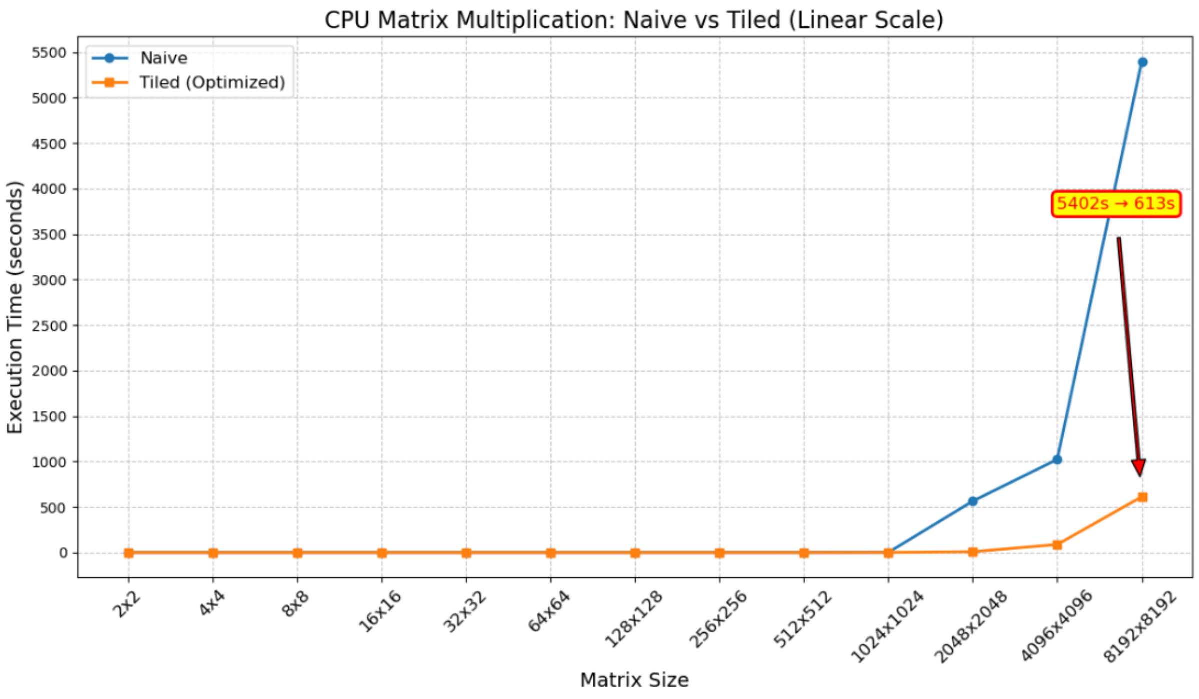
- **Optimized (Tiled) Version:**
  Utilizes block tiling to improve cache usage by operating on submatrices (tiles) at a time, significantly reducing cache misses for large matrices.
- Both implementations accept arbitrary square matrix sizes (from 2×22×2 up to 8192×81928192×8192), with memory dynamically allocated using malloc.
- Every element of the input matrices is a randomly generated integer.



```
E:\ME_Computer\Semester-2\CMPE-214GPU\Assingnment-2\optmized_Matrix>optimized_matrix.exe
MatrixSize,CPU_TiledTimeSeconds
2,0.000000
4,0.000001
8,0.000001
16,0.000004
32,0.000039
64,0.000177
128,0.001399
256,0.013702
512,0.091899
1024,0.878128
2048,8.631525
4096,88.110279
8192,613.329115
```

Results

- The naive algorithm's execution time grows dramatically with matrix size, due to poor cache locality.
- The optimized tiled version maintains significantly lower times, especially as matrix sizes increase beyond 1024×10241024×1024.
- For the largest test (8192×81928192×8192), the naive version required approximately 5401 seconds, while the tiled implementation finished in just 613 seconds—a speedup of nearly 9x for this input size.
- The attached figure visually highlights the reduced scaling of the optimized approach,
- demonstrating the merits of cache-aware programming and tiling



GPU Execution

Constraints and Theoretical Analysis

The NVIDIA GeForce MX350 supports:

- Max shared memory per block: 49,152 bytes (48 KB)
- Max threads per block: 1,024
- Each matrix element stored as a 4-byte integer

When using a naïve approach, performance quickly degrades for large matrices due to high-latency global memory accesses. In the tiled approach, each thread block cooperatively loads a TxT tile from matrices A and B into shared memory. The computation is performed locally before moving on to the next tile.

