

# Comsats University Islamabad, Lahore campus

ASSIGNMENT#5(Lab)

**RAMSHA KHAN (SP23-BCS-112)**

**Section: C Course: PDC**

**Instructor’s name:Akhzar Nazir Due date: 16-10-25**

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

If the total number of elements N cannot be evenly divided by the number of MPI processes (P), then:

With MPI\_Scatter, which sends equal chunks to every process, some elements at the end of the array will be left undistributed (ignored) or may cause memory errors because the last process expects fewer elements.

This leads to incorrect results or even a runtime crash, since each process assumes it received exactly the same number of elements.

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

To fix this, we use **MPI\_Scatterv**, which allows variable-sized chunks per process.

**How it works:**

* Compute:
* long base = N / P;
* long rem = N % P;
* Each process gets base elements, and the first rem processes each get **one extra element**:
* sendcounts[i] = base + (i < rem ? 1 : 0);
* Compute **displacements** (starting indices):
* displs[i] = displs[i-1] + sendcounts[i-1];
* Then call:
* MPI\_Scatterv(A, sendcounts, displs, MPI\_DOUBLE,
* local\_A, sendcounts[rank], MPI\_DOUBLE,
* 0, MPI\_COMM\_WORLD);

**This ensures all elements are distributed correctly**, even when N is not divisible by P.

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

MPI\_Reduce is faster and more efficient because it adds up all partial sums while communicating between processes — only small values (partial sums) are sent.

MPI\_Gather first collects all data from every process to the root, and then the root does the summation.

This takes more time and memory, especially when there are many processes or large data.

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

You can extend the vector-sum idea directly to matrices:

1. Matrix summation

* Treat the matrix as a 2D array (or flatten it into 1D).
* Divide it by rows, columns, or blocks among processes
* Each process computes the partial sum of its assigned portion.
* Use MPI\_Reduce to combine all local sums into a global sum.

Example idea:

local\_sum = sum(matrix\_part);

MPI\_Reduce(&local\_sum, &global\_sum, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

(b) Matrix averaging

* Compute total sum using MPI\_Reduce as above.
* Divide the final sum by the total number of elements:

average = global\_sum / (rows \* cols);