DD2360
Assignment IV: Advanced CUDA

Rui Shi

January 8, 2023

Link to github: `https://github.com/RuiShi324/DD2360`

1 Exercise 1 - 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 threads in the horizon and vertical. From the codes, we can see that the organization of the grid will be used as 2D. So $|600/16| \times |800/16| = 38 \times 50 = 1900$ thread blocks will be used, and 50 in x dimension, 38 in y dimension. In y dimension, not all the threads in the last thread block will be executed, only 8 threads will be used. In this case, because of the linearized row-major order, $8 \times 16/32 = 4$ warps will be generated for each thread block, and in total 200 warps will be generated. So for the thread blocks that are fully executed, $37 \times 50 = 1850$ thread blocks will be used and $1850 \times 16 \times 16/32 = 14800$ warps will be generated during the execution of the kernel. In summary, 15000 warps will be generated.

   For the last thread block in the y dimension, as discussed before, no thread will have control divergence because of the linearized row-major order.

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

   Answer: In this case, only 8 threads will be used for the last thread block in the x dimension, because the last 8 threads in x dimension are not in range of the if statement, which causes control divergence. So each warp will have control divergence in this case, $50 \times 8 = 400$ warps will have control divergence.

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

   Answer: In this case, also 1900 blocks will be used, not all the threads will be used for the last thread block in x dimension and y dimension. For the last warp, as all the last threads in y dimension will not be used, it doesn't have control divergence. So 399 warps will have control divergence.

2 Exercise 2 - CUDA Streams

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)

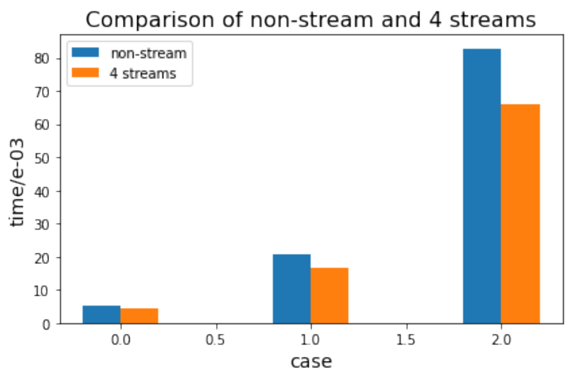


Figure 1: comparison of 4 streams

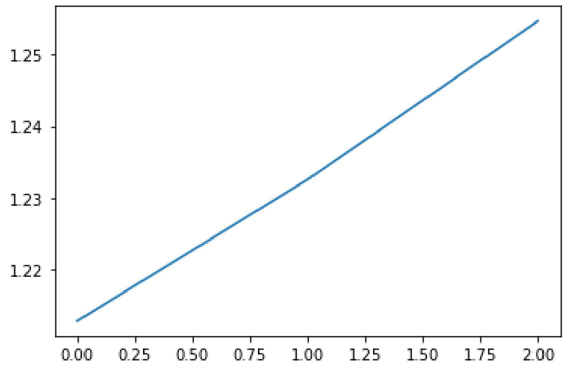


Figure 2: Performance improvement

For non-stream and 4 streams cuda, I set the size of the input vector as 2560000, 10240000 and 40980000. In this range, a larger size of the vector will lead to better improvement of the performance, and it seems to be linear within this range.

2. Use nvprof to collect traces and the NVIDIA Visual Profiler (nvvp) to visualize the overlap of communication and computation. To use nvvp, you can check Tutorial: NVVP - Visualize nvprof Traces

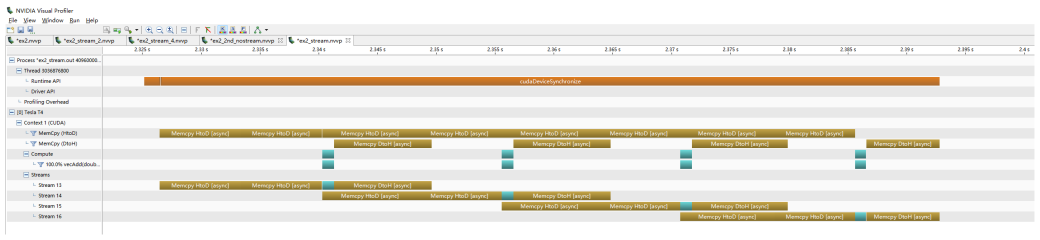


Figure 3: Overlap of communication and computation

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)

