|  |  |
| --- | --- |
| **Name:** | *Sheng-Wei Huang* |
| **NetID:** | *Swhuang3* |
| **Section:** | *AL2* |

**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 | *0.1762* | *0.6354* | *0.8116* | *0.86* | | 1000 | *1.6275* | *6.2637* | *7.8912* | *0.886* | | 10000 | *16.0021* | *62.2321* | *78.2342* | *0.8714* | |
| 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. |
| Use cudaMemcpyToSymbol to copy kernel to constant memory. It is very easy to implement, and the improvement is significant. |
| * 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? |
| * Use cudaMemcpyToSymbol to copy kernel to constant memory in conv\_forward\_gpu\_prolog.      * The optimization would increase performance, because constant memory access is much faster than global memory access. * The optimization synergize with all other optimizations. |
| * 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 | *0.1597* | *0.5726* | *0.7323* | *0.86* | | 1000 | *1.4825* | *5.6488* | *7.1313* | *0.886* | | 10000 | *14.5445* | *57.1707* | *71.7152* | *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). |
| * The Total Execution Time decrease to 92% of baseline (8% faster) after the optimization. (Use batch size = 10000 to compare) * This optimization successful in improving performance. * From nsys: With this optimization, conv\_forward\_kernel total time decrease.  |  |  |  | | --- | --- | --- | |  | baseline | This optimization | | conv\_forward\_kernel | 79482366 | 72374484 |  * From Memory Chart we can see that data transfer between kernel and Global -49.49% => Less global memory access => improve performance |
| * 1. What references did you use when implementing this technique? |
| *Mp4: 3D Convolution* |
| 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. |
| I choose Tuning with restrict and loop unrolling because it is also easy to implement. And loop unrolling can avoiding some Branch Divergence, which will improve performance. |
| 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? |
| * Add \_\_restrict\_\_ prefix to all pointers declartion. * Unroll convolution calculation  |  |  | | --- | --- | | Before Unroll |  | | After Unroll | **.**  **.**  **.** |  * I think the optimization would increase performance of the forward convolution, because loop unrolling can avoiding some Branch Divergence, which will improve performance. * The optimization synergize with all other optimizations. In the above screenshot you can see that I use k4d\_constant to calculate convolution. |
| 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 | *0.1563* | *0.4187* | *0.575* | *0.86* | | 1000 | *1.2345* | *3.9597* | *5.1942* | *0.886* | | 10000 | *11.5244* | *40.5754* | *52.0998* | *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). |
| * The Total Execution Time decrease to 73% of previous optimization (37% faster) after the optimization. (Use batch size = 10000 to compare) * This optimization successful in improving performance. * From nsys: With this optimization, conv\_forward\_kernel total time decrease.  |  |  |  | | --- | --- | --- | |  | Previous code | This optimization | | conv\_forward\_kernel | 72374484 | 52457030 | |
| 1. What references did you use when implementing this technique? |
| * *https://www.nvidia.com/docs/IO/116711/sc11-unrolling-parallel-loops.pdf* * *https://developer.nvidia.com/blog/cuda-pro-tip-optimize-pointer-aliasing/* |

|  |
| --- |
| **Optimization 3:** Using Streams to overlap computation with data transfer |
| * 1. Which optimization did you choose to implement and why did you choose that optimization technique. |
| I choose to implement using Streams to overlap computation with data transfer because it is not hard to implement, also it only influence dimension x of block (Batch size), so I only have to modify one line in kernel. |
| * 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? |
| * Create cuda stream and use cudaMemcpyAsync to copy data to device asynchronously in conv\_forward\_gpu\_prolog() |
