Floyd-Warshall Parallelization Design

Floyd-Warshalls algorithm is designed to find all shortest paths between all pairs of vertices in a directed and weighted graph and can also be used to detect negative cycles. In this analysis and design document, I will cover my approach and design process for implementing a parallelized version of the Floyd-Warshall algorithm given a 5000 x 5000 matrix. This process will implement the cornerstones of parallel program design including partitioning, communication, agglomeration, and mapping. Through the execution of the parallel program design steps, I have been able to develop optimized and efficient palliation of the Floyd-Warshall algorithm that has been optimized for my specific hardware and can also be optimized for a variety of hardware setups with any number of cores or threading designs. The current hardware that this program was designed to run on in its deliverable state is as follows. AMD Ryzen 7 1700 Eight-Core 3000 Mhz with 16 logical processors.

In the partitioning stage of the parallel program design process, the purpose is to examine opportunities for parallel execution. This process is conducted by breaking down the task into as small pieces as possible. The object is to divide the computational and data handling processes into as small of a problem as possible. This is accomplished by either implementing domain decomposition techniques or functional decomposition techniques. During my process, I was not able to identify areas in the computation where functional decomposition could be utilized regarding parallelizing the problem and chose to implement domain decomposition. The domain decomposition was achieved by dividing the adjacency matrix into various subparts. I attempted to divide the partitions into as many subparts as possible initially to provide for later optimization and give me space to work with varying the partitions in later steps. Given that I had 8 processors and 16 logical I specifically targeted dividing greater than this number. I also designed the partitioning process to have equal sizes when the partitions are decided by dividing the matrix up evenly. I was not able to find an alternative way to divide the partitions up besides the one described.

The communication phase of the design process is used to identify parts of the computation that require communication and to find a way to best optimize this during the parallelization process. This was one of the most difficult parts of the design that I found, and I spent a long time attempting to devise a solution. I realized that when calculating the various paths with multiple threads that there was a dependency on calculations during updating paths that had not been calculated yet but may have had shorter paths during parallel thread execution or future thread execution. My intention in the communication for updating the distance matrix was to seek out a way to communicate the updates but I was not able to find a solution for inter-thread communication while working on the problem in parallel. Assessing my solution all of the tasks were able to communicate operations concurrently and all computations were able to happen with each partitioned part of the problem.

The agglomeration part of the design process requires us to revise the decisioning that we made in the partitioning and communication phases. We now seek to further define what algorithm we will use in the parallelization process. We attempt to see if there are opportunities to combine the various tasks from the previous stages into smaller tasks and also assess if it's reasonable or effective to replicate data or specific tasks in the algorithm that we design. During this process, I later realized that the part of updating the distance matrix resulted in inaccurate results during parallelization. I later decided to implement a solution where during each thread's calculation the matrix was revisited, and this corrected the inaccuracies that previously were being calculated. I had a server reservation regarding this initially due to the recalculation that was being done and was concerned about performance, but I was not able to devise an alternate solution to the problem. This process is called replicated computation in the parallel program design process regarding Agglomeration. This was able to eliminate the communication between threads because the process was contained inside each thread during computation. The way that the code is designed also allowed for the ability to simply create more tasks than threads for optimization for various hardware specifications. Overall, replicating the computations within each thread analysis showed that there was a negligible/minimal decline in performance with this approach. Due to the design that was chosen the tasks were evenly distributed among all threads and the number of tasks can easily scale with the problem size by simply modifying the portion that the problem size is divided.

The Mapping process of the Parallel program design process seeks to define where each task is to be executed. With assessing the number of cores that my host machine was able to utilize, 8 cores, 16 logical, I was able to optimize the workload and obtained the best results by dividing the partitions of work with this specific hardware. After tuning the code, I was able to get a speed up of roughly 3x. Although this was still requiring thread numbers of greater than 30 threads. Since the intercommunication of tasks was not frequent or was nonexistent during parallel execution the there was not much concern regarding this issue and thus there was no need for optimizing the locality between thread tasks.