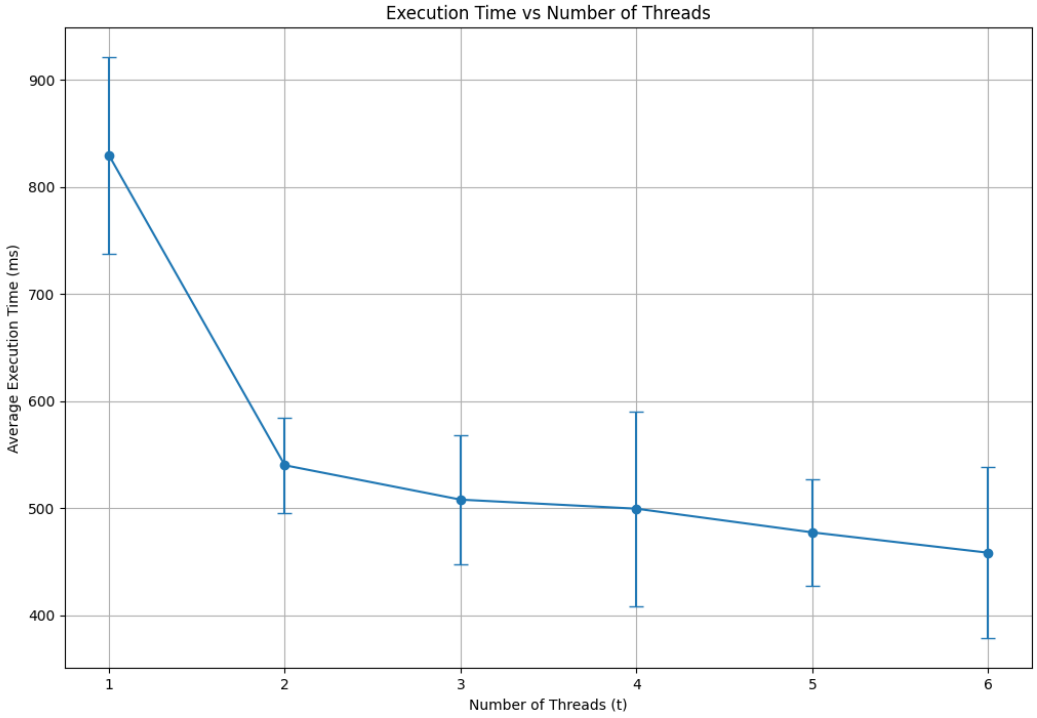
**Samantha Handal**
**260983914**

# Performance Analysis of Multi-threaded Icon Placement




The solution developed for question 2 exploits multi-threading to speed up the process of placing non-overlapping icons onto an image. The synchronization strategy chosen is based on lock striping, which segments the output image into vertical stripes, each protected by its own lock, with the goal to reduce lock contention and increase parallel execution. The original strategy tested involved locking the entire output image, which showed little to no speedup as only one thread could read/write to output image at once, and removing the computationally expensive tasks from the synchronized area could lead to overlapping images.

This lock striping approach seemed like a good solution, instead of having a single lock for the entire shared resource, which can become a bottleneck, the resource is divided into several independent segments, each protected by its own lock. The number of stripes corresponds to the thread count, but if an icon's width is larger than a stripe, the number of stripes is recalculated to fit the icon width into the image width. Threads determine their working segment using the icon's x-coordinate, ensuring that icon placements do not conflict. This means that multiple threads can operate concurrently as long as they are working in different stripes.

As for the performance graph, the average of 10 executions for each thread possibility (1-6), was tested on a 2.3 GHz Quad-Core Intel Core i5 MacBook Pro. The default arguments were used, other than changing the thread count. The program shows a notable decrease in average execution time when moving from a single thread to two threads, suggesting that the program is benefitting from parallel execution across multiple cores. As the number of threads increases beyond two, the speedup starts to plateau, and the variation in execution time becomes more noticeable (indicated by error bars). These results can be due to several factors:

- **Core Utilization**, the MacBook Pro's quad-core processor provides an advantage up to four concurrent threads. Beyond this, the benefits of additional threads reduce due to the overhead of context switching and the limited number of physical cores available for true parallel execution.

- **Lock Contention**, although lock striping was designed to minimize contention, the observed plateau and variability might suggest that threads may still be contending for locks, possibly when attempting to place icons near the boundaries of stripes.

- **Algorithmic Efficiency**, the lock striping algorithm's effectiveness is also dependent on the distribution of icon placement attempts. The randomness of icon placements could lead to scenarios where threads are more likely to compete for the same stripes, increasing the time spent waiting for locks. If attempts are unevenly distributed across stripes, some threads may be idle while others are overburdened, leading to suboptimal utilization of available cores.