| * In kernel I only have to modify one line.      * Defined streamSize than call conv\_forwanr\_kernel with stream[i] asynchronously      * Copy data back to host asynchronously in conv\_forward\_gpu\_epilog() * I think the optimization would increase performance of the forward convolution, because using Streams can overlap computation with data transfer, so that some kernel can start calculating the segment of data that already copied to device while other segments of data are still copying, which will improve performance. * The optimization synergize with all other optimizations. Because all I have to do is modify “int b = blockIdx.x + offset;” in kernel, change cudaMemcpy to cudaMemcpyAsync, and use cudaCreateStream to create Streams. |
|  |
| * 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 | *0.5393* | *0.6613* | *1.2006* | *0.86* | | 1000 | *1.0830* | *4.9036* | *5.9866* | *0.886* | | 10000 | *10.1244* | *38.2340* | *48.3584* | *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). |
| * The Total Execution Time decrease to 93% of previous optimization (7.5% faster) after the optimization. (Use batch size = 10000 to compare) * This optimization successful in improving performance. * From nsys: With this optimization, conv\_forward\_kernel total time decrease.  |  |  |  | | --- | --- | --- | |  | Previous code | This optimization | | conv\_forward\_kernel | 52457030 | 52004476 | |
| * 1. What references did you use when implementing this technique? |
| * https://developer.nvidia.com/blog/how-overlap-data-transfers-cuda-cc/ * ece408-lecture22-GPU-Data-Transfer-sjp-FL21.pdf |
| 1. **Optimization 4:** Multiple kernel implementations for different layer sizes   ***(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. |
| I choose to implement multiple kernel implementations for different layer sizes because I want to tune kernel code of different layer separately to optimize performance. |
| * 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? |
| * I use C to identify which layer is using the code, C == 1 means layer and C == 4 means layer2. * I copy the previous kernel to “conv\_forward\_kernel\_C1” and “conv\_forward\_kernel\_C4”. * For conv\_forward\_kernel\_C1, when doing convolution, c is not iterated because c is always 0. * For conv\_forward\_kernel\_C4, when doing convolution, C is set to 4, c is iterated 4 times, and I use “#pragma unroll 4” to unroll the for loop. * I think the optimization would increase performance of the forward convolution, because conv\_forward\_kernel\_C1 doesn’t have to iterate through c, and the iterate of c in conv\_forward\_kernel\_C4 is unrolled. * The optimization synergize with all other optimizations. Because both conv\_forward\_kernel\_C1 and conv\_forward\_kernel\_C4 are copied from conv\_forward\_kernel, which has all the optimization I have done before. |
| * 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 | *0.2336* | *0.7981* | *1.0317* | *0.86* | | 1000 | *1.0609* | *6.0966* | *7.1575* | *0.886* | | 10000 | *10.1129* | *38.1551* | *48.2680* | *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). |
| * The Total Execution Time decrease to 99.8% of previous optimization (0.2% faster) after the optimization. (Use batch size = 10000 to compare) * This optimization successful in improving performance. * From nsys: With this optimization, conv\_forward\_kernel total time decrease.  |  |  |  | | --- | --- | --- | |  | Previous code | This optimization | | conv\_forward\_kernel | 52457030 | Sum: 52067454 | | conv\_forward\_kernel\_C1 | - | 11029638 | | conv\_forward\_kernel\_C4 | - | 41037816 | |
| * 1. What references did you use when implementing this technique? |
| * Readme of the project. * https://forums.developer.nvidia.com/t/pragma-unroll/3042 |
| 1. **Optimization 5:** Sweeping various parameters to find best values   ***(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. |
| I choose Sweeping various parameters to find best values because it is easy to implement, just change the constants and try to find the parameters that produce the best performance. |
|  |
| * 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? |
