Name: Shruti Tulshidas Pangare
NetID: stp8232
High Performance Machine Learning - Homework 4

Part A
Problem 1: Vector add and Coalescing memory access

Solution Q1: Performs vector addition without coalesced memory reads usingvecaddKernel00.cu.

Sbatch (run_vecadd00.sbatch) and Output (h4_q1.out)

```
#!/bin/bash
#SBATCH --job-name=h4_q1
#SBATCH --output=%x.out
#SBATCH --account=ece_gy_9143-2025sp
#SBATCH --partition=n1s8-v100-1  # Using a V100 GPU partition as an example
#SBATCH --cpus-per-task=4
#SBATCH --time=10:00:00
#SBATCH --mem=10GB
#SBATCH --gres=gpu:1

cd /scratch/stp8232/PartA


module load singularity
module load cuda

#Singularity container
CONTAINER=/scratch/work/public/singularity/cuda12.1.1-cudnn8.9.0-devel-ubuntu22.04.2.sif

# Run all commands in a single Singularity session
singularity exec --nv ${CONTAINER} bash -c "
cd /scratch/stp8232/PartA
make clean
make 2>/dev/null
make
./vecadd00 500
./vecadd00 1000
./vecadd00 2000
"
```

```
rm -f vecadd00 matmult00 vecadd01 matmult01 *.o
/usr/local/cuda/bin/nvcc vecaddKernel00.cu -c -o vecaddKernel00.o -O3
/usr/local/cuda/bin/nvcc timer.cu -c -o timer.o -O3
/usr/local/cuda/bin/nvcc vecadd.cu vecaddKernel00.o -o vecadd00 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3
/usr/local/cuda/bin/nvcc matmultKernel00.cu -c -o matmultKernel00.o -O3
/usr/local/cuda/bin/nvcc matmult.cu matmultKernel00.o -o matmult00 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3
/usr/local/cuda/bin/nvcc vecaddKernel01.cu -c -o vecaddKernel01.o -O3
/usr/local/cuda/bin/nvcc vecadd.cu vecaddKernel01.o -o vecadd01 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3
/usr/local/cuda/bin/nvcc matmultKernel01.cu -c -o matmultKernel01.o -O3  -DFOOTPRINT_SIZE=32
/usr/local/cuda/bin/nvcc matmult.cu matmultKernel01.o -o matmult01 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3  -
DFOOTPRINT_SIZE=32
make: Nothing to be done for 'all'.
Total vector size: 3840000
Time: 0.000307 (sec), GFlopsS: 12.504757, GBytesS: 150.057087
Test PASSED
Total vector size: 7680000
Time: 0.000545 (sec), GFlopsS: 14.091100, GBytesS: 169.093201
Test PASSED
Total vector size: 15360000
Time: 0.001066 (sec), GFlopsS: 14.409418, GBytesS: 172.913020
Test PASSED
```

Solution 2: implements coalesced memory access to improve performance
uses vecaddKernel01.cu

```
#include "vecaddKernel.h"

__global__ void AddVectors(const float* A, const float* B, float* C, int N)
{
   int offset = blockDim.x * gridDim.x;
   int index;

   for (int i = 0; i < N; i++) {
      index =  offset * i + (blockIdx.x * blockDim.x + threadIdx.x);
      C[index] = A[index] + B[index];
   }
}
```

Sbatch (run_vecadd01.sbatch) and Output (h4_Q2.out)

```
#!/bin/bash
#SBATCH --job-name=h4_Q2
#SBATCH --output=%x.out
#SBATCH --account=ece_gy_9143-2025sp
#SBATCH --partition=n1s8-v100-1  # Using a V100 GPU partition as an example
#SBATCH --cpus-per-task=4
#SBATCH --time=10:00:00
#SBATCH --mem=10GB
#SBATCH --gres=gpu:1

cd /scratch/stp8232/PartA


module load singularity
module load cuda

#Singularity container
CONTAINER=/scratch/work/public/singularity/cuda12.1.1-cudnn8.9.0-devel-ubuntu22.04.2.sif

# Run all commands in a single Singularity session
singularity exec --nv ${CONTAINER} bash -c "
cd /scratch/stp8232/PartA
make clean
make 2>/dev/null
make
./vecadd01 500
./vecadd01 1000
./vecadd01 2000
"
```

```
rm -f vecadd00 matmult00 vecadd01 matmult01 *.o
/usr/local/cuda/bin/nvcc vecaddKernel00.cu -c -o vecaddKernel00.o -O3
/usr/local/cuda/bin/nvcc timer.cu -c -o timer.o -O3
/usr/local/cuda/bin/nvcc vecadd.cu vecaddKernel00.o -o vecadd00 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3
/usr/local/cuda/bin/nvcc matmultKernel00.cu -c -o matmultKernel00.o -O3
/usr/local/cuda/bin/nvcc matmult.cu matmultKernel00.o -o matmult00 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3
/usr/local/cuda/bin/nvcc vecaddKernel01.cu -c -o vecaddKernel01.o -O3
/usr/local/cuda/bin/nvcc vecadd.cu vecaddKernel01.o -o vecadd01 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3
make: *** No rule to make target 'matmultKernel01.cu', needed by 'matmultKernel01.o'.  Stop.
Total vector size: 3840000
Time: 0.000238 (sec), GFlopsS: 16.122250, GBytesS: 193.466995
Test PASSED
Total vector size: 7680000
Time: 0.000468 (sec), GFlopsS: 16.418071, GBytesS: 197.016848
Test PASSED
Total vector size: 15360000
Time: 0.000931 (sec), GFlopsS: 16.497954, GBytesS: 197.975445
Test PASSED
```

Coalesced vector addition significantly improves GPU performance metrics (execution time and throughput) compared to non-coalesced approaches. This performance gain occurs because coalesced memory access patterns align with the GPU's memory architecture, allowing threads within a warp to access contiguous memory locations in a single transaction, rather than requiring multiple separate memory transactions

Problem 2 : Shared CUDA Matrix Multiply

Solution Q3: Executing Cuda matrix multiplication (matmult00)