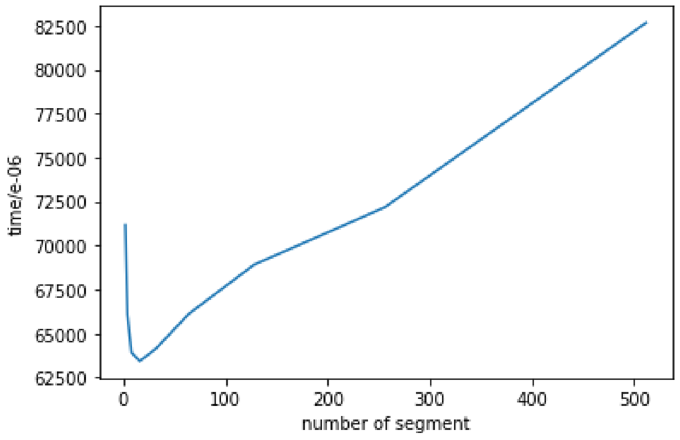


Figure 4: Impact of segment size

From the figure, we can see that at first, more streams could help to improve the performance, as it could be more parallel to save time. However, too many streams will spend more time on task blocking and switching, and especially in this vector addition program, most of the time is spent on memory copy, then too many streams will cost more time, even more time than non-stream Cuda.

3 Exercise 3 - Pinned Memory and Unified Memory

1. What are the differences between pageable memory and pinned memory, what are the tradeoffs?

   Answer: Pageable memory is allowed to be paged in or paged out, while pinned memory is not. Too much pageable memory will cause worse performance, as GPU can not directly access it, but pageable memory could extend the space of memory to run more programs.

2. Compare the profiling results between your original code and the new version using pinned memory. Do you see any difference in terms of the breakdown of execution time after changing to pinned memory?

   Answer: For the original code, I tested with matrix lengths of 128, 256, 512, and 1024. For the new version of pinned memory, there is no significant difference considering kernel execution. But for memory copy, it spends much less when using pinned memory when the vector length is large.

