DD2360
Assignment 4 - Report

Heng Kang hengkang@kth.se
Jian Wen jianwe@kth.se

Git repo: *https://github.com/SinGuLaRiTy2001/DD2360HT23*

Ex1. Thread Scheduling and Execution Efficiency

1. Assume X=800 and Y=600. Assume that we decided to use a grid of 16X16 blocks. That is, each block is organized as a 2D 16X16 array of threads. How many warps will be generated during the execution of the kernel? How many warps will have control divergence? Please explain your answers.

   ANSWER: For each block, there are 16×16=256 threads. The block number is (800/16)×(600/16)=50×38=1900 blocks. Therefore, the total number of threads is 1900×256=486400 threads. In CUDA devices, each warp contains 32 threads, so the number of warps is 486400/32=15200.

   Control divergence occurs in the last row and last column of blocks due to some threads being out of the image bounds.On the X axis, the number of warp with divergence is (800%16)/32=0. On the Y axis, the number of warp with divergence is (600%16)/32=1. Considering there are 38 blocks on the Y axis, so there are a total of 38 warps with divergence.

2. Now assume X=600 and Y=800 instead, how many warps will have control divergence? Please explain your answers.

   ANSWER: By using similar calculations, the number of warp on the X axis will be (600%16)/32=1 while for Y axis they are perfectly aligned. And there are 600/16=38 blocks on X axis. So the total number of warps is still 38.

3. Now assume X=600 and Y=799, how many warps will have control divergence? Please explain your answers.

   ANSWER: By using similar calculations, the number of warp on the X axis will be (600%16)/32=1, and the total number of warps will be 1×38=38 on X axis; For Y axis, there are 799/16=50 blocks. And the warps number will be 50×(799%16)/32=50×8=400 on Y axis. Therefore, the total number of warps with control divergence is 400+38=438.

