CSE4269– Parallel and Distributed Systems Lab

Lab Assignment 03

PI calculation using the Master-Worker paradigm.

Submitted by

Kazi Shadman Sakib
Roll: CSE-FH-097
Reg: 2017-614-973

Submitted to

Dr. Upama Kabir
Professor
Department of Computer Science & Engineering
University of Dhaka

Submission Date

24.10.2023

1 Own Machine-Specific Speed-Up Analysis:

The Master-Worker paradigm is a common approach in parallel computing, where a master process or thread coordinates the execution of multiple worker processes or threads to solve a particular task. In this assignment, we will investigate the speedup achieved in PI calculation programs when implemented using parallel version and dynamic spawning of workers version. Speedup, often denoted as 'S,' is a measure of how much faster a parallel program runs compared to a sequential (single-threaded) version.

To calculate speedup for both the parallel and dynamic implementations of the PI calculation program, we will use the following formula:

$$\text{Speedup}(S) = \frac{T_{\text{sequential}}}{T_{\text{parallel/dynamic}}}$$

The speedup in parallel computing is often influenced by various factors, including the communication overhead, load balancing, and inherent limitations of the algorithm being parallelized. In our case, we've compared two MPI-based programs: parallelPI and dynamicSpawnPI.

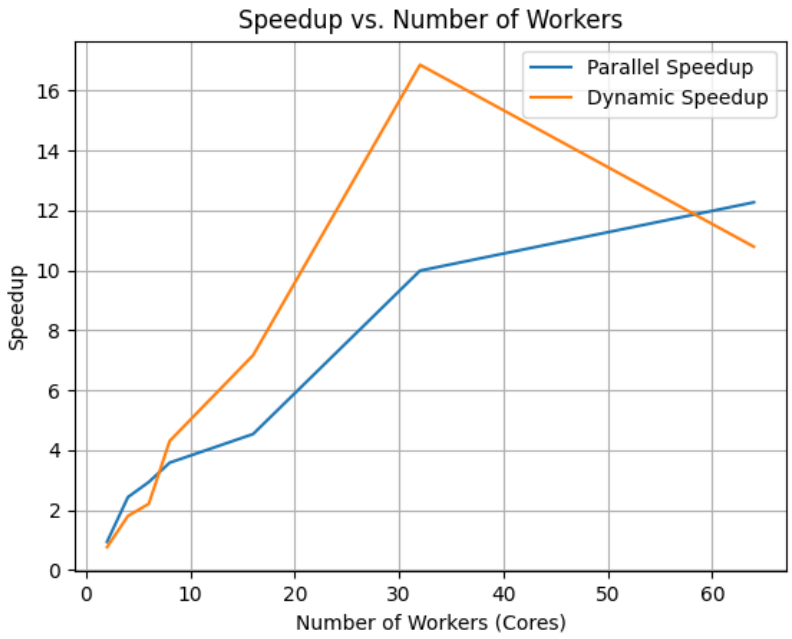

Figure 1: Speedup Performance Analysis in Parallel Computing

The graph depicted in the Figure 1 illustrates the correlation between the number of workers (CPU cores) and the speedup of both parallelPI and dynamicSpawnPI algorithms. The blue curve signifies the expected speedup for parallelPI, which exhibits a linear growth pattern with an increasing number of workers. However, in the case of dynamicSpawnPI, a more complex behavior is observed.

Prior to approximately 30-32 workers or CPU cores, the speedup for dynamicSpawnPI surpasses the expected parallel speedup, demonstrating a superlinear trend. This phenomenon indicates that, with this specific number of workers, dynamicSpawnPI reaches its optimal performance. Nevertheless, as the number of workers or CPU cores surpasses this threshold, around 35 or more, the dynamicSpawnPI's speedup begins to slow down, although it continues to outperform the parallel speedup.

This distinctive behavior in dynamicSpawnPI can be attributed to certain overhead associated with coordinating multiple workers, encompassing communication and synchronization. This overhead intensifies as the number of workers increases, leading to a sub-linear speedup in practical scenarios. The graph illustrates this dynamic speedup with an orange curve.

The discrepancy between dynamic speedup and parallel speedup becomes more pronounced when the number of workers or CPU cores exceeds approximately 58. Beyond this point, the dynamic speedup lags behind the parallel speedup, resulting in a less favorable performance comparison.

The decrease in speedup of dynamicSpawnPI when we increase the number of workers or cores from approximately 35 to 64 can be attributed to a few factors:

1. **Communication Overhead:** When we have more processes, the overhead of coordinating and communicating between them can increase. In dynamicSpawnPI, each worker process communicates its local results back to the master process, which may involve more communication overhead as the number of worker processes increases. This can lead to slower performance compared to the more direct communication in parallelPI.
2. **Load Imbalance:** As we distribute the workload among more processes, it becomes more challenging to achieve load balancing. Some worker processes may finish their tasks faster than others, leading to idle times where they are waiting for the slower processes to catch up. Load imbalance can reduce the overall efficiency of the parallel computation.
3. **Synchronization and Waiting:** In dynamicSpawnPI, the master process waits for all worker processes to complete their tasks and send results back. As we increase the number of processes, waiting for them to finish and synchronize can become a bottleneck.