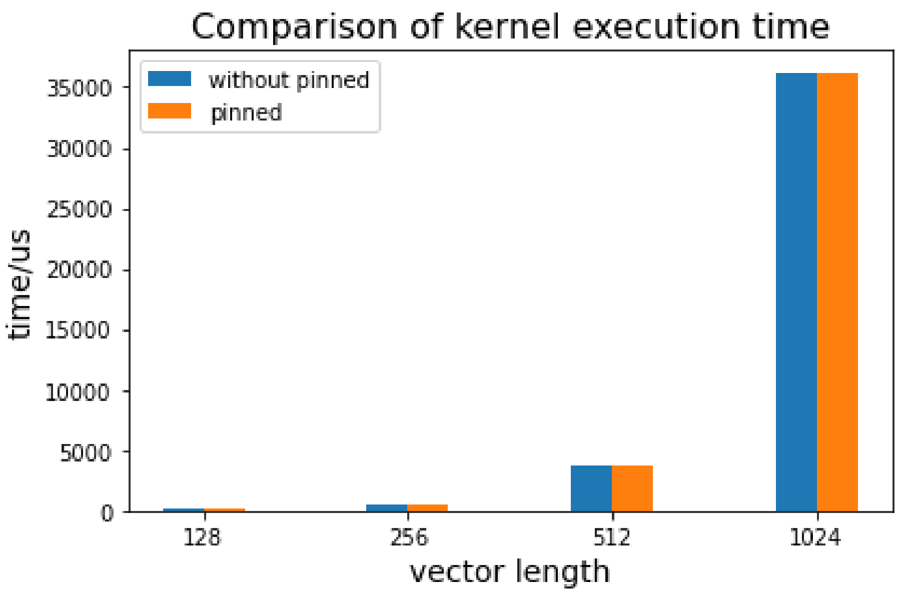


Figure 5: Comparison of kernel execution time

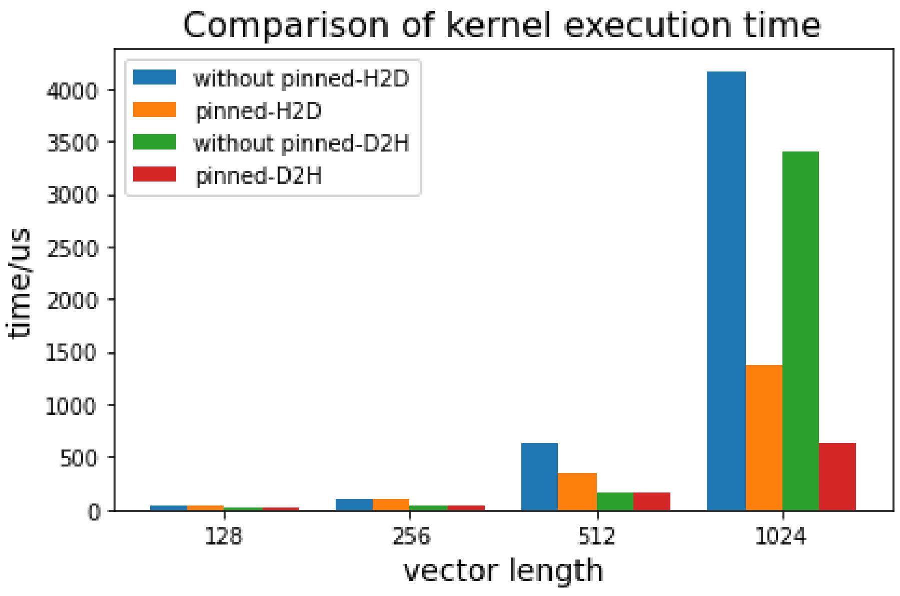


Figure 6: Comparison of memory copy time

As pinned memory on the host allows us to avoid intermediate transfers and achieve higher data transfer rate (bandwidth), it is more efficient when data is larger.

3. What is a managed memory? What are the implications of using managed memory?

   Answer: Managed memory is a single memory address space which is accessible from any processor in a system. Allocate data can be read or written from cude running on either CPUs or GPUs, and the CUDA system software takes care of migrating memory pages to the memory of the accessing processor.

4. Compare the profiling results between your original code and the new version using managed memory. What do you observe in the profiling results?

   Answer: For the version using unified memory, the execution time of the kernel increases significantly, because of the host-to-device page faults, reducing the throughput achieved by the CUDA kernel. So I tried to use Unified Memory prefetching to move the data to the GPU after initializing it, using cudaMemPrefetchAsync() for it. After that, the execution time of kernel is similar to the origin version, and it improves the performance because it doesn't need to copy the data from host to device or device to host.

4 Exercise 4 - 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: For each sparse matrix, there are $3 \times (dimX - 2)$ non-zero elements in total, so the FLOP can be calculated as the equation 1.

