|  |  |
| --- | --- |
| **Name:** | *Jiatong* Li |
| **NetID:** | *Jl180* |
| **Section:** | *Tuesday, Thursday 9:30am* |

**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 | *0.181132ms* | *0.657373ms* | *0m1.279s* | *0.86* | | 1000 | *1.65316ms* | *6.32952ms* | *0m9.856s* | *0.886* | | 10000 | *16.2296ms* | *63.0129ms* | *1m35.102* | *0.8714* | |
| 1. **Optimization 1:** Tiled shared memory convolution |
| * 1. Which optimization did you choose to implement? Chose from the optimization below by clicking on the check box and explain why did you choose that optimization technique. |
| Tiled shared memory convolution (**2 points**)  Shared memory matrix multiplication and input matrix unrolling (**3 points**)  Kernel fusion for unrolling and matrix-multiplication (**2 points**)  Weight matrix in constant memory (**1 point**)  Tuning with restrict and loop unrolling (**3 points**)  Sweeping various parameters to find best values (**1 point**)  Multiple kernel implementations for different layer sizes (**1 point**)  Input channel reduction: tree (**3 point**)  Input channel reduction: atomics (**2 point**)  Fixed point (FP16) arithmetic. (**4 points**)  Using Streams to overlap computation with data transfer (**4 points**)  An advanced matrix multiplication algorithm (**5 points**)  Using Tensor Cores to speed up matrix multiplication (**5 points**)  Overlap-Add method for FFT-based convolution (**8 points**)  Other optimizations: please explain  *This is one of the easiest implementation and easy to understand, I use this as a practice to see if the performance gets better.*   * 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?   *<your answer here>*  *Constant memory access is faster than original way of global memory, here, I just made a constant memory and use copy to symbol. I do believe this would increase performance due to our lecture theory. This is my first opt, so no synergize tested now.* |
| * 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 | *0.162359ms* | *0.664323ms* | *0m1.159s* | *0.86* | | 1000 | *1.48301ms* | *6.45443ms* | *0m9.638s* | *0.886* | | 10000 | *14.686ms* | *64.3614ms* | *1m33.727s* | *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, successful, we can see improvements in OP time1 and 2. Only for batch size 10000, the OP time2 was not shortened.*  The time was saved for cudaMalloc, free and Memcpy, since cudaMemcpy+cudaMemcpyToSymbol seems to have a sum less than baseline Memcpy. And the time in Kernel is better too. Comparing (baseline)7977507 and 7834691.  Baseline    On the left is the baseline conv\_forward \_kernel, and the second one is the constant weight matrix applied.  One of the improvement is the SM utilization, And the SM active cycle is smaller comparing to baseline. another part, where we consider about memory, we can find that the Memory Workload, the memory throughput increased about 9 percent, for it is faster to transfer data between constant memory and threads. Similarly for the second conv\_forward\_kernel, see the following images. |
| * 1. What references did you use when implementing this technique?   Previous mp and lecture slides. |
| * 1. Please Paste your kernel code for this optimization. Your code should include the non-trivial code that you have changed for this optimization.   For example, it can be the complete kernel code for Tiled shared memory convolution several lines of code for Weight matrix in constant memory, or the “for” loop for loop unrolling |

|  |
| --- |
| 1. **Optimization 2:** Tiled shared memory convolution |
| * 1. Which optimization did you choose to implement? Chose from the optimization below by clicking on the check box and explain why did you choose that optimization technique. |
| Tiled shared memory convolution (**2 points**)  Shared memory matrix multiplication and input matrix unrolling (**3 points**)  Kernel fusion for unrolling and matrix-multiplication (**2 points**)  Weight matrix in constant memory (**1 point**)  Tuning with restrict and loop unrolling (**3 points**)  Sweeping various parameters to find best values (**1 point**)  Multiple kernel implementations for different layer sizes (**1 point**)  Input channel reduction: tree (**3 point**)  Input channel reduction: atomics (**2 point**)  Fixed point (FP16) arithmetic. (**4 points**)  Using Streams to overlap computation with data transfer (**4 points**)  An advanced matrix multiplication algorithm (**5 points**)  Using Tensor Cores to speed up matrix multiplication (**5 points**)  Overlap-Add method for FFT-based convolution (**8 points**)  Other optimizations: please explain  *Same as previous opt, it is commonly used in mp.*   * 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?   *By shared matrix, we can avoid loading input from global memory, instead, for each tile, we load some input data that would be used to the block shared memory, so that all the threads in that block accesses the input they need very fast. Yes, it can be done together with constant memory weight matrix.* |
| * 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 | *0.173876ms* | *2.80599ms* | *0m3.219s* | *0.86* | | 1000 | *1.50765ms* | *6.24129ms* | *0m10.173s* | *0.886* | | 10000 | *14.9069ms* | *62.4968ms* | *1m34.879s* | *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   *Not really, the result is worse comparing total execution time, op time, and kernel time. Maybe because some declaration issues such as defining the shared memory(I guess it would be similar to malloc, that takes a lot of time), and declaring the argument of calling kernel(I think the declare of shared memory space we need might be time consuming). Here are some profiling results from nsys and Nsight-Compute, left is the previous opt, and right hand side is the shared memory used result.*  We can find that the total time of conv\_forward\_kernel dropped but the cudaMalloc time increased a lot.    Based on this two charts and the bottom shared memory analysis on the first conv\_forward of previous and  with shared mem implementation, we can see that the shared load takes a lot of time, which resulted in the performance did not get better. |
| * 1. What references did you use when implementing this technique? |
| When you use **extern** in front of a variable declaration, it tells the compiler that the variable is defined in another module or translation unit, and the linker will resolve the reference at link time.  *From chatgpt, cause I ran into a problem that how to define the shared memory*  *Also, lecture notes and mp3*   * 1. Please Paste your kernel code for this optimization. Your code should include the non-trivial code that you have changed for this optimization.   For example, it can be the complete kernel code for Tiled shared memory convolution several lines of code for Weight matrix in constant memory, or the “for” loop for loop unrolling        *This is for knowing the block starting x and y we are working on using by as requested.*    *Define the shared space we needed* |

