Answer 1: -

The CUDA version of the given serial code is executed as-

Number of threads per block = 512. This is because at most 1024 threads per block can be executed and to stay on the safe side, the decision to make the number of threads per block as 512 is taken.

Total number of blocks in the grid = (Number of rows in 2D array) / (Number of threads per block).
= SIZE / 512

If SIZE = 8196 , then grid dimension will be 16.
Even if SIZE = 8200, the grid dimension will still be 16 as the above division will give only integral value.

Total number of threads = 16 * 512 = 8196.
Since the number of threads = number of rows in the matrix , threads are scheduled in such a way that each thread will execute its computation on a row of the matrix.

Thread 0	A[0][0]	[0][1]	[0][2]	…	…	…	A[0][8191]
Thread 1	A[1][0]	A[1][1]	A[1][2]	…	…	…	A[1][8191]
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
Thread 8191	A[8191][0]	A[8191][1]	A[8191][2]	…	…	…	A[8191][8191]

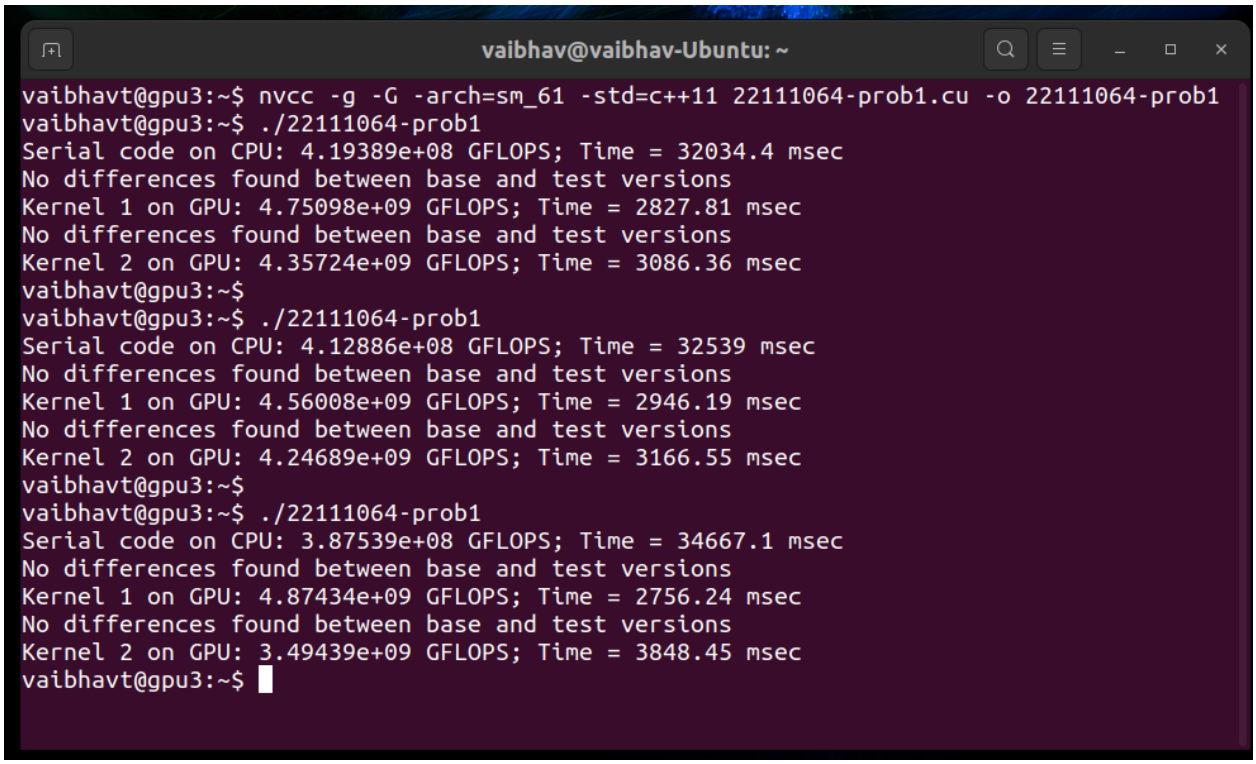
But if SIZE = 8200 , then there will be some extra rows left to compute. To cover them up, the i^{th} thread will execute $(i + 8192)^{th}$ row.
This will cause the program to slow down as compared to when SIZE = 8196 because only now 8 threads will do the remaining 8200 computations and more cache misses.

