|  |  |
| --- | --- |
| **Name:** | *Zheyu Fu* |
| **NetID:** | *zheyufu2* |
| **Section:** | *ZJUI section* |

**ECE 408/CS483 Milestone 3 Report**

|  |
| --- |
| 1. List Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images from your basic forward convolution kernel in milestone 2. This will act as your baseline this milestone. |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *0.268992ms* | *1.22407 ms* | *2.336s* | *0.86* | | 1000 | *2.64686ms* | *11.049 ms* | *11.598s* | *0.886* | | 10000 | *26.2595ms* | *225.062 ms* | *3m31.496s* | *0.8714* |   *Baseline nsys:*    *Baseline Nsight-Compute:* |
| 1. **Optimization 1: Weight matrix (kernel values) in constant memory (1 point)** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| **Weight matrix (kernel values) in constant memory**  *Because it is rather simple but also useful. I just need to declare constant memory for storing kernel values.* |
| * 1. How does the optimization work? Did you think the optimization would increase performance of the forward convolution? Why? Does the optimization synergize with any of your previous optimizations?   Reading from global memory is very slow. In this optimization, kernel values are read from constant memory, which is faster.  Yes, I think the optimization would increase performance of the forward convolution, because once we store the kernel value in constant memory, later we don’t need to read kernel from global memory but from constant memory.  No, it doesn’t synergize with any previous optimization. It was just built based on baseline. |
|  |
| * 1. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used). |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *0.242638ms* | *1.09994ms* | *2.37s* | *0.86* | | 1000 | *2.2353ms* | *7.9752ms* | *9.213s* | *0.886* | | 10000 | *22.5069ms* | *70.9055ms* | *1m56.185s* | *0.8714* | |
| * 1. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of). |
| *Yes, implementing this optimization successful in improving performance.*  *From nsys profiling we can see that conv\_forward\_kernel cost 93.3683ms，*  *which is faster than baseline: 252.26ms*    Also, from Nsight compute, the profiling result shows that SM % becomes higher with  the help of constant memory.      *The memory throughput also increases from 147.07 to 165.98 compared to baseline.* |
| * 1. What references did you use when implementing this technique? |
| *From lectures.* |
| 1. **Optimization 2: Input channel reduction: tree (3 point)** |
| 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| Input channel reduction: tree  Because I thought channel reduction using tree could be efficient. Also, I can get some Implementation reference from lecture. |
| 1. How does the optimization work? Did you think the optimization would increase performance of the forward convolution? Why? Does the optimization synergize with any of your previous optimizations? |
| *Before applying input channel reduction by tree, each thread needs to go through three*  *for loop, while after applying, each thread just need to go through two for loop plus several steps to add up the results from different channel. On the whole, applying the method reduces time complexity and the optimization works.*  *Yes, I think the optimization* would increase performance of the forward convolution with the *previous reason stated.*  *Yes, it synergizes with Optimization 1.* |
| 1. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used). |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *1.83567ms* | *2.56969ms* | *2.416s* | *0.86* | | 1000 | *5.9846ms* | *9.83527ms* | *19.232s* | *0.886* | | 10000 | *19.4723ms* | *99.9019ms* | *1m59.733s* | *0.8714* | |
| 1. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of). |
| *We see that conv\_forward\_kernel time doesn’t shrink compared to optimization 1, and it is not successful.*  *Now: 119.37ms*  *Optimization1: 93.3683ms* |
| In Nsight-compute analysis, we see that SM % almost doesn’t change compared to optimization 1. However, Memory % increases from 50.21% to 57.86%.    Memory throughput increases from 165.98 to 226.76.     1. What references did you use when implementing this technique? |
|  |
| *From textbooks and textbook as well as google.* |

