|  |  |
| --- | --- |
| **Name:** | *Wenjun Wu* |
| **NetID:** | *wenjun5* |
| **Section:** | *AL1* |

**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 | *0.243959 ms* | *0.876509 ms* | *0m1.258s* | *0.86* | | 1000 | *2.28637 ms* | *8.52961 ms* | *0m9.773s* | *0.886* | | 10000 | *22.4504 ms* | *84.8044 ms* | *1m35.581s* | *0.8714* |   The following is the results of profiling and Nsight-Compute:  baseline_data |
| 1. **Optimization 1: Tiled shared memory convolution (2 points)** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| I choose to implement the optimization: Tiled shared memory convolution.  Reason: How to use shared memory is taught in the MP4 in this class, so I am very familiar with this method and it does improve the speed of convolution. |
| * 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 think by using shared memory, we can increase memory reuse, and because accessing to the shared memory is more efficient than global memory. So, initially, I thought that it would increase the performance of the forward convolution. However, when testing I found that this optimization actually can’t improve the performance. |
| * 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 | *0.350216 ms* | *1.36298 ms* | *0m1.602s* | *0.86* | | 1000 | *3.32689 ms* | *13.3513 ms* | *0m9.973s* | *0.886* | | 10000 | *32.7419 ms* | *128.071 ms* | *1m37.672s* | *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). |
| This optimization did not improve performance. Here is the profilling result:  *op1_share_memo_data*  the total time for conv\_forward\_kernel increases from 108216435 to 168307599, which means that it becomes slower than before. It’s true that using shared\_memory reduce the GPU utilization of Memory:    and the access to shared memory is faster than global memory:    But the difference between using shared\_memory and baseline is not large and the declaration needs time: |
| * 1. What references did you use when implementing this technique? |
| *I am using the knowledge that showed in the lectures and the code in MP3 and MP4.* |
| 1. **Optimization 2: 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. |
| I choose to implement the optimization: Weight matrix (kernel values) in constant memory.  Reason: this optimization is easy to implement and it my improve the speed, because by defining constant memory to store mask, we can reduce the time by reading the mask from the global memory. |
| 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? |
| *This optimization work because the access to the constant memory is much faster than the global memory and reduce the bandwidth for accessing the global memory. I think it will* increase performance of the forward convolution because once we store the mask into the constant value, it will be accessed a lot of time without asking the global memory as it did before. This optimization does not synergize with my previous optimizations. |
| 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 | *0.162261 ms* | *0.662555 ms* | *0m1.179s* | *0.86* | | 1000 | *1.47058 ms* | *6.45876 ms* | *0m9.786s* | *0.886* | | 10000 | *14.5587 ms* | *64.4992 ms* | *1m36.276s* | *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). |
| This optimization improves performance. Here is the profilling result:  *op4_constant_memo*  it reduced the conv\_forward\_kernel time from 108216435 to 78377879. The following shows the profiling result from Nsight-Compute:    It turns out that the SM is even more efficient, compared to the result of baseline. And the usage of the memory is reduced compared to the baseline.  Here is the compute workload analysis. Compared to the baseline, the utilization incresed dramatically to reach a higher performance:    Here is the memory workload analysis. the Memory Thoughput increses from 58 to 91, nearly double the original output speed. |
| 1. What references did you use when implementing this technique? |
| The lectures, textbook and google. |