Sbatch (run_matmul00.sbatch) and Output (h4_Q3.out)

```
#!/bin/bash
#SBATCH --job-name=h4_q3
#SBATCH --output=%x.out
#SBATCH --account=ece_gy_9143-2025sp
#SBATCH --partition=n1s8-v100-1  # Using a V100 GPU partition as an example
#SBATCH --cpus-per-task=4
#SBATCH --time=10:00:00
#SBATCH --mem=10GB
#SBATCH --gres=gpu:1

cd /scratch/stp8232/PartA


module load singularity
module load cuda

#Singularity container
CONTAINER=/scratch/work/public/singularity/cuda12.1.1-cudnn8.9.0-devel-ubuntu22.04.2.sif

# Run all commands in a single Singularity session
singularity exec --nv ${CONTAINER} bash -c "
cd /scratch/stp8232/PartA
make clean
make 2>/dev/null
make
# 256×256 matrices (16 × 16)
./matmult00 16
# 512×512 matrices (32 × 16)
./matmult00 32
# 1024×1024 matrices (64 × 16)
./matmult00 64
"
```

```
rm -f vecadd00 matmult00 vecadd01 matmult01 *.o
/usr/local/cuda/bin/nvcc vecaddKernel00.cu -c -o vecaddKernel00.o -O3
/usr/local/cuda/bin/nvcc timer.cu -c -o timer.o -O3
/usr/local/cuda/bin/nvcc vecadd.cu vecaddKernel00.o -o vecadd00 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3
/usr/local/cuda/bin/nvcc matmultKernel00.cu -c -o matmultKernel00.o -O3
/usr/local/cuda/bin/nvcc matmult.cu matmultKernel00.o -o matmult00 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3
/usr/local/cuda/bin/nvcc vecaddKernel01.cu -c -o vecaddKernel01.o -O3
/usr/local/cuda/bin/nvcc vecadd.cu vecaddKernel01.o -o vecadd01 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3
make: *** No rule to make target 'matmultKernel01.cu', needed by 'matmultKernel01.o'.  Stop.
Data dimensions: 256x256
Grid Dimensions: 16x16
Block Dimensions: 16x16
Footprint Dimensions: 16x16
Time: 0.000026 (sec), nFlops: 33554432, GFlopsS: 1291.169618
Data dimensions: 512x512
Grid Dimensions: 32x32
Block Dimensions: 16x16
Footprint Dimensions: 16x16
Time: 0.000101 (sec), nFlops: 268435456, GFlopsS: 2655.424309
Data dimensions: 1024x1024
Grid Dimensions: 64x64
Block Dimensions: 16x16
Footprint Dimensions: 16x16
Time: 0.000697 (sec), nFlops: 2147483648, GFlopsS: 3081.491363
```

Solution Q4: CUDA kernel to perform matrix mult on the GPU (matmul01)

```
#include "matmultKernel.h"

__global__ void MatMulKernel(Matrix A, Matrix B, Matrix C){
   // Compute offsets for coalesced memory access.
   int outputRowOffset = BLOCK_SIZE * B.width / FOOTPRINT_SIZE;
   int outputColOffset = BLOCK_SIZE * A.height / FOOTPRINT_SIZE;

   // Calculate row and column indices for the current thread within the output matrix.
   int row = blockIdx.y * blockDim.y + threadIdx.y;
   int col = blockIdx.x * blockDim.x + threadIdx.x;

   // Variables to accumulate the products for the output matrix C.
   float sumForC1 = 0, sumForC2 = 0, sumForC3 = 0, sumForC4 = 0;

  for (int k = 0; k < A.width; ++k) {
      sumForC1 += A.elements[row * A.width + k] * B.elements[k * B.width + col];
      sumForC2 += A.elements[(row + outputRowOffset) * A.width + k] * B.elements[k *
B.width + col];
      sumForC3 += A.elements[row * A.width + k] * B.elements[k * B.width + (col +
outputColOffset)];
      sumForC4 += A.elements[(row + outputRowOffset) * A.width + k] * B.elements[k *
B.width + (col + outputColOffset)];
   }
   C.elements[row * C.width + col] = sumForC1;
   C.elements[(row + outputRowOffset) * C.width + col] = sumForC2;
```

```
    C.elements[row * C.width + (col + outputColOffset)] = sumForC3;
    C.elements[(row + outputRowOffset) * C.width + (col + outputColOffset)] = sumForC4;
}
```

Each thread computes four elements in the output matrix using calculated offsets, which increases computational efficiency. direct dot product computation for each of the four output elements assigned to a thread. This approach avoids the shared memory overhead present in tiled implementations while still increasing the computational density per thread.

