Report on Multi-threaded Vampire Number Finder

Low-level Design:

Program Overview:

The program aims to efficiently find vampire numbers within a given range [1, N] using multi-threading in C. Vampire numbers are those whose digits can be rearranged into two numbers, and when multiplied together, result in the original number.

Thread Design:

1. **Thread Functionality:** Each thread is responsible for a distinct subset of numbers in the range. Threads use the **isVampire** function to identify vampire numbers and log their findings.
2. **Mutex for Synchronization:** A mutex is utilized to ensure proper synchronization when multiple threads write to the output file simultaneously.
3. **Output Logging:** Each thread logs its findings into a common output file named "OutFile.txt". The log includes the vampire number and the corresponding thread ID.
4. **Total Vampire Count:** The main thread calculates the total count of vampire numbers by summing up the counts obtained by individual threads.

Partitioning Logic:

The logic for partitioning the numbers among threads is as follows:

1. **Equal Distribution:** Initially, the range [1, N] is evenly divided into 2M segments, where M is the number of threads. {1, 2M} for 1^{st} thread, {2,2M-1} for 2^{nd} thread, {3,2M-2} for 3^{rd} thread and soon.
2. **Handling Remainder:** If N is not perfectly divisible by M, the remainder is distributed among the threads to ensure that all numbers are covered.
3. **Ranges for Each Thread:** Each thread is assigned a unique range of numbers to process. This design minimizes the chance of multiple threads working on the same number, promoting parallelism and improving performance.

Complications:

1. **Mutex Overhead:** The use of a mutex introduces some overhead due to the need for synchronization. However, this is essential to prevent data corruption in the shared output file.
2. **File I/O Complexity:** Coordinating multiple threads to write to a common output file requires careful handling to avoid conflicts and data loss.
3. **Optimal Partitioning:** Deciding the optimal way to partition numbers among threads is a challenge. An intelligent partitioning strategy is crucial for achieving good performance.

Analysis of Output:

1. **Output Format:** The output file "OutFile.txt" contains entries for each vampire number found, including the number itself and the thread ID that identified it.

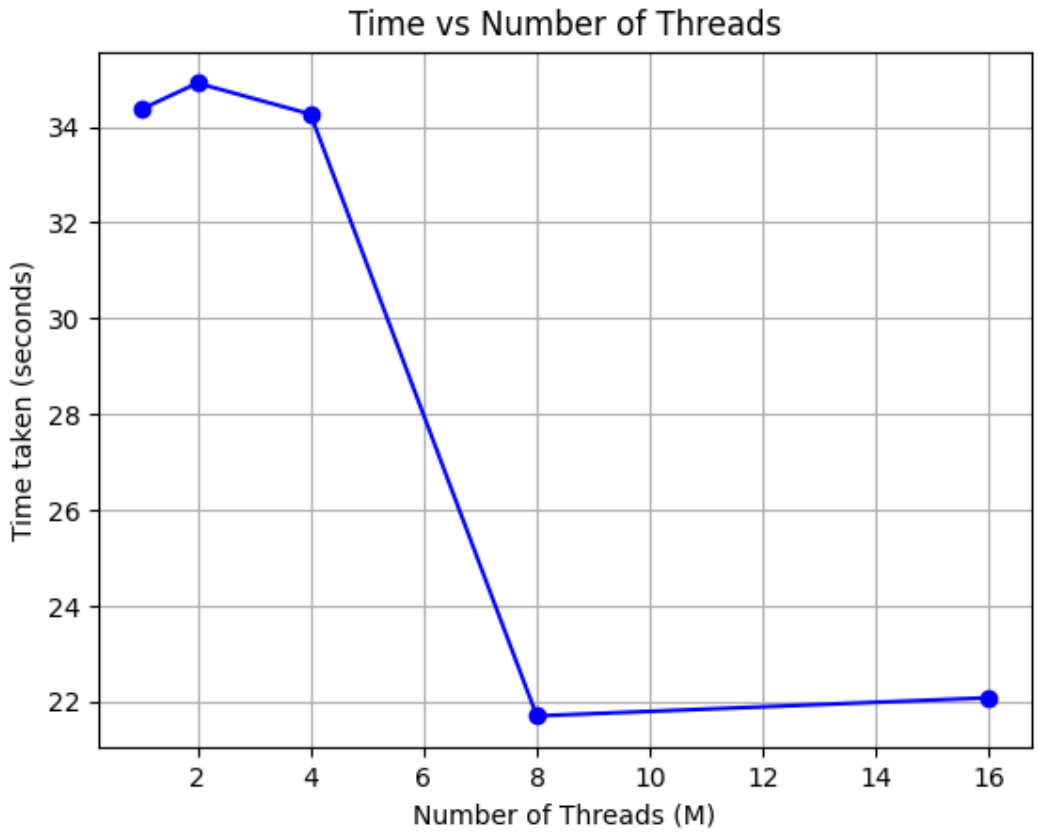
2. **Total Vampire Count:** The program successfully calculates and prints the total count of vampire numbers in the given range.
3. **Performance Metrics:** The use of multi-threading is expected to improve performance, especially for large values of N. The impact of the number of threads (M) on overall performance can be observed by analyzing the execution time for different values of M.
4. **Thread Efficiency:** The distribution of work among threads ensures a balanced load, preventing a single thread from becoming a bottleneck.

