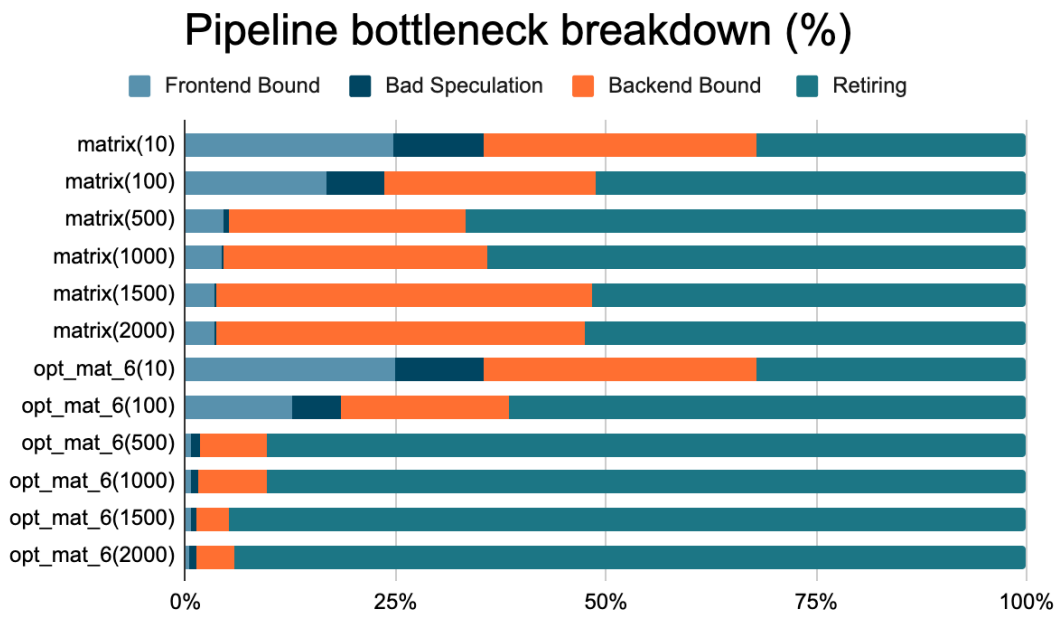
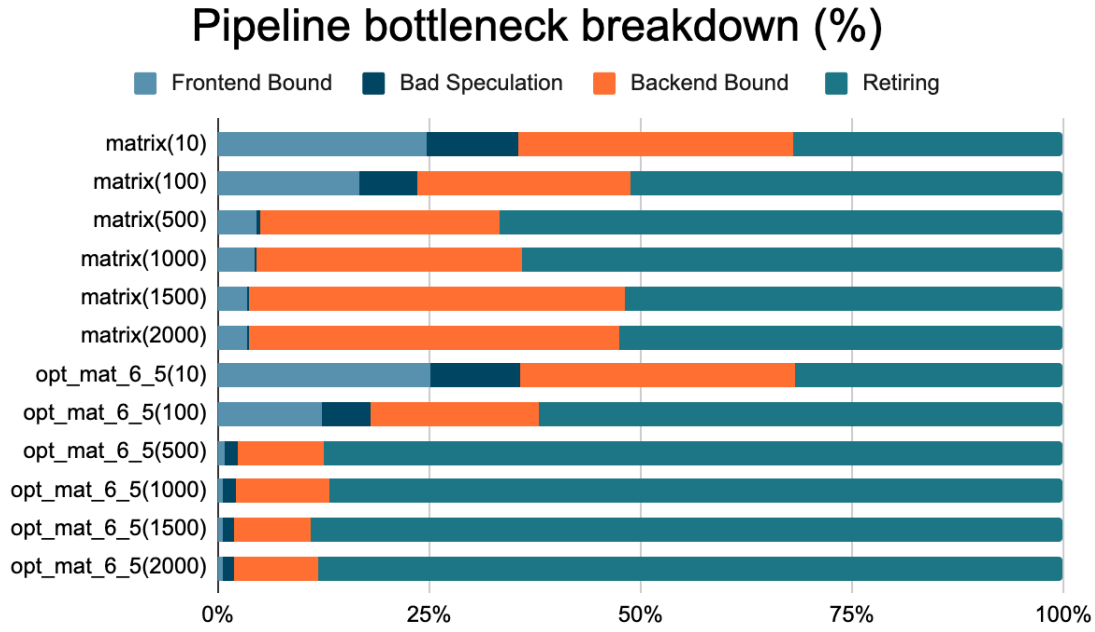
Report - Lab2