   $$FLOP = \frac{no\_timesteps * (3 \times (dimX - 2))}{time_{SMPV}} \tag{1}$$

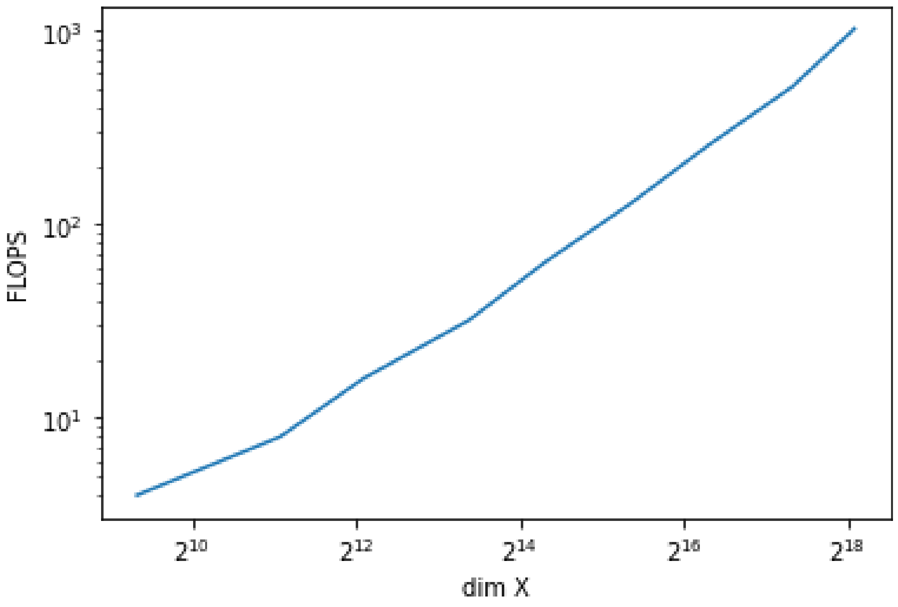
   

   Figure 7: FLOPS at different input sizes

   As the figure shown, the larger input size is, the larger FLOP is. However, it will stay stable even increasing the input size, and from the experiments it cannot reach the peak throughput.

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: A larger number of timesteps is, a smaller value of relative error is, because the Finite Differences Method can be more accurate to estimate the Heat equation.

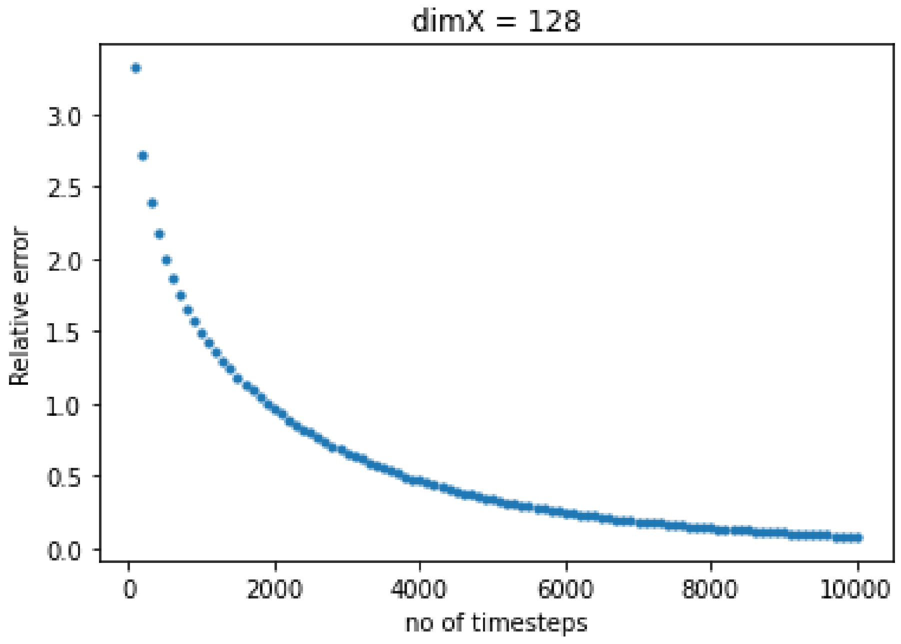
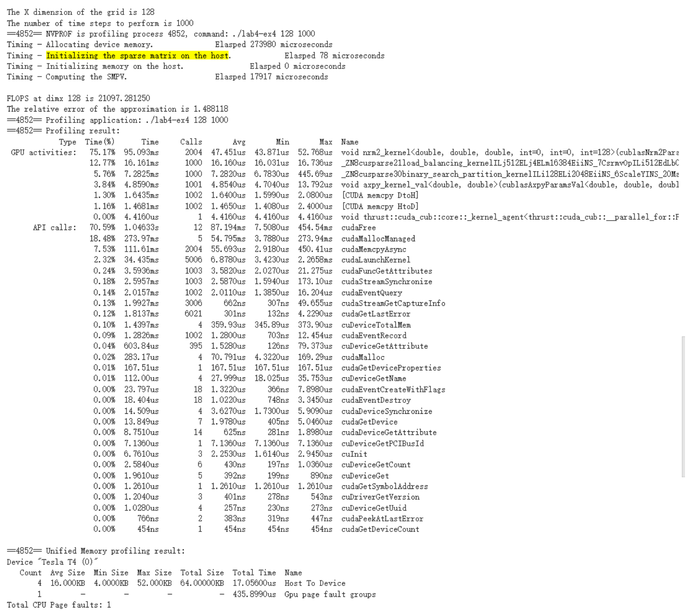