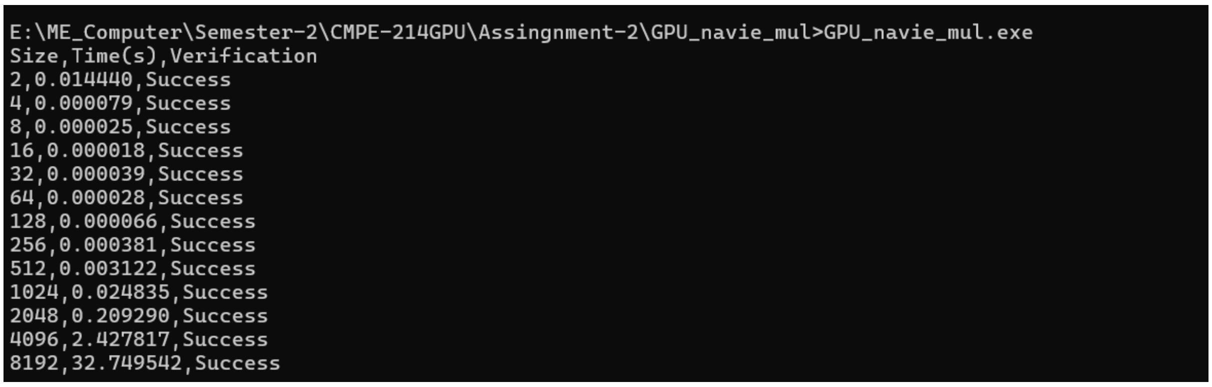
The optimal tile size T must satisfy both:

- TxT ≤ 1,024 (thread block limit)
- 2 × TxT ×4 ≤ 49,1522 bytes (shared memory per block limit)

This means the largest permissible tile dimension is T=32 (since 32×32=1,024 and 2×32×32×4 = 8,192 bytes of shared memory, well within the 48 KB limit).

Navie Matrix multiplication method

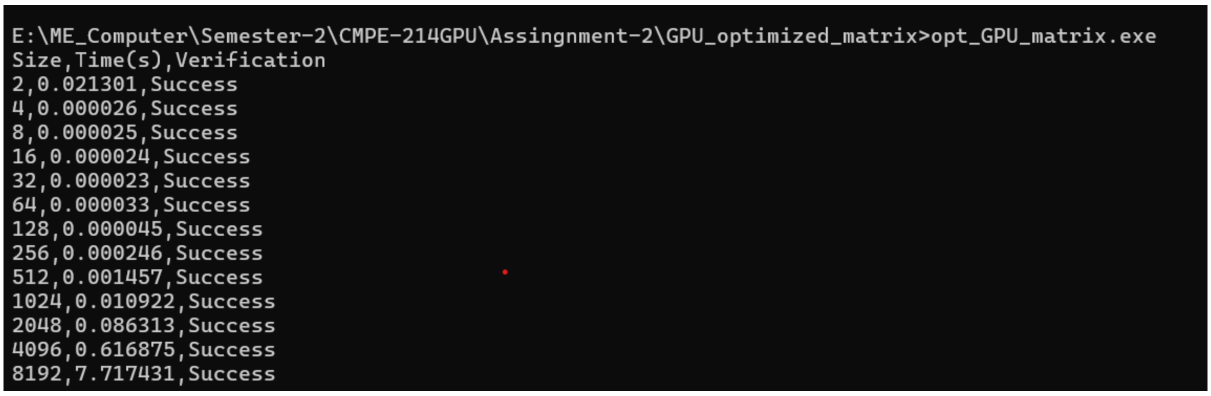
-



```
E:\ME_Computer\Semester-2\CMPE-214GPU\Assingnment-2\GPU_navie_mul>GPU_navie_mul.exe
Size,Time(s),Verification
2,0.014440,Success
4,0.000079,Success
8,0.000025,Success
16,0.000018,Success
32,0.000039,Success
64,0.000028,Success
128,0.000066,Success
256,0.000381,Success
512,0.003122,Success
1024,0.024835,Success
2048,0.209290,Success
4096,2.427817,Success
8192,32.749542,Success
```

