|  |  |
| --- | --- |
| **Name:** | *Aditya Gupta* |
| **NetID:** | *adityag5* |
| **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 | *0.32 ms* | *1.197 ms* | *1.169 s* | *0.86* | | 1000 | *3.08 ms* | *11.985 ms* | *9.941 s* | *0.886* | | 10000 | *30.44 ms* | *120.418 ms* | *36.117 s* | *0.8714* | |
| 1. **Optimization 1: Tiled shared memory convolution** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *I chose the shared memory convolution as an optimization technique as it would be the most straightforward to implement starting from the baseline version.* |
| * 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 uses shared memory, which is shared by all threads in a given block, which results in lower latency and higher bandwidth than using global memory, in turn resulting in better performance of the kernel. This technique involves storing both input data and kernel data into shared memory before convolution is done, and since global memory is usually slow, this technique leads to a faster convolution process.* |
| * 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 | *0.203 ms* | *1.1016 ms* | *1.085 ms* | *0.86* | | 1000 | *1.93 ms* | *10.075 ms* | *9.224 s* | *0.886* | | 10000 | *19.002 ms* | *100.42 ms* | *32.989 s* | *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).   Text  Description automatically generated  Timeline  Description automatically generated with medium confidence |
| *Timeline  Description automatically generated*  *Yes, this optimization was successful in improving performance as seen from the clear reduction in Op times and execution times. However, it still hasn’t met the desired performance of Op times less than 70 ms. The switch from using global memory entirely to using shared memory for partitioning data clearly reduced total execution time, which is the most apparent in the 10,000 dataset.*  *The SOL utilization screenshot also shows how much the SM usage % has increased and the memory usage % has decreased, indicating that performance was improved when using shared memory.* |
| * 1. What references did you use when implementing this technique? |
| [*https://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86\_64\_website/projects/convolutionSeparable/doc/convolutionSeparable.pdf*](https://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_64_website/projects/convolutionSeparable/doc/convolutionSeparable.pdf) |
| 1. **Optimization 2: Shared memory matrix multiplication and input matrix unrolling** |
| 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *I chose loop unrolling as my second technique as to reap the benefits of instruction level benefits at the expense of space complexity.* |
| 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 is also a standalone optimization utilizing shared memory matrix multiplication and unrolling of the input matrix. For this optimization, both the input data and kernel data are stored in shared memory, which has the same benefits as optimization 1. On top of that, before convolution, the input data X will be modified as an unrolled matrix to introduce instruction level parallelism which improves 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 | *1.51 ms* | *1.196 ms* | *1.159 s* | *0.86* | | 1000 | *13.02 ms* | *9.57 ms* | *10.138 s* | *0.886* | | 10000 | *130.8 ms* | *92.94 ms* | *38.1 s* | *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 was not successful in improving in improving performance as suggested by the OP times and execution times which have increased from the baseline for all batch sizes* |
| 1. What references did you use when implementing this technique? |
| *<answer here>* |

|  |
| --- |
| 1. **Optimization 3: Kernel fusion for unrolling and matrix-multiplication** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *This optimization involves fusing the two kernels which independently unrolled the input matrices and performed the matrix multiplication into one kernel. I chose this as this will be an attempt to improve optimization 2.* |
| * 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 is based on optimization 2 as it uses the two kernels created there and fuses them to improve performance since the total number of operations would be reduced by not requiring the GPU to transfer the input matrix to a new unrolled matrix and thus saving memory on the GPU.* |
| * 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 | *<op\_time>* | *<op\_time>* | *<exec\_time>* | *<accuracy>* | | 1000 | *<op\_time>* | *<op\_time>* | *<exec\_time>* | *<accuracy>* | | 10000 | *<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 4: Sweeping parameters to find best values** |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| *From the baseline kernel, this was a pretty simple optimization to implement as it would only involve fine tuning parameters such as block sizes, tile widths.* |
| * 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 would involve sweeping parameters like tile width and block size which would result in more threads per block increasing parallelism or more elements in a single tiled matrix multiplication operation. This is also a standalone optimization. For this particular optimization different tile widths like 16,32 and 64 were tried. The tile width of 16 provided the best 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 | *0.225 ms* | *1.008 ms* | *4.401 ms* | *0.86* | | 1000 | *2.182 ms* | *8.894 ms* | *10.284 ms* | *0.886* | | 10000 | *15.97 ms* | *62.2325 ms* | *42.99 s* | *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 can be seen from the reduction in OP times and execution rimes that the optimization significantly improved performance for all datasets while retaining accuracy. The SOL utilization screenshot also shows that the SM usage % has increased from around 40% to around 70% for both kernels and the memory usage % has decreased from around 95% to 55%, indicating that performance was improved when parameters were sweeped.*  *Timeline  Description automatically generated*  *Timeline  Description automatically generated* |
| * 1. What references did you use when implementing this technique? |
| *ECE 408 Lecture Slides* |
| 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 10k 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>* | | 10000 | *<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 10k 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>* | | 10000 | *<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>* |