|  |  |
| --- | --- |
| **Name:** | *Jiadong Hong* |
| **NetID:** | *jiadong5* |
| **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 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 | *0.188722 ms* | *0.668105 ms* | *0m1.620s* | *0.86* | | 1000 | *1.72154 ms* | *6.4301 ms* | *0m11.110s* | *0.886* | | 5000 | *8.53825 ms* | *31.9946 ms* | *0m51.045s* | *0.871* | |
| 1. **Optimization 1:** *Weight matrix (kernel values) in constant memory* |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique |
| *Weight matrix (kernel values) in constant memory, since it is the easiest optimization method to implement.*  *Final Optimization picked: Tuning with restrict and loop unrolling.* |
| * 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? * **How it works:** Storing the weight matrix in constant memory allows the GPU to cache the values, taking advantage of its read-only characteristics. This can reduce the memory access time during convolution operations. * **Expectation:** This optimization is effective when the weight matrix remains constant throughout the convolution operation. It helps reduce memory access latency during the computation, potentially improving performance. * **Synergy:** This optimization works independently of the others but complements them by ensuring efficient use of the weight matrix during computation. |
| * 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 | *0.17194 ms* | *0.653659 ms* | *0m1.617s* | *0.86* | | 1000 | *1.55587 ms* | *6.3645 ms* | *0m10.607s* | *0.886* | | 5000 | *7.71476 ms* | *31.7326 ms* | *0m51.176s* | *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). |
| *Actually from my observation yes but not much.*  *Let’s see baseline kernel execution duration for grid size = 483136:*    *Then Constant Memory duration for grid size = 483136:*    *We can directly say that the kernel speed has been speed up by constant memory. Also we can figure out that the pressure of the optimized cache and DRAM memory is much lower.*  *Let’s see nsys timeline:*    *The first graph is baseline and the second is Mask in Constant Memory, as we can see that the speed is faster for Mask in Constant memory.* |
| * 1. What references did you use when implementing this technique? |
| *From ChatGPT and MP4’s implemented code. MP4 we have that approach to put the mask into the constant memory.* |
| 1. **Optimization 2:** *Tuning with restrict and loop unrolling* |
| 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *Tuning with restrict and loop unrolling, since I think the main task to reduce the op time is to cut down the execution of calculation and make full use of the compiler, this implementation directly reduce the divergence and made full use of complier optimization.* |
| 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? |
| * **How it works:** Using the **restrict** keyword helps the compiler assume that pointers do not alias, allowing for more aggressive optimizations. Loop unrolling reduces loop overhead and enables better utilization of pipeline stages. * **Expectation:** This optimization is expected to improve the efficiency of memory access and computation by providing the compiler with more opportunities for optimization. * **Synergy:** Works well with constant memory optimization as it allows the compiler to optimize memory access patterns more effectively. |
| 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 | *0.166246 ms* | *0.580914 ms* | *0m1.617s* | *0.86* | | 1000 | *1.50082 ms* | *5.61239 ms* | *0m10.604s* | *0.886* | | 5000 | *7.4336 ms* | *28.0054 ms* | *0m51.159s* | *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). |
| *Yes,*  *Let’s see baseline kernel execution duration for grid size = 483136:*    *we can see the optimized kernel execution time of grid size = 483136:*  *图形用户界面, 文本  描述已自动生成*  *Optimized Version is much faster.*  *For Timeline:*    *First is baseline and second is restircted and unrolled loop optimization, as we can see that the kernel speed is mauch more faster.* |
| 1. What references did you use when implementing this technique? |
| *From ChatGPT and some other CUDA tutorials on YouTube.* |

|  |
| --- |
| 1. **Optimization 3:** *FP16 arithmetic*   ***(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. |
| *FP16 arithmetic since I think reduce calculation complexity is the most direct way to reduce consumption..* |
| * 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? |
| * **How it works:** Utilizing half-precision floating-point (FP16) arithmetic reduces memory bandwidth requirements and computational complexity compared to single-precision (FP32) arithmetic. * **Expectation:** Expected to reduce the time and resources required for arithmetic operations, improving overall computational efficiency. * **Synergy:** May interact with constant memory optimization by potentially reducing the size of the data transferred between global memory and the processing units. |
| * 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 | *0.230797 ms* | *0.86191 ms* | *0m1.651s* | *0.86* | | 1000 | *2.14797 ms* | *8.31054 ms* | *0m11.005s* | *0.887* | | 5000 | *10.665 ms* | *41.3822 ms* | *0m51.406s* | *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). |
| *Actually it is not.*  *Let’s see baseline kernel execution duration for grid size = 483136:*    *we can see the optimized kernel execution time of grid size = 483136:*    *As you can see the kernel takes actually more time.*  *For Timeline:*    *Actually when the kernel is not too large it may be faster, but when too large kernel results in much data needs to transfer from float to half precision, it may cost too much.* |
| * 1. What references did you use when implementing this technique? |
| *From conducting a survey on how to use Tensor Core on website finding out there is no time to implement tensor core and tensor core basic is to implement the FP16, a little bit adjustment of code.* |
| 1. **Optimization 4:** *Using Streams to overlap computation with data transfer*   ***(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, it’s a topic that recently covered in the lecture and I what to practice it in real code and development.* |
| * 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? |
| * **How it works:** Utilizing CUDA streams allows for asynchronous execution of kernel computation and data transfer operations. This can overlap the communication and computation phases, reducing overall execution time. * **Expectation:** Expected to decrease the total execution time by concurrently executing computation and data transfer operations. * **Synergy:** Can work well with other optimizations, especially when there are opportunities to overlap computation with data transfer, reducing idle time. |
| * 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).   # Warning: May not be really meaningful compared to baseline since the layer-level execution is made separately. |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy | | 100 | *0.000653 ms* | *0.000604 ms* | *0m1.808s* | *0.86* | | 1000 | *0.000703 ms* | *0.001064 ms* | *0m11.377s* | *0.886* | | 5000 | *0.000689 ms* | *0.001685 ms* | *0m56.619s* | *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).   *Actually yes, since the stream method is use to reduce the transfer time loss, so, let’s see the layer-level time consumption, for baseline, the first big layer in CNN is:*    *For Stream Method:*    *We can figure out that in the layer sense, the execution become much faster.* |
|  |
| * 1. What references did you use when implementing this technique? |
| *From lecture 22. Actually I think three way may be enough, but the timeline shocked me, more stream may led to better performance.* |