Sbatch (run_matmul01.sbatch) and Output (h4_Q4.out)

```
#!/bin/bash
#SBATCH --job-name=h4_q4
#SBATCH --output=%x.out
#SBATCH --account=ece_gy_9143-2025sp
#SBATCH --partition=nls8-v100-1  # Using a V100 GPU partition as an example
#SBATCH --cpus-per-task=4
#SBATCH --time=10:00:00
#SBATCH --mem=10GB
#SBATCH --gres=gpu:1

cd /scratch/stp8232/PartA


module load singularity
module load cuda

#Singularity container
CONTAINER=/scratch/work/public/singularity/cuda12.1.1-cudnn8.9.0-devel-ubuntu22.04.2.sif

# Run all commands in a single Singularity session
singularity exec --nv ${CONTAINER} bash -c "
cd /scratch/stp8232/PartA
make clean
make 2>/dev/null
make
./matmult01 8
./matmult01 16
./matmult01 32
"
```

```
rm -f vecadd00 matmult00 vecadd01 matmult01 *.o
/usr/local/cuda/bin/nvcc vecaddKernel00.cu -c -o vecaddKernel00.o -O3
/usr/local/cuda/bin/nvcc timer.cu -c -o timer.o -O3
/usr/local/cuda/bin/nvcc vecadd.cu vecaddKernel00.o -o vecadd00 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3
/usr/local/cuda/bin/nvcc matmultKernel00.cu -c -o matmultKernel00.o -O3
/usr/local/cuda/bin/nvcc matmult.cu matmultKernel00.o -o matmult00 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3
/usr/local/cuda/bin/nvcc vecaddKernel01.cu -c -o vecaddKernel01.o -O3
/usr/local/cuda/bin/nvcc vecadd.cu vecaddKernel01.o -o vecadd01 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3
/usr/local/cuda/bin/nvcc matmultKernel01.cu -c -o matmultKernel01.o -O3  -DFOOTPRINT_SIZE=32
/usr/local/cuda/bin/nvcc matmult.cu matmultKernel01.o -o matmult01 -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 timer.o -O3  -
DFOOTPRINT_SIZE=32
make: Nothing to be done for 'all'.
Data dimensions: 256x256
Grid Dimensions: 8x8
Block Dimensions: 16x16
Footprint Dimensions: 32x32
Time: 0.000069 (sec), nFlops: 33554432, GFlopsS: 485.301684
Data dimensions: 512x512
Grid Dimensions: 16x16
Block Dimensions: 16x16
Footprint Dimensions: 32x32
Time: 0.000129 (sec), nFlops: 268435456, GFlopsS: 2081.145854
Data dimensions: 1024x1024
Grid Dimensions: 32x32
Block Dimensions: 16x16
Footprint Dimensions: 32x32
Time: 0.001105 (sec), nFlops: 2147483648, GFlopsS: 1943.300810
```

As matrix size increases, the execution time grows, GFLOPS increases with matrix size, indicating better GPU utilization for larger matrices.

Solution Q5:
Based on my experimentation with matrix multiplication in CUDA

1. **Maximizing Work per Thread**: Assigning multiple output elements to each thread significantly reduces thread management overhead and improves computational efficiency. The 2.65× speedup for 256×256 matrices demonstrates this benefit clearly.
2. **Balancing Thread Count and Work per Thread**: While increasing work per thread is beneficial, there's an optimal balance point that varies with problem size. Too few threads may underutilize the GPU, while too many create excessive overhead.
3. **Memory Access Patterns**: Strategic memory access patterns that reduce bank conflicts and promote coalescing can significantly impact performance. The varying speedups across different matrix sizes in my implementation indicate the importance of memory considerations.
4. **Adapt Optimizations to Problem Size**: Different optimizations yield different benefits depending on the problem size. For smaller matrices (256×256), reducing thread overhead provided the largest benefit, while for larger matrices (1024×1024), memory access patterns became more critical.
5. **Measure Actual Performance**: Theoretical improvements may not always translate to real-world gains. The performance improvements I observed varied substantially across different matrix sizes, highlighting the importance of empirical measurement.
6. **Grid and Block Dimensions Matter**: The optimal choice of grid and block dimensions depends on both the algorithm and the hardware. In my implementation, adjusting the FOOTPRINT_SIZE and consequently the grid dimensions was essential for performance gains.

Part B: CUDA Unified Memory

Solution Q1: Vector Addition on CPU

C++ code - CPUVecAdd.cpp

```
#include <iostream>
#include <chrono>
#include <cmath>
// Function to add arrays
void sumVectors(float *vector1, float *vector2, int size) {
for (int idx = 0; idx < size; ++idx) {
vector2[idx] = vector1[idx] + vector2[idx];
 }
}
int main(int argc, char** argv) {
int scale = 1;
int size = 1 << 20;
if (argc == 2) {
sscanf(argv[1], "%d", &scale);
 }
 size = scale * size;
float *firstArray = (float*)malloc(size * sizeof(float));
float *secondArray = (float*)malloc(size * sizeof(float));
 // Initialize arrays
for (int idx = 0; idx < size; ++idx) {
firstArray[idx] = 1.0f;
secondArray[idx] = 2.0f;
 }
auto timeStart = std::chrono::high_resolution_clock::now();
sumVectors(firstArray, secondArray, size);
auto timeEnd = std::chrono::high_resolution_clock::now();
auto elapsedTime = std::chrono::duration_cast<std::chrono::microseconds>(timeEnd -
timeStart);
std::cout << "K: " << scale << " million, " << "Time: " << (float)elapsedTime.count()/1000000
<< " (sec)" << std::endl;
 // Check for errors (all values should be 3.0f)
float errorMax = 0.0f;
for (int idx = 0; idx < size; idx++) {
 errorMax = std::fmax(errorMax, std::fabs(secondArray[idx] - 3.0f));
 }
std::cout << "Max error: " << errorMax << std::endl;
 // Free memory
free(firstArray);
```

```
free(secondArray);
return 0;
}
```

Sbatch (run_CPUVecAdd.sbatch) and Output (CPUVecAdd.out)

```
#!/bin/bash
#SBATCH --job-name=CPUVecAdd
#SBATCH --output=%x.out
#SBATCH --account=ece_gy_9143-2025sp
#SBATCH --partition=n1s8-v100-1  # Using a V100 GPU partition as an example
#SBATCH --cpus-per-task=4
#SBATCH --time=10:00:00
#SBATCH --mem=10GB
#SBATCH --gres=gpu:1

cd /scratch/stp8232/PartB


module load singularity
module load cuda

#Singularity container
CONTAINER=/scratch/work/public/singularity/cuda12.1.1-cudnn8.9.0-devel-ubuntu22.04.2.sif

# Run all commands in a single Singularity session
singularity exec --nv ${CONTAINER} bash -c "
cd /scratch/stp8232/PartB
make clean
make 2>/dev/null
make
./CPUVecAdd 1
./CPUVecAdd 5
./CPUVecAdd 10
./CPUVecAdd 50
./CPUVecAdd 100
"
```

```
rm -f CPUVecAdd GPUVecAdd GPUVecAddUM *.o
g++ -o CPUVecAdd CPUVecAdd.cpp -std=c++11
make: *** No rule to make target 'GPUVecAdd.cu', needed by 'GPUVecAdd'.  Stop.
K: 1 million, Time: 0.003176 (sec)
Max error: 0
K: 5 million, Time: 0.016369 (sec)
Max error: 0
K: 10 million, Time: 0.032471 (sec)
Max error: 0
K: 50 million, Time: 0.160803 (sec)
Max error: 0
K: 100 million, Time: 0.325852 (sec)
Max error: 0
```

