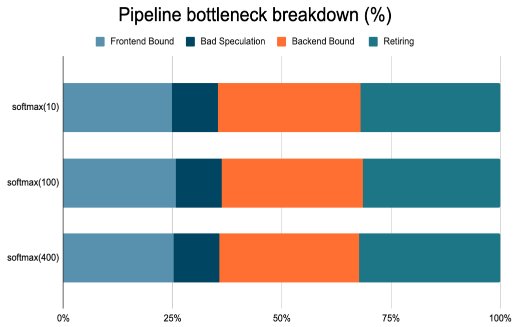
Report - Lab1

Siyu Meng sm2659

SoftMax:

Implementation: First, I allocate memory for output and calculate the sum of all the inputs' exponential. Then go over the input list again to calculate the input's exponential over the total exponential sum and put it in the output list.

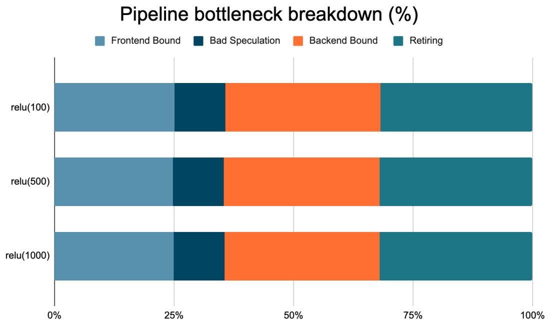
Analysis: I increased the input list length from 10 to 400 to do the profile. I find that even though I increased the size, the pipeline breakdown does not change a lot. What I think the reason behind is that this function is simple and already highly-optimized. The pipeline breakdown might be well-balanced for the given workload, and increasing the input size might not reveal new bottlenecks. The main the bottleneck for this function is backend bound and retiring. I think it is caused by accessing memory to get input and execution and put the result back to the output memory.



Relu:

Implementation: If the input is larger than zero, I return the input directly. Otherwise, return 0.0.

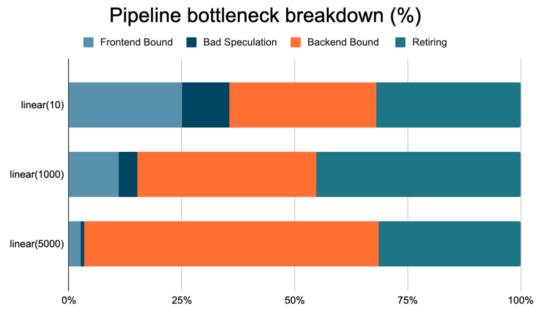
Analysis: Similar to the softmax function breakdown. I think the reason for the stable breakdown as the input size increases is also the same. The function code is simple and already highly-optimized. The main bottleneck is backend bond and retirement. However, except the bad speculation, the remaining three percent are similar. I think the bad speculation is small because I only have one if condition, which makes it have less mispredictions.



Linear:

Implementation: I allocate output memory and use a double for loop. The inner loop goes through the input and weights to do the product. The outer loop adds the bias value to the product and puts it into the output list.

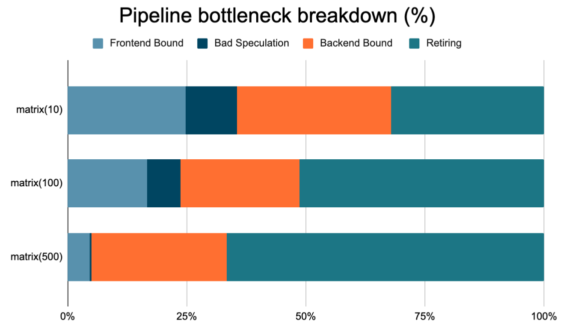
Analysis: The backend bound percent increases significantly as the input size increases. I think the reason is that when I increase the input size, the L1 cache is not enough to store the value I calculated in the for loop. Therefore, it needs to go to L2 cache or main memory, which will largely reduce the runtime.



Matrix:

Implementation: First, I allocate memory for the output matrix. Then I use three for loops to go over each row of the matrix A and each column of the matrix B at the same time and sum all the products. In the second for loop, I put that sum into the output C matrix.

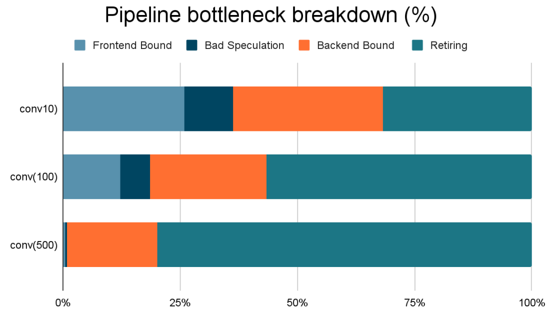
Analysis: The retiring percentage increases a lot as the matrix size increases. I think it is caused by instruction dependencies. There are lots of data dependencies between instructions. Because the inner for loop calculates the sum of all the products depending on each number in row A and each number in column B, the retiring stage might have a hard time to find independent instruction to commit, which leads to a retiring bottleneck.



Conv:

Implementation: Within the convolution function, I use seven for loop. First one to get the num filters, the second one to get the numchannels and then calculate one image channel with one filter sum by element-wise calculation and put sum into outputconv based on the numfilter and outputsize. At the end of each channel loop, I use another for loop to add bias and apply relu function to outputconv.

Analysis: Similar to matrix multiplication, the retiring increases a lot as the input size increases. I think the reason is similar. Because there are lots of instruction dependencies in the convolution function. We need to wait for product data to get sum. And continue the following dependent instructions.



test_matrix_ops.c :

Similar to test linear, I allocate memory and create two matrices with shape (2,3) and (3,1). I calculated the result by hand as the expected result, which compares with the output from the matrix multiplication function. After performing the test, free all the allocated memory.

test_conv.c:

Within the test convolution file, I used the example given in the lab1 pdf to test the correctness. First, I allocate memory for image, kernel and bias. Then call the convolution method with these parameters. I create the expected output based on the example and compare each value of convolution method result with it. After performing the test, free all the allocated memory.

test_linear.c:

I create inputs with size 3, weights with shape(3,2) and bias with size 2. First, I allocate memory for inputs, weights and bias. Then I create my own data points and calculate results by hand, which are used to compare with the result from the linear function. After performing the test, free all the allocated memory. I also create a test case to see if the input is empty and input size is 0, the result should return null.

Discussion:

- I find that when the input size is larger than 500, the softmax sum of the float will add a small float or sub small float number such as 0.00001 to the result, which will make the result unable to pass the unit test because it is not within the range. I googled for the reason and found that float is not precise in C and it will automatically add some small number at some points. Therefore, I control the input size within the 500.
- The output of the matrix has a huge number. I print out the input value to check if it is correct. Then I print out the temp result within the for loop. I find that some of them are correct and some of them are really huge. I realize that it is not the value that I put in or calculate. It is a random number. Then I find that I did not initialize the output and directly use it.
- One more interesting observation is that for linear function, if I only increase the input size and keep the output size constant, the pipeline does not change a lot. I think it is because I do not need much memory to store the output and access them again for bias. I think I could improve by creating a temp value to store the sum rather than access the output list every time I want to add on.
- I think for matrix and convolution functions, I could reduce the use of for loop. Or if I could store some data for later use, which could make more instructions parallelized. Moreover, I might be able to create a local fixed length n-dimensional matrix, which does not need to allocate any memories. After I calculate and get the result, I could copy all the values from the local parameter to memory, which will largely reduce the memory access.