OBSERVATIONS AND ANALYSIS

1. GRAPH GENERATOR:

In order to devise the test cases for execution, we have created a program that generates graphs based on certain parameters that are input to it. The number of nodes in the graph must strictly be of the form:

$$v=[(st-2)(st-3)*n]+2$$

Here,
v=number of nodes in the graph to be generated.
st=number of stages
n=1,2,3,....
The constant '+2' is for the source and sink nodes.

We input these values to the graph generator **"gen.c"** and we get a random graph as output, that is generated based on input parameters specified.

2. COMPARISON OF EXECUTION TIMES:

Sr. No.	Number of Nodes in Graph (n)	Serial Execution Time (a) (seconds)	Parallel Execution Time(b) (seconds)	Speed-up (a/b)
1.	10	0.003	0.026	**0.011**
2.	18	0.003	0.025	**0.012**
3.	42	0.003	0.026	**0.011**
4.	82	0.008	0.042	**0.190**
5.	122	0.011	0.021	**0.524**
6.	162	0.039	0.047	**0.829**
7.	202	0.068	0.033	**2.060**
8.	402	0.543	0.056	**9.696**
9.	602	1.477	0.056	**26.375**
10.	802	3.191	0.073	**43.712**

3. ANALYSIS:

The parallelized version of "Excess Scaling Algorithm" gives very satisfactory results at higher values of n.

Reason for lower speed-up at lower values of n:
When n is small, the time spent over communication between the processes (MPI_Send and MPI_Recv) is high. This time is spent basically for synchronization between the master process and slave processes. The **time spent over communication is much higher than the time saved in computation,** hence the lower speed-up.

However, **as the value of n increases, the speedup increases exponentially.** The time saved by parallelizing computation over a number of processes is much higher than the time lag in synchronization. The parallel version of algorithm thus gives speed-ups in excess of 40 at values of n which are close to 1000. That is, 40 times faster execution than the corresponding time taken by executing the algorithm serially.