The given code template is modified by adding an following extras:

- Serial2() method : For executing the serial operation on the matrix of SIZE = 8200.
- Extra execution of serial2() method which takes around same time as taken by serial() method

Due to this methods, the overall execution time of the program will increase but it will not affect the performance of kernel methods.
Compile Command : nvcc -g -G -arch=sm_61 -std=c++11 22111064-prob1.cu -o 22111064-prob1
Run Command : ./22111064-prob1

Output (Same code but with different values of SIZE) : -



Time Taken : -

Serial version : 33080.2 ms

CUDA code (Non-optimised) : 2843.4 ms

CUDA code (Optimised) : 3367.12 ms

Speedup (Compared to serial version) :-

Serial version : 1

CUDA code (Non-optimised) : 11.6

CUDA code (Optimised) : 9.8

The one way of optimizing the program is: -

In the first iteration, let the threads execute in the same manner as discussed above (Each thread corresponds to a row).

In the next iteration, let the threads execute on the remaining rows, but this time, each thread will correspond to a column so that 8192 threads will perform 8 computations. This time, run the code as it was in the previous version(i^{th} thread will execute $(i + 8192)^{th}$ column).

Thread 0	A[0][0]	[0][1]	…	A[0][8191]	A[0][8192]	…	A[0][8199]
Thread 1	A[1][0]	A[1][1]	…	A[1][8191]	A[1][8192]	…	A[1][8199]
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
Thread 8191	A[8191][0]	A[8191][1]	…	A[8191][8191]	A[8191][8192]	…	A[8191][8199]
	A[8192][0]	A[8192][1]	…	A[8192][8191]	A[8192][8192]	…	A[8192][8199]
	.	.	.	.	.	.	.
	A[8199][0]	A[8199][1]	…	A[8199][8191]	A[8199][8192]	…	A[8192][8199]

The coloured section will be executed in the first step.

Thread 0	Thread 1	…	Thread 8191	Thread 0	…	Thread 8
A[0][0]	[0][1]	…	A[0][8191]	A[0][8192]	…	A[0][8199]
A[1][0]	A[1][1]	…	A[1][8191]	A[1][8192]	…	A[1][8199]
.	.	.	.	.	.	.
.	.	.	.	.	.	.
A[8191][0]	A[8191][1]	…	A[8191][8191]	A[8191][8192]	…	A[8191][8199]
A[8192][0]	A[8192][1]	…	A[8192][8191]	A[8192][8192]	…	A[8192][8199]
.	.	.	.	.	.	.
A[8199][0]	A[8199][1]	…	A[8199][8191]	A[8199][8192]	…	A[8192][8199]

The coloured section will be executed in the next step.

Output (Optimizes code vs Non - Optimized come with same values of SIZE) : -

```
vaibhavt@gpu3:~$ nvcc -g -G -arch=sm_61 -std=c++11 22111064-prob1.cu -o 22111064-prob1
vaibhavt@gpu3:~$ ./22111064-prob1
Serial code on CPU: 4.16626e+08 GFLOPS; Time = 32278.3 msec
No differences found between base and test versions
Kernel 1 on GPU: 3.57942e+09 GFLOPS; Time = 3757.03 msec
No differences found between base and test versions
Kernel 2 on GPU: 5.73132e+09 GFLOPS; Time = 2346.4 msec
vaibhavt@gpu3:~$
vaibhavt@gpu3:~$ ./22111064-prob1
Serial code on CPU: 4.14769e+08 GFLOPS; Time = 32422.9 msec
No differences found between base and test versions
Kernel 1 on GPU: 3.56169e+09 GFLOPS; Time = 3775.74 msec
No differences found between base and test versions
Kernel 2 on GPU: 5.5148e+09 GFLOPS; Time = 2438.53 msec
vaibhavt@gpu3:~$
vaibhavt@gpu3:~$ ./22111064-prob1
Serial code on CPU: 4.1494e+08 GFLOPS; Time = 32409.5 msec
No differences found between base and test versions
Kernel 1 on GPU: 3.49881e+09 GFLOPS; Time = 3843.59 msec
No differences found between base and test versions
Kernel 2 on GPU: 5.50698e+09 GFLOPS; Time = 2441.99 msec
vaibhavt@gpu3:~$
```

