Operating Systems - Assignment 1

Aryan Nath

April 1, 2025

1 Explanation for Multithreading:

I have a function for running the Floyd Warshall algorithm on a graph input represented by the dist matrix. In the non-threaded approach there are three loops, the loop with the k iterator, the loop with the i iterator, and the loop with the j iterator. For implementing this using multithreading, for each iteration of the i loop, I have created a new thread and passed the value of the row (i), k, and a pointer to the dist matrix as a struct to the function thread_rowcompute() which computes updates all of the column in the row i of the dist matrix. Then after the i loop has terminated, I joined the threads and repeated this in the next iteration of the k loop.

Pseudocode for Multithreading:
(Next page)

Running the code:

1. Add input to input.txt
2. Run make
3. Run ./main
4. Check output.txt

2 Runtime Comparison

Execution time for loop k=0: 0.000126 seconds
Execution time for loop k=1: 0.000095 seconds
Execution time for loop k=2: 0.000067 seconds
Execution time for loop k=3: 0.000089 seconds
Execution time(without threading): 0.000002 seconds

Despite using threading, we could not achieve a lower runtime than the standard sequential approach. This is probably because of the extra overhead caused by the struct creation, which is passed as an argument in to the function run by each thread, thread creation, and thread joining.

Algorithm 1 Multithreaded Floyd-Warshall Algorithm

Define $INF = 100000$
Structure *distrow*:
    row: integer
    distmatrix: pointer to integer array
    squaremat_width: integer
    intermediate_node: integer
function ThreadRowCompute(distmat_struct)
    $distmat \leftarrow distmat\_struct.distmatrix$
    $row \leftarrow distmat\_struct.row$
    $k \leftarrow distmat\_struct.intermediate\_node$
    $N \leftarrow distmat\_struct.squaremat\_width$
    for $j \leftarrow 0$ to $N - 1$ **do**
        if $distmat[row][k] + distmat[k][j] < distmat[row][j]$ **then**
            $distmat[row][j] \leftarrow distmat[row][k] + distmat[k][j]$
        end if
    end for
    Free *distmat_struct*
    return NULL
end function
function FloydWarshall(N, $dist[N][N]$)
    for $k \leftarrow 0$ to $N - 1$ **do**
        $tid \leftarrow$ Allocate array of N thread IDs
        $start\_time \leftarrow$ Current time
        for $i \leftarrow 0$ to $N - 1$ **do**
            $distmat\_struct \leftarrow$ Allocate new structure of type *distrow*
            $distmat\_struct.distmatrix \leftarrow dist$
            $distmat\_struct.row \leftarrow i$
            $distmat\_struct.intermediate\_node \leftarrow k$
            $distmat\_struct.squaremat\_width \leftarrow N$
            Create thread $tid[i]$ running ThreadRowCompute($distmat\_struct$)
        end for
        $end\_time \leftarrow$ Current time
        $execution\_time \leftarrow (end\_time - start\_time)/CLOCKS\_PER\_SEC$
        Output "Execution time for loop $k = k$: $execution\_time$ seconds"
        for $i \leftarrow 0$ to $N - 1$ **do**
            Wait for thread $tid[i]$ to complete
        end for
        Free *tid*
    end for
end function

3 Achieved Output

0 1 2 1
1 0 1 2
2 1 0 1
1 2 1 0

(written to output.txt)

4 Theoretical Time Complexity

Since the threads in the 2nd loop are running in parallel, the theoretical time complexity of the the program should be $O(n^2)$ instead of $O(n^3)$. However, this definitely does not execute as two nested for loops due to various possible delays from process scheduling and the overhead from thread creation and joining.