Design Document – HW1

Brandon Callender

1/11/2016

What data structures will your Serial program use internally?

The serial implementation will represent the graph as an adjacency matrix, so we will need a matrix struct to store it in memory containing a 2D array (int **) to represent the vertices and their edge lengths.

What additional data structure will your Parallel program use internally? / How will work be divided up among threads? / What data (if any) must be communicated between threads? / What synchronization (if any) is needed to ensure correctness?

In the parallel implementation, each thread will take on the inner two loops (i and j) for a specific i. For example, with two threads, T1 can work on i=1 and T2 can work on i=2 in parallel, when one is done, it moves on to the next available i. This means we need some sort of shared counter that will keep track of the next i available for processing with a mutual exclusion guarantee making checking-and-incrementing the i value atomic. Once all the I's are completed for a given k, k will increment and i will reset to 0. We need to make sure that two threads aren't processing the same i at the same time.

We also need to prevent two threads from accessing the same element of the matrix at the same time. As each thread is limited to a specific i, all operations modifying the matrix for that thread will be performed on the ith column, so no lock is necessary – as long as we can prevent two threads from working on the same i value.

• What invariants must be maintained to ensure correctness? You should think about what invariants are required for the Serial implementation to be correct and what (if any) additional invariants are required for the Parallel implementation.

The primary invariant that makes the algorithm correct is that after the kth iteration of the triple-loop, for any element (i,j), the weight represented there is the minimum weight path from i->j using only the vertices 0-k. In order to maintain that loop invariant in the parallel version, all threads must be working on the same k value.

Hypothesis

Parallel Overhead

Parallelization won't have too much overhead associated with it (only synchronization of i and k and no lock on the primary matrix), so I estimate that the parallel version with 1 thread will perform at about 90% the speed of the serial version.

Parallel Speedup

Increased threading will have little effect until N is greater than T for any given T. At this point, there will be a big speed up (up to 2x) but then diminishing returns as the number of threads grows (1.5x, 1.25x …)

Testing Framework

Testing will come in two rounds. First, a reference implementation (in Python) will generate outputs for a series of inputs designed to stress the invariants of both the parallel and serial implementations. The serial and parallel implementations will then be tested on the same inputs (with the sizes and thread # requirements specified by the requirements document), with their outputs tested against the reference outputs for correctness and timed for performance. The entire process will be automated in Python (generating random graphs and calling the appropriate C program to test it. Some early bad path test cases include:

Input: Empty/Non-existent file.
Expected Output: Error – Nothing to process.

Input: Malformed Graph
Expected Output: Error – Unable to parse Input Graph.

Input: Graph with negative cycle
Expected Output: Error – Negative cycle exists, unable to calculate shortest path.

Input: Disconnected Subgraphs
Expected output: No optimized paths should be found between subgraphs.

Other test cases include:

Input: Sparse Graph with many improved paths.
Expected Output: Improved paths in output matrix as generated by reference implementation.

Input: Sparse Graph with many improved paths (with negative edges)
Expected Output: Improved paths in output matrix as generated by reference implementation.

Input: Sparse Graph with single improved path.
Expected Output: Improved path in output matrix.

Input: Sparse Graph with no improved paths.
Expected Output: Output matrix = input matrix.