Time Taken : -

Serial version : 32370.2 ms

CUDA code (Non-optimised) : 3792.12 ms

CUDA code (Optimised) : 2409 ms

Speedup (Compared to serial version) :-

Serial version : 1

CUDA code (Non-optimised) : 8.5

CUDA code (Optimised) : 13.4

Output (Optimizes code vs Non - Optimized come with different values of SIZE) : -

```
vaibhavt@gpu3:~$ nvcc -g -G -arch=sm_61 -std=c++11 22111064-prob1.cu -o 22111064-prob1
vaibhavt@gpu3:~$ ./22111064-prob1
Serial code on CPU: 3.99383e+08 GFLOPS; Time = 33639.1 msec
No differences found between base and test versions
Kernel 1 on GPU: 3.59587e+09 GFLOPS; Time = 3736.19 msec
No differences found between base and test versions
Kernel 2 on GPU: 5.42317e+09 GFLOPS; Time = 2479.73 msec
vaibhavt@gpu3:~$
vaibhavt@gpu3:~$ ./22111064-prob1
Serial code on CPU: 4.22011e+08 GFLOPS; Time = 31835.4 msec
No differences found between base and test versions
Kernel 1 on GPU: 4.77073e+09 GFLOPS; Time = 2816.11 msec
No differences found between base and test versions
Kernel 2 on GPU: 7.61947e+09 GFLOPS; Time = 1764.95 msec
vaibhavt@gpu3:~$
vaibhavt@gpu3:~$ ./22111064-prob1
Serial code on CPU: 4.21661e+08 GFLOPS; Time = 31861.8 msec
No differences found between base and test versions
Kernel 1 on GPU: 4.58441e+09 GFLOPS; Time = 2930.56 msec
No differences found between base and test versions
Kernel 2 on GPU: 7.59237e+09 GFLOPS; Time = 1771.25 msec
vaibhavt@gpu3:~$
```

Time Taken : -

Serial version : 32445.43 ms

CUDA code (Non-optimised) : 3161 ms

CUDA code (Optimised) : 2005.31 ms

Speedup (Compared to serial version) :-

Serial version : 1

CUDA code (Non-optimised) : 10.3

CUDA code (Optimised) : 16.2

Answer 2 : -

The CUDA version of the given serial code is executed as-

Number of threads per block = 256. This is because at most 1024 threads per block can be executed and of threads per block as 256 is taken.
So, the block dimension = 16. (16 * 16 = 256).

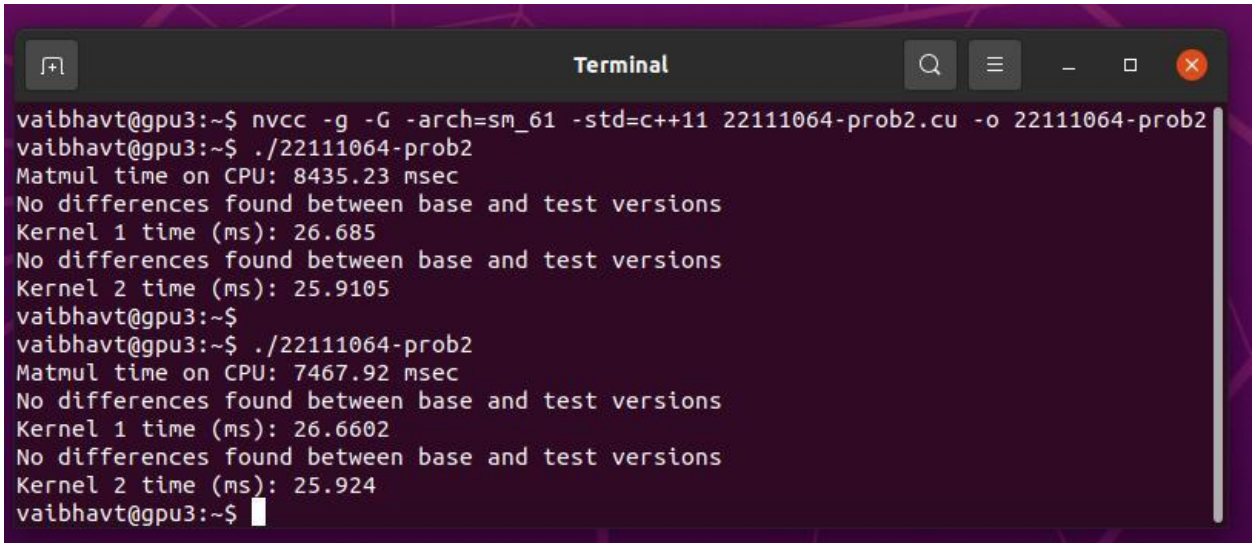
Total number of blocks in the grid = (Number of rows in 2D array) / (Number of threads per block).
= ceil (SIZE / 256)

Here, Each (i , j)th thread will access the ith row of matrix A and the jth column of matrix B and store it at location C[i][j];.

The optimized version of the above CUDA code uses the concept of tiling.
In this, a chunk of data is already stored in the shared memory and it exploits the locality of data for computation.

