|  |  |
| --- | --- |
| **Name:** | *Zihao Zheng* |
| **NetID:** | *zihao10* |
| **Section:** | *AL2* |

**ECE 408/CS483 Milestone 3 Report**

|  |
| --- |
| 1. List Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 5k images from your basic forward convolution kernel in milestone 2. This will act as your baseline this milestone. Note: **Do not** use batch size of 10k when you profile in *--queue rai\_amd64\_exclusive*. We have limited resources, so any tasks longer than 3 minutes will be killed. Your baseline M2 implementation should comfortably finish in 3 minutes with a batch size of 5k (About 1m35 seconds, with nv-nsight). |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *0.243364 ms* | *1.07161 ms* | *0m3.935s* | *0.86* | | 1000 | *2.48539 ms* | *9.70919 ms* | *0m10.256s* | *0.886* | | 5000 | *12.2784 ms* | *51.4837 ms* | *0m48.961s* | *0.871* |   *For 5000:*  *Layer Time: 335.19 ms;*  *Layer Time: 289.152 ms* |
| 1. **Optimization 1:** *FP16 arithmetic. (note this can modify model accuracy slightly) (4 points)* |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *I implemented FP16 arithmetic to reduce computing and data transmission costs due to its lower precision, anticipating an enhancement in overall performance.*  *(to use fp16, change the flag of use\_fp16 to 1 and others to 0)* |
| * 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? |
| *Transform floating-point values to half2 before and after performing memcopy, then revert back to float before output. Anticipating improved performance through this optimization, as FP16's half-precision reduces computing and transfer costs compared to float. This optimization aligns synergistically with the baseline.* |
| * 1. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 5k images using this optimization (including any previous optimizations also used). |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *0.239316 ms* | *1.03725 ms* | *0m1.615s* | *0.86* | | 1000 | *2.41886 ms* | *10.3409 ms* | *0m11.172s* | *0.887* | | 5000 | *12.078 ms* | *50.0602 ms* | *0m49.109s* | *0.8712* | |
| * 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 implementation improve a little ,according to picture it has lower SM and memory throughput.*  *According to profile, about 50% from device memory to L2 cache is reduced, due to the reason that fp16 is only half size of float, so this save us a little time on the transmission. The reason why it did not significant helped improving the code could be the reason that the type conversion helped a little in cache hitting but the most time it still miss for it needs stride in convolution.* |
| * 1. What references did you use when implementing this technique? |
| *https://docs.nvidia.com/cuda/cuda-math-api/group\_\_CUDA\_\_MATH\_\_\_\_HALF\_\_MISC.html* |
| 1. **Optimization 2: *Shared memory matrix multiplication and input matrix unrolling (****3 points****)*** |
| 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *I used Shared memory matrix multiplication and input matrix unrolling,*I believed that after unroll the matrix it will increase the hit rate of the cache which will speed up the computation and the matrix multiply with shared memory is better than calculating them one by one and reduce the memory request from 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? |
| *The optimization involves utilizing shared memory for matrix multiplication and input matrix unrolling. Shared memory is a small but fast memory space shared among threads in a thread block, which can significantly improve memory access efficiency. Matrix unrolling involves transforming the input matrix to a more cache-friendly format, which can enhance data locality and reduce memory access latency.*  *The optimization works by minimizing the time spent on memory access and maximizing computational throughput. Shared memory allows threads within a block to exchange data more efficiently, reducing the need to fetch data from slower global memory. Matrix unrolling helps improve cache utilization, enabling more efficient processing of matrix elements.*  *I do think this optimization could increase the performance of forward convolution. By reducing memory access times and maximizing computational efficiency, the overall execution time for the convolution operation is likely to decrease.*  *This optimization aligns synergistically with the baseline.* |
| 1. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 5k images using this optimization (including any previous optimizations also used). |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *0.92349 ms* | *0.57506 ms* | *0m1.534s* | *0.86* | | 1000 | *9.05535 ms* | *5.52085 ms* | *0m14.031s* | *0.886* | | 5000 | *45.4207 ms* | *27.6578 ms* | *0m51.245s* | *0.871* | |
| 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 implementation greatly improve the performance for layer2.but worse performance in layer1.*  *From below we can see that it has a higher through put in SM and lower throughput in Memory.*    *From the memory chart we see that the transmission between L1 and L2 cache is reduced by around 66%, and device memory greatly make use of L2 cache(orange arrow) and increase the hit rate. The reason why L1 cache hit rate decreased could be the reason of unrolling.*    *From this chart we see the total time for matrix multiply shared is increased, this could be the reason that the unrolled matrix in bigger than the original input and requires extra time to do the transmission.* |