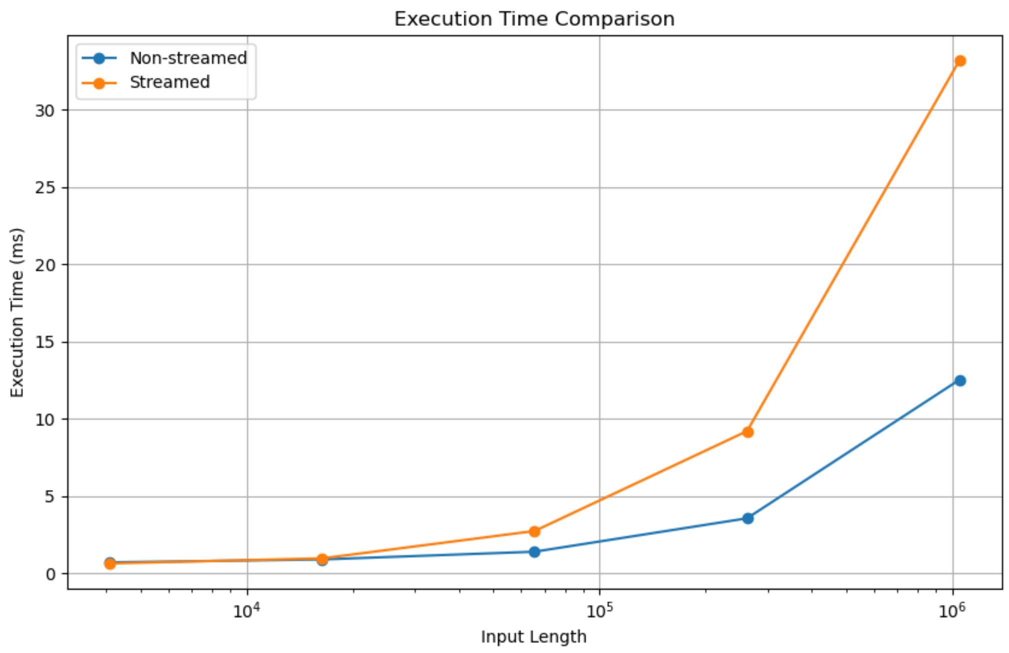
Ex2. CUDA Streams

Updated Implementation:

```
cudaStream_t streams[numStreams];
for (int i = 0; i < numStreams; i++) {
   cudaStreamCreate(&streams[i]);
}
startTime = getTime();
for (int i = 0; i < inputLength; i += S_seg) {
   int segSize = min(S_seg, inputLength - i);
   int streamIdx = i / S_seg % numStreams;
       cudaMemcpyAsync(deviceInput1  +  i,  hostInput1  +  i,  segSize  *  sizeof(DataType),
cudaMemcpyHostToDevice, streams[streamIdx]);
       cudaMemcpyAsync(deviceInput2  +  i,  hostInput2  +  i,  segSize  *  sizeof(DataType),
cudaMemcpyHostToDevice, streams[streamIdx]);
   int Block_dim = ThreadNum;
   int Grid_dim = segSize / ThreadNum + (segSize % ThreadNum ≠ 0);
    vecAdd<<<Grid_dim, Block_dim, 0, streams[streamIdx]>>>(deviceInput1 + i, deviceInput2 + i,
deviceOutput + i, segSize);
       cudaMemcpyAsync(hostOutput  +  i,  deviceOutput  +  i,  segSize  *  sizeof(DataType),
cudaMemcpyDeviceToHost, streams[streamIdx]);
}
for (int i = 0; i < numStreams; i++) {
   cudaStreamSynchronize(streams[i]);
}
endTime = getTime();
printf("Total asynchronous operation time: %.6lf ms.\n", (endTime - startTime) * 1000);
```

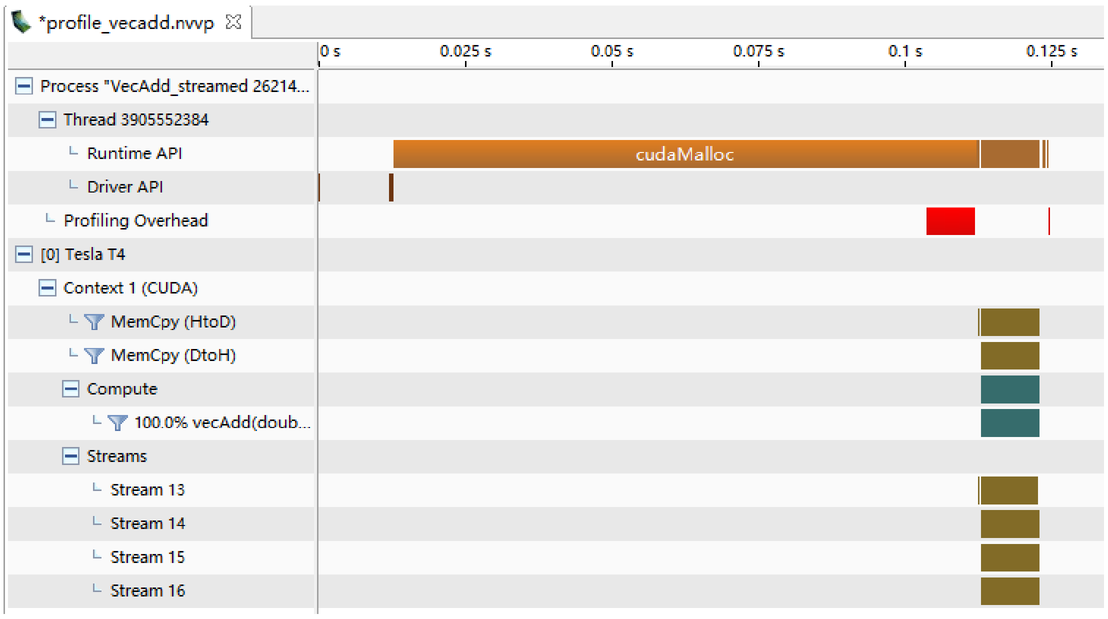
1. Compared to the non-streamed vector addition, what performance gain do you get? Present in a plot (you may include comparison at different vector length)

   ANSWER: The two implementations are evaluated with different input lengths: 4096, 16384, 65536, 262144, 1048576. The S_seg is set as 1024.