|  |
| --- |
| 1. **Optimization 3:** Input channel reduction: tree |
| * 1. Which optimization did you choose to implement? Chose from the optimization below by clicking on the check box and explain why did you choose that optimization technique. |
| Tiled shared memory convolution (**2 points**)  Shared memory matrix multiplication and input matrix unrolling (**3 points**)  Kernel fusion for unrolling and matrix-multiplication (**2 points**)  Weight matrix in constant memory (**1 point**)  Tuning with restrict and loop unrolling (**3 points**)  Sweeping various parameters to find best values (**1 point**)  Multiple kernel implementations for different layer sizes (**1 point**)  Input channel reduction: tree (**3 point**)  Input channel reduction: atomics (**2 point**)  Fixed point (FP16) arithmetic. (**4 points**)  Using Streams to overlap computation with data transfer (**4 points**)  An advanced matrix multiplication algorithm (**5 points**)  Using Tensor Cores to speed up matrix multiplication (**5 points**)  Overlap-Add method for FFT-based convolution (**8 points**)  Other optimizations: please explain  *This one is also done before and easy to understand*   * 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?   *By making the results of each channel’s convolution stored in shared memory, and then add them up using parallel execution, which would be quicker than* |
| * 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 | *0.707728ms* | *2.81728ms* | *0m3.299s* | *0.86* | | 1000 | *1.50392ms* | *7.98942ms* | *0m10.207s* | *0.886* | | 10000 | *14.8876ms* | *80.3098ms* | *1m38.385s* | *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   *it seems that the kernel time with tree reduction was even longer than simply go through all the channels and sum the up. This might be because the channels are not that much, so logarithm time reduction is little comparing to a simpler code without the effort on blocks and shared memory to fulfill tree reduction.* |
| * 1. What references did you use when implementing this technique? |
| *Lecture note.*   * 1. Please Paste your kernel code for this optimization. Your code should include the non-trivial code that you have changed for this optimization.   For example, it can be the complete kernel code for Tiled shared memory convolution several lines of code for Weight matrix in constant memory, or the “for” loop for loop unrolling |

|  |
| --- |
| 1. **Optimization 4: *<optimization name>*** |
| * 1. Which optimization did you choose to implement? Chose from the optimization below by clicking on the check box and explain why did you choose that optimization technique. |
| Tiled shared memory convolution (**2 points**)  Shared memory matrix multiplication and input matrix unrolling (**3 points**)  Kernel fusion for unrolling and matrix-multiplication (**2 points**)  Weight matrix in constant memory (**1 point**)  Tuning with restrict and loop unrolling (**3 points**)  Sweeping various parameters to find best values (**1 point**)  Multiple kernel implementations for different layer sizes (**1 point**)  Input channel reduction: tree (**3 point**)  Input channel reduction: atomics (**2 point**)  Fixed point (FP16) arithmetic. (**4 points**)  Using Streams to overlap computation with data transfer (**4 points**)  An advanced matrix multiplication algorithm (**5 points**)  Using Tensor Cores to speed up matrix multiplication (**5 points**)  Overlap-Add method for FFT-based convolution (**8 points**)  Other optimizations: please explain  *<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?   Memcpy the same time do kernel. Yes, cause it is overlapped.  Synergize with constant mask weight. |
| * 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 | *0.00649ms* | *0.006409ms* | *0m1.207s* | *0.86* | | 1000 | *0.005653ms* | *0.006149ms* | *0m9.618s* | *0.886* | | 10000 | *0.007823ms* | *0.007768ms* | *1m 40.014s* | *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, slightly for 1000 size, but got worse when size comes to 10000, comparing to only constant weight mask used. The execution time is around that and op time is decreased a lot.* |
| * 1. What references did you use when implementing this technique? |
| *Course notes and Chatgpt*   * 1. Please Paste your kernel code for this optimization. Your code should include the non-trivial code that you have changed for this optimization.   For example, it can be the complete kernel code for Tiled shared memory convolution several lines of code for Weight matrix in constant memory, or the “for” loop for loop unrolling |

For different parameters, I tried constant size 6000, 8000, 10000, and 6000 works and small. And the stream number needs to be 10 to be better than 8 or 12 or 16.