Compile Command : nvcc -g -G -arch=sm_61 -std=c++11 22111064-prob1.cu -o 22111064-prob2
Run Command : ./22111064-prob2

Output (When N = 1 << 10 and Block/Tile Size = 8) :-



Serial version : 7951.575 ms
CUDA version : 26.673 ms
CUDA version (with tiling) : 25.92 ms

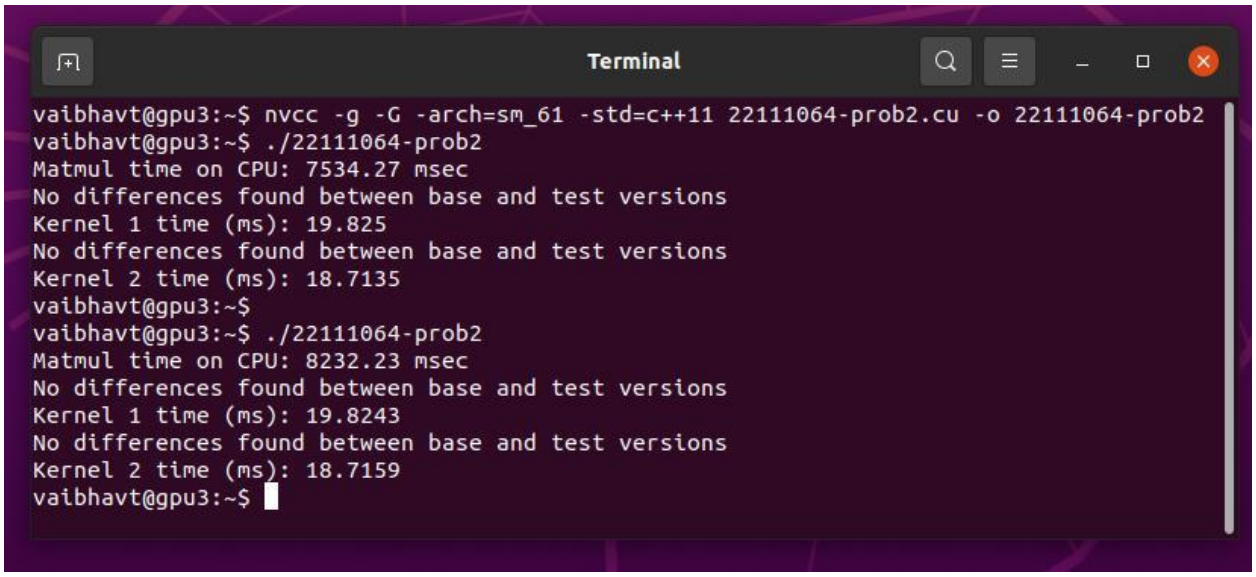
Speed up (Compared to serial version) : -

Serial version : 1

CUDA version : 298.1

CUDA version (with tiling) : 306.77

Output (When N = 1 << 10 and Block/Tile Size = 16) :-



Serial version : 7883.25 ms

CUDA version : 19.82 ms

CUDA version (with tiling) : 18.71 ms

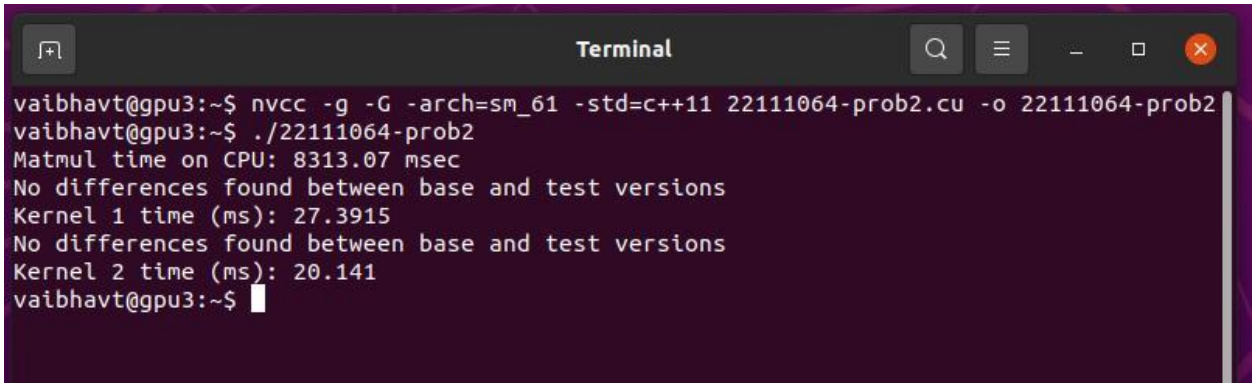
Speed up (Compared to serial version) : -