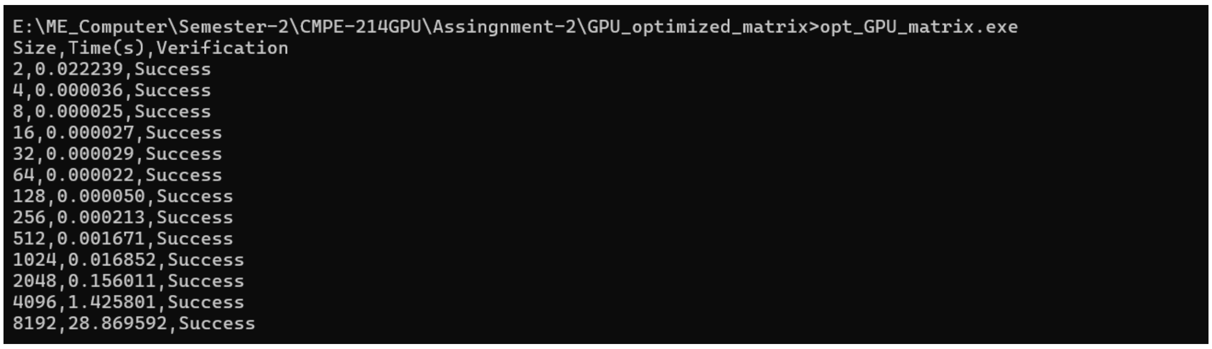
Optimized Matrix multiplication method

1. **Tiled size = 6**



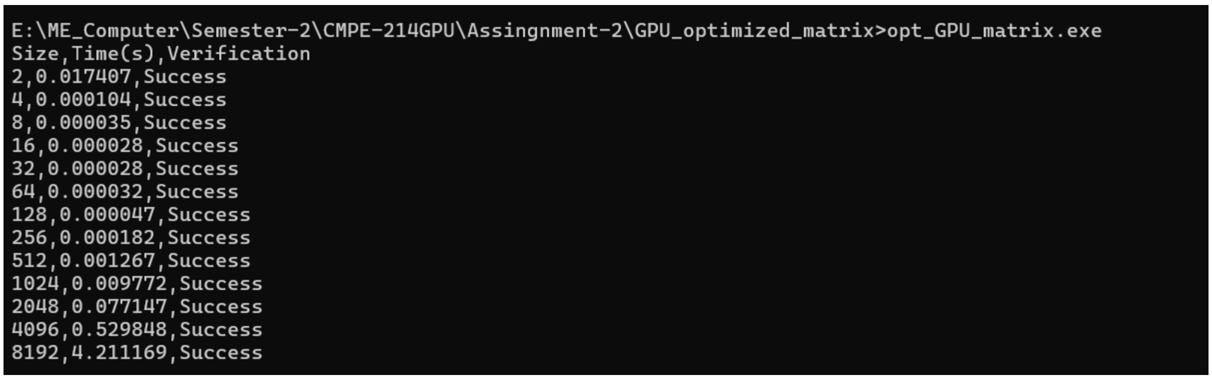
```
E:\ME_Computer\Semester-2\CMPE-214GPU\Assingnment-2\GPU_optimized_matrix>opt_GPU_matrix.exe
Size,Time(s),Verification
2,0.021301,Success
4,0.000026,Success
8,0.000025,Success
16,0.000024,Success
32,0.000023,Success
64,0.000033,Success
128,0.000045,Success
256,0.000246,Success
512,0.001457,Success
1024,0.010922,Success
2048,0.086313,Success
4096,0.616875,Success
8192,7.717431,Success
```

