# Final Project: Optimised matrix multiplication

## Overview:

In this project I have tried to explore two special features of Cuda. They are:

1. Unified Memory
2. Cuda Streams

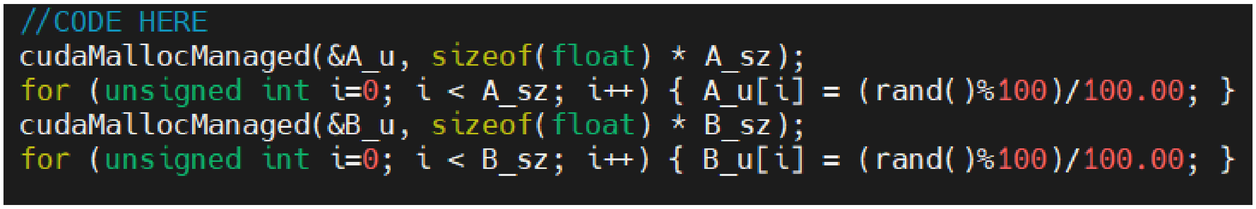
For which I have considered one of the popular Cuda Algorithm Matrix multiplication. I have used the nvprof - which presents an overview of the GPU kernels and memory copies in our application, to demonstrate the differences between Unified memory, Cuda Streams variation of Matrix multiplication.

## Technical Description:

### Unified Memory:

Unified Memory is a single memory address space accessible from any processor in a system. As compared to pre pascal unified memory, with the post pascal systems we don't have to allocate memory for host and device i.e. CPU and GPUs separately. Instead we could store in a single location.

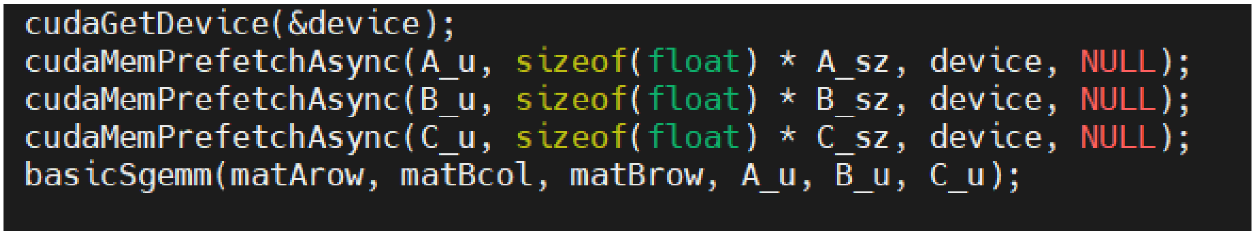
As opposed to cudaMalloc() we use cudaMallocManaged() to allocate memory. In my implementation I did the memory allocation as follows:



```
//CODE HERE
cudaMallocManaged(&A_u, sizeof(float) * A_sz);
for (unsigned int i=0; i < A_sz; i++) { A_u[i] = (rand()%100)/100.00; }
cudaMallocManaged(&B_u, sizeof(float) * B_sz);
for (unsigned int i=0; i < B_sz; i++) { B_u[i] = (rand()%100)/100.00; }
```

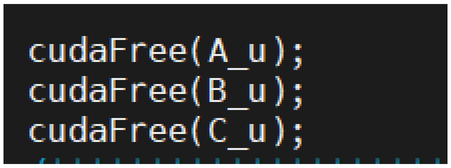
Since only the memory management is different in comparison to matrix multiplication, the rest of the kernel code could be similar.

Also I have used the cudaMemPrefetchAsync() before executing the kernel method to retrieve the data from the unified memory to destination device in our case to GPU.



```
cudaGetDevice(&device);
cudaMemPrefetchAsync(A_u, sizeof(float) * A_sz, device, NULL);
cudaMemPrefetchAsync(B_u, sizeof(float) * B_sz, device, NULL);
cudaMemPrefetchAsync(C_u, sizeof(float) * C_sz, device, NULL);
basicSgemm(matArow, matBcol, matBrow, A_u, B_u, C_u);
```

Then after executing the kernel I cleared the memory using cudaFree().