Serial version : 1

CUDA version : 397.8

CUDA version (with tiling) : 418.7

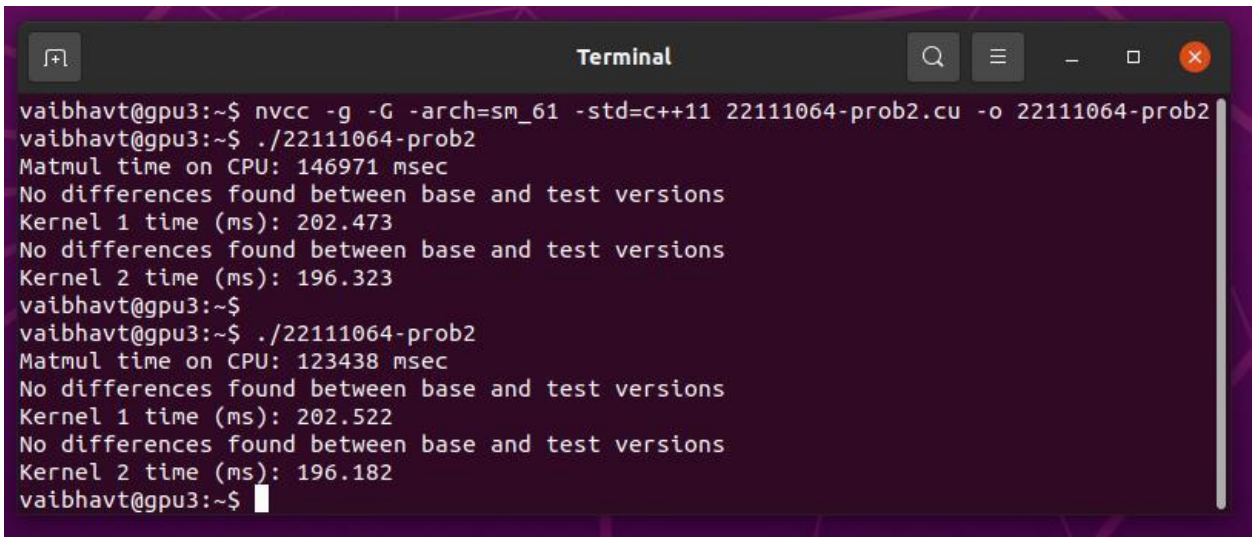
Output (When N = 1 << 10 and Block/Tile Size = 32) :-



Serial version : 8313.07 ms
CUDA version : 27.3915 ms
CUDA version (with tiling) :- 20.141 ms

Speed up (Compared to serial version) : -
Serial version : 1
CUDA version : 303.5
CUDA version (with tiling) : 412.7

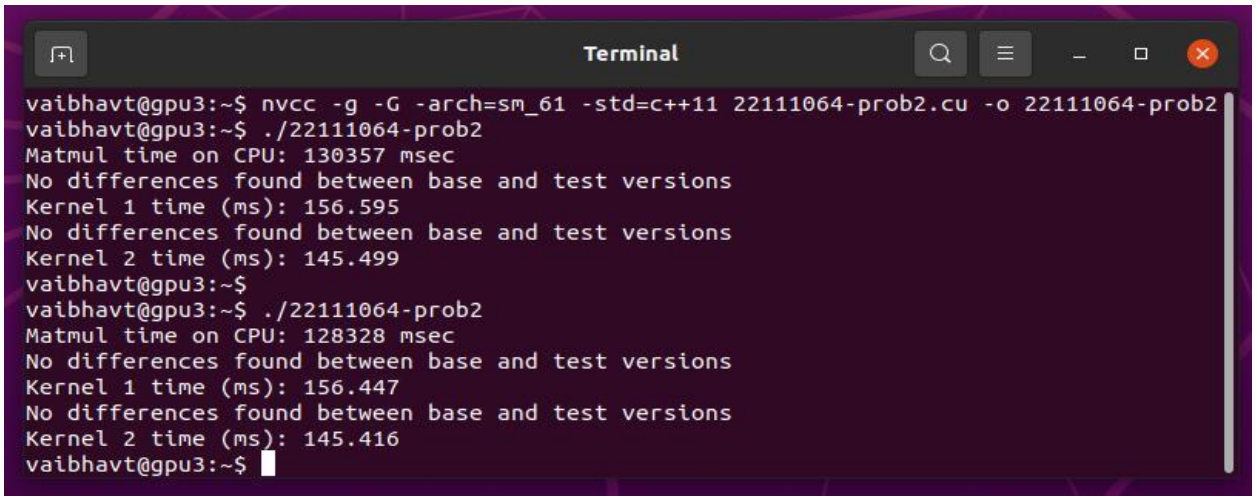
Output (When N = 1 << 11 and Block/Tile Size = 8) :-



Serial version : 135204.5 ms
CUDA version : 202.5 ms
CUDA version (with tiling) : 196.25 ms

Speed up (Compared to serial version) : -
Serial version : 1
CUDA version : 667.7
CUDA version (with tiling) : 688.93