| 1. What references did you use when implementing this technique? |
| *Lec12 and Lec5* |
| 1. **Optimization 3: *Using Streams to overlap computation with data transfer (****4 points****)*** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *I use streams to overlap computation for multiple streams allow for parallelism within a GPU. Different streams can execute different kernels concurrently, exploiting the parallel processing power of the GPU.* |
| * 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? |
| *CUDA stream optimization enables asynchronous execution of tasks on the GPU, allowing for overlap between computation and communication. In convolution, this can hide latencies associated with data transfers, increase GPU throughput, and exploit parallelism. The optimization is likely to enhance forward convolution performance by keeping the GPU busy with concurrent tasks, and it synergizes with basline.* |
| * 1. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 5k images using this optimization (including any previous optimizations also used). |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *0.003062 ms* | *0.003425 ms* | *0m1.536s* | *0.86* | | 1000 | *0.004376 ms* | *0.004003 m* | *0m10.090s* | *0.886* | | 5000 | *0.006444 ms* | *0.006156 ms* | *0m50.395s* | *0.871* | |
| * 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 implementation did not improve the performance much.*    *From upper we see that it has a lower throughput in SM and lower throughput in Memory in minor degree.*  *The pipe got better used, which speed up the data transfer.*    From above memory chart we find no great difference, which is expected.  *baseline:*  *Layer Time: 335.19 ms;*  *Layer Time: 289.152 ms*  *stream:*  *Layer Time: 336.131 ms*  *Layer Time: 260.404 ms*  *however from layer time we found the stream did not helped much in saving time. This could be the reason that the number of stream is too small and make little use in*  *saving the data transmission. This might be better if more number of stream apllied.*  *from below we can see the average time for convforward is greatly reduced compared to baseline* |
| * 1. What references did you use when implementing this technique? |
| *https://docs.nvidia.com/cuda/cuda-runtime-api/group\_\_CUDART\_\_STREAM.html* |
| 1. **Optimization 4: *sweeping various parameter and kernel in constant memory(0.5+0.5)*** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *I choose to find different tile size and picked Tile width as 20 which outperformed 32 much and kernel in constant 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 change the tile size by better utilizing the available thread which make more use of the hardware, also the filter matrix in constant memory reduce the cost to fetch compared to store in host. This optimization synergize with baseline* |
| * 1. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 5k images using this optimization (including any previous optimizations also used). |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *0.182797 ms* | *0.519764 ms* | *0m1.530s* | *0.86* | | 1000 | *1.72726 ms* | *5.13407 ms* | *0m10.091s* | *0.886* | | 5000 | *8.5622 ms* | *25.3927 ms* | *0m48.079s* | *0.871* | |
| * 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). |
| *overall it improved the performance.*    *from above we see it use less memory, which means the constant memory reduce the bandwidth pressure. Also the SM throughput is still the same.*  *From this memory chart we found the kernel and global communication is reduced by 43.39%. and L2 Cache hit rate is improved by 9.32%, at the same time L1 cahce hit rte only reduce by 0.05%, which is minor influence. Overall it increase the hit rate of cache and speed up the computation.*  *From below we see the total time is greatly reduce compared to baseline with around 60000000.* |
| * 1. What references did you use when implementing this technique? |
| *Lectures.* |
| 1. **Optimization 5: *<optimization name>***   ***(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. |
| *<answer here>* |
| * 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? |
| *<answer here>* |
| * 1. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 5k images using this optimization (including any previous optimizations also used). |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *<op\_time>* | *<op\_time>* | *<exec\_time>* | *<accuracy>* | | 1000 | *<op\_time>* | *<op\_time>* | *<exec\_time>* | *<accuracy>* | | 5000 | *<op\_time>* | *<op\_time>* | *<exec\_time>* | *<accuracy>* | |
| * 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). |
| *<answer here>* |
| * 1. What references did you use when implementing this technique? |
| *<answer here>* |
| 1. **Optimization 6: *<optimization name>***   ***(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. |
| *<answer here>* |
| * 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? |
| *<answer here>* |
| * 1. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 5k images using this optimization (including any previous optimizations also used). |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *<op\_time>* | *<op\_time>* | *<exec\_time>* | *<accuracy>* | | 1000 | *<op\_time>* | *<op\_time>* | *<exec\_time>* | *<accuracy>* | | 5000 | *<op\_time>* | *<op\_time>* | *<exec\_time>* | *<accuracy>* | |
| * 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). |
| *<answer here>* |
| * 1. What references did you use when implementing this technique? |
| *<answer here>* |