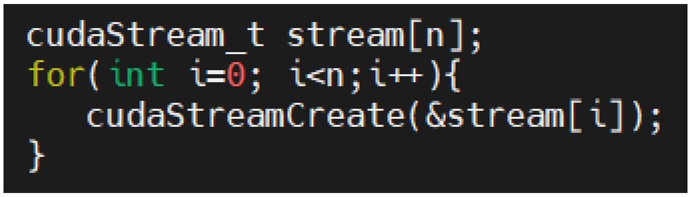
```
cudaFree(A_u);
cudaFree(B_u);
cudaFree(C_u);
```

## Streams:

In my second implementation I have tried a concurrency optimization technique Cuda Streams. Streams gives us a flexibility of forming multi ques as opposed to the traditional serial queues implementation in Cuda programming.

So In my approach of optimizing matrix multiplication I have made use of 4 streams implemented on Memory allocation and executing kernel using unified memory allocation.

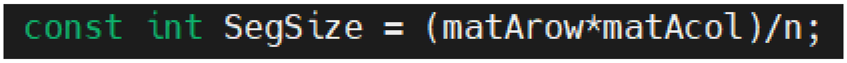
The following the initialization of cuda Stream:



```
cudaStream_t stream[n];
for( int i=0; i<n;i++){
   cudaStreamCreate(&stream[i]);
}
```

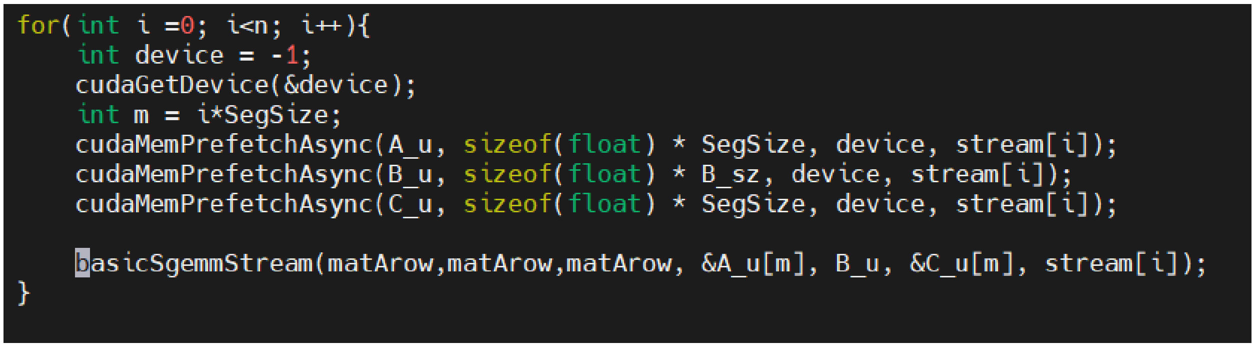
Where n = 4.

Then I have divided the memory into segments of 4 using:



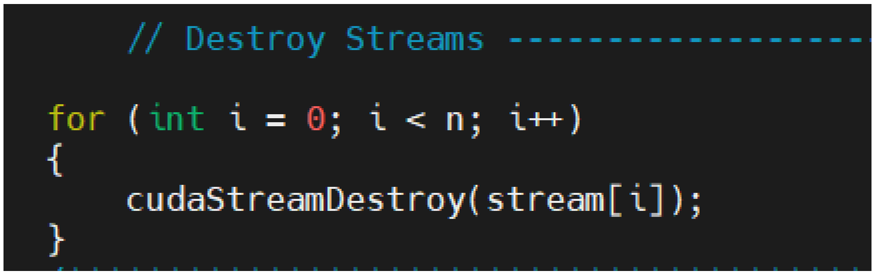
```
const int SegSize = (matArow*matAcol)/n;
```

Then I have implemented the streams as follows:



```
for( int i =0; i<n; i++){
    int device = -1;
    cudaGetDevice(&device);
    int m = i*SegSize;
    cudaMemPrefetchAsync(A_u, sizeof(float) * SegSize, device, stream[i]);
    cudaMemPrefetchAsync(B_u, sizeof(float) * B_sz, device, stream[i]);
    cudaMemPrefetchAsync(C_u, sizeof(float) * SegSize, device, stream[i]);

    basicSgemmStream(matArow,matArow,matArow, &A_u[m], B_u, &C_u[m], stream[i]);
}
```

In the end after kernel execution, I have cleared the stream with the following:



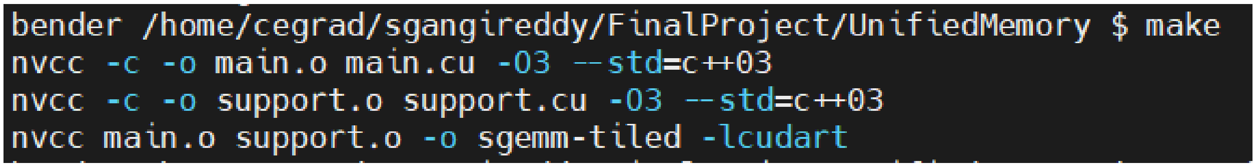
```
      // Destroy Streams ------------------

  for (int i = 0; i < n; i++)
  {
      cudaStreamDestroy(stream[i]);
  }
```

## Status of the Project:

Both the programs are running well without any errors and exceptions. However there were few issues while compiling and optimizing matrix multiplication using streams.
Possible reasons are:

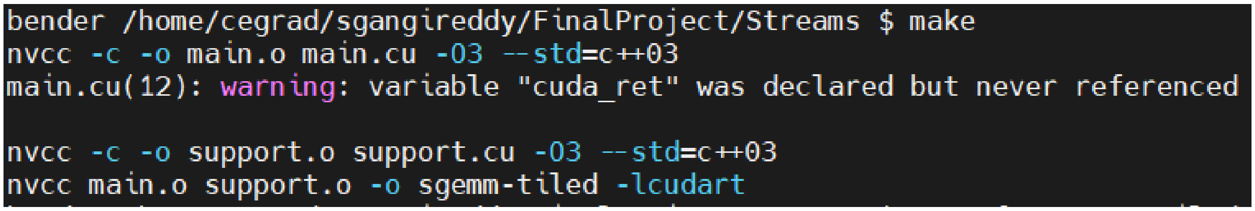
1. A complete optimization hasn't been achieved because of the use case of Streams with the matrix multiplication kernel. A Possible reason is with not passing a transposed B matrix.
2. Using cudaDeviceSynchronize(), resolved some memory buffer issues with streams.

Unified memory:



```
bender /home/cegrad/sgangireddy/FinalProject/UnifiedMemory $ make
nvcc -c -o main.o main.cu -O3 --std=c++03
nvcc -c -o support.o support.cu -O3 --std=c++03
nvcc main.o support.o -o sgemm-tiled -lcudart
```

Streams:

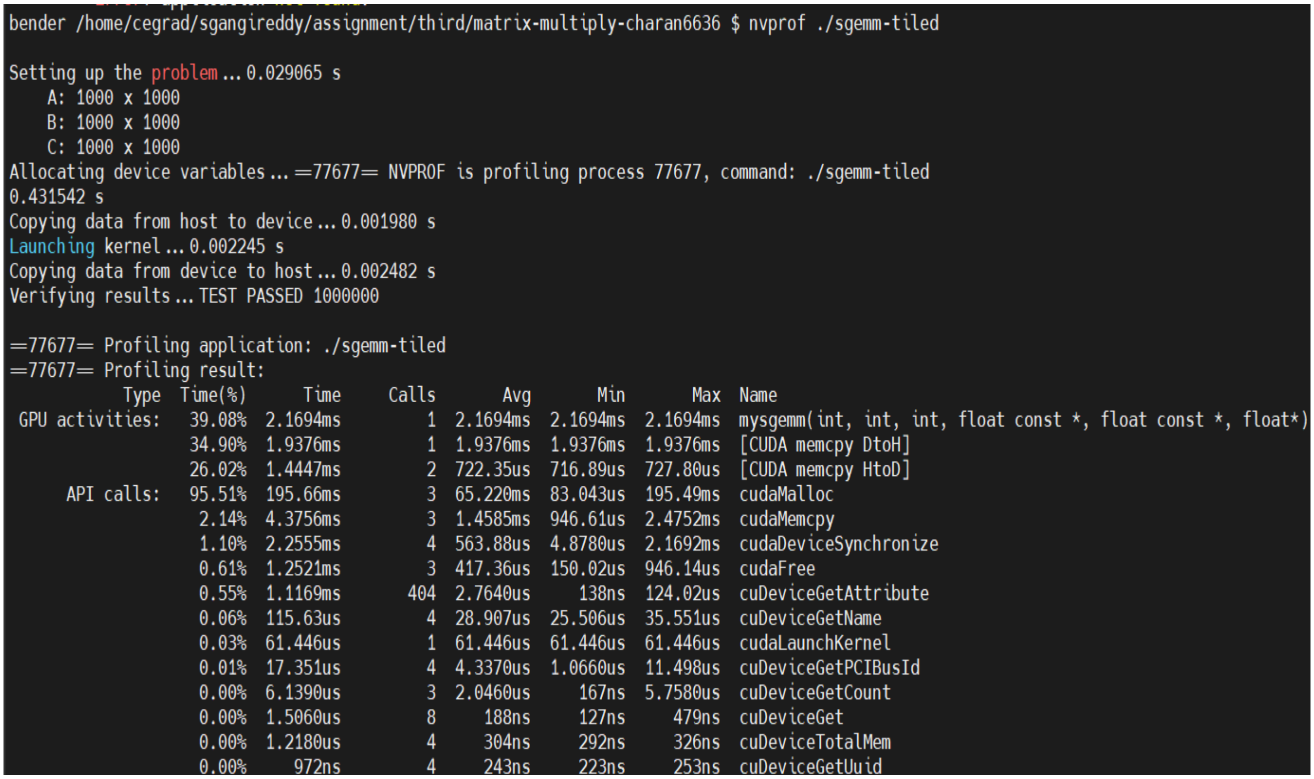


```
bender /home/cegrad/sgangireddy/FinalProject/Streams $ make
nvcc -c -o main.o main.cu -O3 --std=c++03
main.cu(12): warning: variable "cuda_ret" was declared but never referenced

nvcc -c -o support.o support.cu -O3 --std=c++03
nvcc main.o support.o -o sgemm-tiled -lcudart
```