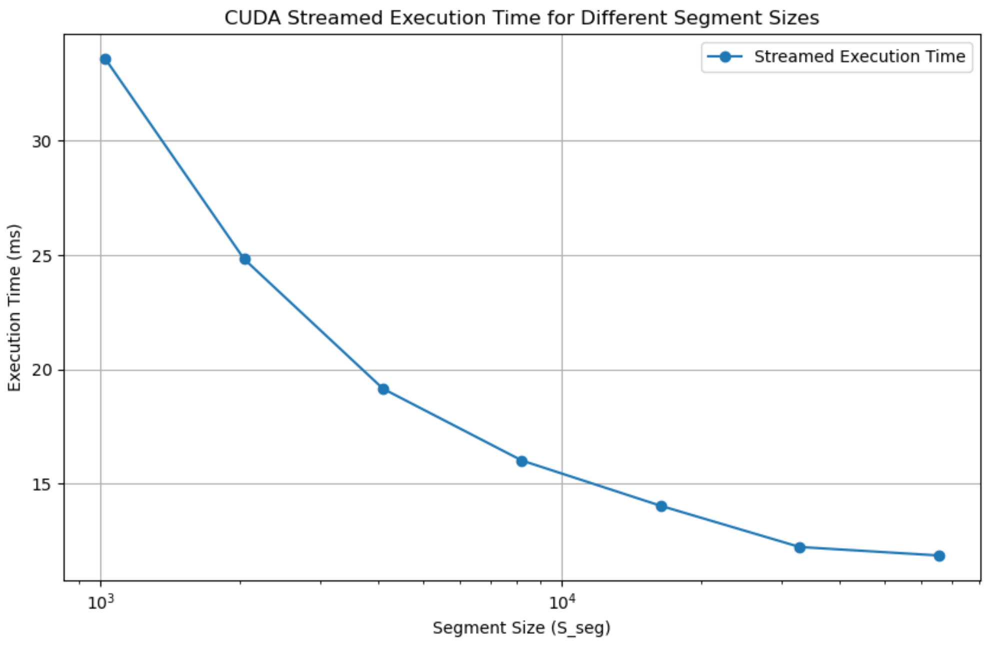
2. Use nvprof to collect traces and the NVIDIA Visual Profiler (nvvp) to visualize the overlap of communication and computation.

   ANSWER: The NVVP timeline indicates the use of four CUDA streams.It can be seen that the memory copy and computation are carried out at the same time on 4 different streams.



3. What is the impact of segment size on performance? Present in a plot (you may choose a large vector and compare 4-8 different segment sizes)

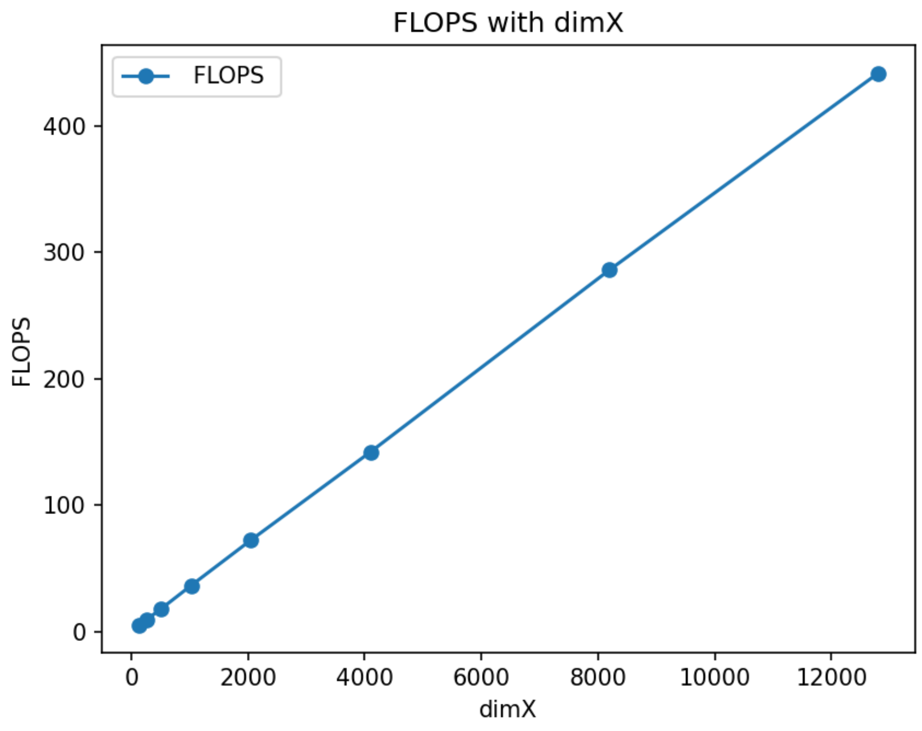
   ANSWER: A series evaluation with S_seg=1024, 2048, 4096, 8192, 16384, 32768, 65536 are performed. It can be seen from the figure that, the execution time decreases as segment size increases. The x-axis of the figure is in log unit.



