Assignment -2

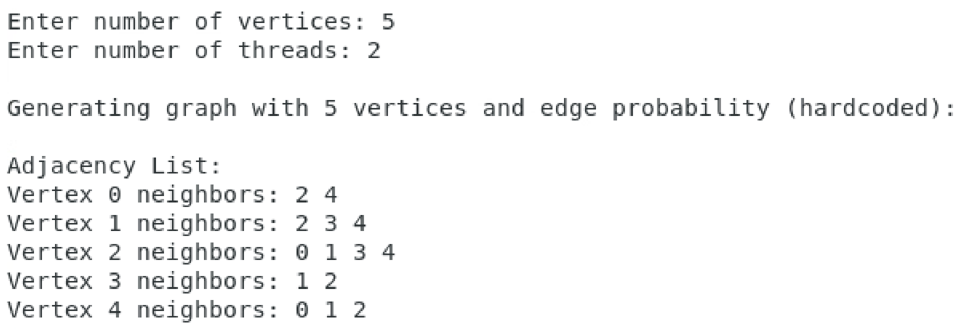
Question – 3

Graph coloring involves coloring a graph G (V, E) such that no two adjacent vertices have the same color. This algorithm is used in several important applications, including register allocation in compilers, sparse matrix ordering and VLSI routing. The goal is to use the least number of colors to color the graph. An exact solution to graph coloring is NP-hard. You don't need to obtain the optimal solution (i.e., you can use a greedy approach). Make sure to test your solution with different graphs that you will generate. Describe how you tested your implementation.

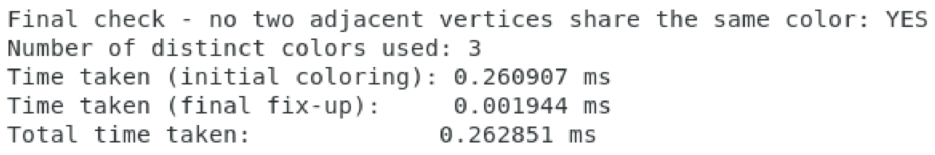
a) There are a few approaches you can take to solve this problem. Develop your solution using OpenMP.
b) Evaluate the strong scaling and weak scaling performance of your implementation. Compare the runtime taken to the algorithmic complexity (big-O) of the algorithm you have chosen for your implementation.

Answer:

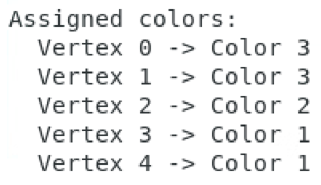
In case-1, I am going to choose the number of vertices as 5 and threads as 2. I am going to print the adjacency list just to display the neighbors of the particular vertex. This is to show the user that the color of the adjacent matrices aren't the same by seeing the adjacency list.



```
Enter number of vertices: 5
Enter number of threads: 2

Generating graph with 5 vertices and edge probability (hardcoded):

Adjacency List:
Vertex 0 neighbors: 2 4
Vertex 1 neighbors: 2 3 4
Vertex 2 neighbors: 0 1 3 4
Vertex 3 neighbors: 1 2
Vertex 4 neighbors: 0 1 2
```

