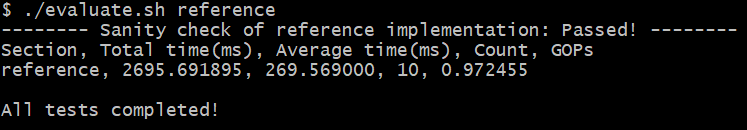
EfficeintML.ai Lab 5 Report

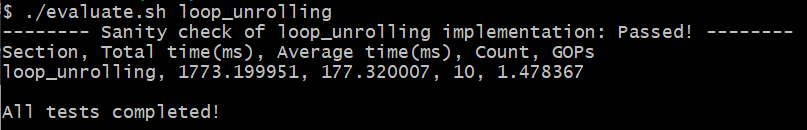
1. Loop Unrolling (20pt): Please fill in the starter code in *kernel/template/loop\_unrolling.cc* to implement loop unrolling and run the `./evaluate.sh loop\_unrolling` to evaluate performance improvement.
   1. Please copy and paste your implementation in *kernel/template/loop\_unrolling.cc*: (15pt)

具体代码请查看：<https://github.com/LordLKY/MIT-6.5940-Labs/blob/main/EfficientML_lab5/tinychat-tutorial/kernels/starter_code/loop_unrolling.cc>

* 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)  
     Reference测试结果：



Loop\_unrolling测试结果：

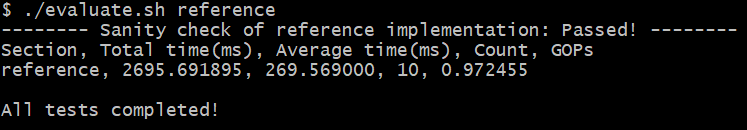


从图中可见Loop\_unrolling取得了的speed up，这是因为Loop\_unrolling有效减少了分支跳转带来的开销，同时unrolling后的循环内部语句可能有助于更高效的底层流水化

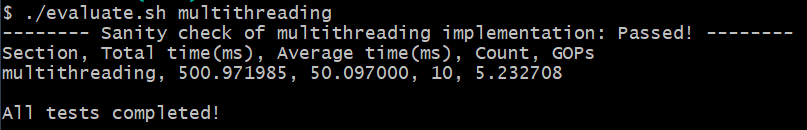
1. Multithreading (20pt): Please fill in the starter code in *kernel/template/multithreading.cc* to implement multithreading and run the `./evaluate.sh multithreading` to evaluate performance improvement.
   1. Please copy and paste your implementation in *kernel/template/multithreading.cc*: (15pt)

具体代码请查看：<https://github.com/LordLKY/MIT-6.5940-Labs/blob/main/EfficientML_lab5/tinychat-tutorial/kernels/starter_code/multithreading.cc>

* 1. How does the performance in GOPs, achieved through multithreading on your computer, compare to the reference implementation? Please explain the performance difference. (5pt)  
     Reference测试结果：



Multithreading测试结果：

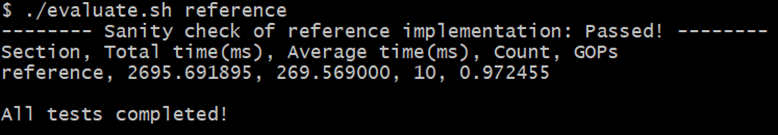


从图中可见Multithreading取得了的speed up，这是因为首先4个线程可以在多核处理器上并行执行，直观地带来了4倍的加速。此外，Multithreading中每个线程需要遍历的权重为Reference的1/4，对cache的利用可能更加充分并带来额外的加速

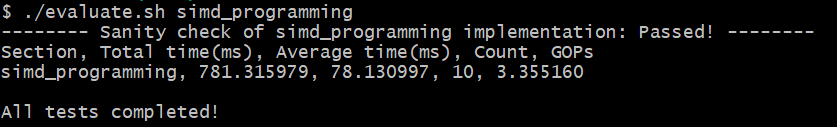
1. SIMD Programming (20pt): Please fill in the starter code in *kernel/template/simd\_programming.cc* to implement SIMD programming and run the `./evaluate.sh simd\_programming` to evaluate the performance improvement.
   1. Please copy and paste your implementation in *kernel/template/simd\_programming.cc*: (15pt)

具体代码请查看：<https://github.com/LordLKY/MIT-6.5940-Labs/blob/main/EfficientML_lab5/tinychat-tutorial/kernels/starter_code/simd_programming.cc>

* 1. How does the performance in GOPs, achieved through SIMD programming on your computer, compare to the reference implementation? Please explain the performance difference. (5pt)  
     Reference测试结果：