## Evaluation/Results:

Using nvprof I have generated the overall picture of execution in each algorithm.

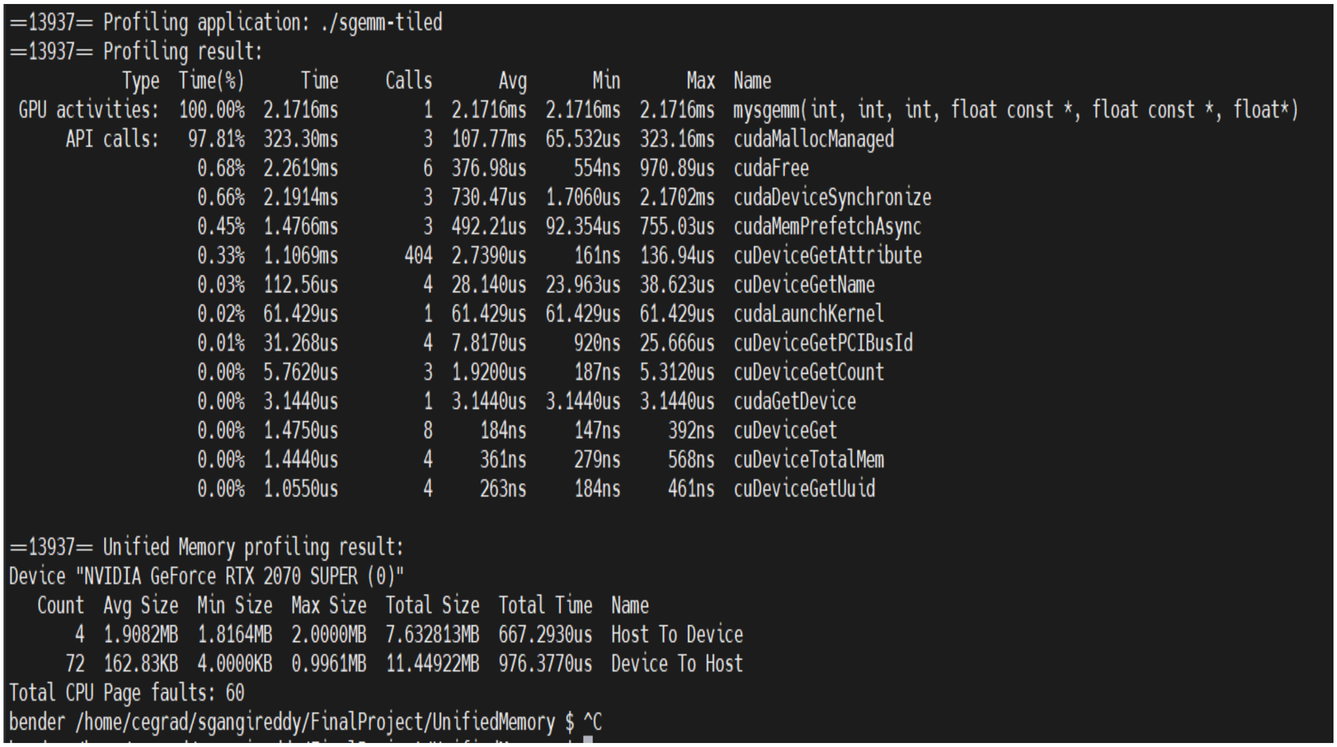
## Profiling for naive matrix multiplication:



```
bender /home/cegrad/sgangireddy/assignment/third/matrix-multiply-charan6636 $ nvprof ./sgemm-tiled

Setting up the problem... 0.029065 s
    A: 1000 x 1000
    B: 1000 x 1000
    C: 1000 x 1000
Allocating device variables... ==77677== NVPROF is profiling process 77677, command: ./sgemm-tiled
0.431542 s
Copying data from host to device... 0.001980 s
Launching kernel... 0.002245 s
Copying data from device to host... 0.002482 s
Verifying results... TEST PASSED 1000000

==77677== Profiling application: ./sgemm-tiled
==77677== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   39.08%  2.1694ms         1  2.1694ms  2.1694ms  2.1694ms  mysgemm(int, int, int, float const *, float const *, float*)
                   34.90%  1.9376ms         1  1.9376ms  1.9376ms  1.9376ms  [CUDA memcpy DtoH]
                   26.02%  1.4447ms         2  722.35us  716.89us  727.80us  [CUDA memcpy HtoD]
      API calls:   95.51%  195.66ms         3  65.220ms  83.043us  195.49ms  cudaMalloc
                    2.14%  4.3756ms         3  1.4585ms  946.61us  2.4752ms  cudaMemcpy
                    1.10%  2.2555ms         4  563.88us  4.8780us  2.1692ms  cudaDeviceSynchronize
                    0.61%  1.2521ms         3  417.36us  150.02us  946.14us  cudaFree
                    0.55%  1.1169ms       404  2.7640us     138ns  124.02us  cuDeviceGetAttribute
                    0.06%  115.63us         4  28.907us  25.506us  35.551us  cuDeviceGetName
                    0.03%  61.446us         1  61.446us  61.446us  61.446us  cudaLaunchKernel
                    0.01%  17.351us         4  4.3370us  1.0660us  11.498us  cuDeviceGetPCIBusId
                    0.00%  6.1390us         3  2.0460us     167ns  5.7580us  cuDeviceGetCount
                    0.00%  1.5060us         8     188ns     127ns     479ns  cuDeviceGet
                    0.00%  1.2180us         4     304ns     292ns     326ns  cuDeviceTotalMem
                    0.00%     972ns         4     243ns     223ns     253ns  cuDeviceGetUuid
```