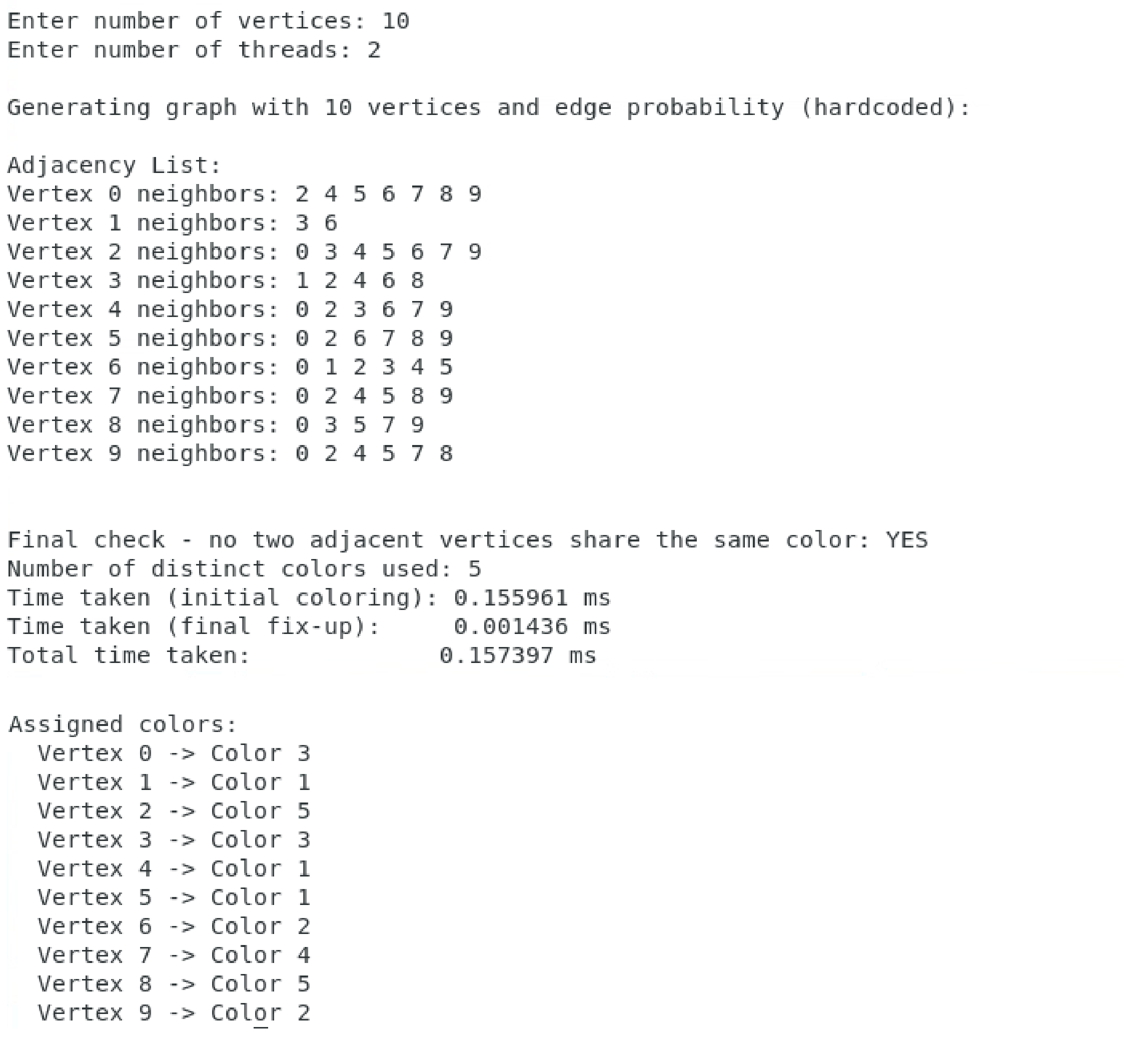


```
Final check - no two adjacent vertices share the same color: YES
Number of distinct colors used: 3
Time taken (initial coloring): 0.260907 ms
Time taken (final fix-up):     0.001944 ms
Total time taken:             0.262851 ms
```



```
Assigned colors:
  Vertex 0 -> Color 3
  Vertex 1 -> Color 3
  Vertex 2 -> Color 2
  Vertex 3 -> Color 1
  Vertex 4 -> Color 1
```

In case-2, I take the number of vertices as 10 and threads as 2. The output is:

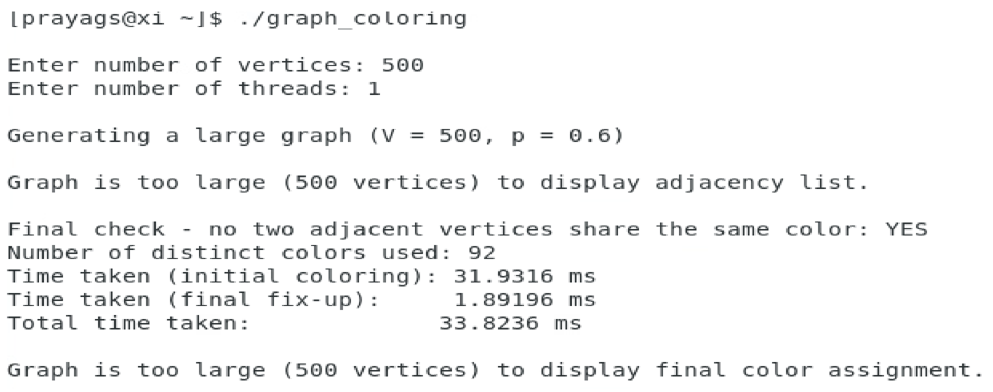


```
Enter number of vertices: 10
Enter number of threads: 2

Generating graph with 10 vertices and edge probability (hardcoded):

Adjacency List:
Vertex 0 neighbors: 2 4 5 6 7 8 9
Vertex 1 neighbors: 3 6
Vertex 2 neighbors: 0 3 4 5 6 7 9
Vertex 3 neighbors: 1 2 4 6 8
Vertex 4 neighbors: 0 2 3 6 7 9
Vertex 5 neighbors: 0 2 6 7 8 9
Vertex 6 neighbors: 0 1 2 3 4 5
Vertex 7 neighbors: 0 2 4 5 8 9
Vertex 8 neighbors: 0 3 5 7 9
Vertex 9 neighbors: 0 2 4 5 7 8


Final check - no two adjacent vertices share the same color: YES
Number of distinct colors used: 5
Time taken (initial coloring): 0.155961 ms
Time taken (final fix-up):     0.001436 ms
Total time taken:             0.157397 ms

Assigned colors:
  Vertex 0 -> Color 3
  Vertex 1 -> Color 1
  Vertex 2 -> Color 5
  Vertex 3 -> Color 3
  Vertex 4 -> Color 1
  Vertex 5 -> Color 1
  Vertex 6 -> Color 2
  Vertex 7 -> Color 4
  Vertex 8 -> Color 5
  Vertex 9 -> Color 2
```

2 (b):

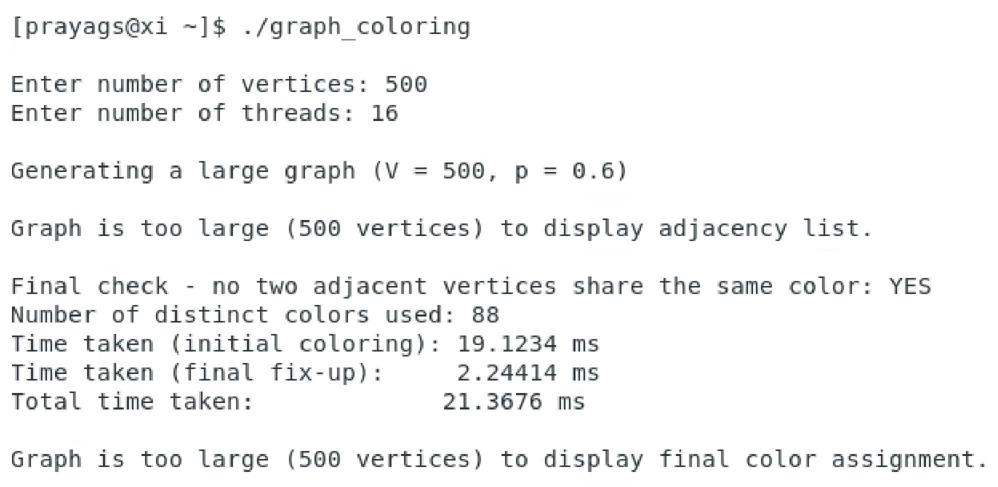
Strong scaling efficiency is calculated as

Efficiency= [Execution Time at 1 Thread × 100/ (Execution Time at N Threads × N)]