Figure 8: Relative error at different number of steps

3. Compare the performance with and without the prefetching in Unified Memory. How is the performance impact? [Optional: using nvprof to get metrics on UM]

   Answer: With prefetching in Unified Memory, the execution time of computing the SMPV decreases from 17917 to 14948 ms, because there is no GPU page fault group, and it is faster to initialize the sparse matrix on the host. And prefechting only takes 46.664 μs, so it could improve the performance.

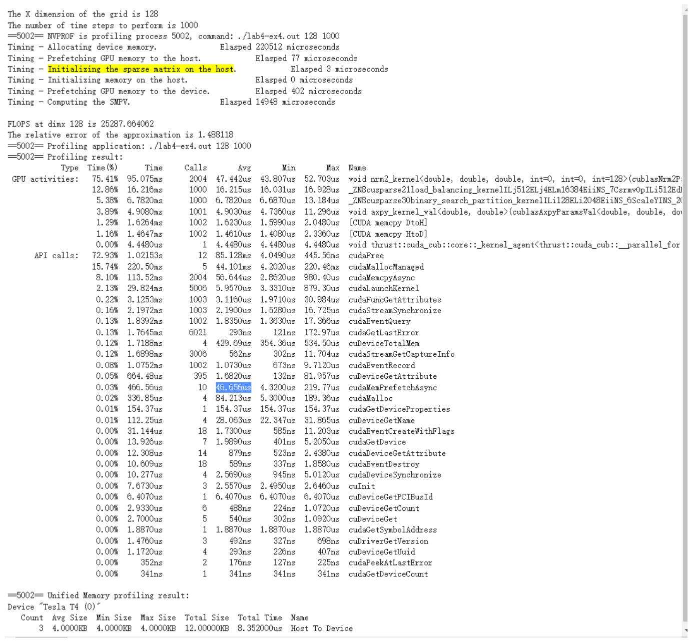


```
The X dimension of the grid is 128
The number of time steps to perform is 1000
==4852== NVPROF is profiling process 4852, command: ./lab4-ex4 128 1000
Timing - Allocating device memory.              Elasped 273980 microseconds
Timing - Initializing the sparse matrix on the host.            Elasped 78 microseconds
Timing - Initializing memory on the host.               Elasped 0 microseconds
Timing - Computing the SMPV.                    Elasped 17917 microseconds

FLOPS at dimx 128 is 21097.281250
The relative error of the approximation is 1.488118
==4852== Profiling application: ./lab4-ex4 128 1000
==4852== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   75.17%  95.093ms      2004  47.451us  43.871us  52.768us  void nrm2_kernel<double, double, double, int=0, int=0, int=128>(cublasNrm2Para
                   12.77%  16.161ms      1000  16.160us  16.031us  16.736us  _ZN8cusparse21load_balancing_kernelILj512ELj4ELm16384EiiNS_7CsrmvOpILi512EdLbC
                    5.76%  7.2825ms      1000  7.2820us  6.7830us  445.69us  _ZN8cusparse30binary_search_partition_kernelILi128ELi2048EiiNS_6ScaleYINS_20Ma
                    3.84%  4.8590ms      1001  4.8540us  4.7040us  13.792us  void axpy_kernel_val<double, double>(cublasAxpyParamsVal<double, double, doubl
                    1.30%  1.6435ms      1002  1.6400us  1.5990us  2.0800us  [CUDA memcpy DtoH]
                    1.16%  1.4681ms      1002  1.4650us  1.4080us  2.4000us  [CUDA memcpy HtoD]
                    0.00%  4.4160us         1  4.4160us  4.4160us  4.4160us  void thrust::cuda_cub::core::_kernel_agent<thrust::cuda_cub::__parallel_for::P
      API calls:   70.59%  1.04633s        12  87.194ms  7.5080us  454.54ms  cudaFree
                   18.48%  273.97ms         5  54.795ms  3.7880us  273.94ms  cudaMallocManaged
                    7.53%  111.61ms      2004  55.693us  2.9180us  450.41us  cudaMemcpyAsync
                    2.32%  34.435ms      5006  6.8780us  3.4230us  2.2658ms  cudaLaunchKernel
