|  |  |
| --- | --- |
| **Course:** | **CSC510, Spring 2024** |
| **Assignment #2:** | **Estimate π in Parallel Using C and MPI** |
| **Name:** | **Hardikkumar Malaviya** |
| **Student Id:** | **w10142959** |
| **Submission Date:** | **02/28/2024** |

**Index**

1. **Overview of the problem**
2. **Description of parallel algorithm design and implementations**
3. **Results in Table 1**
4. **Results in Table 2**
5. **Screenshots corresponding to three different cases from Table 1 and Table 2**
6. **Discussion and conclusion**
7. **Appendix**

**1. Overview of the Problem:**

The problem involves estimating the value of π using a specific formula for an infinite series. with both serial and parallel implementations. The formula used is known as the Leibniz formula for π:

In this formula, i varies from 0 to n, and the more terms you include in the series (n increases), the closer the sum gets to the actual value of π.

A math problem with numbers

Description automatically generated

This is a basic parallel implementation using MPI in C. The program divides the work among different processes, calculates local sums, and then combines them to get a global sum. The goal is to write a serial C program to calculate this sum and then parallelize it using MPI (Message Passing Interface). Two parallel programs are developed using point-to-point communication functions (MPI\_Send and MPI\_Recv) and collective communication functions.

**2. Description of Parallel Algorithm Design and Implementations:**

Load all three files in magnolia server:

File\_1: Sum\_Serial.c

File\_2: Sum\_MPI\_v1.c (Point to point communication)

File\_3: Sum\_MPI\_v2.c (Collective communication)

A screenshot of a computer

Description automatically generated

Login to magnolia:

A screenshot of a computer program

Description automatically generated

**Serial Implementation (**Sum\_Serial.c**):** The serial program calculates the sum using the given formula in a sequential manner. It serves as a baseline for comparison with parallel implementations.

A screenshot of a computer program

Description automatically generated

**Parallel Implementation (**Sum\_MPI\_v1.c**):**

The first parallel program uses point-to-point communication functions, MPI\_Send and MPI\_Recv. Each process calculates its partial sum, and communication occurs to aggregate the partial sums into the final result.

Compiling and running parallel code:

|  |  |
| --- | --- |
| First we load the mpi module: | **module load openmpi -2.0/gcc** |
| Next compile the MPI program: | **mpicc -O2 -o Sum\_MPI\_v1 Sum\_MPI\_v1.c -lm** |
| Execution of parallel code: | **mpirun -c 1 ./ Sum\_MPI\_v2** |

A computer screen with white text

Description automatically generated

**Parallel Implementation (**Sum\_MPI\_v2.c**):**

The second parallel program utilizes collective communication functions. The MPI collective operations (e.g., MPI\_Reduce) are employed to efficiently combine the partial sums calculated by individual processes.

Compiling and running parallel code:

|  |  |
| --- | --- |
| First we load the mpi module: | **module load openmpi -2.0/gcc** |
| Next compile the MPI program: | **mpicc -O2 -o Sum\_MPI\_v2 Sum\_MPI\_v2.c -lm** |
| Execution of parallel code: | **mpirun -c 1 ./ Sum\_MPI\_v2** |
|  |  |

A computer screen with white text

Description automatically generated

Results are shown in Table 1,2 and in screenshots section.

**Discussion Regarding Parallel Performance:**

The results in Table 1 compare the sum obtained by the three programs for different **n** and **p** values. It ensures the correctness of the parallel implementations by validating against the serial results.

Table 2 provides a detailed performance analysis of the two parallel implementations for **n= 1,000,000,000**. It includes the parallel runtime (T), speedup (S), and efficiency (E) for different numbers of processes (p).

Speedup (S): Calculated as the ratio of serial runtime (Ts) to parallel runtime(Tp), the speedup indicates how much faster the parallel program is compared to the serial version.

Efficiency (E): Efficiency is the ratio of speedup to the number of processes (p). It measures how well the parallel algorithm scales with an increasing number of processes.

The analysis helps in understanding the parallel performance, identifying bottlenecks, and evaluating the scalability of the parallel implementations.

