Dhruv Maniar R#11713343
Brandon Bratcher R#11520450
James Gibson R#11671129
Richard Pearson R#11786900

We were tasked with parallelized a section of code such that each process holds a sub-matrix D and D0, which make up a larger matrix. These matrices can be arranged into a grid-like pattern of sqrt(p) * sqrt(p) where p is the number of processes.

Each submatrix must access row and column data that is outside of its bounds. This data updates each iteration. To achieve this, we performed a row-wise and column-wise one-to-many communication each k-iteration. K behaves as a global index for the matrix and each process has local i and j indexes for its submatrix. For any given k iteration, other processes require the row data given by the submatrix[i][k] and the column data given by submatrix[k][j]. If the matrices share an i index, then they require column data from the submatrix k intersects; if they share a j index, they require row data. Each communication sends an n/sqrt(p) package of data where n is the length of one dimension of the full matrix. Each process receives or already has the row and column data it needs. There is sqrt(p) one-to-many communications happening for each column and each row at the same time. Since they are all parallel using our methodology, the communication complexity is log2(sqrt(p)) times the data sent n/sqrt(p).

To achieve the one-to-many communication, the process containing the required data is chosen as the process to begin iteration 0. At iteration 0, this process sends to its immediate neighbor. Every following iteration, each process that has received data sends to a process that has yet to receive data. With this method, a log2(sqrt(p)) time complexity is achieved, where sqrt(p) is the number of processes involved in the communication. Since the data proliferates rightward or downward(process ids to send to increase), a process could attempt to send to a process outside of the row or column bounds. To address this, a wrapping technique was used so that any data being sent “out-of-bounds” is instead sent to previous processes which have yet to receive data. This ensures that all members of a row or column receive the correct data.

Our algorithms used for implementation match how our communication pattern works on paper, and it achieves the required level of parallelism.