                    0.24%  3.5936ms      1003  3.5820us  2.0270us  21.275us  cudaFuncGetAttributes
                    0.18%  2.5957ms      1003  2.5870us  1.5940us  173.10us  cudaStreamSynchronize
                    0.14%  2.0157ms      1002  2.0110us  1.3850us  16.204us  cudaEventQuery
                    0.13%  1.9927ms      3006     662ns     307ns  49.655us  cudaStreamGetCaptureInfo
                    0.12%  1.8137ms      6021     301ns     132ns  4.2290us  cudaGetLastError
                    0.10%  1.4397ms         4  359.93us  345.89us  373.90us  cuDeviceTotalMem
                    0.09%  1.2826ms      1002  1.2800us     703ns  12.454us  cudaEventRecord
                    0.04%  603.84us       395  1.5280us     126ns  79.373us  cuDeviceGetAttribute
                    0.02%  283.17us         4  70.791us  4.3220us  169.29us  cudaMalloc
                    0.01%  167.51us         1  167.51us  167.51us  167.51us  cudaGetDeviceProperties
                    0.01%  112.00us         4  27.999us  18.025us  35.753us  cuDeviceGetName
                    0.00%  23.797us        18  1.3220us     366ns  7.8980us  cudaEventCreateWithFlags
                    0.00%  18.404us        18  1.0220us     748ns  3.3450us  cudaEventDestroy
                    0.00%  14.509us         4  3.6270us  1.7300us  5.9090us  cudaDeviceSynchronize
                    0.00%  13.849us         7  1.9780us     405ns  5.0460us  cudaGetDevice
                    0.00%  8.7510us        14     625ns     281ns  1.8980us  cudaDeviceGetAttribute
                    0.00%  7.1360us         1  7.1360us  7.1360us  7.1360us  cuDeviceGetPCIBusId
                    0.00%  6.7610us         3  2.2530us  1.6140us  2.9450us  cuInit
                    0.00%  2.5840us         6     430ns     197ns  1.0360us  cuDeviceGetCount
                    0.00%  1.9610us         5     392ns     199ns     890ns  cuDeviceGet
                    0.00%  1.2610us         1  1.2610us  1.2610us  1.2610us  cudaGetSymbolAddress
                    0.00%  1.2040us         3     401ns     278ns     543ns  cuDriverGetVersion
                    0.00%  1.0280us         4     257ns     230ns     273ns  cuDeviceGetUuid
                    0.00%     766ns         2     383ns     319ns     447ns  cudaPeekAtLastError
                    0.00%     454ns         1     454ns     454ns     454ns  cudaGetDeviceCount

==4852== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
       4  16.000KB  4.0000KB  52.000KB  64.00000KB  17.05600us  Host To Device
       1         -         -         -           -  435.8990us  Gpu page fault groups
Total CPU Page faults: 1
```