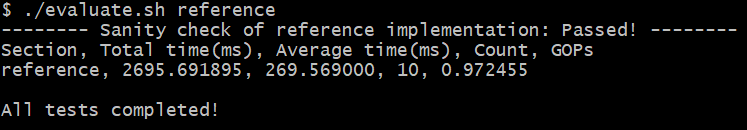


SIMD\_programming测试结果：

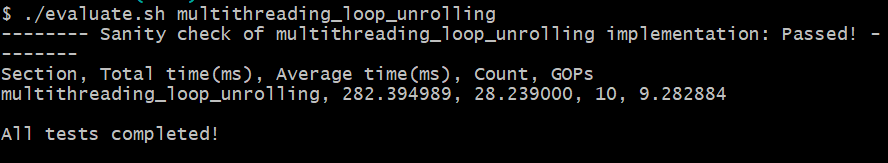


从图中可见SIMD\_programming取得了的speed up，CPU上的SIMD无法带来太显著的效果，从代码中也可以看出SIMD算子的不灵活性也引入了额外的计算，但SIMD还是可以极大地减少指令开销

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)  
      具体代码请查看：<https://github.com/LordLKY/MIT-6.5940-Labs/blob/main/EfficientML_lab5/tinychat-tutorial/kernels/starter_code/multithreading_loop_unrolling.cc>
   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)  
      Reference测试结果：

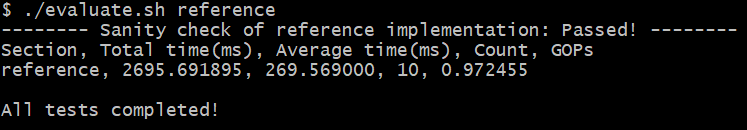


Multithreading\_loop\_unrolling测试结果：

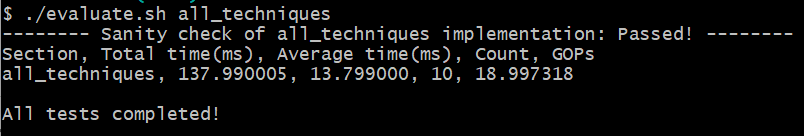


从图中可见Multithreading\_loop\_unrolling取得了×9.55的speed up，与两种优化各自取得的speed up相乘大致相符

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)  
      具体代码请查看：<https://github.com/LordLKY/MIT-6.5940-Labs/blob/main/EfficientML_lab5/tinychat-tutorial/kernels/starter_code/all_techniques.cc> （注意将extra\_techniques设置为false）
   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)  
      Reference测试结果：



All\_techniques测试结果：



从图中可见All\_techniques取得了×19.55的speed up，主要来源于线程数相较之前增加了一倍，SIMD能带来的加速比较有限

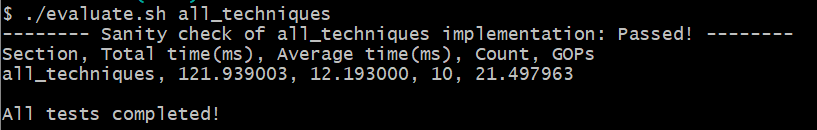
1. Bonus (20pt): Any optimization techniques on your mind? Try to implement them to improve the performance further! If you can further improve the performance compared to the optimized kernel in [TinyChatEngine](https://github.com/mit-han-lab/TinyChatEngine), you can get bonus points here! Each percent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

在All\_techniches的基础上可以考虑如下优化：

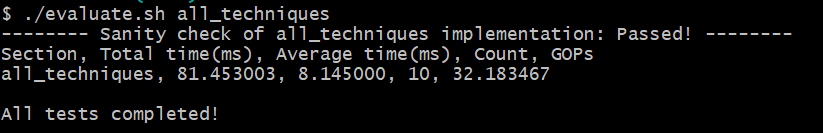
* 1. 进行Loop\_tiling以适配各级cache
  2. 在循环读取SIMD数据时使用Double Buffering

由于测试的是推理时矩阵向量乘法的性能，优化a的效果可能不显著，因此选择优化b进行实现，具体代码参见：<https://github.com/LordLKY/MIT-6.5940-Labs/blob/main/EfficientML_lab5/tinychat-tutorial/kernels/starter_code/all_techniques.cc> （注意将extra\_techniques设置为true）

All\_techniques测试结果：



Extra\_techniques测试结果（All\_techniques + Double Buffering）：



可见Double Buffering在All\_techniques基础上取得了×1.50的speed up，Double Buffering可以有效overlap对SIMD数据的访存与计算，效果还是比较显著的