Project Report

Parallel and Distributed Computing

# Group Members: 1-Muhammad Talha(21K-3288)

# 2-Syed Ismail (21K-4571)

# 3-Abdul Ahad Nauman(21K-4618)

# **1.Project Title:**

## **Cramers Rule and Quick Sort Using OpenMP And MPI**

# 2.Project Introduction

Parallel computing plays a huge role in meeting the computational requirements of applications with data. This project evolves around the parallelization of two fundamental algorithms, Cramer's Rule and Quick Sort, utilizing both OpenMP and Message Passing Interface (MPI). The primary goal is to investigate the impact of parallelization on key performance metrics such as computation time, communication time, and scalability across varying data sizes and comparing them through graphs.The project entails the parallel implementation of Cramer's Rule using OpenMP and MPI, as well as parallel Quick Sort utilizing the same frameworks. We analyze the execution times for different data sizes, assessing the efficiency of parallelization concerning both computation and communication..

3.Methodology (Pseudocode of serial and parallel version)

1-Cramers Rule with OpenMP

Shared-memory parallelism, we applied OpenMP directives to parallelize the Cramer's Rule algorithm. Our focus was on parallelizing the computation of determinants and solving equations. This approach aimed to improve the algorithm's overall efficiency.

Pseudocode:

1. Function SolveUsingCramer\_OpenMP(matrix A, vector B)
   * Ensure that A is a square matrix and B is a column vector of the same length.
   * n = number of rows in A
   * Calculate detA = Determinant\_OpenMP(A)
   * If detA == 0, return "No unique solution"
   * Initialize solution vector X of size n
   * #pragma omp parallel for
     + For i = 0 to n-1:
       - Create matrix Ai by replacing the i-th column of A with B
       - X[i] = Determinant\_OpenMP(Ai) / detA
   * Return X
2. Function Determinant\_OpenMP(matrix A)
   * Implement the determinant calculation (e.g., LU decomposition, expansion by minors) with OpenMP parallel constructs

2-Cramers Rule with MPI

For the MPI implementation of Cramer's Rule, we distributed the data and computations across multiple nodes. Each node handled a portion of the data, and MPI communication facilitated the iterative exchange of matrix information until the final solution was reached.

Pseudocode:

1. Function SolveUsingCramer\_MPI(matrix A, vector B, rank, size)
   * Ensure that A is a square matrix and B is a column vector of the same length.
   * n = number of rows in A
   * If rank == 0 (master process):
     + Calculate detA = Determinant\_MPI(A)
     + If detA == 0, broadcast "No unique solution" to all processes and return
   * Broadcast detA to all processes
   * Divide the task of calculating determinants of modified matrices Ai among the MPI processes
   * Each process calculates a subset of Determinant\_MPI(Ai) for some i
   * Gather all determinants at the master process
   * If rank == 0:
     + Compute solution vector X[i] = Determinant\_MPI(Ai) / detA for each i
     + Return X
2. Function Determinant\_MPI(matrix A)
   * Implement the determinant calculation with MPI parallel constructs

3-Quick Sort with OpenMP

In the Quick Sort implementation, we used the power of OpenMP directives to parallelize the sorting process. Breaking down the algorithm into segments suitable for simultaneous execution allowed for an efficient sorting of the data.

Pseudocode (Sources given at the end):

OpenMP Version

1. Function QuickSort\_OpenMP(arr, left, right)
   * If left < right:
     + pivotIndex = Partition(arr, left, right)
     + #pragma omp parallel sections
       - #pragma omp section
         * QuickSort\_OpenMP(arr, left, pivotIndex - 1)
       - #pragma omp section
         * QuickSort\_OpenMP(arr, pivotIndex + 1, right)
2. Function Partition(arr, left, right)
   * Choose pivot = arr[right]
   * i = left - 1
   * For j = left to right-1:
     + If arr[j] < pivot:
       - i++
       - Swap arr[i] and arr[j]
   * Swap arr[i + 1] and arr[right]
   * Return i + 1

4-Quick Sort with MPI

For the MPI implementation of Quick Sort, we adapted to a distributed-memory environment. Each node in the cluster took on a portion of the data, and MPI communication was employed to exchange information about the sorted segments, ensuring a synchronized and effective sorting operation across the distributed system.