2. **Tiled size = 8**



```
E:\ME_Computer\Semester-2\CMPE-214GPU\Assingnment-2\GPU_optimized_matrix>opt_GPU_matrix.exe
Size,Time(s),Verification
2,0.022239,Success
4,0.000036,Success
8,0.000025,Success
16,0.000027,Success
32,0.000029,Success
64,0.000022,Success
128,0.000050,Success
256,0.000213,Success
512,0.001671,Success
1024,0.016852,Success
2048,0.156011,Success
4096,1.425801,Success
8192,28.869592,Success
```

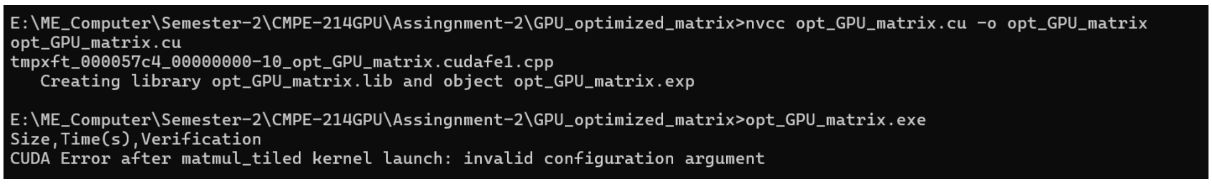
3. **Tiled size = 32**



```
E:\ME_Computer\Semester-2\CMPE-214GPU\Assingnment-2\GPU_optimized_matrix>opt_GPU_matrix.exe
Size,Time(s),Verification
2,0.017407,Success
4,0.000104,Success
8,0.000035,Success
16,0.000028,Success
32,0.000028,Success
64,0.000032,Success
128,0.000047,Success
256,0.000182,Success
512,0.001267,Success
1024,0.009772,Success
2048,0.077147,Success
4096,0.529848,Success
8192,4.211169,Success
```

