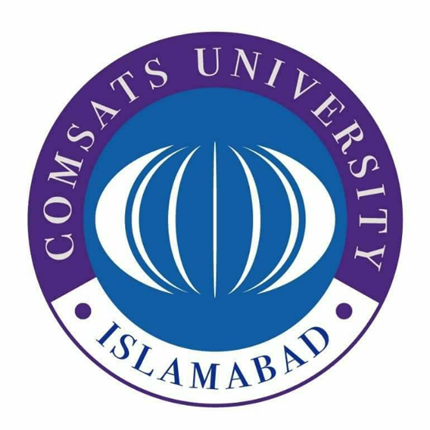
**COMSATS UNIVERSITY ISLAMABAD**

**LAHORE CAMPUS**

**ASSIGNMENT NO#5**

**NAME: ZOHA NISAR GILL**

**ROLL NUMBER: SP23-BCS-130**

**SECTION: C**

**Discussion Questions**

**1. What happens if the vector size N is not divisible by the number of processes?**

When the total number of elements N is not evenly divisible by the number of processes, some processes will have more data to handle than others, or a few elements may remain unassigned.  
If the program assumes equal-sized divisions, this imbalance can lead to incorrect results or wasted computational capacity.  
To ensure every element contributes to the total sum, the program must adjust how data is divided among processes.

**2. How can you modify the program to handle uneven partitions?**

To handle uneven partitions, the program can use **MPI\_Scatterv** instead of MPI\_Scatter.  
MPI\_Scatterv allows the programmer to:

* Define how many elements each process receives (counts[] array), and
* Where each process’s portion starts in the main vector (displs[] array).

By calculating these arrays based on the base division and the remainder, each process gets the correct amount of data, even when N is not divisible by the number of processes.

**3. How would performance differ between using MPI\_Reduce vs. MPI\_Gather + local summation?**

Both methods combine partial sums from multiple processes, but their performance differs:

* **MPI\_Reduce**
  + Combines partial results directly within MPI using an internal tree-like reduction.
  + Data is reduced step by step as it travels between processes.
  + Faster and more efficient because it minimizes data transfer and distributes the workload of combining results.
* **MPI\_Gather + local summation**
  + Each process sends its partial sum to the root process.
  + The root process adds them all sequentially.
  + Slower and less scalable since all data is sent to one process, increasing communication overhead and creating a bottleneck.

**In summary:**  
MPI\_Reduce is faster and preferred for large-scale parallel programs, while MPI\_Gather is simpler but less efficient for larger process counts.

**4. How could this same approach be extended to matrix summation or averaging?**

The same distributed approach can be applied to matrices by dividing the matrix by rows or columns among processes.  
Each process:

1. Computes the sum (or average) of its assigned rows or columns.
2. Sends the partial result to the root process using MPI\_Reduce or MPI\_Allreduce.
3. The final result is the total or average of all elements in the matrix.

This method improves performance for large matrices by parallelizing computation across multiple processes.