PARALLEL COMPUTING

Converting Kruskal's Algorithm Parallel

SUBMITTED BY
0908
0914
0921
0923
0927
0945

Introduction

This project is intended towards converting a problem that has a serialized solution available, into a parallel one using Foster's methodology. We have chosen the Kruskal's algorithm of finding the minimum spanning tree for a given graph. We have used MPI to code the solution and measured the performance of our approach. The solution scope and limitations are also discussed in this report.

We have parallelized the serial program according to Foster's Methodology, that is, we have proceeded towards the following steps:

Partition: We have divided the problem into tasks to identify and expose opportunities for parallel execution.

Communication: We have shared data between computations and determined the amount and pattern of communication.

Agglomeration: Then we have combined tasks to improve performance.

Mapping: After that, agglomerated tasks had been assigned to threads.

Source Code Description:

1. We have taken a 2D matrix arr[12][12], as we are working with 12 nodes to find the minimum spanning tree using Kruskal's algorithm. (Line 4)
2. An array parent[12] is taken to keep track whether the parent of a pair of nodes is same or not.
3. In line 20, there's a call to MPI_INIT that tells the MPI system to do all of the necessary setup. The arguments, argc and argv are pointers to the arguments to main, argc and argv.
4. Double start = MPI_Wtime() is there at line 21 to keep track of the execution time. Here the execution time starts.
5. MPI_Init defined a communicator that consists of all of the processes started by the user when she started the program. This communicator is called MPI COMM WORLD. The function calls in Lines 23 and 25 are getting information about MPI COMM WORLD. Their syntax is
    i. MPI_Comm_rank(MPI_COMM_WORLD,&world_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

ii. For both functions, the first argument is a communicator and has the special type defined by MPI for communicators, MPI_Comm. MPI_Comm_size returns in its second argument the number of processes in the communicator, and MPI_Comm_rank returns in its second argument the calling process' rank in the communicator.

6. int slot=12/(world_size-1); with this variable named "slot", we are determining the number of rows per process will get from the 2D matrix.
7. The tags are there at line 27-32 for working as send tag and receive tag in MPI_Send and MPI_Recv
8. At line 34, the index tag is declared. It will later be used to keep track of the number of rows per process will receive.
9. The variable source is there to determine the root process.
10. The variable minCost will contain the final cost. (Line 36)
11. From line 38 to 66, the responsibility of the root process has been handled.
12. For all the 12 nodes, the parent array has been assigned value -1. We will be working here with 7 processes. That is world_size=7.
13. Now the for loop from line 45-59 will send the index, 2D matrix and parent array to the slave processes, and will receive the partial sum (the partially calculated minimum cost for the minimum spanning tree created by each slave process), the updated 2D matrix and the updated parent array. Then the partial sum is added with the minCost and the minimum cost is printed.
14. Double end = MPI_Wtime(); is used to keep track when the execution ended. The total execution time is calculated by subtracting start from end (end-start) and is printed at line 63. We will this execution time later to compare the performance of the serial code and parallelized code.
15. Line 68-111 handles the responsibility of the non-root processes, that is,the slave processes.
16. From line 72-74, the slave processes are receiving the index, 2D matrix and the parent array.
17. The while loop at line 79, what actually happening is, each process is finding the min edge from each of the assigned two rows from the 2D array. While finding the min edge between 2 nodes, the parent of the 2 nodes are checked. If the parents are same, then the edge is rejected. Finding parent is done in the function find() at line 117. And the checking whether the parents are the same, is done in the function uni() at line 124.
18. The minimum edges obtained from the assigned 2 rows are the added to the variable sum, and the parent array and the 2D matrix are updated.
19. From line 107-109 the values of sum, updated 2D array and updated parent array are sent to the root process using MPI_Send.
20. At line 113, we called MPI_Finalize because any resources allocated for MPI can be freed now as we're done using MPI.

Solution Scope:

The solution works with a 12*12 2D matrix. As we are working with 7 nodes, each node gets 2 rows from the matrix and finds out the minimum edges from the assigned 2 rows. They send the minimum edges to the root process. The responsibility of the root process is to add up the minimum edges received from each of the slave processes and calculate the total minimum cost to build a minimum spanning tree.

Limitations:

Performance of the will decrease with increasing size of dataset.

Solution Performance:

For 7 processes,

The Execution Time for the parallel code is 0.0467374325 ms.

The Execution Time for the serial code is 0.0003980000 ms.

Here we see, the execution time for the parallel code is more than the execution time of the serial code. It happened due to Parallel slowdown. Each processing node here is spending more time doing communication than useful processing. The communications overhead is surpassing the increased processing power and parallel slowdown is occurring.