Where we could see that, it took “2.1694”secs for executing the kernel method and also we could observe time taken to execute various methods in the program.

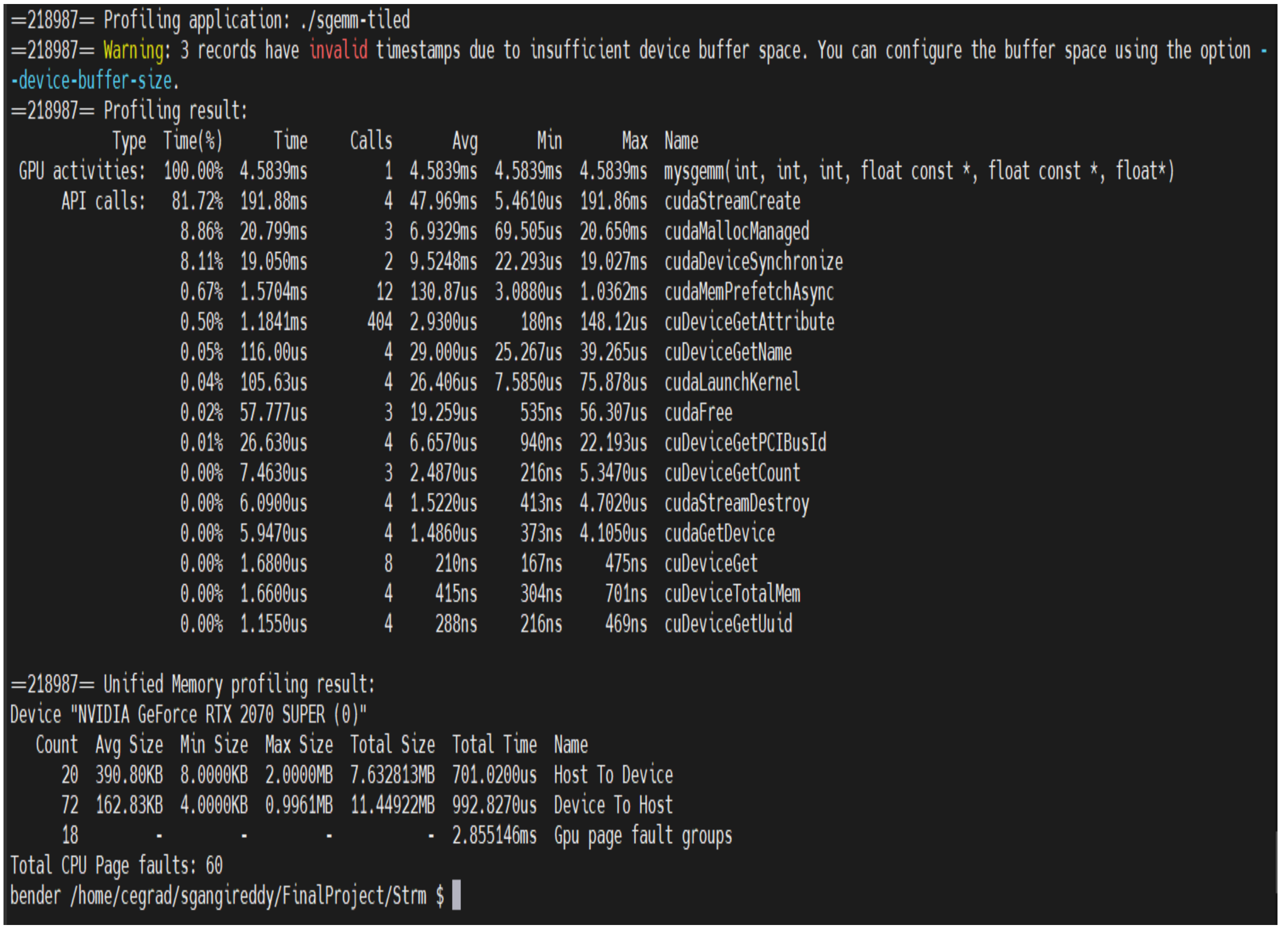
## Profiling for Unified Memory matrix multiplication:



```
==13937== Profiling application: ./sgemm-tiled
==13937== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  2.1716ms         1  2.1716ms  2.1716ms  2.1716ms  mysgemm(int, int, int, float const *, float const *, float*)
      API calls:   97.81%  323.30ms         3  107.77ms  65.532us  323.16ms  cudaMallocManaged
                    0.68%  2.2619ms         6  376.98us     554ns  970.89us  cudaFree
                    0.66%  2.1914ms         3  730.47us  1.7060us  2.1702ms  cudaDeviceSynchronize
                    0.45%  1.4766ms         3  492.21us  92.354us  755.03us  cudaMemPrefetchAsync
                    0.33%  1.1069ms       404  2.7390us     161ns  136.94us  cuDeviceGetAttribute
                    0.03%  112.56us         4  28.140us  23.963us  38.623us  cuDeviceGetName
                    0.02%  61.429us         1  61.429us  61.429us  61.429us  cudaLaunchKernel
                    0.01%  31.268us         4  7.8170us     920ns  25.666us  cuDeviceGetPCIBusId
                    0.00%  5.7620us         3  1.9200us     187ns  5.3120us  cuDeviceGetCount
                    0.00%  3.1440us         1  3.1440us  3.1440us  3.1440us  cudaGetDevice
                    0.00%  1.4750us         8     184ns     147ns     392ns  cuDeviceGet
                    0.00%  1.4440us         4     361ns     279ns     568ns  cuDeviceTotalMem
                    0.00%  1.0550us         4     263ns     184ns     461ns  cuDeviceGetUuid

==13937== Unified Memory profiling result:
Device "NVIDIA GeForce RTX 2070 SUPER (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
       4  1.9082MB  1.8164MB  2.0000MB  7.632813MB  667.2930us  Host To Device
      72  162.83KB  4.0000KB  0.9961MB  11.44922MB  976.3770us  Device To Host
Total CPU Page faults: 60
bender /home/cegrad/sgangireddy/FinalProject/UnifiedMemory $ ^C
```

By the figure we could see that time taken to for execution is almost similar between matrix multiplication and unified memory as essentially we didn’t try to execute optimization here.

