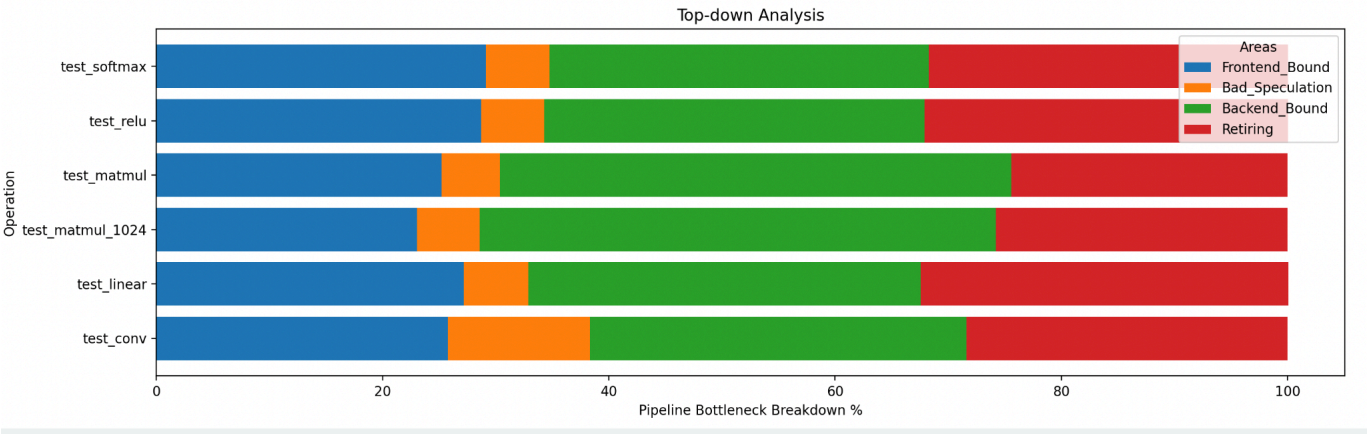
Lab 1

Aneesh Pandoh 10/09/2024

Setup:

Ran Top Down Analysis on each test 10 times. For each operation, I ran the tests 100 times (besides MatMul_1024 as it would log me out of the server before completing, so it is set to 5). There are **3 tests (inputs) per operation**, that would test different inputs sizes and numbers.

I decided to combine all of these different sizes to get the impact of the functions themselves. The only input size that is separate is matmul_1024 as it a very mem bound operation and was simply just taking too long and thought it would be interesting to compare it to the matmul of a smaller inputs.



Results:

The most prominent result from the graph was the percentage of bad speculation contributed to convolution(). This is probably due to the amount of loops causing more branching along with the fact that to run the test, multiple functions had to be called just to set up data. We can also see that softmax and relu had very similar results, I assume this is because both function simply perform arithmetic operations while linearly iterating through the array, thus not jumping around too much. We can see that the matmul operation was significantly more backend bound, which only increased with the larger 1024x1024 matrix size. This is probably due to that we have to load large amount of data to do a $O(n^2)$ calculation, the larger performance hit as the 1024 float size will take more than a single cache line compared to the other matmul test.

Implementation:

Conv:

- Handled edge cases of NULL inputs and image being smaller than the kernel being applied
- First we iterate through the filters, then we go through the x and y dimensions of the image which is where we add the bias data to start. We then iterate through the channels and the kernel's dimensions where so that we can implement the formula: output[f][i][j] += image[c][i + ki][j + kj] * kernel[f][c][ki][kj], which is where we take the image value, image[c][i + ki][j + kj] and multiply it by the kernel weight, kernel[f][c][ki][kj] and store it in the position (all the elements in the current kernel dimension get added to the same output index).
- It is important to note that we calculate the output size beforehand (primarily for allocation), this can be done through the formula inputSize - kernelSize + 1

Relu:

- If x is greater than 0, return x, else return 0
- Apply this function to all the values in the given input array

Linear:

- Handled edge cases of NULL input
- We start by setting the output array index values to corresponding biases, then we compute the dot product of the input and corresponding weight col by iterating through it.

MatMul:

- Handled edge cases of NULL input and the dimensions not being valid for matrix multiplication
- Go through the rows and columns of A and B matrices respectively and multiply those values to get a single value to put in the corresponding output index
- Did not do any blocking optimizations to fit within cache lines, etc.

Softmax:

- Handled edge cases of NULL input

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- First we find the maximum value of the input vector to improve numerical stability, preventing overflow. Then compute the exponential of each adjusted input, sum these values for normalization, and apply the natural logarithm to produce log-softmax values.

Tests:
test_conv:

- Was restricted by the data_utils.c func so could not test different image and kernel sizes
- Added test for edges case where input image is less than kernel size
- Added test with different numbers

test_linear:

- Added test for edges cases, empty input or 0 weights/biases
- Added test for sizes 512 input length (medium) and 1024 input length (large)

test_matrix_ops:

- Added test for non square and different dimensions (small)
- Added test for matmul of 1024x1024 matrices (large)

test_others:

- Added more flatten() tests with different numbers (not sizes, because was still limited by utils)
- Added tests for applyRelu() with different numbers and input sizes
- Added tests for softmax() with different numbers and edge cases

Issues:
Had general issues logging onto the VM (especially through vscode ssh), similar to the posts on EdDiscussion.

*Used github copilot / and chatGPT to speed up the process writing tests and implementation

Improvement:

- Implement blocking to have the data better utilize cache lines
- Implement data/thread level parallelization