**Micro-Processors Report Milestone 3**

in this milestone we started using broadcast and reduce rather than using send and receive,

the percentage of the parallelized code on the source code level is around 20%

the use of parallelism in our code is just the matrix multiplication part of the code

hence the remaining part of the code which serves the rest of the functionalities of the code is not parallelized, moreover, it’s safe to say that the other parts of the code though not part of our idea or plan can also be parallelized since we can parallelize the nodes on each layer or even if we tweak the code a bit we can parallelize each layer somehow, which in a way is how distributed machine learning works.

However, out of the matrix multiplications we can say around 70% of the work if not more can be parallelized since we give every process a pointer to work with and then send the results back to the root process to do combine the answers together in a larger array of all pointers.

On the instruction level parallelism there are 2 major loops that we should look observe, the one at dotprod which is around r+4 instructions that loops over the number of rows in an array to preform the dot product of a 1xr array with an rxr array to produce another 1xr array which is the nodes of the next layer, so each proc takes a row and they do the weighted sum on it.

The other loop is almost the same for dotprod2 where the input is now rx1 for matrix 1 and 1xc for matrix 2 which runs for c+2 instructions where c is the number of columns in the other array where we send the second array along with a the ith value in the first array to compute the dot product and then return a pointer that should be concatenated together to form the rxc matrix

So, we got epochs\*(numInputs\*(r+4+c+2)) where epochs is the number of iterations num inputs is the num inputs of each training (which is the batch size but for our implementation we use a batch size of one) out of the whole code we can say that 40% of the code in the instruction level is parallelized since the loops are mostly in the parallelized part which increases the number of parallelized instructions

The main difference between using broadcast and reduce rather than send and receive is the fact that with send and receive you need to do it separately for every process rather than sending once from the root to every other or reducing from all the processes at the root

So, we don’t have to use a for loop to receive from/send to all processes we can simply use broadcast/reduce to reduce the lines of code used

both broadcast and reduce can be used in the code

broadcast to send to all processes stuff like when to start executing, the input received from the user, and anything that one node has that should be sent to all other nodes,

reduce can be used when doing the dot product as concatenating the output array into an array of array for the input, simply as if we’re doing gather. However, reduce operators don’t contain such function so using it would mean having to create one ourselves, it’s a hassle to try to create the function ourselves since we need to know who’s array is sent first and define the operation for all processes since the order of the pointers in the array matter, Moreover, not all procs in our implementation always have something to do, so for example if the 2d matrix is of size 3x3 and we have 4 processes only 3 of those would be running since each one will take a pointer of size 3 to work with, and with reduce we need all processes to be able to send the data back to the root, this could be fixed by the use of dummy data that the unused nodes send back to the root and then having the root remove the unneeded data but we thought that would be processing power and execution time wasted.

So, we used broadcast in our implementation however we chose not to implement reduce since the operations supported doesn’t help in concatenating arrays into an array of arrays along with the reasons stated above.

That being said, broadcast could be done with better implementation allowing for faster execution time, rather than sending from proc 0 to all other procs we can use a tree layout such that proc 0 sends to 1 and 2, then 1 sends to 3 and 4 while 2 sends to 5 and 6 for example,

We Can say that whether using broadcast or send and receive the execution time is mostly the same, but however better than using the sequential code