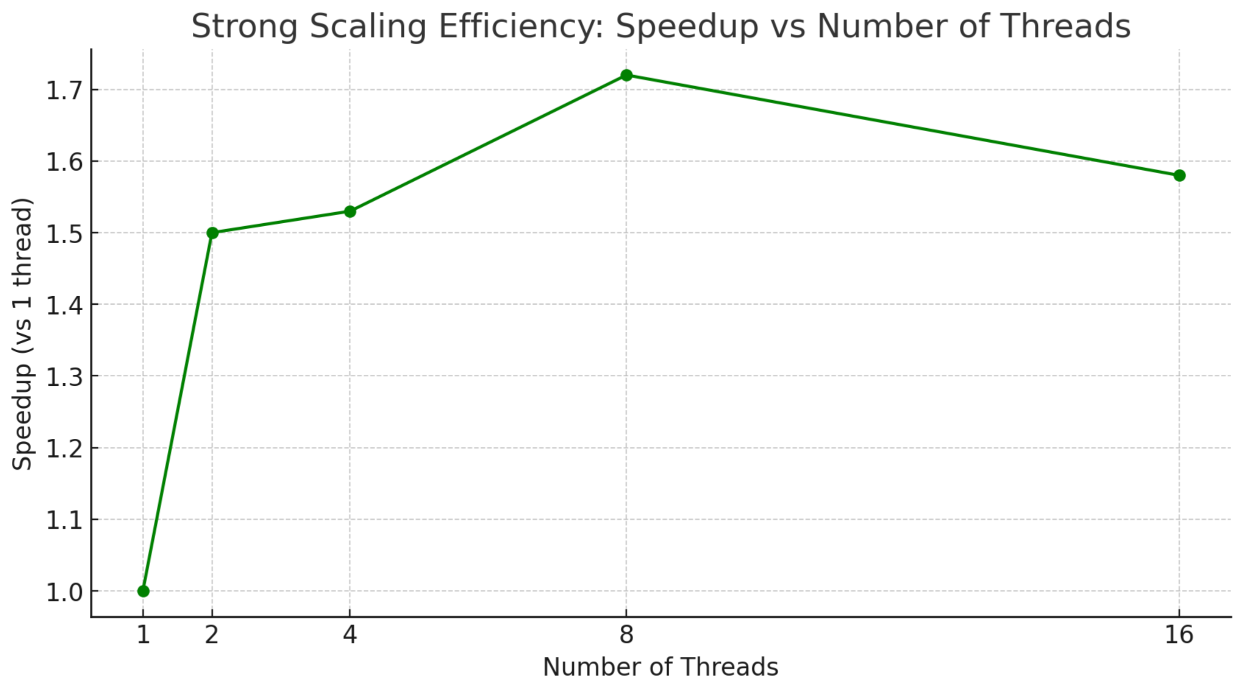
```
[prayags@xi ~]$ ./graph_coloring

Enter number of vertices: 500
Enter number of threads: 1

Generating a large graph (V = 500, p = 0.6)

Graph is too large (500 vertices) to display adjacency list.

Final check - no two adjacent vertices share the same color: YES
Number of distinct colors used: 92
Time taken (initial coloring): 31.9316 ms
Time taken (final fix-up):     1.89196 ms
Total time taken:             33.8236 ms

Graph is too large (500 vertices) to display final color assignment.
```

```
Enter number of vertices: 500
Enter number of threads: 2

Generating a large graph (V = 500, p = 0.6)

Graph is too large (500 vertices) to display adjacency list.

Final check - no two adjacent vertices share the same color: YES
Number of distinct colors used: 90
Time taken (initial coloring): 20.8071 ms
Time taken (final fix-up):     1.81418 ms
Total time taken:             22.6213 ms

Graph is too large (500 vertices) to display final color assignment.


[prayags@xi ~]$ ./graph_coloring

Enter number of vertices: 500
Enter number of threads: 4

Generating a large graph (V = 500, p = 0.6)

Graph is too large (500 vertices) to display adjacency list.

Final check - no two adjacent vertices share the same color: YES
Number of distinct colors used: 88
Time taken (initial coloring): 19.9112 ms
Time taken (final fix-up):     2.14161 ms
Total time taken:             22.0528 ms

Graph is too large (500 vertices) to display final color assignment.


[prayags@xi ~]$ ./graph_coloring

Enter number of vertices: 500
Enter number of threads: 8

Generating a large graph (V = 500, p = 0.6)

Graph is too large (500 vertices) to display adjacency list.

Final check - no two adjacent vertices share the same color: YES
Number of distinct colors used: 88
Time taken (initial coloring): 17.5147 ms
Time taken (final fix-up):     2.2777 ms
Total time taken:             19.7924 ms

Graph is too large (500 vertices) to display final color assignment.
```