Output (When N = 1 << 11 and Block/Tile Size = 16) :-



Serial version : 129342.5 ms

CUDA version : 156.5 ms

CUDA version (with tiling) : 145.5 ms

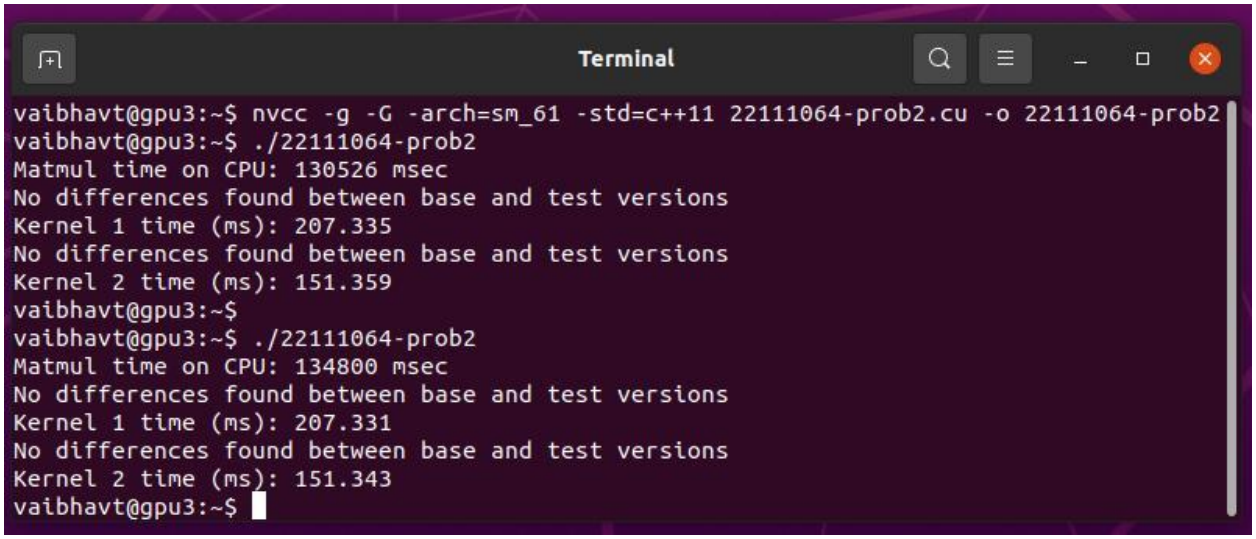
Speed up (Compared to serial version) : -

Serial version : 1

CUDA version : 826.5

CUDA version (with tiling) : 889

Output (When N = 1 << 11 and Block/Tile Size = 32) :-



Serial version : 132663 ms

CUDA version : 207.333 ms

CUDA version (with tiling) : 151.351 ms

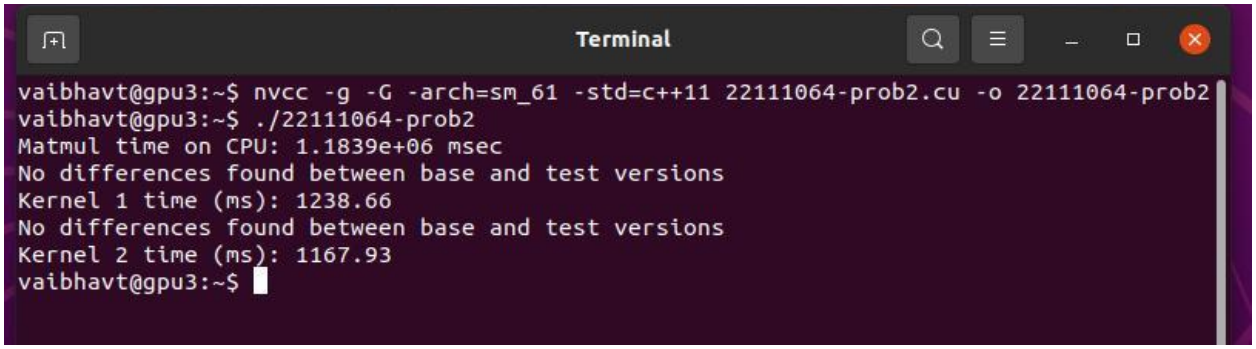
Speed up (Compared to serial version) : -

Serial version :- 1

CUDA version : 639.9

CUDA version (with tiling) : 876.5

Output (When N = 1 << 12 and Block/Tile Size = 16) :-



Serial version : - ~ 1200000 ms (20 min)

CUDA version : 27.3915 ms

CUDA version (with tiling) : 20.141 ms

Speed up (Compared to serial version) : -

Serial version : 1

CUDA version : 968.7

CUDA version (with tiling) : 1027.4

Answer 3 : -

Compile Command : nvcc -g -G -arch=sm_61 -std=c++11 22111064-prob1.cu -o 22111064-prob3

