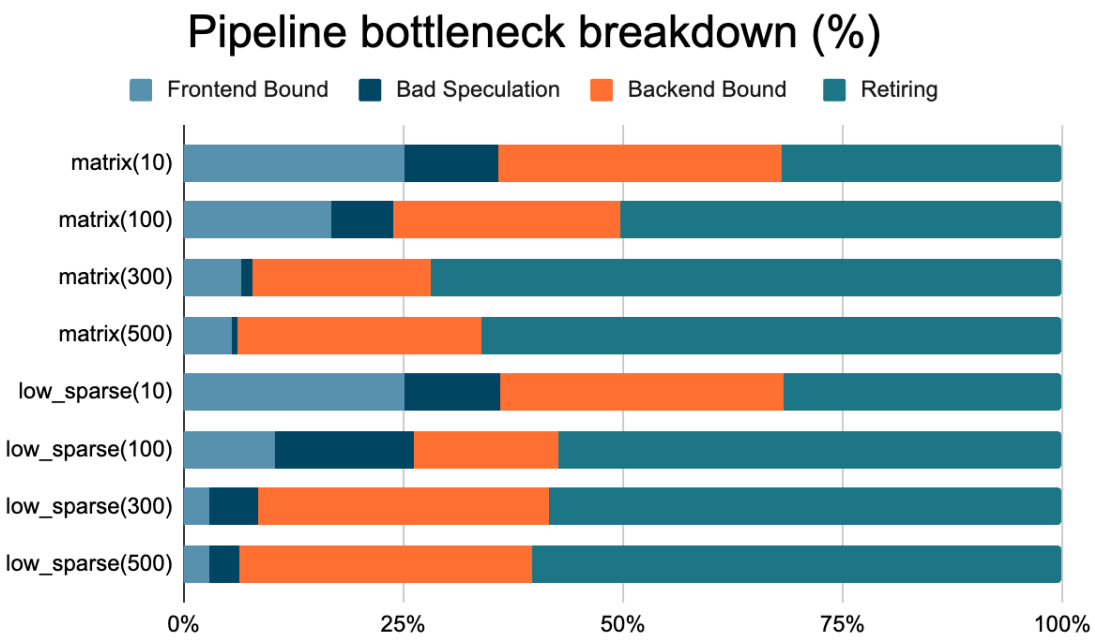
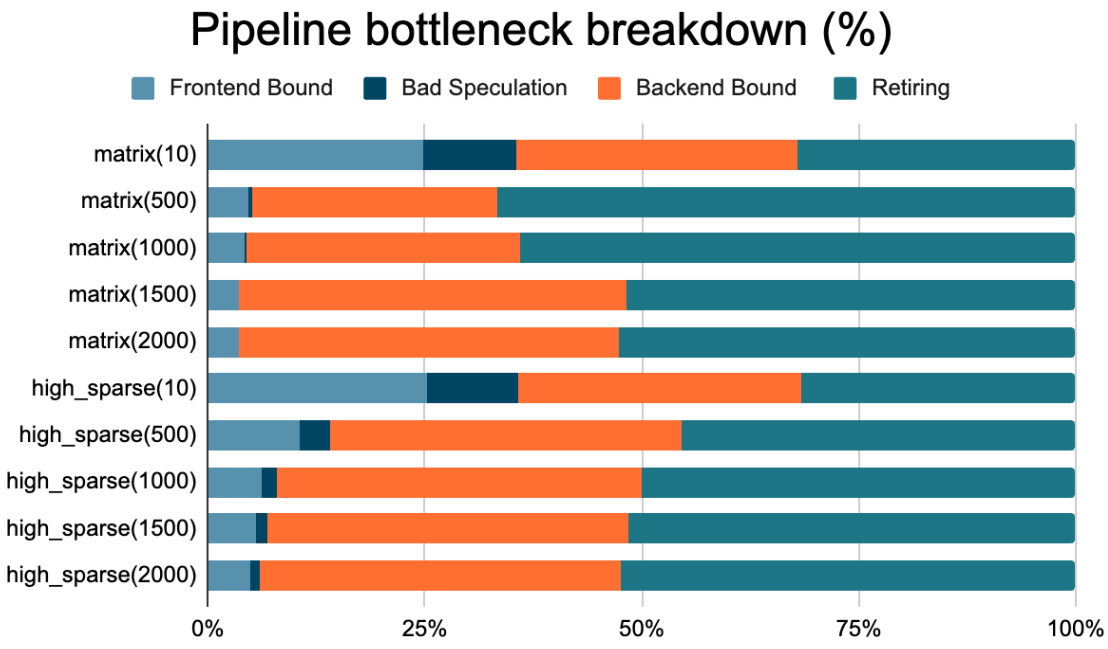
Report - Lab3

Siyu Meng sm2659



Low sparsity: 0.2 (20 percent of matrix is non zero value)