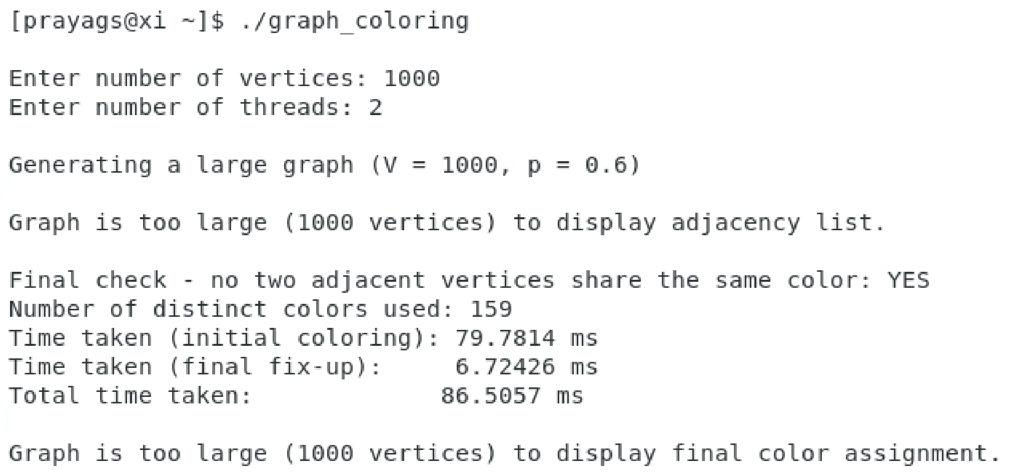
```
[prayags@xi ~]$ ./graph_coloring

Enter number of vertices: 500
Enter number of threads: 16

Generating a large graph (V = 500, p = 0.6)

Graph is too large (500 vertices) to display adjacency list.

Final check - no two adjacent vertices share the same color: YES
Number of distinct colors used: 88
Time taken (initial coloring): 19.1234 ms
Time taken (final fix-up):     2.24414 ms
Total time taken:             21.3676 ms

Graph is too large (500 vertices) to display final color assignment.
```

Threads	**Vertex Size**	**Execution Time (s)**	**Speedup (vs 1 thread)**	**Efficiency**
1	500	0.0338	1.0x	100%
2	500	0.0226	1.5x	74.78%
4	500	0.02205	1.53x	38.32%
8	500	0.0197	1.72x	21.44%
16	500	0.02136	1.58x	9.89%

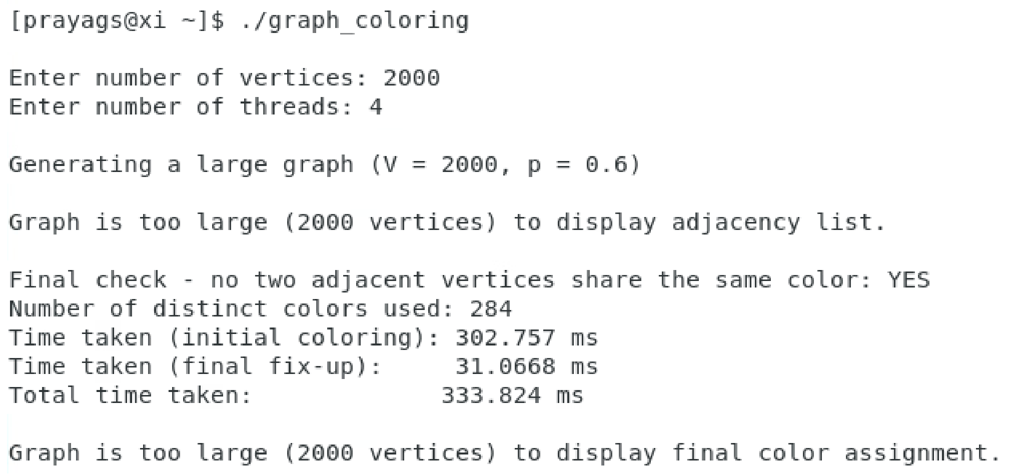


Weak scaling efficiency is calculated as:

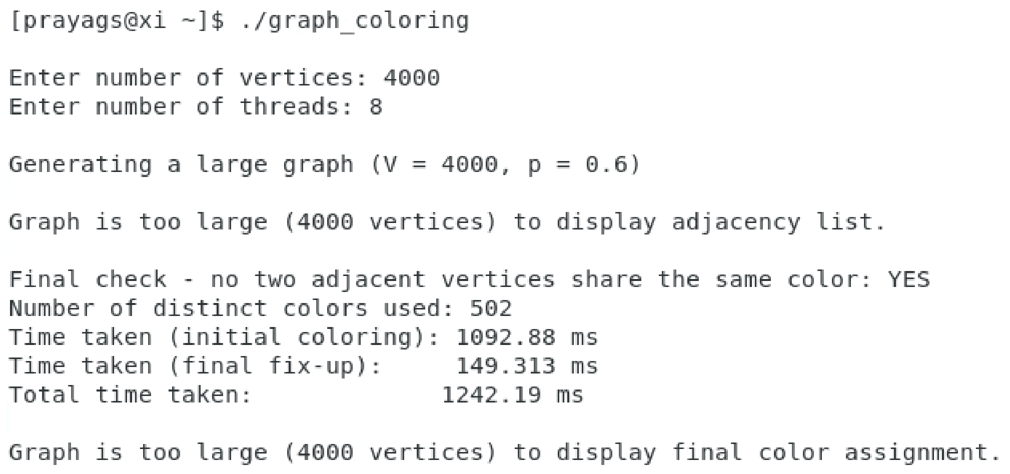
Efficiency= (Execution Time at 1 Thread/Execution Time at N Threads)×100



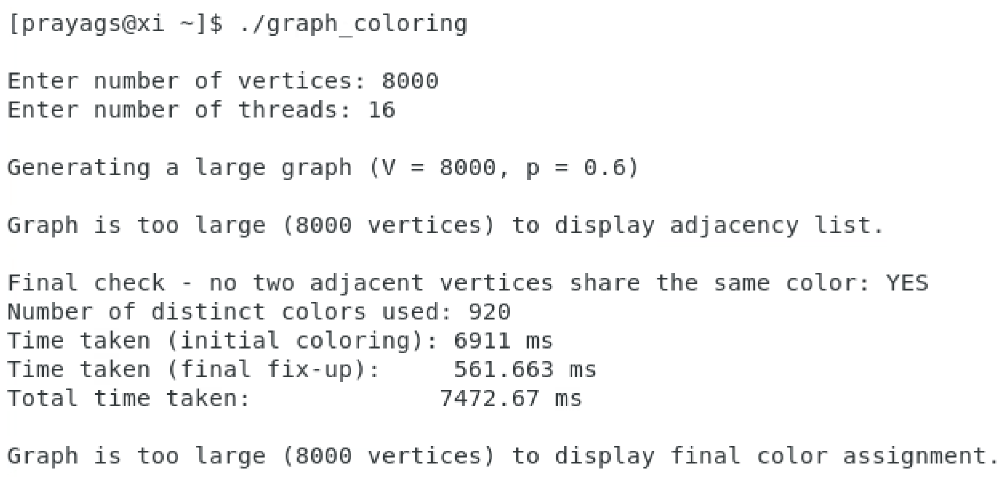
```
[prayags@xi ~]$ ./graph_coloring

Enter number of vertices: 1000
Enter number of threads: 2

Generating a large graph (V = 1000, p = 0.6)

Graph is too large (1000 vertices) to display adjacency list.

Final check - no two adjacent vertices share the same color: YES
Number of distinct colors used: 159
Time taken (initial coloring): 79.7814 ms
Time taken (final fix-up):     6.72426 ms
Total time taken:             86.5057 ms

Graph is too large (1000 vertices) to display final color assignment.
```



```
[prayags@xi ~]$ ./graph_coloring

Enter number of vertices: 2000
Enter number of threads: 4

Generating a large graph (V = 2000, p = 0.6)

Graph is too large (2000 vertices) to display adjacency list.

Final check - no two adjacent vertices share the same color: YES
Number of distinct colors used: 284
Time taken (initial coloring): 302.757 ms
Time taken (final fix-up):     31.0668 ms
Total time taken:             333.824 ms

Graph is too large (2000 vertices) to display final color assignment.
```