Ex3. Heat Equation with Using NVIDIA Libraries

1. Run the program with different dimX values. For each one, approximate the FLOPS (floating-point operation per second) achieved in computing the SMPV (sparse matrix multiplication). Report FLOPS at different input sizes in a FLOPS. What do you see compared to the peak throughput you report in Lab2?

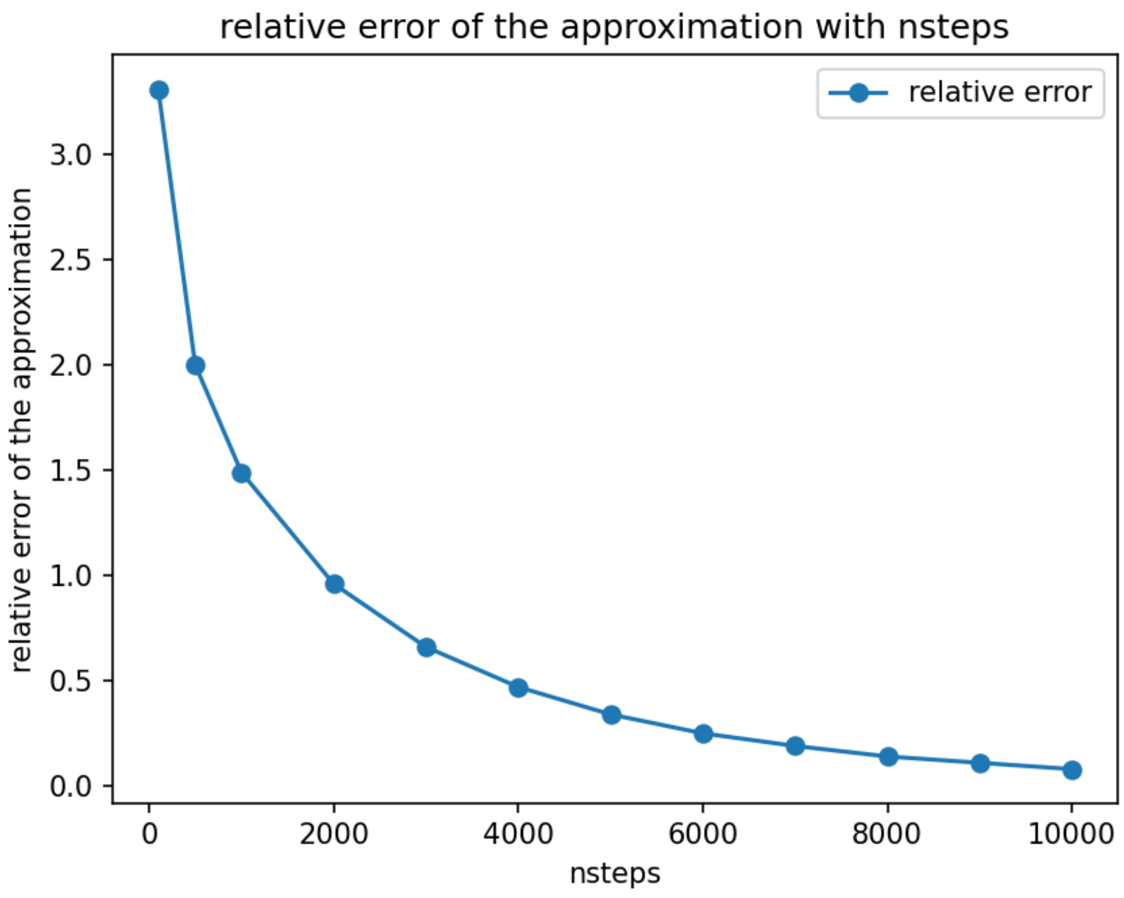
ANSWER: Values in the figure are lower than theoretical values.