### Profiling for Cuda Streams matrix multiplication:



```
==218987== Profiling application: ./sgemm-tiled
==218987== Warning: 3 records have invalid timestamps due to insufficient device buffer space. You can configure the buffer space using the option -
-device-buffer-size.
==218987== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  4.5839ms         1  4.5839ms  4.5839ms  4.5839ms  mysgemm(int, int, int, float const *, float const *, float*)
      API calls:   81.72%  191.88ms         4  47.969ms  5.4610us  191.86ms  cudaStreamCreate
                    8.86%  20.799ms         3  6.9329ms  69.505us  20.650ms  cudaMallocManaged
                    8.11%  19.050ms         2  9.5248ms  22.293us  19.027ms  cudaDeviceSynchronize
                    0.67%  1.5704ms        12  130.87us  3.0880us  1.0362ms  cudaMemPrefetchAsync
                    0.50%  1.1841ms       404  2.9300us     180ns  148.12us  cuDeviceGetAttribute
                    0.05%  116.00us         4  29.000us  25.267us  39.265us  cuDeviceGetName
                    0.04%  105.63us         4  26.406us  7.5850us  75.878us  cudaLaunchKernel
                    0.02%  57.777us         3  19.259us     535ns  56.307us  cudaFree
                    0.01%  26.630us         4  6.6570us     940ns  22.193us  cuDeviceGetPCIBusId
                    0.00%  7.4630us         3  2.4870us     216ns  5.3470us  cuDeviceGetCount
                    0.00%  6.0900us         4  1.5220us     413ns  4.7020us  cudaStreamDestroy
                    0.00%  5.9470us         4  1.4860us     373ns  4.1050us  cudaGetDevice
                    0.00%  1.6800us         8     210ns     167ns     475ns  cuDeviceGet
                    0.00%  1.6600us         4     415ns     304ns     701ns  cuDeviceTotalMem
                    0.00%  1.1550us         4     288ns     216ns     469ns  cuDeviceGetUuid

==218987== Unified Memory profiling result:
Device "NVIDIA GeForce RTX 2070 SUPER (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      20  390.80KB  8.0000KB  2.0000MB  7.632813MB  701.0200us  Host To Device
      72  162.83KB  4.0000KB  0.9961MB  11.44922MB  992.8270us  Device To Host
      18         -         -         -           -  2.855146ms  Gpu page fault groups
Total CPU Page faults: 60
bender /home/cegrad/sgangireddy/FinalProject/Strm $ 
```

The Matrix multiplication using streams have achieved a “4.5839s” runtime on kernel, which is moderately optimized.

# Compiling and running:

Unified memory:

- cd Unified-Memory/
- Make
- nvprof ./sgemm-tiled (To get profiler details)

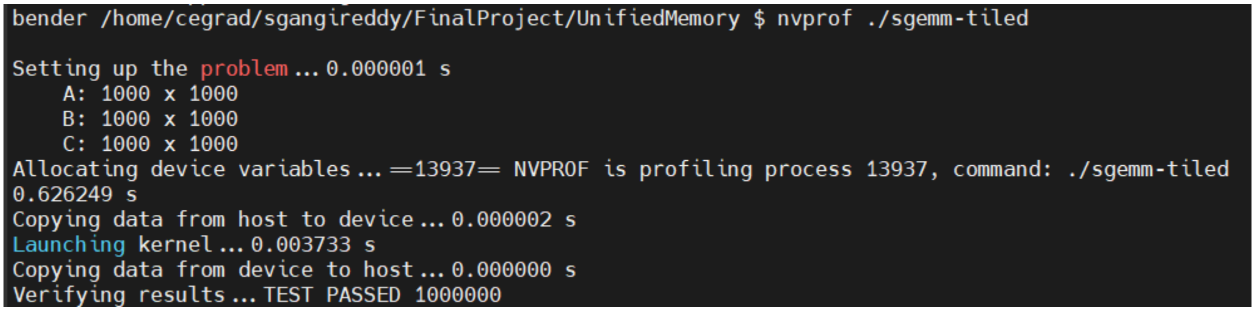
Stream matrix multiplication:

- cd Streams/
- Make

nvprof ./sgemm-tiled (To get profiler details)