Solution Q2: Vector Addition on GPU without Unified Memory
addition using cudaMalloc() and cudaMemcpy() for memory management. GPUVecAdd

Sbatch (run_GPUVecAdd.sbatch) and Output (GPUVecAdd.out)

```
#!/bin/bash
#SBATCH --job-name=GPUVecAdd
#SBATCH --output=%x.out
#SBATCH --account=ece_gy_9143-2025sp
#SBATCH --partition=n1s8-v100-1
#SBATCH --cpus-per-task=4
#SBATCH --time=10:00:00
#SBATCH --mem=10GB
#SBATCH --gres=gpu:1

cd /scratch/stp8232/PartB


module load singularity
module load cuda

#Singularity container
CONTAINER=/scratch/work/public/singularity/cuda12.1.1-cudnn8.9.0-devel-ubuntu22.04.2.sif

# Run all commands in a single Singularity session
singularity exec --nv ${CONTAINER} bash -c "
cd /scratch/stp8232/PartB
make clean
make 2>/dev/null
make GPUVecAdd
nvprof ./GPUVecAdd 1 1 1
nvprof ./GPUVecAdd 1 1 5
nvprof ./GPUVecAdd 1 1 10
nvprof ./GPUVecAdd 1 1 50
nvprof ./GPUVecAdd 1 1 100
nvprof ./GPUVecAdd 1 256 1
nvprof ./GPUVecAdd 1 256 5
nvprof ./GPUVecAdd 1 256 10
nvprof ./GPUVecAdd 1 256 50
nvprof ./GPUVecAdd 1 256 100
nvprof ./GPUVecAdd 16 256 1
nvprof ./GPUVecAdd 16 256 5
nvprof ./GPUVecAdd 16 256 10
nvprof ./GPUVecAdd 16 256 50
nvprof ./GPUVecAdd 16 256 100
"
```

