|  |  |
| --- | --- |
| **Name:** | *Yiming Li* |
| **NetID:** | *Yiming22* |
| **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 | *0.175098 ms* | *0.632573 ms* | *0m1.190s* | *0.86* | | 1000 | *1.63944 ms* | *6.2645 ms* | *0m9.660s* | *0.886* | | 10000 | 16.0852 ms | 62.7071 ms | 1m37.766s | 0.8714 |   *Launch: 1 - 122 - conv\_forward\_kernel*    Launch: 4 - 143 - conv\_forward\_kernel    *Baseline:*  *The first conv\_forward\_kernel GPU utilization SM is 75.05%, the Memory is 88.46%.*  *The second conv\_forward\_kernel GPU utilization SM is 75.07%, the Memory is 78.90%.* |
| 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 (1 point)* |
| * 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 weighted matrix is never changed in this project. And if we move the matrix to constant memory, I think it can speed up the operation time and can get better use of bandwidth.*  *No synergy.* |
| * 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 | *0.165795 ms* | *0.57739 ms* | *0m1.595s* | 0.86 | | 1000 | *1.48544ms* | *5.65347 ms* | *0m9.798s* | *0.886* | | 10000 | *14.5932 ms* | *14.5932 ms* | *1m38.642s* | *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). |
| I think it’s successful to optimization.  profiling results from *nsys* and *Nsight-Compute:*  *Launch: 1 - 122 - conv\_forward\_kernel*    Launch: 4 - 142 - conv\_forward\_kernel    *The first conv\_forward\_kernel GPU utilization SM is 83.18%(compared to base line: 75.05%), the Memory is 71.32%(compared with base line: 88.46%)*  *The second conv\_forward\_kernel GPU utilization SM is 80.84% (compared to base line: 75.07%), the Memory is 57.46%(compared with base line: 78.90%)*  *The constant memory help kernel to access less global memory and the time of this is cut down largely.* |
| * 1. What references did you use when implementing this technique? |
| *Constant memory in C language, chapter 7* |
| 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 (considered as one optimization only if you do both) (3 points)* |
| 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? |
| *Loop unrolling can help optimizing the execution time. It speeds up the program by getting rid of the control/test instructions in the loop. This is implemented by removing and reducing number of iterations.*  *No synergy.* |
| 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 | *0.167156 ms* | *0.598132 ms* | *0m1.387s* | 0.86 | | 1000 | *1.5526 ms* | *5.97613 ms* | *0m10.103s* | *0.886* | | 10000 | *15.382 ms* | *60.2823 ms* | *1m43.120s* | *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). |
| I think it’s successful to optimization.  profiling results from *nsys* and *Nsight-Compute:*  *Launch: 1 - 122 - conv\_forward\_kernel*    Launch: 4 - 143 - conv\_forward\_kernel    *The first conv\_forward\_kernel GPU utilization SM is 55.34%(compared to base line: 75.05%), the Memory is 91.09%(compared with base line: 88.46%)*  *The second conv\_forward\_kernel GPU utilization SM is 56.1%(compared to base line: 75.07%), the Memory is 81.42%(compared with base line: 78.90%)*  *The variable in loop unrolling will be placed in registers and increase the access speed. It cut down the utilization of SM but raise Memory, which increases the efficiency of execution.* |
| 1. What references did you use when implementing this technique? |
| *Restrict: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#restrict>*  *Loop unrolling: chapter 16* |

|  |
| --- |
| **Optimization 3:** *Tiled shared memory convolution* |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *Tiled shared memory convolution (2 points)* |
| * 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 input data and weighted matrix are put in the shared memory. It is a very common technique used in convolution and can help to increase reuse rate and to reduce global memory access rate, in order to speed up the process.*  *No synergy.* |
| * 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 | *0.209385 ms* | *1.01973 ms* | *0m1.142s* | 0.86 | | 1000 | *1.96323 ms* | *10.0182 ms* | *0m12.081s* | *0.886* | | 10000 | *19.2785 ms* | *100.08 ms* | *1m42.191s* | *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). |
