OS-1
ASSIGNMENT REPORT
(VAMPIRE NUMBERS)

NAME : RUVVA SURAJ KUMAR
ROLL NO : AI22BTECH11022

1. Program Design:

The program is designed to identify vampire numbers within a given range using a multi-threaded approach. Below are key design elements:

- The program reads input parameters (num1 and num2) from a file named "input.txt".
- It dynamically allocates memory for an array 'input' containing numbers from 1 to num1.
- Later the program creates num2 threads, each responsible for checking a subset of numbers (1 to num1) for vampire numbers.
- The 'is_vampire' function checks thread subset numbers for vampire numbers using ‘vamoire_num_check’.
  The 'vampier_num_check' function checks if a number is a vampire number or not.

2. Complications Faced:

- Memory Allocation: The program dynamically allocates memory for arrays, and it checks for allocation failures.

3. Logic of Partitioning N Numbers Among M Threads:

- The program divides the numbers from 1 to num1 among num2 threads into subsets using nums_per_thread.
- If num1 is divisible evenly by num2, each thread is assigned 'num1/num2' numbers.
- If not, the first 'num1%num2' threads are assigned 'num1/num2 + 1' numbers, and the remaining threads are assigned by order from 1 to num2 threads.

4. Output Analysis:

- The program prints each vampire number found along with the corresponding thread number.
- The total count of vampire numbers is shown at the end of execution.
- The execution time is measured and the elapsed time is printed.

Time vs size analysis, N:

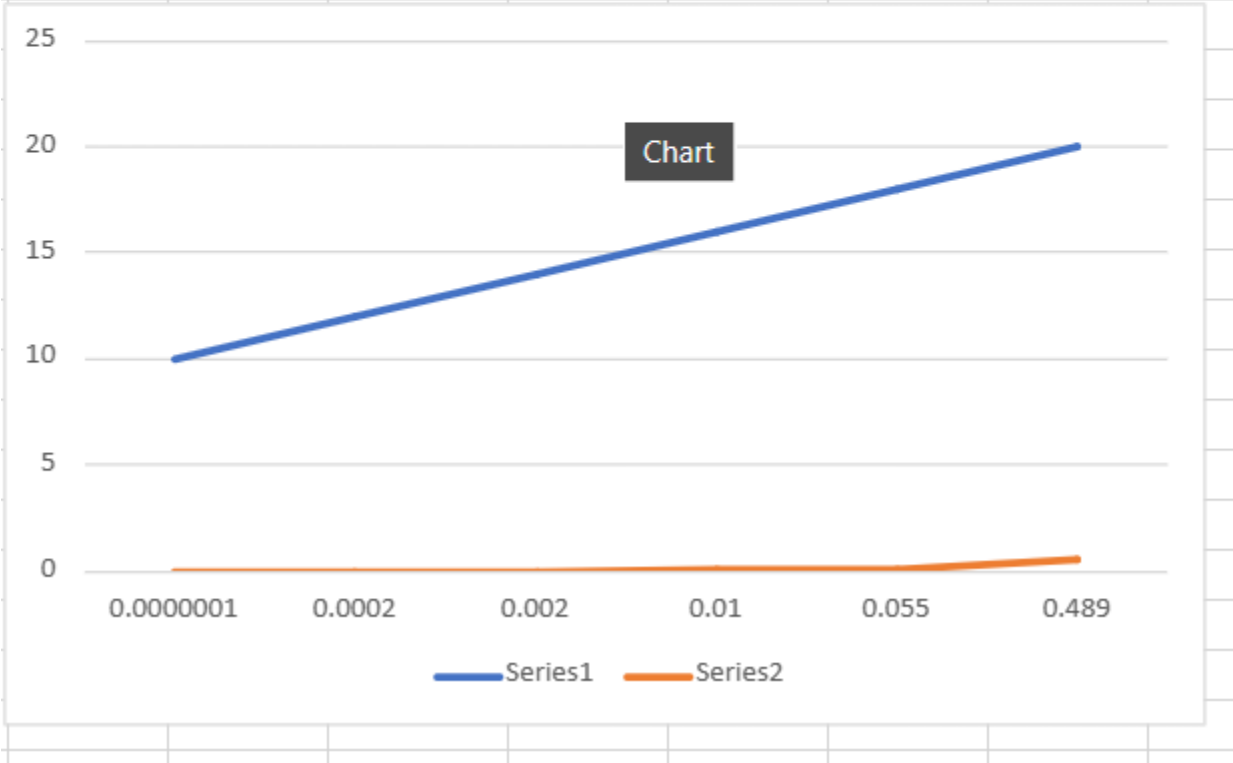
From the Graph :
Y-axis: Values of k varying from 10 to 20 in increments of 2 (N=2^k).
X-axis: Time taken by the algorithms. Number of threads (M) fixed at 8 for all experiments.

Observations:

The graphical representation illustrates how the execution time of the program is influenced by varying input sizes. This analysis shows assessing the scalability and efficiency of the multi-threaded approach across diverse inputs. The program adeptly employs multi-threading to identify vampire numbers within a specified range. In the context of Multi-Threaded systems, each thread is typically assigned a subset of

values from 1 to n/k, with each thread executing n/k operations.

SIZE(N)	N	TIME
1024	10	0.0000001
4096	12	0.0002
16384	14	0.002
65536	16	0.01
262144	18	0.055
1048576	20	0.489



Time vs Number of Threads:

Graph Analysis:
Y-axis: Values of M, the number of threads, varying from 1 to 16
X-axis: Time taken by the algorithm.
Observation :

As the thread count rises, a discernible reduction in execution time is observed, reaching its lowest point at 16 threads. Best performance is attained with 16 threads, showing efficient workload management among them. The correlation suggests that as the number of threads increases, the parallelization and workload distribution become more effective. Consequently, the compiler operates optimally, accomplishing tasks more efficiently in less time with multi-threaded systems, thereby achieving enhanced parallelism.

TIME VS NUMBERS OF THREADS

M	TIME
1	1.356
2	0.905
4	0.568
8	0.47
16	0.379