16 Threads per block, 256 of blocks and k 1, 5, 10, 50,100

```
rm -f CPUVecAdd GPUVecAdd GPUVecAddUM *.o
g++ -o CPUVecAdd CPUVecAdd.cpp -std=c++11
/usr/local/cuda/bin/nvcc -o GPUVecAdd GPUVecAdd.cu -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 -O3
make: 'GPUVecAdd' is up to date.
K: 1, Grid size: 1, Block size: 1
==5064== NVPROF is profiling process 5064, command: ./GPUVecAdd 1 1 1
Max error: 0
==5064== Profiling application: ./GPUVecAdd 1 1 1
==5064== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   96.89%  69.752ms         1  69.752ms  69.752ms  69.752ms  add(int, float*, float*)
                    2.23%  1.6021ms         2  801.04us  799.94us  802.15us  [CUDA memcpy HtoD]
                    0.88%  634.56us         1  634.56us  634.56us  634.56us  [CUDA memcpy DtoH]
      API calls:   44.15%  142.76ms         2  71.381ms  76.563us  142.69ms  cudaMalloc
                   33.19%  107.33ms         1  107.33ms  107.33ms  107.33ms  cudaLaunchKernel
                   22.50%  72.757ms         3  24.252ms  975.47us  70.757ms  cudaMemcpy
                    0.10%  331.85us         2  165.92us  163.09us  168.76us  cudaFree
                    0.06%  179.32us       114  1.5720us     161ns  60.857us  cuDeviceGetAttribute
                    0.00%  12.444us         1  12.444us  12.444us  12.444us  cuDeviceGetName
                    0.00%  7.5290us         1  7.5290us  7.5290us  7.5290us  cuDeviceGetPCIBusId
                    0.00%  1.7670us         3     589ns     204ns  1.3000us  cuDeviceGetCount
                    0.00%  1.2910us         2     645ns     190ns  1.1010us  cuDeviceGet
                    0.00%     401ns         1     401ns     401ns     401ns  cuDeviceTotalMem
                    0.00%     375ns         1     375ns     375ns     375ns  cuModuleGetLoadingMode
                    0.00%     275ns         1     275ns     275ns     275ns  cuDeviceGetUuid
K: 5, Grid size: 1, Block size: 1
==5082== NVPROF is profiling process 5082, command: ./GPUVecAdd 1 1 5
Max error: 0
==5082== Profiling application: ./GPUVecAdd 1 1 5
==5082== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   96.32%  328.76ms         1  328.76ms  328.76ms  328.76ms  add(int, float*, float*)
                    2.47%  8.4166ms         2  4.2083ms  4.2042ms  4.2124ms  [CUDA memcpy HtoD]
                    1.21%  4.1342ms         1  4.1342ms  4.1342ms  4.1342ms  [CUDA memcpy DtoH]
      API calls:   69.31%  342.11ms         3  114.04ms  4.4114ms  333.26ms  cudaMemcpy
                   27.84%  137.42ms         2  68.708ms  97.909us  137.32ms  cudaMalloc
                    1.95%  9.6183ms         1  9.6183ms  9.6183ms  9.6183ms  cudaLaunchKernel
                    0.86%  4.2571ms         2  2.1286ms  557.45us  3.6997ms  cudaFree
                    0.03%  142.23us       114  1.2470us     141ns  54.298us  cuDeviceGetAttribute
                    0.00%  11.904us         1  11.904us  11.904us  11.904us  cuDeviceGetName
                    0.00%  7.3110us         1  7.3110us  7.3110us  7.3110us  cuDeviceGetPCIBusId
                    0.00%  2.2260us         3     742ns     170ns  1.7330us  cuDeviceGetCount
                    0.00%  1.2400us         2     620ns     171ns  1.0690us  cuDeviceGet
                    0.00%     436ns         1     436ns     436ns     436ns  cuDeviceTotalMem
                    0.00%     265ns         1     265ns     265ns     265ns  cuModuleGetLoadingMode
                    0.00%     220ns         1     220ns     220ns     220ns  cuDeviceGetUuid
K: 10, Grid size: 1, Block size: 1
==5101== NVPROF is profiling process 5101, command: ./GPUVecAdd 1 1 10
Max error: 0
==5101== Profiling application: ./GPUVecAdd 1 1 10
==5101== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   95.99%  625.81ms         1  625.81ms  625.81ms  625.81ms  add(int, float*, float*)
                    2.67%  17.390ms         2  8.6948ms  8.6797ms  8.7099ms  [CUDA memcpy HtoD]
                    1.34%  8.7281ms         1  8.7281ms  8.7281ms  8.7281ms  [CUDA memcpy DtoH]
      API calls:   80.80%  652.70ms         3  217.57ms  8.8556ms  634.91ms  cudaMemcpy
```

```
                    0.17%  1.3632ms         2  681.58us  257.39us  1.1058ms  cudaFree
                    0.02%  147.23us       114  1.2910us     154ns  57.085us  cuDeviceGetAttribute
                    0.00%  12.529us         1  12.529us  12.529us  12.529us  cuDeviceGetName
                    0.00%  8.6030us         1  8.6030us  8.6030us  8.6030us  cuDeviceGetPCIBusId
                    0.00%  1.6600us         3     553ns     192ns  1.2410us  cuDeviceGetCount
                    0.00%     976ns         2     488ns     174ns     802ns  cuDeviceGet
                    0.00%     337ns         1     337ns     337ns     337ns  cuModuleGetLoadingMode
                    0.00%     328ns         1     328ns     328ns     328ns  cuDeviceTotalMem
                    0.00%     262ns         1     262ns     262ns     262ns  cuDeviceGetUuid
K: 50, Grid size: 1, Block size: 1
==5115== NVPROF is profiling process 5115, command: ./GPUVecAdd 1 1 50
Max error: 0
==5115== Profiling application: ./GPUVecAdd 1 1 50
==5115== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   95.93%  3.12700s         1  3.12700s  3.12700s  3.12700s  add(int, float*, float*)
                    2.71%  88.229ms         2  44.114ms  44.084ms  44.145ms  [CUDA memcpy HtoD]
                    1.36%  44.339ms         1  44.339ms  44.339ms  44.339ms  [CUDA memcpy DtoH]
      API calls:   95.54%  3.26035s         3  1.08678s  44.257ms  3.17173s  cudaMemcpy
                    4.11%  140.15ms         2  70.077ms  93.554us  140.06ms  cudaMalloc
                    0.27%  9.1748ms         1  9.1748ms  9.1748ms  9.1748ms  cudaLaunchKernel
                    0.08%  2.6479ms         2  1.3240ms  396.88us  2.2511ms  cudaFree
                    0.01%  176.38us       114  1.5470us     142ns  73.674us  cuDeviceGetAttribute
                    0.00%  12.172us         1  12.172us  12.172us  12.172us  cuDeviceGetName
                    0.00%  7.0850us         1  7.0850us  7.0850us  7.0850us  cuDeviceGetPCIBusId
                    0.00%  1.4420us         3     480ns     189ns  1.0570us  cuDeviceGetCount
                    0.00%  1.0500us         2     525ns     167ns     883ns  cuDeviceGet
                    0.00%     494ns         1     494ns     494ns     494ns  cuDeviceTotalMem
                    0.00%     291ns         1     291ns     291ns     291ns  cuDeviceGetUuid
                    0.00%     257ns         1     257ns     257ns     257ns  cuModuleGetLoadingMode
K: 100, Grid size: 1, Block size: 1
==5132== NVPROF is profiling process 5132, command: ./GPUVecAdd 1 1 100
Max error: 0
==5132== Profiling application: ./GPUVecAdd 1 1 100
==5132== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   95.94%  6.25703s         1  6.25703s  6.25703s  6.25703s  add(int, float*, float*)
                    2.71%  176.60ms         2  88.302ms  88.097ms  88.506ms  [CUDA memcpy HtoD]
                    1.35%  88.141ms         1  88.141ms  88.141ms  88.141ms  [CUDA memcpy DtoH]
      API calls:   97.77%  6.52257s         3  2.17419s  88.334ms  6.34557s  cudaMemcpy
                    2.05%  136.58ms         2  68.292ms  101.40us  136.48ms  cudaMalloc
                    0.13%  8.8276ms         1  8.8276ms  8.8276ms  8.8276ms  cudaLaunchKernel
                    0.04%  2.8960ms         2  1.4480ms  496.00us  2.4001ms  cudaFree
                    0.00%  148.32us       114  1.3010us     154ns  56.553us  cuDeviceGetAttribute
                    0.00%  12.172us         1  12.172us  12.172us  12.172us  cuDeviceGetName
                    0.00%  7.4840us         1  7.4840us  7.4840us  7.4840us  cuDeviceGetPCIBusId
                    0.00%  2.0920us         3     697ns     214ns  1.5660us  cuDeviceGetCount
                    0.00%     693ns         2     346ns     163ns     530ns  cuDeviceGet
                    0.00%     396ns         1     396ns     396ns     396ns  cuDeviceTotalMem
                    0.00%     258ns         1     258ns     258ns     258ns  cuModuleGetLoadingMode
                    0.00%     235ns         1     235ns     235ns     235ns  cuDeviceGetUuid
K: 1, Grid size: 1, Block size: 256
==5151== NVPROF is profiling process 5151, command: ./GPUVecAdd 1 256 1
Max error: 0
==5151== Profiling application: ./GPUVecAdd 1 256 1
==5151== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
```

Solution Q3: Vector Addition on GPU with Unified Memory
cudaMallocManaged() to handle memory which simplifies data handling between CPU and GPU VecAddUM

