Exercise: MPI Matrix Multiplication Benchmark

Before attempting the exercises in this document please ensure that you have read and understood the key topics covered in tutorial.

Objective:

Benchmark the performance of matrix multiplication using MPI on an SCC cluster (your account). This exercise will involve writing a simple MPI program, varying the number of processes, and measuring the execution time to understand scalability.

To do this exercise you will need:

- Access to an SCC cluster with MPI installed.
- Basic knowledge of MPI programming (e.g., sending and receiving messages).
- Familiarity with compiling and running MPI programs.
- You will require python packages like MPI and numpy.

Contents

Task 1: Matrix Multiplication (20 min)

- Write an MPI program to perform matrix multiplication.
  1. You could take help from "Parallel Computing: Basic Principles" course by Oswald Haan on Friday (05.04.2024).
  2. a) Skeleton code for function 1 (language python):

```python
def matrix_multiply(A, B):
    return np.dot(A, B)
```

  b) Skeleton Code for Function 2 (language python):

```python
 /* Main function. */
from mpi4py import MPI
import numpy as np


```

```
if __name__ == "__main__":
  /* Declaring the required variables and the required MPI functions. */

    # Matrix dimensions
    N = 100
    M = 100
    K = 100


    # Create random matrices A and B
    np.random.seed(42)
    A = np.random.rand(N, M)
    B = np.random.rand(M, K)


  # Your code for MPI Functions




    # Your code for splitting matrices among processes




    # Your code for perform local matrix multiplication




    # Your code for gathering results




    if rank == 0:
        # Combine results
        result = np.concatenate(C, axis=0)
        print("Result Matrix (C):")
        print(result)
}

/* Finally, test and compile the code. Keep the file name "mpi_matrix_multiply.py"*/
```

3. Compile and run the program.
   - Compile the program using mpicc or mpifort, depending on your MPI compiler:

```
mpicc -o mpi_matrix_multiply mpi_matrix_multiply.py
```

4. Run the program with varying numbers of MPI processes using the bash script:

```
mpiexec -n <num_processes> ./mpi_matrix_multiply
```

5. Measure the execution time for each run using time or mpiexec's built-in timing options.

Task 2: Benchmarking (25 min)

- Do the benchmarking using your program (known as strong and week scaling benchmark).
- Vary the number of MPI processes (-n) from 1 to a suitable maximum based on the available cores in your cluster.

- Record the execution time for each run.
- Plot a graph showing the scalability of the matrix multiplication with respect to the number of processes.
- X-axis: Number of MPI processes
- Y-axis: Execution time

Task 3: Analysis (20 min)

- Analysis:
  - Analyze the benchmark results to understand the scalability of the matrix multiplication program.
  - Look for trends in execution time as the number of processes increases.
  - Identify any bottlenecks or inefficiencies in the program or the cluster configuration.
  - Summarize (short-document) and complete your benchmarking analysis.
  - Include all the details you argue relevant on your critical thinking.
  - For convenience you could also store your records in a CSV file, with the file name "hpctrainingNN.csv".
- Discussion:
  - How does the execution time change with an increasing number of processes?
  - Does the program exhibit strong or weak scaling?
  - What factors might contribute to the observed scalability?
  - How could the program or the cluster configuration be optimized for better performance?

Hints

- You could also create and include graphs and chart plots as felt necessary.
- Here "NN" is your user code. You could also share your findings during the session for the feedback.
- You can modify the matrix dimensions (N, M, K) to vary the size of the matrices and observe their impact on performance.
- We encourage you to experiment with different MPI functions, matrix sizes, and cluster configurations to deepen your understanding of HPC benchmarking.