Oscar Galindo

Lab 1

Dr. Shirley Moore

Lab 1

1. Graphs

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Graphical user interface, application, email

Description automatically generated

The results presented here may be divided in two categories. The first category is the results of the graphs that are “balanced” and the second category is the results of the graphs that are “unbalanced”. In the balanced category there is a single graph which interestingly requires a significantly higher amount to time to complete as marked in all graphs with a blue line. Possibly, this “balanced” example takes a significantly higher amount of time due to the impossibility to discard a number of nodes by just finding local minimum in the execution. This situation might cause the program to have to evaluate longer the paths of during execution to be able to discard such options. Hence, the reason it takes longer. With regards to execution time, it is interesting to observe how the graph of the time taken between the static and dynamic implementations remains almost the same (I consulted this with Dr. Moore, and she suggested that the graph was balanced and hence this happened). And finally, it was also interesting to see how the magnitude does not decrease as expected in both the static and dynamic implementations, as seen, the time taken with a single thread is only four times as big as the time taken with 8 threads.

In the case of the “unbalanced” or, maybe better put, more predictable global minimum graphs, it can be observed those graphs lie on the same neighborhood of time for each of their executions, except in the case of the mat\_16\_2 example which actually jumps in time taken from executing with a single thread to executing with 2 threads. It is also interesting to see that in general there is a pattern in which regardless of the graph there seems to be an increase in the amount of time taken between the first iterative method execution and the second iterative method execution. In addition, it is interesting to observe how the recursive implementation provided in this lab does as well, or better than the second iterative version and the other algorithms (compare the execution times for mat\_16\_5 for a glance into this kind of results). This, at least to me, seems to be counterintuitive because of the overhead time-cost incurred by opening activation records as the code executes. With regards to the performance with multiple threads, it was observed that the magnitudes are closer to the expected savings of time around 6 times less between when a single thread executes the code and when 8 execute the code.

1. Execution of br-17(modified as needed to actually run) in static:

Text

Description automatically generated

Execution of br-17(modified as needed to actually run) in dynamic:

Text

Description automatically generated

Given that if we add a new node, we would a problem 18 times as big as the br-17, then we can see that if were to “evenly” divide the work then every core would have 2.25times more work, hence if we were to multiply the time take by the current times we would see that the static implementation would take 454.29 seconds \* 4 ~ 1022seconds ~17 minutes, which is in this case non-feasible. On the other hand the dynamic implementation could vary quite significantly, taking the relationship linearly as previously stated would output a time of 317.76 seconds \* 2.25 distributed work factor ~ 714.96 seconds ~ 12 minutes, close, quite possibly in some cases we may not see a termination and in some others we may see a termination.

Experiment with br-17 (modified to have 18 nodes):

Dynamic execution with partition set to 5:

Yellow text on a green background

Description automatically generated with medium confidence

Static execution:

Text

Description automatically generated

1. For this problem the control of the reading and writing of the best tours is controlled by a read/write lock instead of using the mutex that was originally included in the code. The modifications done are:

To control the access in the reading sense:

Text

Description automatically generated

To control the access to the best tour in the writing sense:

Text

Description automatically generated

Output times

1 thread:

Text

Description automatically generated

2 threads:

Text

Description automatically generated

4 threads:

Text

Description automatically generated

8 threads:

Text

Description automatically generated

Conclusion: The performance is positively affected by the inclusion of a read/write lock. Theoretically, this is possible due to the less overhead of the lock mechanism operations (lock/unlock) than the mutex lock and unlock procedures. Also, the fact that now no threads will get outdated data could help to finish the execution faster.