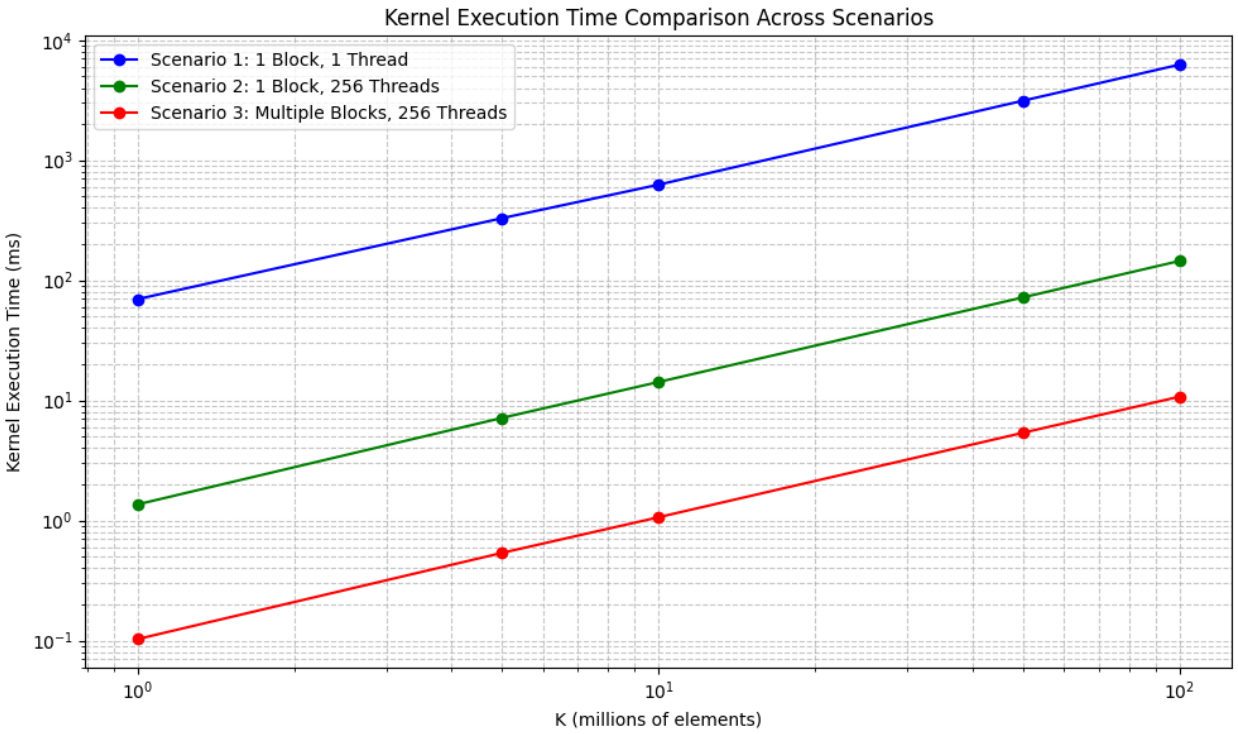
Sbatch (run_VecAddUM.sbatch) and Output (VecAddUM.out)