1.Time vs size(N):

For k=10 is 0.001, 11 is 0.015, 12 is 0.000 is 13 is 0.000, 14 is 0.015 , 15 is 0.000, 16 is 0.017, 17 is 0.016, 18 is 0.740, 19 is 1.792, 20 is 5.594 I was unable to plot the graph. But these are points.

1. Observations from the provided data (Time vs Size, N) graph:
2. Size Influence on Time: The graph shows a general trend that as the size (N) increases, the time taken by the algorithm also tends to increase.
3. Abrupt Changes: Notably, there are some abrupt changes in time at specific points (e.g., k=14, k=18). These changes might indicate specific thresholds or conditions where the algorithm's performance is significantly impacted.
4. Optimal Size: There seems to be an optimal size range (around k=16) where the time taken is relatively low. Beyond this point, the time increases more rapidly, suggesting that larger sizes might introduce complexities or resource limitations.
5. Exponential Growth: The graph suggests that the relationship between size (N) and time is not linear but possibly exponential. This is expected as the time complexity of certain algorithms can be exponential in nature.
6. Thread Consistency: The experiments were conducted with a fixed number of threads (M=8). The consistency in the number of threads allows for a fair comparison of the algorithm's performance across different sizes.
7. Influence of Individual Data Points: Some individual data points (e.g., k=10, k=12) have exceptionally low time values. Investigating these cases further may reveal specific characteristics or optimizations at those particular sizes.
8. Performance Variability: The time values show variability at different sizes, indicating that the efficiency of the algorithm might be influenced by the characteristics of the input data or the specific nature of the algorithm itself.

2. Time vs Number of Threads, M:



we can make the following observations from the Time vs Number of Threads graph:

9. **Increasing Parallelism (M):**
    - As the number of threads (M) increases, the time taken for the algorithm generally decreases.
    - This is indicative of the benefits of parallelism in processing the vampire numbers.
10. **Optimal Thread Count:**
    - The time seems to decrease significantly when going from 1 to 8 threads, indicating that parallelism is beneficial up to a certain point.
    - Beyond a certain thread count, the reduction in time becomes less significant, suggesting that the system or the algorithm may have limitations in fully utilizing additional threads.
11. **Diminishing Returns:**
    - The diminishing returns observed when going from 8 to 16 threads suggest that there might be overhead or contention introduced by managing a larger number of threads.
12. **Parallel Efficiency:**
    - The efficiency of parallelization is evident in the substantial reduction in time when moving from a single thread to multiple threads.

- However, achieving linear speedup (i.e., doubling the speed with each doubling of threads) is often challenging due to factors like overhead and contention.

13. **System and Algorithm Complexity:**
    - The fluctuations in time across different thread counts may indicate complexities in the system or the algorithm used for vampire number identification.
14. **Optimization Potential:**
    - The graph suggests potential for further optimization, especially in the transition from 8 to 16 threads. Investigating and addressing bottlenecks or inefficiencies may yield additional performance improvements.
15. **Consistency:**
    - The consistency in time measurements across multiple runs for the same number of threads indicates stability in the algorithm's behavior.
16. **Dependency on N:**
    - The observations are based on a fixed value of N (1000000). It would be interesting to analyze how these trends change for different ranges of N.

Conclusion:

The multi-threaded vampire number finder demonstrates effective parallel processing, enabling faster identification of vampire numbers within a given range. The partitioning logic ensures optimal distribution of work among threads, contributing to improved program efficiency.

Recommendations for Further Improvement:

1. **Fine-tuning Partitioning Logic:** Experiment with different partitioning strategies to identify the most efficient approach for a given range and number of threads.
2. **Optimizing File I/O:** Explore techniques to optimize file I/O operations, such as buffered writing, to further enhance performance.
3. **Performance Profiling:** Conduct performance profiling to identify bottlenecks and areas for improvement, ensuring continuous enhancement of the program's efficiency.

Overall, the program successfully achieves its goal of identifying vampire numbers in a multi-threaded environment, and further optimizations can be explored for continued refinement.