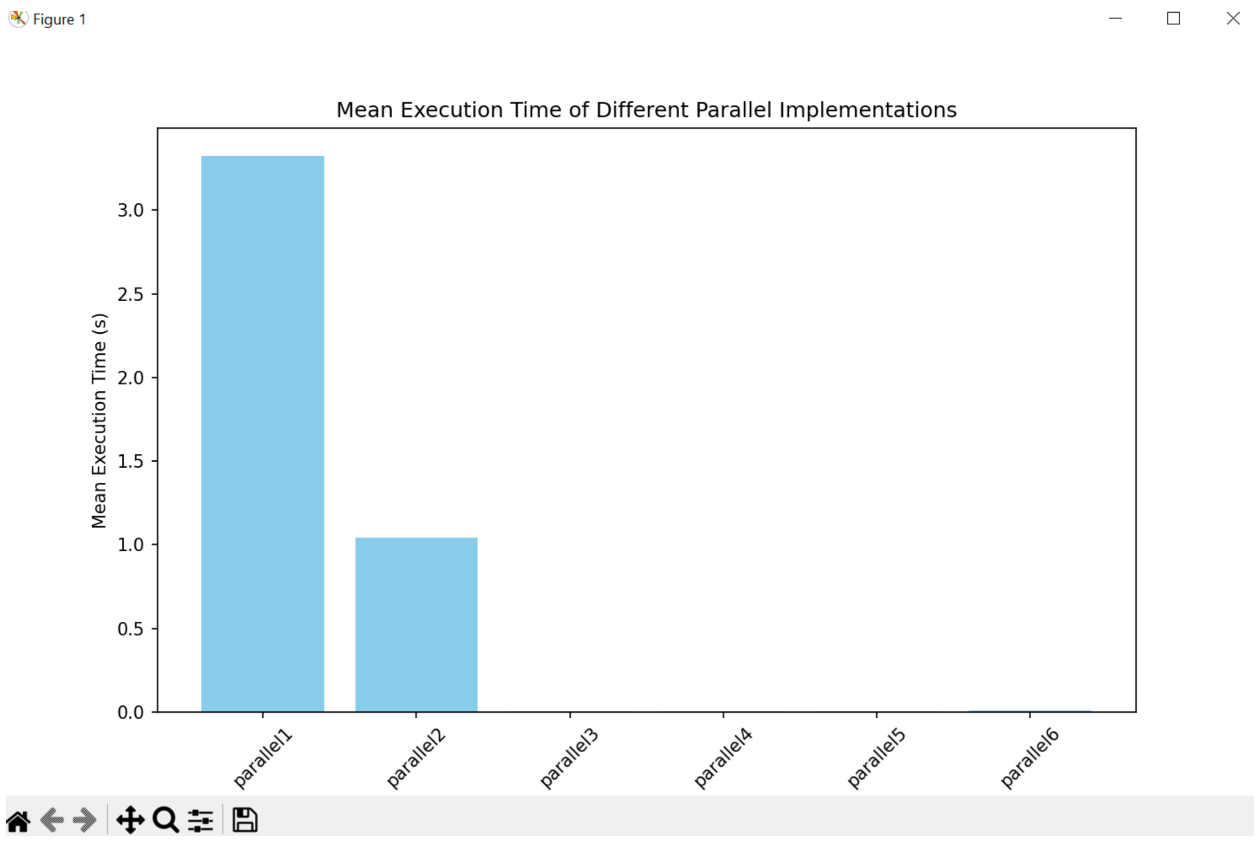

From the data, it is clear that using mutexes significantly affects the performance of the program. The most performant implementations are those that either minimize the use of mutexes or avoid them entirely.

Observations:

- parallel1 has the highest execution time, likely due to the high contention on a single global mutex.
- parallel2 performs better than parallel1 by using an array of mutexes, reducing contention but still incurring synchronization overhead.
- parallel3 and parallel4 show the best performance, as they use local counters and only synchronize at the final update. This minimizes the time spent holding locks.
- parallel5 and parallel6 also perform well but slightly worse than parallel3 and parallel4. This is due to the overhead of managing separate arrays for each thread or digit without the need for synchronization.

Role of Mutex Locks

- Mutex locks prevent race conditions and ensure data integrity in multithreaded programming but add overhead.
- High Contention: Using a single global mutex (as in parallel1) results in high contention and increased execution time.
- Reduced Contention: Using multiple mutexes (as in parallel2) reduces but does not eliminate contention.

- Minimal Synchronization: Local counters with final synchronization (as in parallel3 and parallel4) provide the best performance by minimizing the use of mutexes.
- No Mutexes: Avoiding mutexes (as in parallel5 and parallel6) can be efficient but depends on implementation details.

Conditions for Using Mutexes

- Shared Resources: Use mutexes to safely update shared resources.
- High Contention: Use multiple mutexes to reduce contention.
- Minimal Synchronization: Minimize mutex use by performing most operations on local data and synchronizing only when necessary.
- Independent Operations: Design programs to allow threads to work independently, avoiding the need for mutexes when possible.

The parallel versions show significant improvements in execution time compared to the serial version. As the number of threads increases, the execution time decreases, demonstrating the benefits of parallel processing.

Optimal Thread Count: The optimal thread count is the one that provides the best balance between performance improvement and resource usage. Based on the measured execution times:

- 4 Threads: Achieved the lowest execution time (2.0 seconds), indicating the best performance.
- 2 Threads: Also provided significant improvement (3.0 seconds) with potentially less resource contention compared to 4 threads.