**3. Table 1:** Comparison of Sum results with respect to different n and p values.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sum | Sum\_Serial.c | Sum\_MPI\_v1.c | | Sum\_MPI\_v2.c | |
| p = 1 | p = 8 | p = 1 | p = 8 |
| n = 1 | **2.666667** | **2.666667** | **2.666667** | **2.666667** | **2.666667** |
| n = 3 | **2.895238** | **2.895238** | **2.895238** | **2.895238** | **2.895238** |
| n = 10 | **3.232316** | **3.232316** | **3.232316** | **3.232316** | **3.232316** |
| n = 1000 | **3.142592** | **3.142592** | **3.142592** | **3.142592** | **3.142592** |
| n = 1000000 | **3.141594** | **3.141594** | **3.141594** | **3.141594** | **3.141594** |

**4. Table 2:** Comparison of the parallel performance of the two parallel implementations **(n=1000,000,000)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sum S | Sum\_MPI\_v1.c | | | Sum\_MPI\_v2.c | | |
| T1 | S1 | E1 | T2 | S2 | E2 |
| p = 1 | **42.854759** | **0.946129** | **0.946129** | **41.804953** | **0.917506** | **0.917506** |
| p = 2 | **21.310554** | **1.991845** | **0.995923** | **22.352730** | **1.819020** | **0.909510** |
| p = 4 | **11.161973** | **3.788267** | **0.947067** | **10.104263** | **3.797137** | **0.949284** |
| p = 8 | **5.637753** | **7.840941** | **0.980118** | **5.739658** | **7.370813** | **0.921352** |
| p = 16 | **3.065641** | **14.719159** | **0.919947** | **2.868254** | **15.824588** | **0.989037** |

**5. Screenshots:**

**Table 1 : Sum\_MPI\_v1.c** **( P=1 & P=8 )**

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated  Fig. P=1 | A screenshot of a computer program  Description automatically generated  Fig. P=8 |

**Table 1: Sum\_MPI\_v2.c** **(P=1 & P=8)**

|  |  |
| --- | --- |
| A screenshot of a computer program  Description automatically generated  Fig. P=1 | A screenshot of a computer  Description automatically generated  Fig. P=2 |

**Table 2: Sum\_MPI\_v1.c &** **Sum\_MPI\_v2.c**

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated  Fig. Sum\_MPI\_v1.c | A screenshot of a computer program  Description automatically generated  Fig. Sum\_MPI\_v2.c |

**6. Discussion and conclusion** (discussion regarding the parallel performance observed in both implementations):

In a serial implementation, a single processor executes all operations, making it suitable for small programs. However, employing serial code for large programs results in prolonged execution times due to the constraint of having only one processor.

On the other hand, parallel implementation involves utilizing multiple processors concurrently to address a problem swiftly compared to a single processor. This approach divides the task among multiple processors, enabling parallel execution and significantly reducing the time required for large programs.

In summary, parallel code is more time-efficient for extensive programs since the workload is distributed among multiple processors and executed simultaneously. In contrast, a serial implementation is a straightforward C program well-suited for smaller programs.

The performance analysis of the parallel implementations, Sum\_MPI\_v1.c and Sum\_MPI\_v2.c, reveals a compelling trend as the number of processes increases. The parallel runtime consistently decreases, indicative of the effective distribution of workload across multiple processors. This reduction in runtime corresponds to an escalating speedup, demonstrating the advantage of parallel execution in significantly reducing computation time compared to a single processor. Moreover, the efficiency values, consistently exceeding 0.9, emphasize the successful scaling of the parallel implementations. The findings collectively highlight the commendable performance improvement achieved through parallelization, showcasing its efficacy in handling large-scale computations.

In conclusion, the analysis underscores the efficiency gains achieved by parallel implementations, emphasizing their suitability for handling substantial computational workloads. The decreasing runtime, increasing speedup, and consistently high efficiency affirm the successful parallelization strategy, providing valuable insights into the scalable and effective nature of the parallel algorithms employed in Sum\_MPI\_v1.c and Sum\_MPI\_v2.c.

**7. Appendix**

1. <https://github.com/Hardik-Malaviya/CSC-510---Assignment-2>

2. Source code:

|  |  |
| --- | --- |
| A screenshot of a computer program  Description automatically generated  Fig. Sum\_MPI\_v1.c | A screenshot of a computer program  Description automatically generated  Fig. Sum\_MPI\_v2.c |
| A screenshot of a computer program  Description automatically generated  Fig. Sum\_Serial.c |

**Student Name**: Hardikkumar Malaviya (w10142959)

**Assignment #2: Grading Policy**

|  |  |  |  |
| --- | --- | --- | --- |
| **Grade Components** | **Points** | **Points Earned** | **Comments** |
| C/MPI Programs (comments, clarity, efficient implementation) | 20 |  |  |
| Correctness of problem solving  (Table 1) | 10 |  |  |
| Compilation and Execution (Screenshots and consistent results) | 10 |  |  |
| HPC performance measure and analysis (Tables 2, 3, 4 and two figures) | 10 |  |  |
| Report  (all components, Word format and clarity) | 20 |  |  |
| Total | 70 |  |  |