Run Command : ./22111064-prob3

Output (When N << 10 and Block Size = 16): -

```
! nvcc -g -G -arch=sm_61 -std=c++11 22111064-prob3.cu -o 22111064-prob3
```

```
! nvprof ./22111064-prob3
```

```
Matmul time on CPU: 11702 msec
==345== NVPROF is profiling process 345, command: ./22111064-prob3
No differences found between base and test versions
Kernel 1 time (ms): 128.213
No differences found between base and test versions
Kernel 2 time (ms): 79.3074
No differences found between base and test versions
Kernel 3 time (ms): 77.3753
No differences found between base and test versions
Kernel 4 time (ms): 56.6701
==345== Profiling application: ./22111064-prob3
==345== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   35.99%  122.52ms         1  122.52ms  122.52ms  122.52ms  opt_matmul_prob2(unsigned long const *, unsigned long const *, unsigned long*)
                   22.71%  77.321ms         1  77.321ms  77.321ms  77.321ms  zerocopy(unsigned long const *, unsigned long const *, unsigned long*)
                   22.68%  77.198ms         1  77.198ms  77.198ms  77.198ms  pinned(unsigned long const *, unsigned long const *, unsigned long*)
                   16.63%  56.614ms         1  56.614ms  56.614ms  56.614ms  uvm(unsigned long const *, unsigned long const *, unsigned long*)
                    1.38%  4.6996ms         4  1.1749ms  681.27us  1.7102ms  [CUDA memcpy HtoD]
                    0.60%  2.0472ms         2  1.0236ms  636.99us  1.4102ms  [CUDA memcpy DtoH]
      API calls:   54.93%  479.62ms         4  119.91ms  91.963us  479.28ms  cudaMalloc
                   38.23%  333.75ms         4  83.436ms  56.622ms  122.59ms  cudaDeviceSynchronize
                    2.33%  20.321ms         3  6.7736ms  10.596us  20.278ms  cudaMallocManaged
                    2.14%  18.728ms         6  3.1213ms  2.4072ms  3.5110ms  cudaHostAlloc
                    0.90%  7.8363ms         6  1.3061ms  1.2359ms  1.3417ms  cudaFreeHost
                    0.87%  7.6388ms         6  1.2731ms  663.77us  1.9004ms  cudaMemcpy
                    0.33%  2.8814ms         9  320.16us     442ns  1.6623ms  cudaFree
                    0.10%  857.01us         1  857.01us  857.01us  857.01us  cuDeviceGetPCIBusId
```

```
                   22.71%  77.321ms         1  77.321ms  77.321ms  77.321ms  zerocopy(unsigned long const *, unsigned long const *, unsig
                   22.68%  77.198ms         1  77.198ms  77.198ms  77.198ms  pinned(unsigned long const *, unsigned long const *, unsigned long*)
                   16.63%  56.614ms         1  56.614ms  56.614ms  56.614ms  uvm(unsigned long const *, unsigned long const *, unsigned long*)
                    1.38%  4.6996ms         4  1.1749ms  681.27us  1.7102ms  [CUDA memcpy HtoD]
                    0.60%  2.0472ms         2  1.0236ms  636.99us  1.4102ms  [CUDA memcpy DtoH]
      API calls:   54.93%  479.62ms         4  119.91ms  91.963us  479.28ms  cudaMalloc
                   38.23%  333.75ms         4  83.436ms  56.622ms  122.59ms  cudaDeviceSynchronize
                    2.33%  20.321ms         3  6.7736ms  10.596us  20.278ms  cudaMallocManaged
                    2.14%  18.728ms         6  3.1213ms  2.4072ms  3.5110ms  cudaHostAlloc
                    0.90%  7.8363ms         6  1.3061ms  1.2359ms  1.3417ms  cudaFreeHost
                    0.87%  7.6388ms         6  1.2731ms  663.77us  1.9004ms  cudaMemcpy
                    0.33%  2.8814ms         9  320.16us     442ns  1.6623ms  cudaFree
                    0.10%  857.01us         1  857.01us  857.01us  857.01us  cuDeviceGetPCIBusId
                    0.08%  670.84us         4  167.71us  4.9020us  654.71us  cudaEventSynchronize
                    0.04%  347.84us         1  347.84us  347.84us  347.84us  cuDeviceTotalMem
                    0.02%  164.75us       101  1.6310us     129ns  79.211us  cuDeviceGetAttribute
                    0.01%  122.83us         4  30.706us  27.605us  35.332us  cudaLaunchKernel
                    0.01%  82.212us         8  10.276us  4.5110us  21.361us  cudaEventRecord
                    0.00%  39.707us         4  9.9260us  6.7670us  14.432us  cudaEventElapsedTime
                    0.00%  28.728us         1  28.728us  28.728us  28.728us  cuDeviceGetName
                    0.00%  9.9350us         2  4.9670us     713ns  9.2220us  cudaEventCreate
                    0.00%  5.2570us         3  1.7520us     451ns  3.7280us  cudaHostGetDevicePointer
                    0.00%  1.4120us         3     470ns     209ns     892ns  cuDeviceGetCount
                    0.00%  1.4070us         2     703ns     210ns  1.1970us  cuDeviceGet
                    0.00%     277ns         1     277ns     277ns     277ns  cuDeviceGetUuid

==345== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
     881  27.896KB  4.0000KB  900.00KB  24.00000MB  3.814656ms  Host To Device
      48  170.67KB  4.0000KB  0.9961MB  8.000000MB  718.5230us  Device To Host
      69         -         -         -           -  13.91817ms  Gpu page fault groups
Total CPU Page faults: 96
```