|  |
| --- |
| 1. **Optimization 3: Input channel reduction: atomics (2 point)**   ***(Delete this section blank if you did not implement this many optimizations.)*** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *Input channel reduction: atomics (2 point)*  *Because this method is very similar to channel reduction by tree that has been done by me in optimization 2, so I decided to try this method as well.* |
| * 1. How does the optimization work? Did you think the optimization would increase performance of the forward convolution? Why? Does the optimization synergize with any of your previous optimizations? |
| *It is very similar to optimization 2.*  *Before applying input channel reduction by tree, each thread needs to go through three for loop, while after applying, each thread just need to go through two for loop plus several steps to add up the results from different channel. On the whole, applying the method reduces time complexity and the optimization works.*  *Yes, I think the optimization* would increase performance of the forward convolution with the *previous reason stated.*  *Yes, it synergizes with Optimization 1.* |
| * 1. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used). |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *0.18692ms* | *0.840482ms* | *0m1.863s* | *0.86* | | 1000 | *1.93856ms* | *8.945328ms* | *0m10.397s* | *0.886* | | 10000 | *16.2979ms* | *98.0769ms* | *1m45.978s* | *0.8714* | |
| * 1. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).   Time for conv\_forward\_kernel is 114.375ms, however, for previous optimization 1, the data is 93.3683ms. So actually this optimization isn’t very successful. |
| *I find that this optimization increases GPU utilization compared to* optimization 1.  *With SM % changes from 80.59% to 82.12%.*  *With Memory% changes from 50.21%, 58.55%*    *Also, memory throughput increases from 165.89 to 294.19.*    *However, atomicAdd() could be an expensive step to do, which potentially caused the run time increases.* |
| * 1. What references did you use when implementing this technique? |
| *From textbooks and lectures as well as google.* |
| 1. **Optimization 4: Using Streams to overlap computation with data transfer (4 points)**   ***(Delete this section blank if you did not implement this many optimizations.)*** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *Using Streams to overlap computation with data transfer (4 points).*  *Because this technique focuses on device overlap, which is different from all techniques I applied before. I think it could be interesting to have overlapping implemented, so I just try it.* |
| * 1. How does the optimization work? Did you think the optimization would increase performance of the forward convolution? Why? Does the optimization synergize with any of your previous optimizations?   It can take effects because it helps overlap overlap transfer and compute of adjacent segments. Operations in different streams can execute in parallel and it is a version of task parallelism.  *Yes, I think the optimization* would increase performance of the forward convolution with the *previous reason stated.*  *Yes, it synergizes with Optimization 1.* |
|  |
| * 1. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used). |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *0.007582ms* | *0.009283ms* | *0m1.429s* | *0.86* | | 1000 | *0.009462ms* | *0.01252ms* | *0m9.274s* | *0.886* | | 10000 | *0.01829ms* | *0.01323ms* | *0m41.238s* | *0.8714* | |
| * 1. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).   From nsys we can see that the time for conv\_forward\_kernel is 106.1979ms. For optimization 1, this data is 93.3683ms. So actually this optimization isn’t successful. |
| *Gpu utilization reduces a lot. SOL SM reduces from 80.59 to 69.54, while SOL memory reduces from 50.21% to 44.23%*    *Memory throughput drops significantly from 165.98 to 12.04, due to the overlapping optimization.* |
| * 1. What references did you use when implementing this technique? |
| *From textbooks and textbook as well as google.* |
| 1. **Optimization 5:** **Sweeping various parameters to find best values (block sizes, amount of thread coarsening) (1 point). *Best performance.***   ***(Delete this section if you did not implement this many optimizations.)*** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *Sweeping various parameters to find best values (block sizes, amount of thread coarsening) (1 point)*  *Because I think tuning parameters is very essential for this project and have a potential in increasing the performance a lot, so I just try it.* |
| * 1. How does the optimization work? Did you think the optimization would increase performance of the forward convolution? Why? Does the optimization synergize with any of your previous optimizations? |
| *I mainly tune the TILE\_WIDTH, leading to the change of dimGrid and dimBlock. Before tuning, my TILE\_WIDTH stays at 16. However, by sweeping several possible TILE\_WIDTHs, it is easy to generalize the best choice of TILE\_WIDTH since there is always a trade-off for TILE\_WIDTH being too high or too low.*  *Yes, I think the optimization* would increase performance of the forward convolution with the *previous reason stated.*  *Yes, it synergizes with Optimization 1.* |
| * 1. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used). |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *1.13282ms* | *1.23829ms* | *0m5.123s* | *0.86* | | 1000 | *2.75962ms* | *5.24797ms* | *0m13.125s* | *0.886* | | 10000 | ***21.2557ms*** | ***48.7562ms*** | 2m20.791ms | *0.8714* | |
| * 1. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).   *Finally, I fix the TILE\_WIDTH to 12, and t*he implementation is very successful, making Op Time reduced a lot.  *From nsys profiling we can see that conv\_forward\_kernel costs 69.40ms, while in optimization 1, conv\_forward\_kernel costs 93.368ms.* |
| *GPU utilization:*  *SOL SM% increases from 50.21% to 63.51% compared to optimization1.*  *Compute workload analysis:*    *Memory workload analysis:*    *Memory workload increases from 165.98 to 275.81.* |
| * 1. What references did you use when implementing this technique? |
| *No reference, as tuning parameters are easy.* |