| I think it’s not successful to optimization.  profiling results from *nsys* and *Nsight-Compute*  *Launch: 1 - 122 - conv\_forward\_kernel*    Launch: 4 - 143 - conv\_forward\_kernel    *The first conv\_forward\_kernel GPU utilization SM is 91.54%(compared to base line: 75.05%), the Memory is (compared with base line: 88.46%)*  *The second conv\_forward\_kernel GPU utilization SM is 94.31%(compared to base line: 75.07%), the Memory is 59.27%(compared with base line: 78.90%)*  *Not successful. Reason: many control divergence are introduced into this program, which slow down the running speed, although the using of shared memory can reduce global memory access.* |
| * 1. What references did you use when implementing this technique? |
| *Tiled shared memory, chapter 7* |
| **Optimization 4:** *Shared memory matrix multiplication and input matrix unrolling* |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *Shared memory matrix multiplication and input matrix unrolling (3 points)* |
| * 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? |
| *Two kernels were launched one by another. One is about making input matrix to unroll, another is to do matrix multiplication. (The trick is on unrolling while matrix multiplication is just normal one) Unrolling can make it easier for GPU access.*  *No synergy.* |
| * 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 | *2.28242 ms* | *3.10461 ms* | *0m1.144s* | 0.86 | | 1000 | *22.7806 ms* | *29.7191 ms* | *0m10.278s* | *0.886* | | 10000 | *225.165 ms* | *296.176 ms* | *1m42.856s* | *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). |
| I think it’s not successful to optimization.  profiling results from *nsys* and *Nsight-Compute*  *Launch: 1 - 124 - unroll\_kernel*    Launch: 2 - 127 - matrixMultiplyShared    *The first unroll\_kernel GPU utilization SM is 4.09 % (compared to base line: 75.05%), the Memory is 8.04% (compared with base line: 88.46%)*  *The first* matrixMultiplyShared *GPU utilization SM is 29.58% (compared to base line: 75.07%), the Memory is 32.31% (compared with base line: 78.90%)*  also there are many launches in kernels.  *Not improve the performance(even worse a lot). The 1st reason is new X\_unroll matrix is used, which leads to more layer time in order to allocate && copy space. And two separate kernels need more additional time to do memory data transfer. 2nd reason is unroll function needs more operations and no coalescing is used, so even with less time in forward function, the overall time is not better.* |
| * 1. What references did you use when implementing this technique? |
| *Chapter 7, Chapter 16* |
| **Optimization 5:** *Kernel fusion for unrolling and matrix-multiplication* |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *Kernel fusion for unrolling and matrix-multiplication (requires previous optimization: Shared memory matrix multiplication and input matrix unrolling) (2 points)* |
| * 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? |
| *Does synergize with previous optimization: Shared memory matrix multiplication and input matrix unrolling.*  *This merges the unrolling kernel and matrix multiplication kernel into one kernel which can speed up the op time.(less duplicate work done for the kernel). And two shared memory are used, one for weighted matrix and another for input 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 | *0.552482 ms* | *0.347056 ms* | *0m1.134s* | 0.86 | | 1000 | *5.3891 ms* | *3.27264 ms* | *0m10.654s* | *0.886* | | 10000 | *53.5837 ms* | *32.6165 ms* | *1m42.579s* | *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). |
| I think it’s not successful to optimization.  profiling results from *nsys* and *Nsight-Compute.*  *Launch: 1 - 122 - conv\_forward\_kernel*    Launch: 4 - 143 - conv\_forward\_kernel    *The first conv\_forward\_kernel GPU utilization SM is 84.01% (compared to base line: 75.05%), the Memory is 82.30% (compared with base line: 88.46%)*  *The second conv\_forward\_kernel GPU utilization SM is 81.72% (compared to base line: 75.07%), the Memory is 81.73% (compared with base line: 78.90%) This method does perform better than previous two kernel version(without kernel fusing), though it isn’t as good as the base line version(const-weight). Too many math operations are introduced during the unroll process can slow down the execution.* |
| * 1. What references did you use when implementing this technique? |
| *Chapter 7, Chapter 16* |