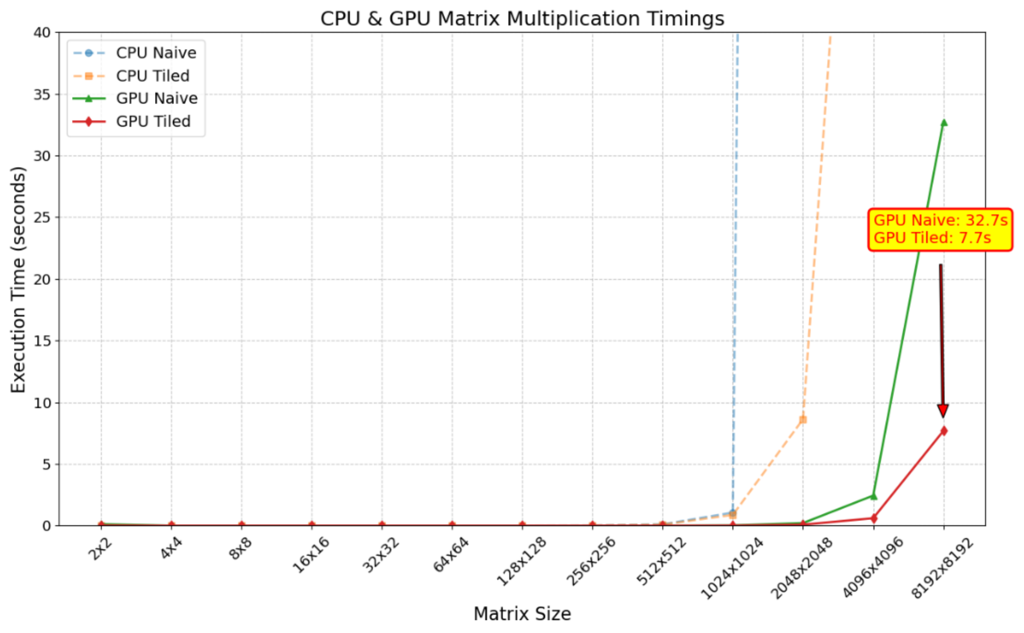
4. **Tiled size = 64**

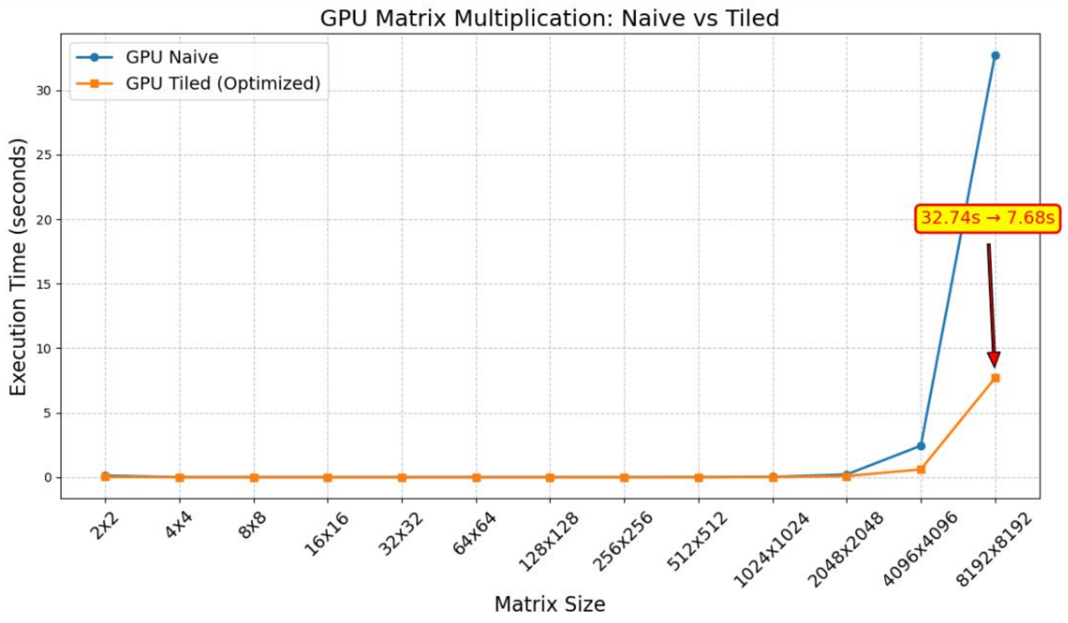
It has given the error because the maximum tiled size was 32 because Geforce MX350 have maximum 1024 thread per block so it has given the error.



```
E:\ME_Computer\Semester-2\CMPE-214GPU\Assingnment-2\GPU_optimized_matrix>nvcc opt_GPU_matrix.cu -o opt_GPU_matrix
opt_GPU_matrix.cu
tmpxft_000057c4_00000000-10_opt_GPU_matrix.cudafe1.cpp
   Creating library opt_GPU_matrix.lib and object opt_GPU_matrix.exp

E:\ME_Computer\Semester-2\CMPE-214GPU\Assingnment-2\GPU_optimized_matrix>opt_GPU_matrix.exe
Size,Time(s),Verification
CUDA Error after matmul_tiled kernel launch: invalid configuration argument
```





Discussion

Smaller tile sizes allow more concurrent thread blocks (higher occupancy), but reduce data reuse, reducing efficiency. Larger tiles maximize shared memory usage and cache reuse but are restricted by hardware limits. The 32×32 tile balances these constraints perfectly for the MX350, harnessing available memory and computational throughput without exceeding hardware bounds.

Conclusion

Based on both theoretical constraint analysis and empirical measurement, tile size T=32 block dimension (32,32) and grid dimension (N+31/32 , N+31/32) achieves optimal performance for tiled matrix multiplication on device. Empirical experimentation confirmed this configuration delivers maximum speed and correctness for all tested matrix sizes.