Time Taken : -

Serial version : 11702 ms
Prob2 version : 128.213 ms
Pinned version : 79.3074 ms
ZeroCopy version : 77.3753 ms
Unified Virtual Memory version: 56.6701 ms

Speed up (Compared to serial version) :-

Serial version : 1
Prob2 version : 91.3
Pinned version : 147.
ZeroCopy version :- 151.24
Unified Virtual Memory version : 206.5

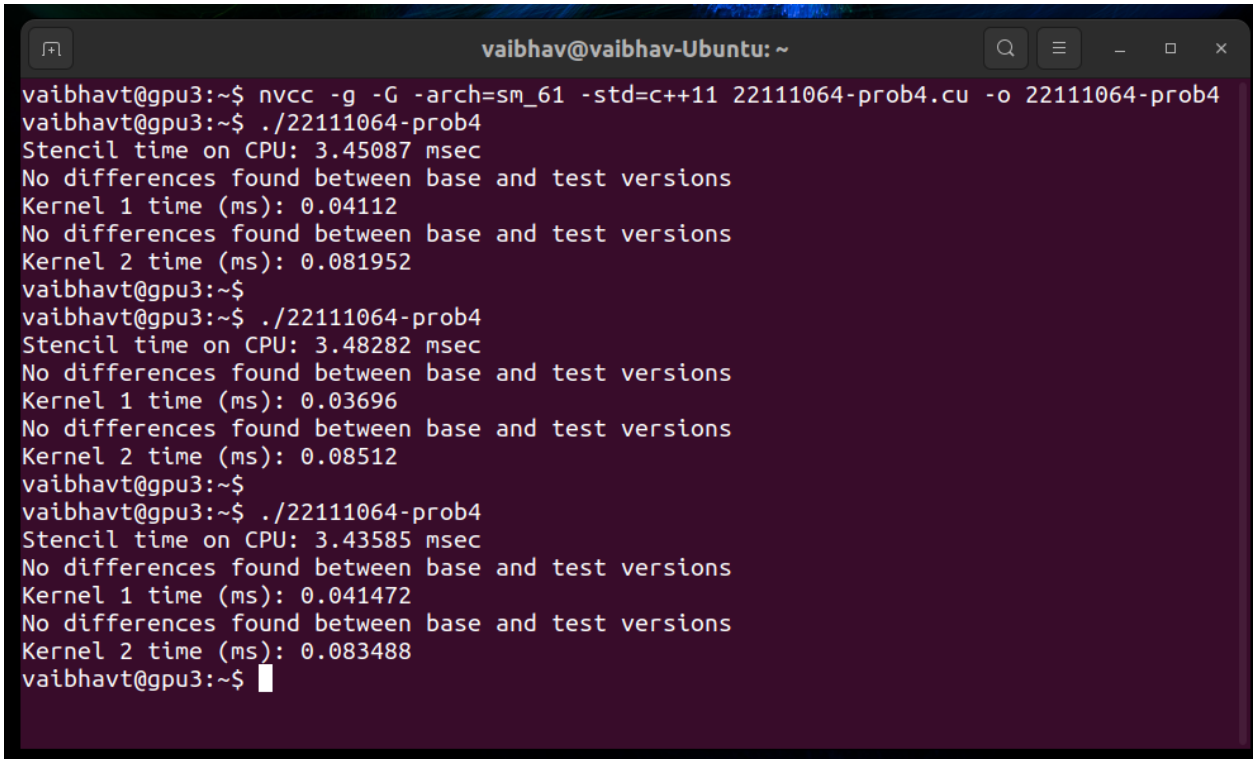
- Here, Zero copy version is faster than the kernel from problem 2 because in that version, copying from and to the memory is done twice but in zero copy, the data transfer happens only once.
- Pinned memory version is faster than Zero copy because the data transfer happens faster in pinned memory as compared to zero copy.
- Unified virtual memory is fastest among all because very few data transfer between host and device and also the page faults are low.

Answer 4 : -

Compile Command : nvcc -g -G -arch=sm_61 -std=c++11 22111064-prob1.cu -o 22111064-prob4

Run Command : ./22111064-prob4

Output: -



Time Taken : -

Serial version : 3.4565 ms
CUDA version (Non Tiling) : 0.0544 ms
CUDA version (Tiling) : 0.08352 ms

Speed up (Compared to serial version) :-

Serial version : 1
CUDA version (Non Tiling) : 53.67
CUDA version (Tiling) : 41.39

The challenging part in tiling this program is, it will have so many conditions that causes the threads divergence and hence gives poor performance compared to the non-tilled version.