CS 610 Semester 2024–2025-I
Assignment 4

Divyansh
(210355)

5th Nov 2024

Note:
1. The makefile for each the problems is in their respective directory.
2. In many cases results varied probably because of the load on gpu server. I have tried to reduce those noise by calculating for higher values of N, wherever possible.

References used:[1, 2, 3, others].

1 Problem 1

Time calculated for `N = 128`.

1.1 Naive and Optimized Kernel

```
Stencil time on CPU: 15.0049 msec
(Naive) Kernel 1 time (ms): 9.19667
(Optimized) Kernel 2 time (ms): 4.18781
```

Analysis: The optimized kernel wins mostly because of two reasons here: first being the use of shared memory leading to faster element access and second one is multiple elements computed per thread leading to lesser bank-conflicts and kernel-launch overheads.

1.2 Loop-unrolling

```
Stencil time on CPU: 14.858 msec
Kernel(Loop-unrolling) time (ms): 5.0185
```

Analysis: For loop unrolling I had two loops in my cuda kernel which were iterating over the z and y direction(direction as in the kernel grid) to cover all points one thread had to handle. So, I unrolled the inner loop with the idea that the compiler optimization might improve the performance but the opposite to what expected the performance remain almost similar or somewhat poorer as compared to the above optimized version. One probale reason for that can be register pressure.

1.3 Loop-unrolling and Pinned Memory

```
Stencil time on CPU: 16.8529 msec
Kernel(Loop-unrolling + pinned) time (ms): 3.49139
```

Analysis: Pinning the memory in the host before copying to host clearly improves the memory bandwidth leading performance gains as compared to just Loop-unrolling.

1.4 Loop-unrolling and UVM

```
Stencil time on CPU: 13.571 msec
Kernel(Loop-unrolling and UVM) time (ms): 18.901
```

Analysis: UVM performs poor than pinned memory(as in the previous case) probably because of the fault overhead getting generated and the memory transfer is not staged as in the case of pinned memory. The profiler showed multiple page-faults

```
360.86ms  1.8560us -  NVIDIA GeForce -  4.000000KB   0x78cdf9940000  [Unified Memory Memcpy DtoH]
360.86ms  10.144us -  NVIDIA GeForce    60.000000KB   0x78cdf9941000  [Unified Memory Memcpy DtoH]
360.89ms         - -               -    PC 0xd9098f59   0x78cdf9960000  [Unified Memory CPU page faul
360.91ms  1.5670us  - NVIDIA GeForce    4.000000KB   0x78cdf9960000  [Unified Memory Memcpy DtoH]
360.91ms  9.8570us  - NVIDIA GeForce    60.000000KB   0x78cdf9961000  [Unified Memory Memcpy DtoH]
360.95ms     -               -        - PC 0xd9098f59   0x78cdf9980000  [Unified Memory CPU page faul
360.96ms  1.5360us -  NVIDIA GeForce    4.000000KB   0x78cdf9980000  [Unified Memory Memcpy DtoH]
360.96ms  9.9210us  - NVIDIA GeForce    60.000000KB   0x78cdf9981000  [Unified Memory Memcpy DtoH]
```

2 Problem 3

2.0.1 Performance Comparison

```
Kernel(Vanilla Port) time (ms): 55813.6
Kernel(Optimized) time (ms): 9994.4
Kernel(Optimized + UVM) time (ms): 18808.7
```

Solution Idea: The vanilla code is done by mapping the outer most loops of the gridsearch function to the grids of cuda kernel. This part wasn't that tricky, the main issue was to know the number of condition satisfying points beforehand so that I can allocate the memory for them before launching the cuda kernel helping me to maintain the thread-wise order of the points when writing into the file. The best solution I could come up with was that I will launch the kernel two times, the first launch calculates the valid number of points for each of the threads individually.
This thread-wise count helps into allocating minimal amount of memory to the buffer in which the points are being written resulting in optimal memory copying overhead and hence better performance(However, the vanilla code also uses this optimization. Further, we can pre-compute the initial values and store them into constant memory so that all the threads need not to compute them again and again.
Switching to UVM with the optimized is better than the vanilla performance but it clearly downgrades the performance as in the just optimized version. The reason for this as in the profiler:

```
275.84ms  244.10us..  -  NVIDIA GeForce 15   0x7b4c60000000  [Unified Memory GPU page faults]

681.13ms  82.977us.. -  NVIDIA GeForce  5   0x7b4c60010000  [Unified Memory GPU page faults]

1.32114s  76.354us -  NVIDIA GeForce  3   0x7b4c60020000  [Unified Memory GPU page faults]

2.36530s  96.577us -  NVIDIA GeForce      1   0x7b4c60040000  [Unified Memory GPU page faults]

4.60860s  101.63us -  NVIDIA GeForce    1   0x7b4c60080000  [Unified Memory GPU page faults]

9.09444s  155.84us -  NVIDIA GeForce    13   0x7b4c60100000  [Unified Memory GPU page faults]
```

3 Problem 4

Time calculated for `N = 512`

```
2D Convolution time on CPU: 6.58894 msec
Kernel2D time (ms): 0.09312
kernel2D(Optimized) time (ms): 0.291107

====================

3D Convolution time on CPU: 10866.1 msec
Kernel3D time (ms): 22.5751
kernel3D(Optimized) time (ms): 10.3476
```