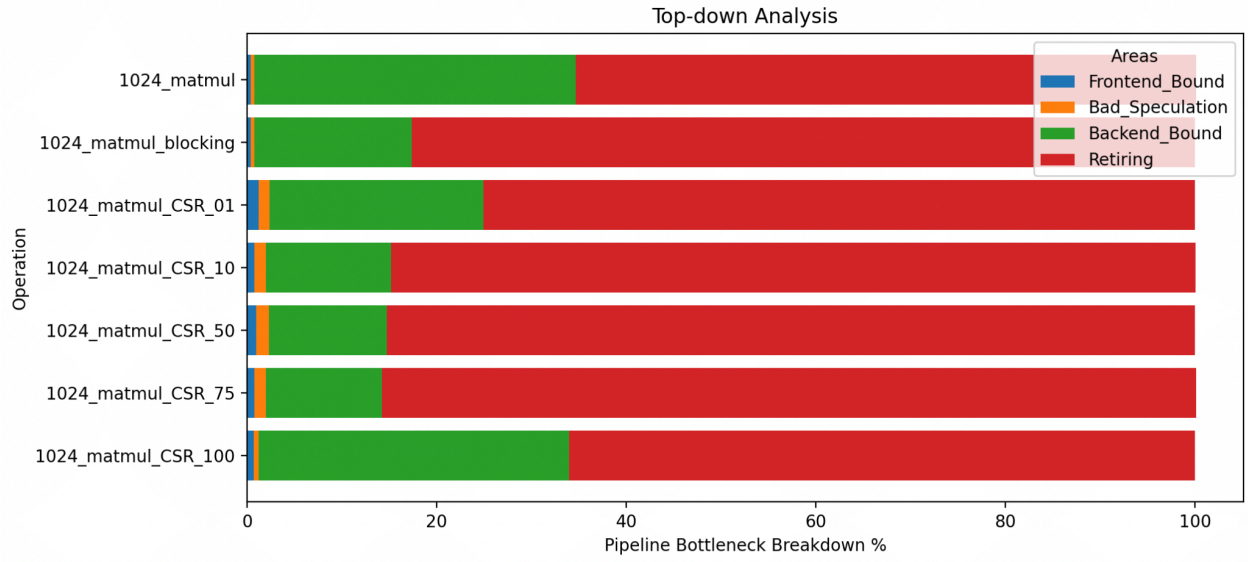
Lab 3

Aneesh Pandoh 11/11/2024

Setup:

Ran Top Down Analysis on each test 5 times. For each operation, I ran the tests 10 times within the function itself, allowing me to test the runtime of matmul specifically and have the overhead of converting to CSR be an insignificant portion of runtime (*Note: I removed/commented the 10x loop in the turned in matrix_ops kernel). I tested on a matrix size of 1024 and compared it to our previous baselines for matmul and matmul_blocking. The tests were set up to have varying levels of density (1%, 10%, 50%, 75%, 100%), this was done by using a random condition conforming to the sparsity level while setting up elements.

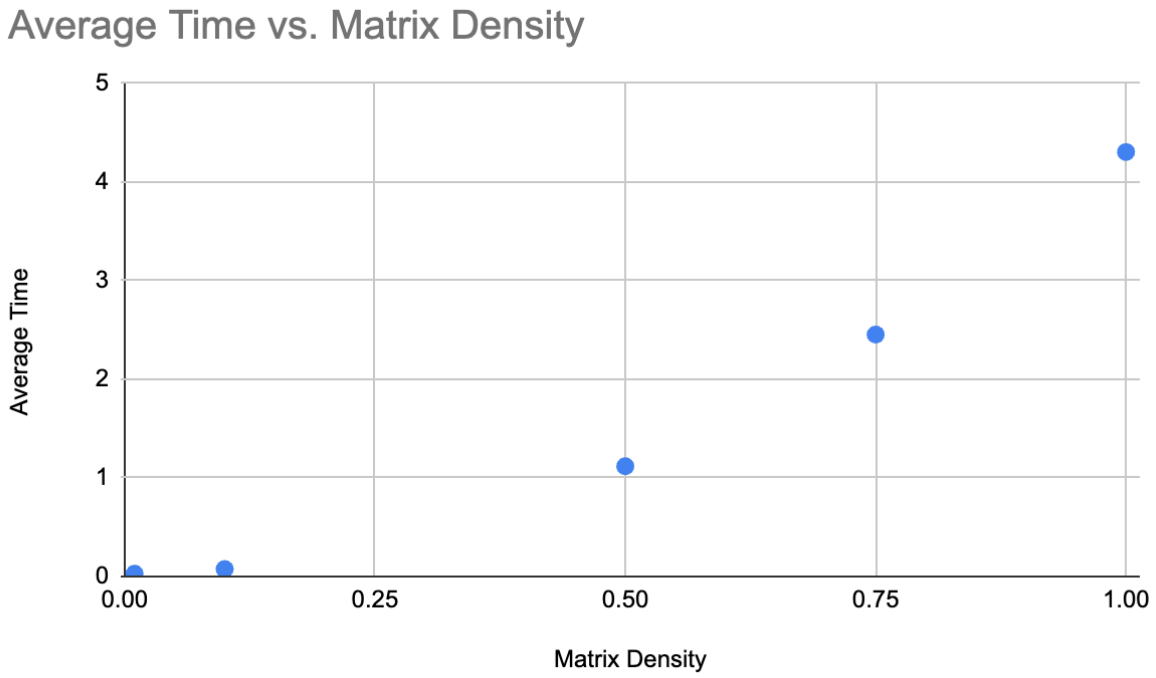
Top-Down Analysis



The results reveal that dense CSR matrices (1024_matmul_CSR_100) and regular matrix multiplication (1024_matmul) exhibit similar performance, with a high percentage of “Backend Bound” instructions relative to other configurations. This is because a 100% dense CSR matrix behaves almost identically to a fully dense matrix as we do not get to skip over any operations. In contrast, matrices with lower densities (10% , 50%, 75%) show an decrease in backend-bound stalls. This is due to the sparsity of the data, which is taken advantage of in CSR format as we can skip performing operations on elements with a value of 0. Surprisingly there was much difference between 10-75% level sparsity regarding bottlenecks, however, what's interesting to see is that when the density is down to 1%, it becomes increasingly backend bound relative to matrices with slightly less sparsity, this is probably due to the fact that 1% of 1024, would be around 10, meaning we would only need to perform 100 multiply-add operations max. This means that the initial load from memory (backend) accounts for a larger share as the time to complete the operation is trivial.

We can also see that the bad speculation portion amount increases which is probably due to the column index array and the row index array being non-standard size but rather depending on where the sparse data lands within the array.

Runtime



We can see that the CSR format saves a significant amount of time, as we can see an almost quadratic increase as the density of the matrix increases. This is probably due to the calculation that we save around x^2 operations per 0 element we skip in CSR. This is also faster than our regular matmul which came in with a time of ~5.2 seconds, in comparison to the 4.3 seconds of a fully dense CSR.

Interesting observations:
CSR was faster than both regular MatMul and MatMul with blocking even at 100% density. I expected it to be around the same as MatMul with no optimizations, but rather it is around the same as MatMul with blocking, I believe this might be due to the memory access pattern as we access sequential data as it is now stored row-wise in CSR, where as in traditional MatMul we need to iterate over one array column-wise results in significant performance hit as we have poor cache utilization of not being able to read data from within a cache line but are instead probably evicting cache lines over and over.

Issues:
Did not have many issues with this lab.
*Used github copilot / and chatGPT to speed up the process writing tests and implementation

Improvement:
Implement dlp and tlp