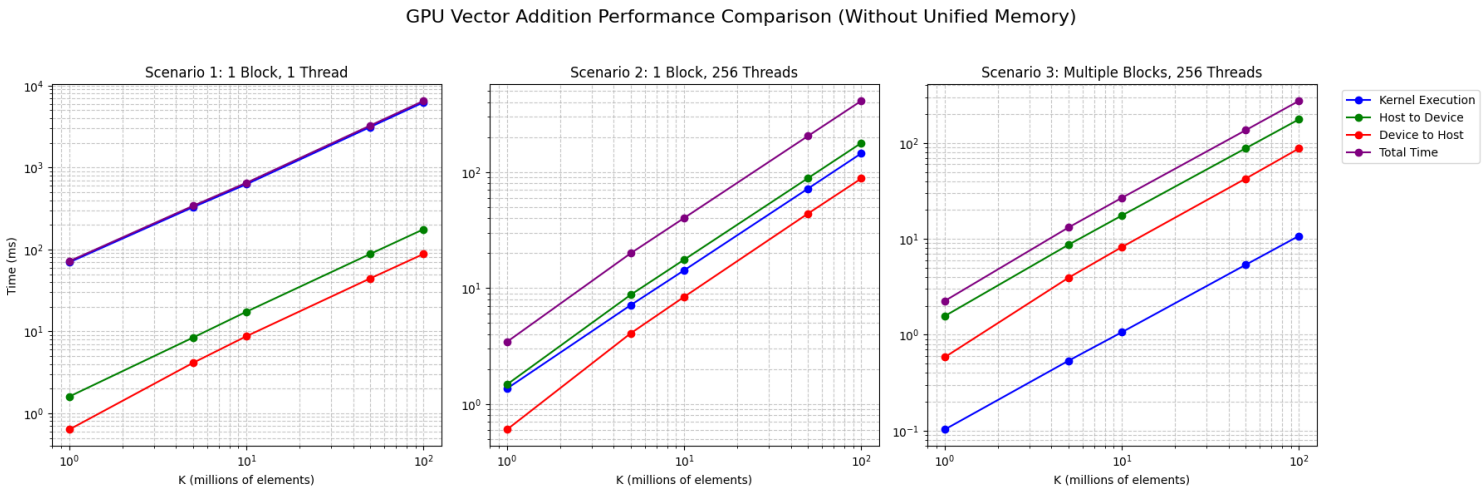
```
#!/bin/bash
#SBATCH --job-name=VecAddUM
#SBATCH --output=%x.out
#SBATCH --account=ece_gy_9143-2025sp
#SBATCH --partition=n1s8-v100-1
#SBATCH --cpus-per-task=4
#SBATCH --time=10:00:00
#SBATCH --mem=10GB
#SBATCH --gres=gpu:1

cd /scratch/stp8232/PartB


module load singularity
module load cuda

#Singularity container
CONTAINER=/scratch/work/public/singularity/cuda12.1.1-cudnn8.9.0-devel-ubuntu22.04.2.sif

# Run all commands in a single Singularity session
singularity exec --nv ${CONTAINER} bash -c "
cd /scratch/stp8232/PartB
make clean
make 2>/dev/null
make VecAddUM
nvprof ./VecAddUM 1 1 1
nvprof ./VecAddUM 1 1 5
nvprof ./VecAddUM 1 1 10
nvprof ./VecAddUM 1 1 50
nvprof ./VecAddUM 1 1 100
nvprof ./VecAddUM 1 256 1
nvprof ./VecAddUM 1 256 5
nvprof ./VecAddUM 1 256 10
nvprof ./VecAddUM 1 256 50
nvprof ./VecAddUM 1 256 100
nvprof ./VecAddUM 16 256 1
nvprof ./VecAddUM 16 256 5
nvprof ./VecAddUM 16 256 10
nvprof ./VecAddUM 16 256 50
nvprof ./VecAddUM 16 256 100
"
```

```
rm -f CPUVecAdd GPUVecAdd VecAddUM *.o
g++ -o CPUVecAdd CPUVecAdd.cpp -std=c++11
/usr/local/cuda/bin/nvcc -o GPUVecAdd GPUVecAdd.cu -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 -O3
/usr/local/cuda/bin/nvcc -o VecAddUM VecAddUM.cu -L/usr/local/cuda/lib64 -L/usr/local/cuda/samples/common/lib/linux/x86_64 -O3
make: 'VecAddUM' is up to date.
K: 1, Grid size: 1, Block size: 1
==5117== NVPROF is profiling process 5117, command: ./VecAddUM 1 1 1
Max error: 0
==5117== Profiling application: ./VecAddUM 1 1 1
==5117== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  62.337ms         1  62.337ms  62.337ms  62.337ms  add(int, float*, float*)
      API calls:   63.50%  166.19ms         2  83.094ms  46.209us  166.14ms  cudaMallocManaged
                   23.82%  62.342ms         1  62.342ms  62.342ms  62.342ms  cudaDeviceSynchronize
                   12.44%  32.570ms         1  32.570ms  32.570ms  32.570ms  cudaLaunchKernel
                    0.17%  454.85us         2  227.43us  227.08us  227.77us  cudaFree
                    0.06%  145.30us       114  1.2740us     150ns  56.466us  cuDeviceGetAttribute
                    0.00%  10.603us         1  10.603us  10.603us  10.603us  cuDeviceGetName
                    0.00%  6.4610us         1  6.4610us  6.4610us  6.4610us  cuDeviceGetPCIBusId
                    0.00%  1.8870us         3     629ns     210ns  1.4570us  cuDeviceGetCount
                    0.00%  1.1250us         2     562ns     154ns     971ns  cuDeviceGet
                    0.00%     477ns         1     477ns     477ns     477ns  cuDeviceTotalMem
                    0.00%     281ns         1     281ns     281ns     281ns  cuModuleGetLoadingMode
                    0.00%     258ns         1     258ns     258ns     258ns  cuDeviceGetUuid

==5117== Unified Memory profiling result:
Device "Tesla V100-SXM2-16GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      48  170.67KB  4.0000KB  0.9961MB  8.000000MB  841.0190us  Host To Device
      24  170.67KB  4.0000KB  0.9961MB  4.000000MB  354.9750us  Device To Host
      12         -         -         -           -  2.135220ms  Gpu page fault groups
Total CPU Page faults: 36
K: 5, Grid size: 1, Block size: 1
==5135== NVPROF is profiling process 5135, command: ./VecAddUM 1 1 5
Max error: 0
==5135== Profiling application: ./VecAddUM 1 1 5
==5135== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  282.54ms         1  282.54ms  282.54ms  282.54ms  add(int, float*, float*)
      API calls:   60.64%  282.55ms         1  282.55ms  282.55ms  282.55ms  cudaDeviceSynchronize
                   34.49%  160.70ms         2  80.351ms  76.202us  160.63ms  cudaMallocManaged
                    4.41%  20.535ms         1  20.535ms  20.535ms  20.535ms  cudaLaunchKernel
                    0.42%  1.9702ms         2  985.11us  864.97us  1.1053ms  cudaFree
                    0.03%  151.79us       114  1.3310us     143ns  55.766us  cuDeviceGetAttribute
                    0.00%  14.765us         1  14.765us  14.765us  14.765us  cuDeviceGetName
                    0.00%  5.7870us         1  5.7870us  5.7870us  5.7870us  cuDeviceGetPCIBusId
                    0.00%  2.0150us         3     671ns     236ns  1.3850us  cuDeviceGetCount
                    0.00%     731ns         2     365ns     200ns     531ns  cuDeviceGet
                    0.00%     473ns         1     473ns     473ns     473ns  cuDeviceTotalMem
                    0.00%     317ns         1     317ns     317ns     317ns  cuModuleGetLoadingMode
                    0.00%     260ns         1     260ns     260ns     260ns  cuDeviceGetUuid

==5135== Unified Memory profiling result:
Device "Tesla V100-SXM2-16GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
     240  170.67KB  4.0000KB  0.9961MB  40.00000MB  4.193828ms  Host To Device
```

