EfficeintML.ai Lab 5 Report

Name:\_\_\_Wei Lu\_\_\_\_\_

1. reference:
   1. Section, Total time(ms), Average time(ms), Count, GOPs

reference, 2546.658936, 254.664993, 10, 1.029364

1. Loop Unrolling (20pt): Please fill in the starter code in *kernel/template/loop\_unrolling.cc* to implement loop unrolling and run the `./evaluate.sh loop\_unrolling` to evaluate performance improvement.
   1. Please copy and paste your implementation in *kernel/template/loop\_unrolling.cc*: (15pt)

* 1. How does the performance in GOPs, achieved through loop unrolling on your computer, compare to the reference implementation? Please explain the performance difference. (5pt)

Section, Total time(ms), Average time(ms), Count, GOPs

loop\_unrolling, 2058.592041, 205.858994, 10, 1.273414

GOPs imroved by 23.7%

1. Multithreading (20pt): Please fill in the starter code in *kernel/template/multithreading.cc* to implement multithreading and run the `./evaluate.sh multithreading` to evaluate performance improvement.
   1. Please copy and paste your implementation in *kernel/template/multithreading.cc*: (15pt)
2. How does the performance in GOPs, achieved through multithreading on your computer, compare to the reference implementation? Please explain the performance difference. (5pt)

Section, Total time(ms), Average time(ms), Count, GOPs

multithreading, 675.494995, 67.549004, 10, 3.880769

GOP improved by 277.1%

Since we have 4 threads, the performance is improved by roughly 3 times.

1. SIMD Programming (20pt): Please fill in the starter code in *kernel/template/simd\_programming.cc* to implement SIMD programming and run the `./evaluate.sh simd\_programming` to evaluate the performance improvement.
   1. Please copy and paste your implementation in *kernel/template/simd\_programming.cc*: (15pt)
   2. How does the performance in GOPs, achieved through SIMD programming on your computer, compare to the reference implementation? Please explain the performance difference. (5pt)  
        
      Section, Total time(ms), Average time(ms), Count, GOPs

simd\_programming, 1154.592041, 115.459000, 10, 2.270447

GOP improved by 120.6%

1. Multithreading with Loop Unrolling (20pt): Please fill in the starter code in *kernel/template/multithreading\_loop\_unrolling.cc* to implement multithreading and loop unrolling and run the `./evaluate.sh multithreading\_loop\_unrolling` to evaluate the performance improvement.
   1. Please copy and paste your implementation in *kernel/template/multithreading\_loop\_unrolling.cc*: (15pt)
   2. How does the performance in GOPs, achieved through multithreading and loop unrolling on your computer, compare to the reference implementation? Please explain the performance difference. (5pt)  
        
      Section, Total time(ms), Average time(ms), Count, GOPs

multithreading\_loop\_unrolling, 544.000977, 54.400002, 10, 4.818814

GOP improved by 368.1%

1. Combination of All Techniques (20pt): Please fill in the starter code in *kernel/template/all\_techniques.cc* to implement all techniques above and run the `./evaluate.sh all\_techniques` to evaluate the performance improvement.
   1. Please copy and paste your implementation in *kernel/template/all\_techniques.cc*: (15pt)
   2. How does the performance in GOPs, achieved through all optimization techniques on your computer, compare to the reference implementation? Please explain the performance difference. (5pt)

Section, Total time(ms), Average time(ms), Count, GOPs

all\_techniques, 133.667999, 13.366000, 10, 19.611574

GOP improved by 1440.1%

1. Bonus (20pt): Any optimization techniques on your mind? Try to implement them to improve the performance further! If you can further improve the performance compared to the optimized kernel in [TinyChatEngine](https://github.com/mit-han-lab/TinyChatEngine), you can get bonus points here! Each percent of performance speedup equals one point (create a pull request in the repo and get verified by the TA), up to 20 points.