Ain Shams University
Faculty of Engineering
Computer and Systems Engineering Department

CSE 352: Distributed Computer Systems – 3th Year CSE – 2st Semester 2024/2025

S H E E T 8

Assume we need to calculate the value of the function e^x, which can be computed by the following formula:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

Write a C program that uses MPI parallelization to compute the value of e^x using the above formula (using the Taylor series). You need to get the upper value of *k* from the user, where your program makes the computation by dividing *k* equally among the processes, it should use 10 processes to do this computation. Then, it displays the computed value of e^x.The value of e^x and the time taken by the program to compute it should be displayed to the user. Assume there are functions that you can use to compute the factorial and the power, which are *fact*(*k*) to compute *y*!, and *pow*(*y*, *k*) to compute y^k.

Solution:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>

// Function to compute k!
double fact(int k) {
    if (k == 0 || k == 1) return 1.0;
    double result = 1.0;
    for (int i = 2; i <= k; i++) {
        result *= i;
    }
    return result;
}

// Function to compute x^k
double pow_val(double x, int k) {
    double result = 1.0;
    for (int i = 0; i < k; i++) {
        result *= x;
    }
    return result;
}

int main(int argc, char** argv) {
    int rank, size, K;
    double x, local_sum = 0.0, global_sum = 0.0;
    double start_time, end_time;

    // Initialize MPI environment
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);  // Get current process rank
    MPI_Comm_size(MPI_COMM_WORLD, &size);  // Get number of processes
```

```
    // Root process gets input
    if (rank == 0) {
        printf("Enter the value of x: ");
        scanf("%lf", &x);
        printf("Enter the number of terms (K): ");
        scanf("%d", &K);
        start_time = MPI_Wtime();  // Start timer
    }

    // Broadcast values to all processes
    MPI_Bcast(&x, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(&K, 1, MPI_INT, 0, MPI_COMM_WORLD);

    // Each process calculates its assigned terms of the series
    for (int k = rank; k <= K; k += size) {
        double term = pow_val(x, k) / fact(k);  // x^k / k!
        local_sum += term;
    }

    // Reduce all local sums to the global sum at root process
    MPI_Reduce(&local_sum, &global_sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    // Output result from root
    if (rank == 0) {
        end_time = MPI_Wtime();  // Stop timer
        printf("\nApproximated value of e^x = %.10f\n", global_sum);
        printf("Time taken: %.6f seconds\n", end_time - start_time);
    }

    MPI_Finalize();  // Finalize MPI
    return 0;
}
```

Notes:

Broadcasting Input Values:

- MPI_Bcast(&k_upper, 1, MPI_INT, 0, MPI_COMM_WORLD): Broadcasts the value of k_upper from process 0 to all other processes in MPI_COMM_WORLD.

- MPI_Bcast(&x, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD): Broadcasts the value of x from process 0 to all other processes.
- Broadcasting ensures that all processes have the same values for k_upper and x

Collective Communication:

- MPI_Bcast: One process (the root) sends the same data to all other processes in the communicator.
- MPI_Reduce: Combines data from all processes using a specified operation (in this case, summation) and returns the result to a designated root process.