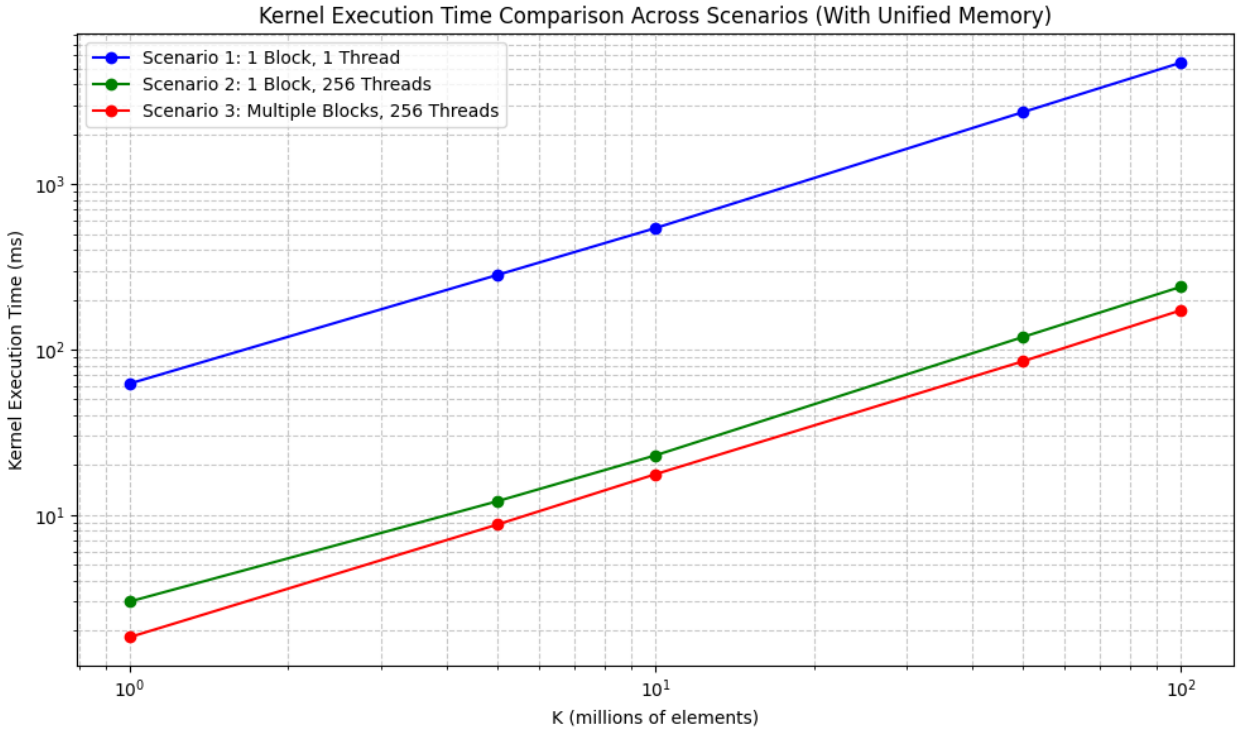
Solution Q4:

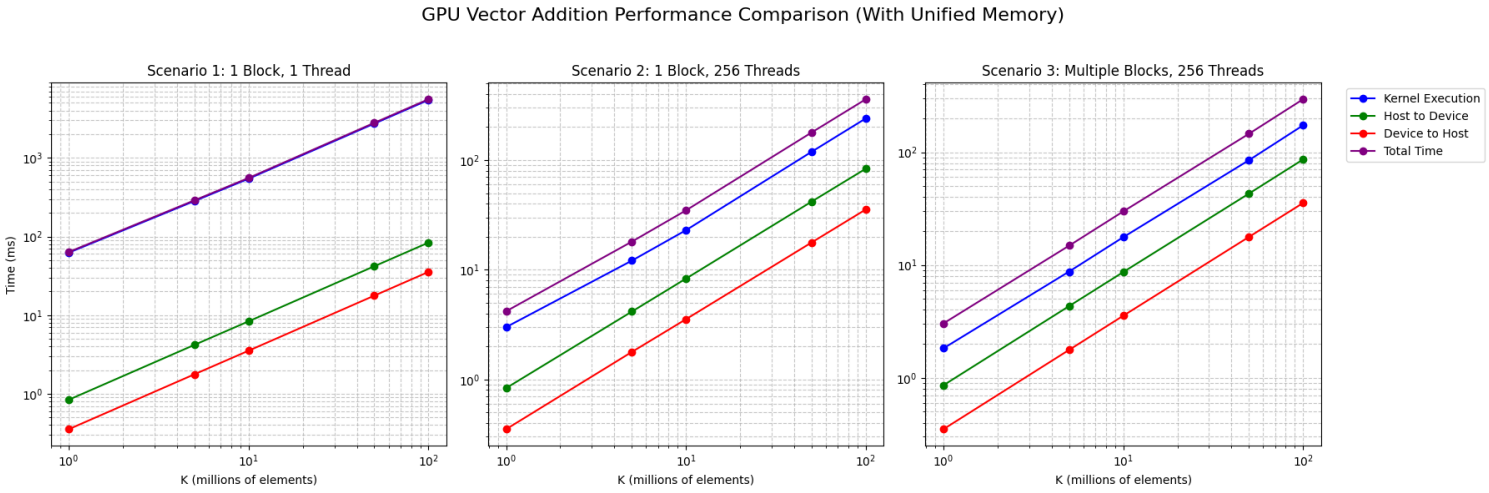
Step 2: Without Unified Memory





Step 3: With Unified Memory





Part C: Convolution in CUDA

Solution C1: Basic Implementation of Convolution
Solution C2: shared memory tiled approach
Solution C3: Cdnn based convolution

```
#!/bin/bash
#SBATCH --job-name=convolution
#SBATCH --output=%x.out
#SBATCH --account=ece_gy_9143-2025sp
#SBATCH --partition=c12m85-a100-1
#SBATCH --cpus-per-task=4
#SBATCH --time=10:00:00
#SBATCH --mem=10GB
#SBATCH --gres=gpu:1

cd /scratch/stp8232/PartC

# Load singularity
module load singularity

# Container path
CONTAINER=/scratch/work/public/singularity/cuda12.1.1-cudnn8.9.0-devel-ubuntu22.04.2.sif

# Run compilation inside the container
singularity exec --nv $CONTAINER bash -c "cd /scratch/stp8232/PartC && nvcc -o convolve convolve.cu -lcudnn"

# Run the program inside the container
singularity exec --nv $CONTAINER bash -c "cd /scratch/stp8232/PartC && ./convolve"
```

```
      |                                                                ^~~~~~~
Checksum : 122756344698240, Time : 2.972 millisec
Checksum : 122756344698240, Time : 2.512 millisec
Checksum : 122756344698240, Time : 2.109 millisec
```

The shared memory optimization provides a good speedup over the basic implementation, and the highly optimized cuDNN library gives you the best performance overall.