To summarize, I complete 5 kinds of optimization:

• Weight matrix (kernel values) in constant memory (1 point)

• Input channel reduction: tree (3 point)

• Input channel reduction: atomics (2 point)

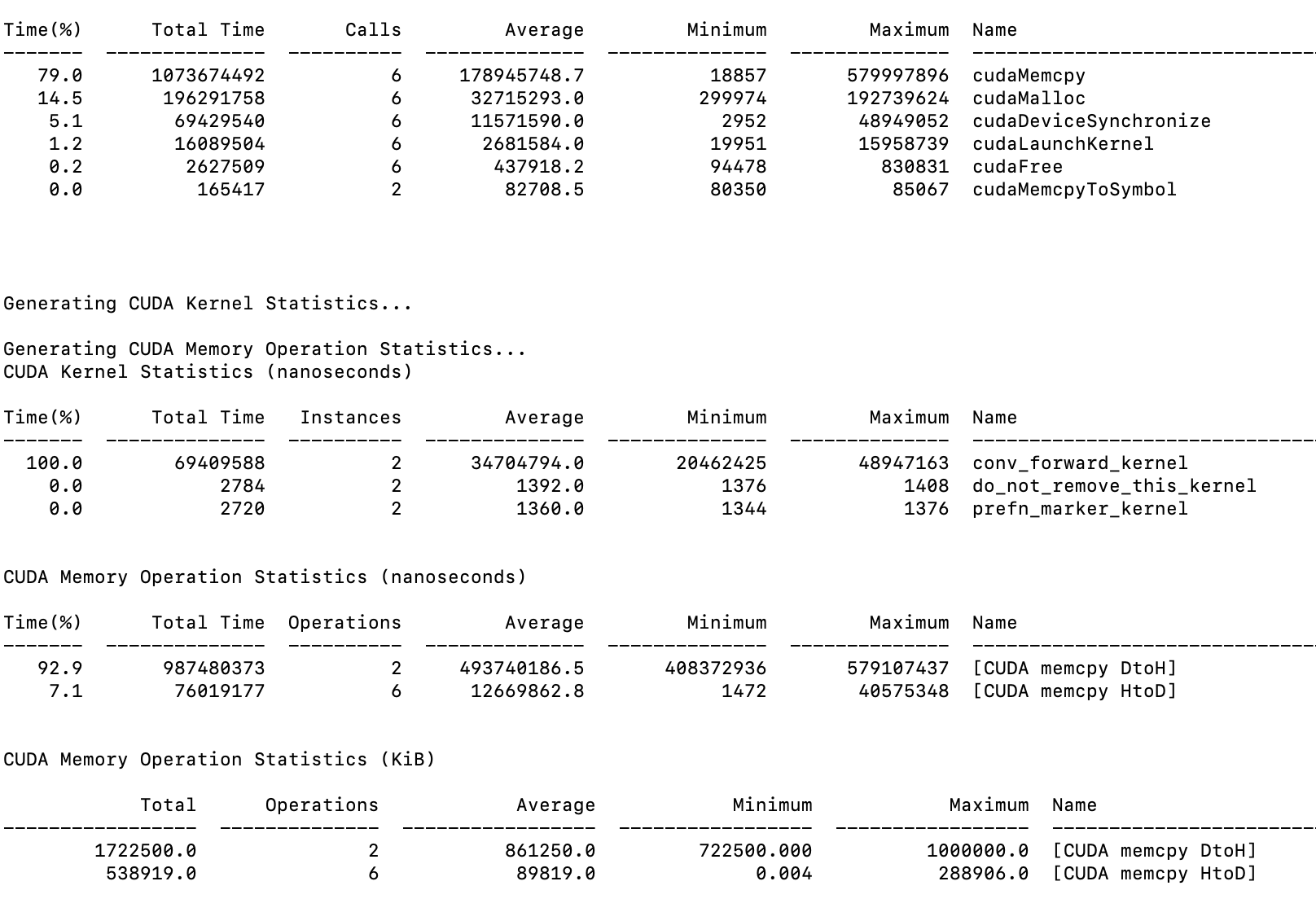
• Using Streams to overlap computation with data transfer (4 points)

• Sweeping various parameters to find best values (block sizes, amount of thread coarsening) (1 point)

In total: 11 points.

The best performance happens when doing both of Sweeping various parameters to find best values and Weight matrix (kernel values) in constant memory.

Shortest time for conv\_forward\_kernel: 69.4096ms, as picture below.

**