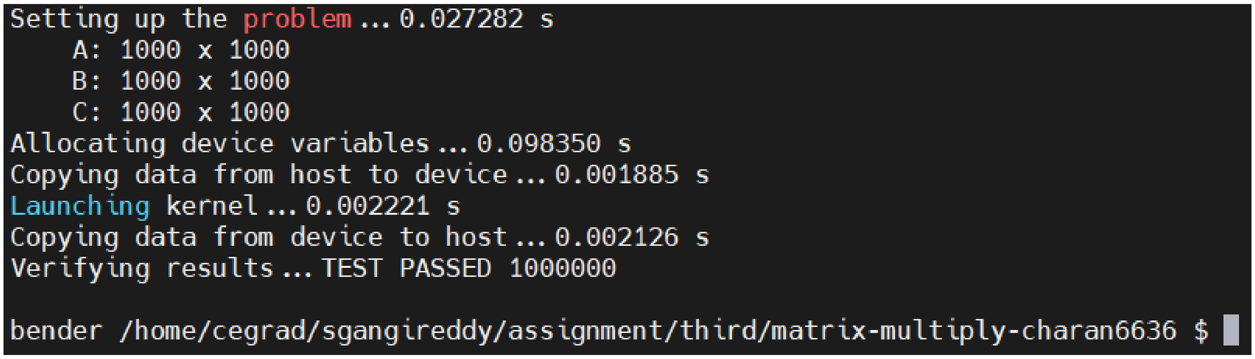
## Results:

Unified memory runtime:



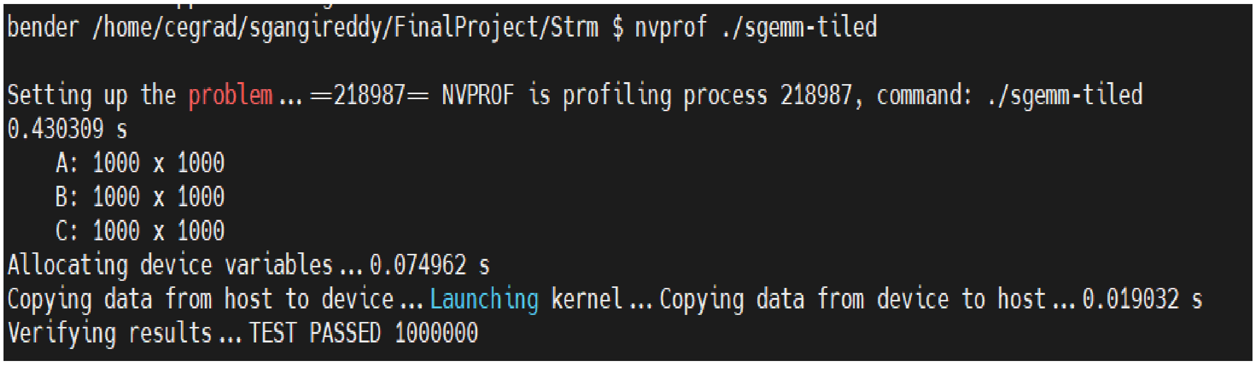
```
bender /home/cegrad/sgangireddy/FinalProject/UnifiedMemory $ nvprof ./sgemm-tiled

Setting up the problem...0.000001 s
    A: 1000 x 1000
    B: 1000 x 1000
    C: 1000 x 1000
Allocating device variables...==13937== NVPROF is profiling process 13937, command: ./sgemm-tiled
0.626249 s
Copying data from host to device...0.000002 s
Launching kernel...0.003733 s
Copying data from device to host...0.000000 s
Verifying results...TEST PASSED 1000000
```

Matrix multiplication runtime:



```
Setting up the problem...0.027282 s
    A: 1000 x 1000
    B: 1000 x 1000
    C: 1000 x 1000
Allocating device variables...0.098350 s
Copying data from host to device...0.001885 s
Launching kernel...0.002221 s
Copying data from device to host...0.002126 s
Verifying results...TEST PASSED 1000000

bender /home/cegrad/sgangireddy/assignment/third/matrix-multiply-charan6636 $
```

Matrix multiplication using streams runtime:



```
bender /home/cegrad/sgangireddy/FinalProject/Strm $ nvprof ./sgemm-tiled

Setting up the problem...==218987== NVPROF is profiling process 218987, command: ./sgemm-tiled
0.430309 s
    A: 1000 x 1000
    B: 1000 x 1000
    C: 1000 x 1000
Allocating device variables...0.074962 s
Copying data from host to device...Launching kernel...Copying data from device to host...0.019032 s
Verifying results...TEST PASSED 1000000
```