Figure 9: Without prefetching



```
The X dimension of the grid is 128
The number of time steps to perform is 1000
==5002== NVPROF is profiling process 5002, command: ./lab4-ex4.out 128 1000
Timing - Allocating device memory.              Elasped 220512 microseconds
Timing - Prefetching GPU memory to the host.            Elasped 77 microseconds
Timing - Initializing the sparse matrix on the host.            Elasped 3 microseconds
Timing - Initializing memory on the host.               Elasped 0 microseconds
Timing - Prefetching GPU memory to the device.          Elasped 402 microseconds
Timing - Computing the SMPV.                    Elasped 14948 microseconds

FLOPS at dimx 128 is 25287.664062
The relative error of the approximation is 1.488118
==5002== Profiling application: ./lab4-ex4.out 128 1000
==5002== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   75.41%  95.075ms      2004  47.442us  43.807us  52.703us  void nrm2_kernel<double, double, double, int=0, int=0, int=128>(cublasNrm2P
                   12.86%  16.216ms      1000  16.215us  16.031us  16.928us  _ZN8cusparse21load_balancing_kernelILj512ELj4ELm16384EiiNS_7CsrmvOpILi512Ed
                    5.38%  6.7820ms      1000  6.7820us  6.6870us  13.184us  _ZN8cusparse30binary_search_partition_kernelILi128ELi2048EiiNS_6ScaleYINS_2
                    3.89%  4.9080ms      1001  4.9030us  4.7360us  11.296us  void axpy_kernel_val<double, double>(cublasAxpyParamsVal<double, double, do
                    1.29%  1.6264ms      1002  1.6230us  1.5990us  2.0480us  [CUDA memcpy DtoH]
                    1.16%  1.4647ms      1002  1.4610us  1.4080us  2.3360us  [CUDA memcpy HtoD]
                    0.00%  4.4480us         1  4.4480us  4.4480us  4.4480us  void thrust::cuda_cub::core::_kernel_agent<thrust::cuda_cub::__parallel_for
      API calls:   72.93%  1.02153s        12  85.128ms  4.0490us  445.56ms  cudaFree
                   15.74%  220.50ms         5  44.101ms  4.2020us  220.46ms  cudaMallocManaged
                    8.10%  113.52ms      2004  56.644us  2.8620us  980.40us  cudaMemcpyAsync
                    2.13%  29.824ms      5006  5.9570us  3.3310us  879.30us  cudaLaunchKernel
                    0.22%  3.1253ms      1003  3.1160us  1.9710us  30.984us  cudaFuncGetAttributes
                    0.16%  2.1972ms      1003  2.1900us  1.5280us  16.725us  cudaStreamSynchronize
                    0.13%  1.8392ms      1002  1.8350us  1.3630us  17.366us  cudaEventQuery
                    0.13%  1.7645ms      6021     293ns     121ns  172.97us  cudaGetLastError
                    0.12%  1.7188ms         4  429.69us  354.36us  534.50us  cuDeviceTotalMem
                    0.12%  1.6898ms      3006     562ns     302ns  11.704us  cudaStreamGetCaptureInfo
                    0.08%  1.0752ms      1002  1.0730us     673ns  9.7120us  cudaEventRecord
                    0.05%  664.48us       395  1.6820us     132ns  81.957us  cuDeviceGetAttribute
                    0.03%  466.56us        10  46.656us  4.3200us  219.77us  cudaMemPrefetchAsync
                    0.02%  336.85us         4  84.213us  5.3000us  189.36us  cudaMalloc
                    0.01%  154.37us         1  154.37us  154.37us  154.37us  cudaGetDeviceProperties
                    0.01%  112.25us         4  28.063us  22.347us  31.865us  cuDeviceGetName
                    0.00%  31.144us        18  1.7300us     585ns  11.203us  cudaEventCreateWithFlags
                    0.00%  13.926us         7  1.9890us     401ns  5.2050us  cudaGetDevice
                    0.00%  12.308us        14     879ns     523ns  2.4380us  cudaDeviceGetAttribute
                    0.00%  10.609us        18     589ns     337ns  1.8580us  cudaEventDestroy
                    0.00%  10.277us         4  2.5690us     945ns  5.0120us  cudaDeviceSynchronize
                    0.00%  7.6730us         3  2.5570us  2.4950us  2.6460us  cuInit
                    0.00%  6.4070us         1  6.4070us  6.4070us  6.4070us  cuDeviceGetPCIBusId
                    0.00%  2.9330us         6     488ns     224ns  1.0720us  cuDeviceGetCount
                    0.00%  2.7000us         5     540ns     302ns  1.0920us  cuDeviceGet
                    0.00%  1.8870us         1  1.8870us  1.8870us  1.8870us  cudaGetSymbolAddress
                    0.00%  1.4760us         3     492ns     327ns     698ns  cuDriverGetVersion
                    0.00%  1.1720us         4     293ns     226ns     407ns  cuDeviceGetUuid
                    0.00%     352ns         2     176ns     127ns     225ns  cudaPeekAtLastError
                    0.00%     341ns         1     341ns     341ns     341ns  cudaGetDeviceCount

==5002== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
       3  4.0000KB  4.0000KB  4.0000KB  12.00000KB  8.352000us  Host To Device
```

Figure 10: With prefetching