| * I modify Layer 1 tile size, Layer 2 tile size, Stream Num, and Row/Col major access pattern, to get the best parameters. * *I think the* optimization will help me find a parameters set that has better performance than before, or maybe the parameters set I used before already has the best performance. * The optimization synergize with all other optimizations. Because all I have to do is modify Layer 1 tile size, Layer 2 tile size, Stream Num. I have already implemented all of them before. |
| * 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 | *0.1335* | *0.7207* | *0.8542* | *0.86* | | 1000 | *1.0468* | *5.7384* | *6.7852* | *0.886* | | 10000 | *10.1221* | *36.8542* | *46.9763* | *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). |
| * The Total Execution Time decrease to 97% of previous optimization (3% faster) after the optimization. (Use batch size = 10000 to compare) * This optimization successful in improving performance. * *In previous optimizations I set StreamNum = 100, but in this optimization I find out that StreamNum = 50 has better performance.* * Below is result of my sweeping. Batch size = 10000  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | *Stream Num = 100*   |  |  |  |  |  | | --- | --- | --- | --- | --- | | *Layer 1 Tile Size* | *Layer 2 Tile Size* | Op Time 1 | Op Time 2 | Total Op Time | | *8* | *8* | *13.5250* | *108.5370* | *122.0620* | | *16* | *16* | *10.1129* | *38.1551* | *48.2680* | | *32* | *32* | *12.2763* | *57.1044* | *69.3807* | | *16* | *32* | *10.2053* | *56.7387* | *66.9440* | | *32* | *16* | *12.2754* | *38.3403* | *50.6157* |   *Layer 1 Tile Size = 16*  *Layer 2 Tile Size = 16*   |  |  |  |  | | --- | --- | --- | --- | | *Stream Num* | Op Time 1 | Op Time 2 | Total Op Time | | *5* | *10.4447* | *39.7660* | *50.2107* | | *10* | *12.1201* | *39.3244* | *51.4445* | | *20* | *10.2465* | *37.6816* | *47.9281* | | *25* | *10.2102* | *37.5382* | *47.7484* | | *50* | *10.1221* | *36.8542* | *46.9763* | | *100* | *10.1129* | *38.1551* | *48.2680* |   *Stream Num = 50*  *Layer 1 Tile Size = 16*  *Layer 2 Tile Size = 16*   |  |  |  |  | | --- | --- | --- | --- | | *Row major access* | *10.1221* | *36.8542* | *46.9763* | | *Col major access* | *10.1824* | *40.5148* | *50.6972* | | |
| * From nsys: With this optimization, conv\_forward\_kernel\_C1 + conv\_forward\_kernel\_C4 total time decrease.  |  |  |  | | --- | --- | --- | |  | Previous code | This optimization | | conv\_forward\_kernel Sum | 52067454 | 49032230 | | conv\_forward\_kernel\_C1 | 11029638 | 10572540 | | conv\_forward\_kernel\_C4 | 41037816 | 38459690 | |
| * 1. What references did you use when implementing this technique? |
| * Readme of the project. |
| 1. **Optimization 6:** Tiled shared memory convolution   ***(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. |
| I choose Tiled shared memory convolution because we already implement it in mp4, and tiled shared memory is supposed to be faster than using global memory. |
| * 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? |
| * I implement strategy two, block size covers input tile. * Shared memory is much faster than global memory, so if we want to access a data multiple times, saving the data to shared memory will improve performance. * The optimization is supposed to synergize with all other optimizations. But after testing the optimization, I find out that it actually make my code slower, so I didn’t merge it into my code. |
| * 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 | *0.3287* | *1.0055* | *1.3342* | *0.86* | | 1000 | *3.1591* | *9.9128* | *13.0719* | *0.886* | | 10000 | *31.2918* | *99.4144* | *130.7062* | *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). |
| * The Total Execution Time increase to 167% of baseline (60% slower) after the optimization. (Use batch size = 10000 to compare) * *The optimization failed in improving performance.* * From nsys: With this optimization, conv\_forward\_kernel total time increase.  |  |  |  | | --- | --- | --- | |  | baseline | This optimization | | conv\_forward\_kernel | 79482366 | 131470546 | |
| * 1. What references did you use when implementing this technique? |
| *Mp4: 3D Convolution* |