|  |  |
| --- | --- |
| **Name:** | *<name>* |
| **NetID:** | *<netid>* |
| **Section:** | *<class section>* |

**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 | *<op\_time>* | *<op\_time>* | *<exec\_time>* | *<accuracy>* | | 1000 | *<op\_time>* | *<op\_time>* | *<exec\_time>* | *<accuracy>* | | 10000 | *<op\_time>* | *<op\_time>* | *<exec\_time>* | *<accuracy>* | |
| 1. **Optimization 1: *<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?   *<your 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. 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  *<answer here>* |

|  |
| --- |
| 1. **Optimization 2: *<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?   *<your 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. 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  *<answer here>* |

|  |
| --- |
| 1. **Optimization 3: *<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?   *<your 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. 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  *<answer here>* |

|  |
| --- |
| 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?   *<your 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. 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  *<answer here>* |