Kevin Nguyen

1) Explain your design of multi-process structure and IPC

The multi-process structure is designed in a way where each child process will calculate a row of the C_parallel matrix. In the parallel_mat_mult method, there is a loop that first fork M child processes and give it an integer that corresponds to the row in the first matrix, the writing pipe, and the crash rate. All child processes will concurrently do the dot product between the given matrix A's row and all the transpose matrix B's rows and communicate it back to the parent process by writing the results to the writing pipe. Afterwards, the child process will close the writing pipe. Meanwhile, the parent process will be waiting for the child processes to finish then check the exit status. If the child process exits successfully, the parent process will read the calculation through the reading pipe. The parent process will then input the result in the C_parallel matrix then close the pipe for that child process once it finishes getting the entire row. If the child process crashes, the parent process will fork a new process with the recovery mechanism explained below. This will repeat until all the rows in C_parallel are done.

2) Explain how your program handles crash

The way that the recovery mechanism works revolves around the idea of shifting crashed child processes towards the beginning of the pid array, then looping through those failed processes again, and repeating this shifting pattern until there are 0 crashed processes left. There are a few variables that are used to get this mechanism to work:

1. pid array (pid) - array of all the pids
2. pipe array (pipefd) - 2d array of the pipe for IPC use
3. pid to pipe array (pid_to_pipe) - array of indexes that map the pid to the correct pipe after shifting the pid to a new index
4. head pointer (fail_p) - point to the head of the pid array where a new pid for a failed process will be overwritten
5. end pointer (end) - marks the ending of the array, changes after every iteration to only account for failed processes to reloop
6. process counter (proc_left) - number of processes left

The parent process is looping until the process counter reaches 0. When a child process fails, the parent process will fork a new pid and place it at the first index in the pid array where a different pid has already finished executing. This is accomplished with the head pointer variable which points to the first index where it is safe to overwrite the index with a new pid. The parent process will also update the pid to pipe array to point to the old index so that we can keep track of the pipe that belongs to that pid. After the first iteration through the pid array, the end pointer will update and point to the last index of the failed processes. The head pointer is reset back to the beginning of the pid array. The parent process will then loop through this revamp list of pids, any failed process will be forked again and the new pid will overwrite the pids that have already finished (shifted towards the beginning of array). This process is repeated until the process counter reaches 0.