|  |
| --- |
| **Optimization 3: Tuning with restrict and loop unrolling (3 points)** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| I choose to implement the optimization: Tuning with restrict and loop unrolling.  Reason: loop unrolling will increase a program's speed by reducing or eliminating instructions that control the loop. |
| * 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? |
| *This optimization work because it* reduces instructions that control the loop. *I think it will* increase performance of the forward convolution because once we the instructions in kernal code reduce, the total time for running this kernal will reduce as well. This optimization does synergize with using shared memory. |
| * 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 | *0.116158 ms* | *0.590386 ms* | *0m1.180s* | *0.86* | | 1000 | *1.02903 ms* | *5.92734 ms* | *0m9.939s* | *0.886* | | 10000 | *10.1514 ms* | *59.2848 ms* | *1m33.919s* | *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). |
| This optimization improves performance. Here is the profilling result:  *op5_tuning*  it reduced the conv\_forward\_kernel time from 78377879 to 66772519. The following shows the profiling result from Nsight-Compute:    It turns out that the SM is more efficient than only using the constant memory(83.80). And the usage of the memory is incresed from 71.42 to 97.15.  Here is the compute workload analysis. it reduces SM busy rate from 83.82 to 43.94, but increse the utilization of ADU from 59.62 to 85.55:    Here is the memory workload analysis. the Memory Thoughput increses from 91 to 130. |
| * 1. What references did you use when implementing this technique? |
| The lectures, textbook and google. |
| **Optimization 4: Sweeping various parameters to find best values (block sizes, amount of thread coarsening) (1 point)** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| I choose to implement the optimization: Sweeping various parameters to find best values.  Reason: this optimization is easy to implement and it may increse the performance of Convolution because the performance of different tile widths differs from each other. |
| * 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? |
| The original is TILE\_WIDTH, but there will be another better TILE\_WIDTH which can reach a higher performance. So, I want to try different possible TILE\_WIDTHs to find the best TILE\_WIDTH to minimize the layer time. I think this optimization would increase performance of the forward convolution because there are various of numbers for me to choose. The optimization synergizes with the constant memory and tuning. |
| * 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 | *0.142651 ms* | *0.416398 ms* | *1.27938 ms* | *0.86* | | 1000 | *1.27938 ms* | *3.99256 ms* | *0m9.744s* | *0.886* | | 10000 | *12.5441 ms* | *39.582 ms* | *1m35.841s* | *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). |
| Althrough this optimization increses the first op time, but it dramatically decrese the second op time. Here is the profilling result:  *op6_sweep_parameters*  it reduced the conv\_forward\_kernel time from 66772519 to 52342229. The following shows the profiling result from Nsight-Compute:    It turns out that the SM is more efficient than only using the previous optimization(84.66). And the usage of the memory is incresed from 52.30 to 87.34.  Here is the compute workload analysis. it increses the utilization of every pipe by at least 5%:    Here is the memory workload analysis. the Memory Thoughput decreases from 130 to 104, because I increses the TILE\_WIDTH, reduce the memory output speed. |
| * 1. What references did you use when implementing this technique? |
| The lectures, textbook and google. |
| 1. **Optimization 5: Input channel reduction: atomics (2 point)** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| I choose to implement the optimization: Input channel reduction: atomics.  Reason: this optimization is easy to implement and it can synergize with other optimizations. |
| * 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? |
| In the previous optimizations, we directly write the result to an intermediate variable placed on the kernal, and then write the result back to the final output after all calculations are completed. Using atomic, the result is directly written back to the output. I don't think this method will increase the efficiency of execution, because atomic will increase the overhead of data collision. This optimization synergizes with the constant memory and 23 TILE\_WIDTH. |
| * 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 | *1.01038 ms* | *3.05715 ms* | *0m1.268s* | *0.86* | | 1000 | *9.88778 ms* | *30.2316 ms* | *0m9.974s* | *0.886* | | 10000 | *98.6176 ms* | *302.202 ms* | *1m34.932s* | *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). |
| This optimization doesn’t improve the speed of convolution. Here is the profilling result:    it incresed the conv\_forward\_kernel time from 52342229 to 381831443. The following shows the profiling result from Nsight-Compute:    It turns out that the SM efficiency reduces dramatically than previous optimization.  Here is the compute workload analysis. The utilization of every pipe is very similar to the previous optimization:    Here is the memory workload analysis. the Memory Thoughput decreases from 130 to 23, because the stuck of the atomic operation. |
| * 1. What references did you use when implementing this technique? |
| *lecture, google and textbook.* |
| **Optimization 6: Using Streams to overlap computation with data transfer** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| I choose to implement the optimization: Using Streams to overlap computation with data transfer.  Reason: We can parallel data transfer process by using streams, which will accelerate the computation. This optimization synergizes with the constant memory, loop unrolling, and 23 TILE\_WIDTH. |
| * 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? |
| We can parallel data transfer process by using streams. I think the optimization would increase performance of the forward convolution, because of the parallel data transfer. This optimization synergizes with the constant memory, loop unrolling, and 23 TILE\_WIDTH. |
| * 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 | *0.000714 ms* | *0.000624 ms* | *0m1.119s* | *0.86* | | 1000 | *0.000794 ms* | *0.000637 ms* | *0m9.791s* | *0.886* | | 10000 | *0.000726 ms* | *0.000672 ms* | *1m36.328s* | *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). |
| The op time reduced dramatically in the graph above. Here is the profilling result:  *op11_streams*  The conv\_forward\_kernel time stays almost the same as before (52342229). The following shows the profiling result from Nsight-Compute:    It turns out that the SM is less efficient than before, but the usage of the memory is incresed from 87.34 to 89.86.  Here is the compute workload analysis. the utilization of every pipe stays the same as before:    Here is the memory workload analysis. the Memory Thoughput stays also the same as before. |
| * 1. What references did you use when implementing this technique? |
| *lecture, google and textbook.* |

***Final Result:***

***I Implemented the following optimizations:***

Tiled shared memory convolution (2 points)

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

Tuning with restrict and loop unrolling (considered as one optimization only if you do both) (3 points)

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

Input channel reduction: atomics (2 point)

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

It turned out that using constant memory, tuning with restrict and loop unrolling, sweeping various parameters to find best values, using Streams to overlap computation with data transfer will improve the performance, while tiled shared memory and atomics will not improve the performance. The best optimization that I achieved is to synergize Stream with the constant memory, loop unrolling, and 23 TILE\_WIDTH.