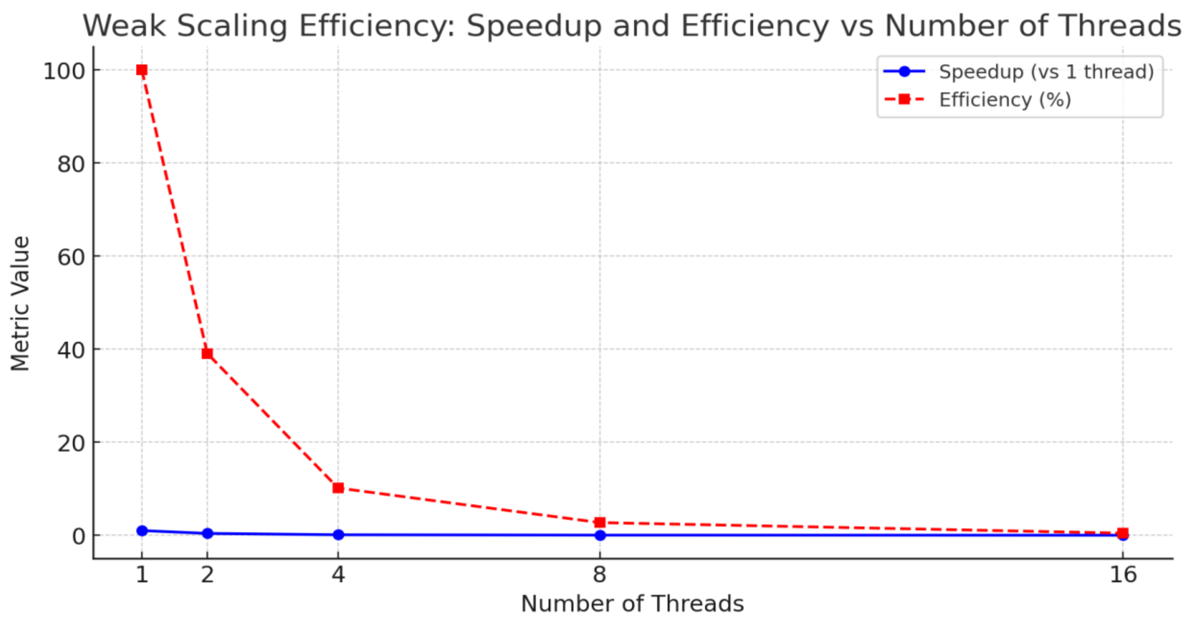


```
[prayags@xi ~]$ ./graph_coloring

Enter number of vertices: 4000
Enter number of threads: 8

Generating a large graph (V = 4000, p = 0.6)

Graph is too large (4000 vertices) to display adjacency list.

Final check - no two adjacent vertices share the same color: YES
Number of distinct colors used: 502
Time taken (initial coloring): 1092.88 ms
Time taken (final fix-up):     149.313 ms
Total time taken:             1242.19 ms

Graph is too large (4000 vertices) to display final color assignment.
```



```
[prayags@xi ~]$ ./graph_coloring

Enter number of vertices: 8000
Enter number of threads: 16

Generating a large graph (V = 8000, p = 0.6)

Graph is too large (8000 vertices) to display adjacency list.

Final check - no two adjacent vertices share the same color: YES
Number of distinct colors used: 920
Time taken (initial coloring): 6911 ms
Time taken (final fix-up):     561.663 ms
Total time taken:             7472.67 ms

Graph is too large (8000 vertices) to display final color assignment.
```

Threads	Vertex Size	Execution Time (s)	Speedup (vs 1 thread)	Efficiency
1	500	0.0338	1.0x	100%
2	1000	0.0865	0.391x	39.08%
4	2000	0.333	0.102x	10.15%
8	4000	1.242	0.027x	2.72%
16	8000	7.472	0.005x	0.45%


Weak Scaling Efficiency: Speedup and Efficiency vs Number of Threads
Speedup (vs 1 thread)
Efficiency (%)
Metric Value
Number of Threads