.

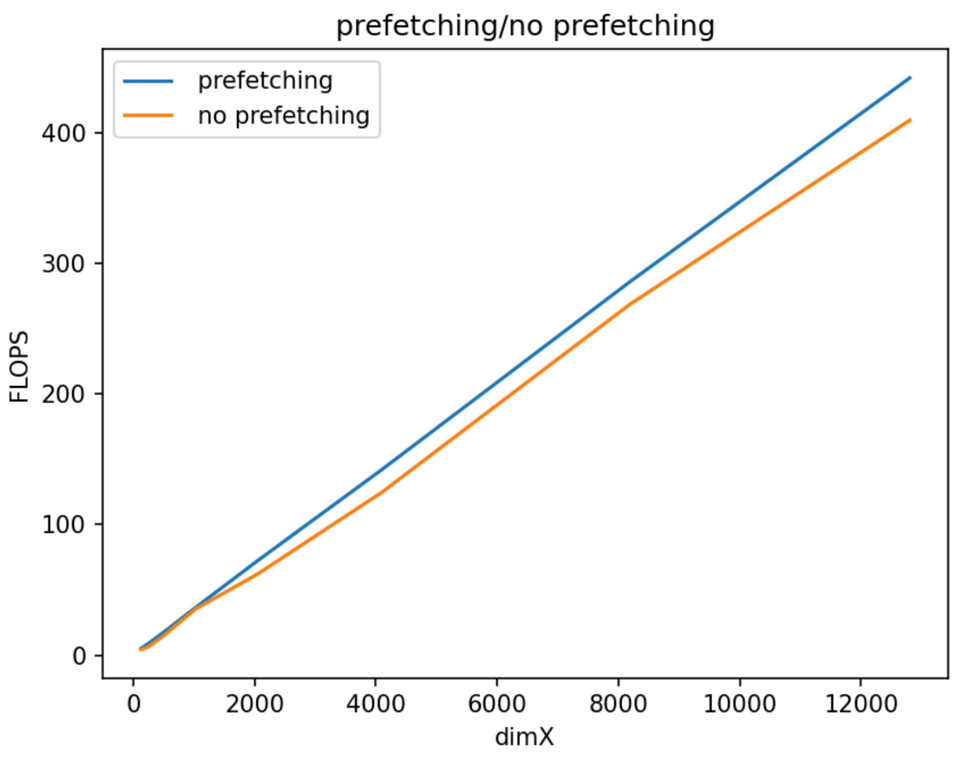
2. Run the program with dimX=128 and vary nsteps from 100 to 10000. Plot the relative error of the approximation at different nstep. What do you observe?

ANSWER: As the nstep increases, the relative error of the approximation decreases.



3. Compare the performance with and without the prefetching in Unified Memory. How is the performance impact?

ANSWER: Without prefetching, the code performs worse than prefetching.



Contributions

Jian Wen

Implemented the exercise 1&2 (Thread Scheduling and Execution Efficiency, CUDA Streams) in the assignment.

Heng Kang

Implemented the exercise 3 (Heat Equation with using NVIDIA libraries) in the assignment.