Pseudocode:

1. Function QuickSort\_MPI(arr, left, right, depth, maxDepth)
   * If depth >= maxDepth:
     + QuickSort\_Serial(arr, left, right) (use a serial quicksort)
     + Return
   * If left < right:
     + pivotIndex = Partition(arr, left, right)
     + Split the array into two sub-arrays
     + Perform MPI asynchronous send and receive to distribute the sub-arrays across processors
     + Each processor recursively calls QuickSort\_MPI on its sub-array
     + Perform MPI gather to collect the sorted sub-arrays
2. Function Partition(arr, left, right) (same as in OpenMP version)
3. Function QuickSort\_Serial(arr, left, right) (standard quick sort implementation for serial computation)

4.Performance Comparison(OpenMP VS MPI )on different Sizes

1.Cramer’s Rule (with 8 processes and 8 threads)

Note: Inefficient method hence the sizes cannot be changed that exponentially

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cramer’s Rule Openmp (Time Taken in seconds) | Cramers’Rule MPI | |
| Size | Computation Time | Computation Time | Communication Time |
| 3 | 0.000014 | 0.002447 | 0.000004 |
| 6 | 0.000032 | 0.000156 | 0.000003 |
| 9 | 0.016223 | 0.009897 | 0.000004 |

Graph

A graph with a line

Description automatically generated

Communication Time MPI

A line graph with a point

Description automatically generated

2.Quick Sort

|  |  |  |  |
| --- | --- | --- | --- |
|  | Quick Sort Openmp (Time Taken in seconds) | Quick Sort MPI | |
| Size | Computation Time | Computation Time | Communication Time |
| 1000 | 0.001747 | 0.016781 | 0.000018 |
| 2000 | 0.002183 | 0.21401 | 0.000027 |
| 5000 | 0.005281 | 0.27854 | 0.000031 |

A graph with a line

Description automatically generated

Communication Time MPI

A graph with a line

Description automatically generated

5.Conclusion

In summary, when comparing the implementation of Quick Sort using OpenMP and Quick Sort using MPI:

Quick Sort with OpenMP benefits significantly from the shared-memory model, leading to reduced sorting time, especially for large datasets. OpenMP's efficient utilization of multicore processors and lower communication cost due to shared memory are key advantages.

Quick Sort with MPI: Implementing Quick Sort in an MPI (distributed memory) environment can be challenging due to the inherent recursive nature of the algorithm, which might not naturally fit into the distributed memory model. The communication overhead between processes in MPI can impact the overall performance, especially for large datasets where frequent data exchange is required. However, for extremely large datasets that exceed the memory capacity of a single machine, MPI can provide a scalable solution.

Now for Cramer’s rule with OpenMP and MPI:

Cramer's Rule with OpenMP: While OpenMP's shared-memory model offers advantages in terms of memory efficiency and lower communication overhead, Cramer's Rule may not fully benefit from this. Since Cramer's Rule involves determinant calculations, which are computationally intensive, the parallelization potential might not be fully realized, especially if the matrix size is large. The shared-memory model, however, does help in reducing the memory footprint compared to a distributed memory approach.

Cramer's Rule with MPI can leverage multiple processors in a distributed system, but its computational intensity and the need for frequent inter-process communication can limit its performance gains. MPI's distributed memory model might lead to higher communication overhead and increased memory usage compared to OpenMP.

6.Sources:

The pseudocodes are based on the sources listed below:

Quick Sort from "Introduction to Algorithms" by Cormen, Leiserson, Rivest, and Stein

Cramer’s Rule from Linear Algebra’ textbook : Elementary Linear Algebra by Howard Anton, Anton Kaul