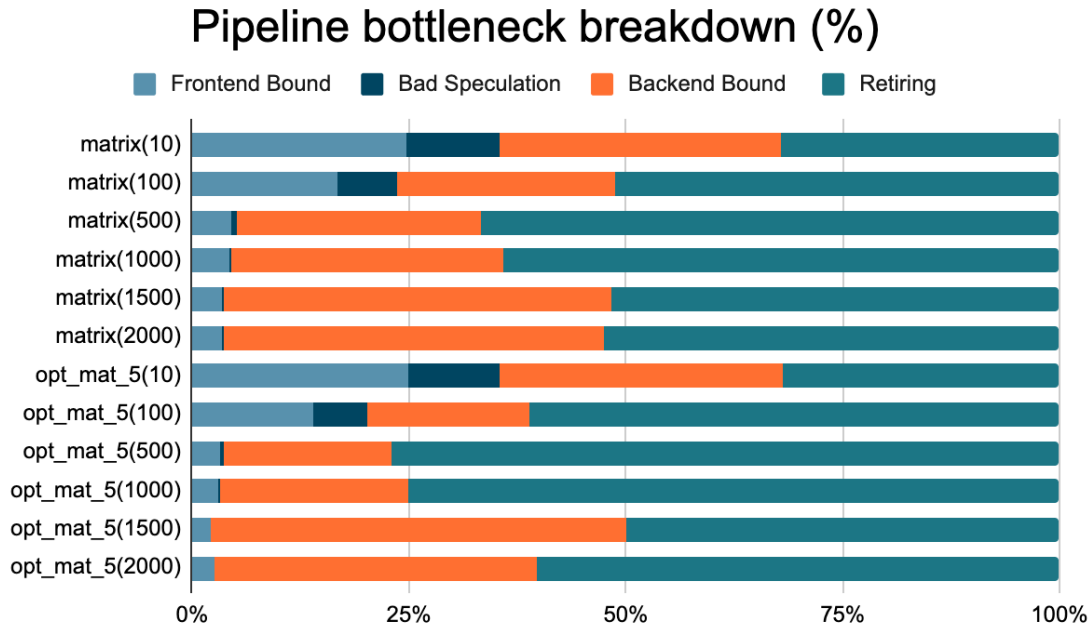
Siyu Meng sm2659



Block Size : 16
Number of Loops: 6



Block Size : 5
Number of Loops: 6



Block Size : 16
Number of Loops: 5

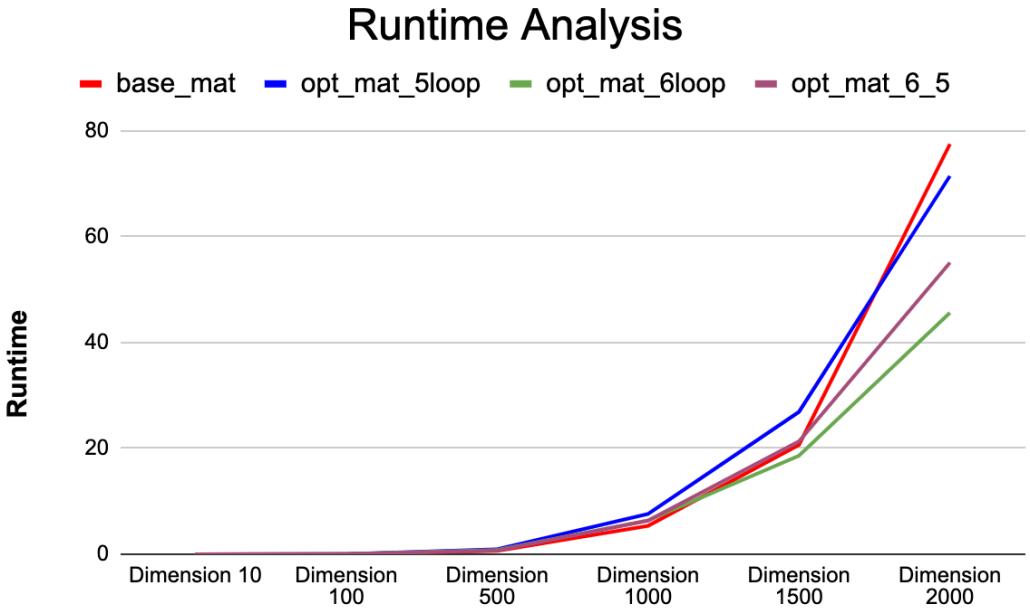
Profiling Process:

- Write both the five and six loops matrix multiplication. (opt_mat_5 and opt_mat_6)
- Using one of them with different input sizes to do the profiling with the base matrix multiplication. (the number in the bracket is input matrix dimension)
- Test six loops matrix multiplication with both 16 and 5 block size to see the block size influence for profiling (opt_mat_6 and opt_mat_6_5)

Top-down Analysis of performance:

- Lower backend bound:
    - Loop tiling, which divides computation into small chunks, enhances cache utilization. This reduces cache misses and, consequently, the time spent fetching data from main memory.
    - The six loops method used more locality than the five loops and base method. The six loops largely reduce the backend bound breakdown compared with five loops and base methods. Even though the five loops do not perform as well as the six loops, compared with the base method, the backend bound still has some improvements.
    - For six loops with different block sizes, the 5 block size compared with 16 block size is not optimal to reduce the cache miss. It might take more time to fetch data from main memory or higher cache levels.
    - When the input size increases, the backend bound reduces a lot because of the improved cache utilization with large matrices.
- Higher retiring:

- When data is fetched into the cache and used multiple times before being evicted, the CPU can execute and retire those instructions without waiting for slower memory accesses.
- Efficient loop structures, like those produced by tiling, can lead to fewer pipeline hazards, allowing more instructions to retire per cycle.
- The regular and predictable access patterns in tiled matrix multiplication can also help branch predictors and other hardware prefetching mechanisms work more efficiently, leading to fewer mispredicted branches and thus a higher retiring rate.



Performance difference:

- Loop tiling introduces additional loop controls and potentially other fixed overheads. For small matrices, these overheads might represent a more significant fraction of the total work, which could increase the runtime. For larger matrices, the actual computation and memory accesses overshadow these fixed overheads, and the relative cost of the overhead diminishes. Thus, the runtime will be faster.

Issue and debugging:

- When I run with the input dimension size 1000 with matmul and matmul_blocking method, the runtime of matmul is faster than blocking. I felt strange and tried several larger sizes to see what happened. After increasing the input dimension, I am able to see the large advantage of using loop tiling doing matrix multiplication.
- When I first write loop tiling, I miswrite the start point of the block loop from 0, which leads to a really long runtime. I discussed with friends to see if they also run for a long time and recheck my code to see where I got wrong and fix that bug.