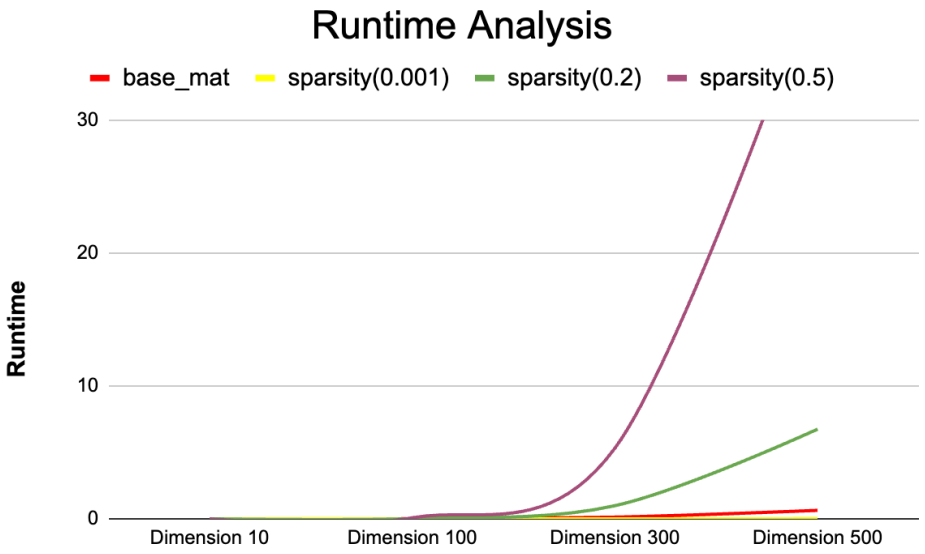
High sparsity: 0.001 (almost all the elements are zero except several non zero elements, more like a one-hot matrix).

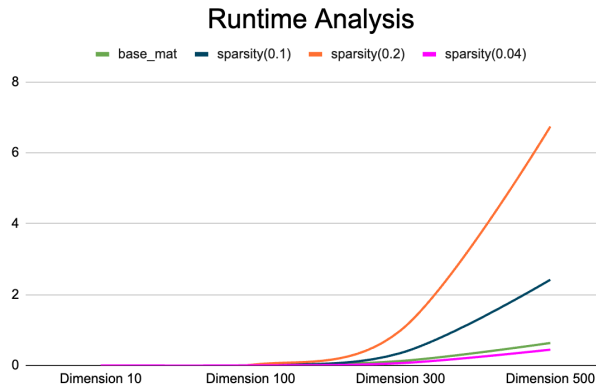
Profiling process:

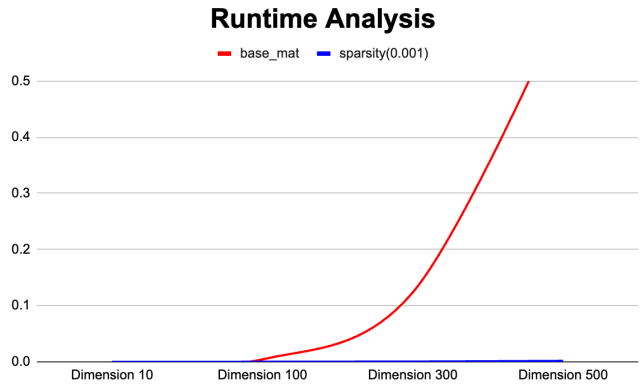
- Set the sparsity as 0.2 and change the input size from 10 to 500.
- Set the sparsity as 0.001 and change the input size from 10 to 2000.

Top-down Analysis of performance:

- Low sparsity matrix multiplication have larger backend bound compared with baseline
    - Low sparsity means there are more nonzero values in the matrix. When we convert dense matrices to CSR matrix representation, which include values, rows and columns. In the baseline, the memory only stores the values. However, for a CSR matrix, it needs to store both columns and values, which will double the access of memory.
- High sparsity matrix multiplication have High frontend bound
    - Each access to a non-zero element involves looking up the row and column indices in separate arrays, which can introduce additional latency in the frontend, particularly if the matrix is large and non-zero elements are scattered.
- Different sparsity matrix have similar backend bound
    - The use of CSR format results in memory access patterns that may be cache-unfriendly, particularly when accessing row and column indices. Both high and low sparsity matrices can exhibit suboptimal cache behavior, leading to backend bottlenecks associated with cache misses and inefficient memory access.







Performance difference:

- The high sparsity runtime is faster than the baseline. Because we extract meaningful numbers from the matrix and reduce the runtime of calculation of zero elements in the baseline.
- The low sparsity runtime is slower than the baseline. Because originally we only store values and need to access values one time. For the CSR matrix, we need to store values and columns, which will double the memory access and reduce the runtime largely.
- I find that after the sparsity value is over 0.04, the sparse matrix multiplication does not have an advantage over the base matrix multiplication.

Issue and debugging:

- I got the segfault when I tested with all zero and only one non-zero element matrix. I posted on piazza and went to the TA's office hours. I used valgrind and gdb to see where I got the memory leak. After finding which line caused the segfault, I went through my logic and double checked the array index and size. I solved the memory leak and was able to run the test case and profiling.
- I find that sparse matrices do not do well for some dense matrices that are over some threshold. Therefore, we should choose appropriate matrix multiplication depending on the properties of the matrix.