Ex.No:5 Implementing All-pass Shortest Path algorithm using Open-Mpi

Nithishkumar.K
18CO66

Aim

To parally run the Floyds algorithm to find all pair shortest path using open-Mpi

Description of the tool:

open MPI is a free and open source libarary for high performance Computing. It use message passing interface to Cordinate between the threads It doesn't have shared memory it is cross-platform and Support Languges C, C++, and fortran.

Steps to Implement

- Install the mpicc Compiler
- modify the all pairs shortest algorithm to work with open-mpi by adding MPI-Gather and MPI scatter methods to do it.
- Also keep track the ranks of the threads and also the root thread.
- Now open the terminal and navigate to the location where you have the Source Code.
- Compile the program using the mpicc Command

Nithishkumar. k.
18 co66

- once Command where users are required to pass the number of threads as arguement, which should match the number of nodes in the input graph.

Result

The all Pair Shortest Path algorithm was executed and the output has been verified.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h> /* for debugging */
#include <mpi.h>

const int INFINITY = 1000000;

void Read_matrix(int local_mat[], int n, int my_rank, int p, MPI_Comm comm);

void Print_matrix(int local_mat[], int n, int my_rank, int p, MPI_Comm comm);

void Floyd(int local_mat[], int n, int my_rank, int p, MPI_Comm comm);

int Owner(int k, int p, int n);

void Copy_row(int local_mat[], int n, int p, int row_k[], int k);

void Print_row(int local_mat[], int n, int my_rank, int i);

int main(int argc, char* argv[])
{
 int n;
 int* local_mat;
 MPI_Comm comm;
 int p, my_rank;

 MPI_Init(&argc, &argv);
 comm = MPI_COMM_WORLD;

 MPI_Comm_size(comm, &p);
 MPI_Comm_rank(comm, &my_rank);

 if (my_rank == 0)
 {
 printf("How many vertices?\n");
 scanf("%d", &n);
 }

 MPI_Bcast(&n, 1, MPI_INT, 0, comm);
 local_mat = malloc(n*n/p*sizeof(int));

 if (my_rank == 0)
    printf("Enter the local_matrix\n");

 Read_matrix(local_mat, n, my_rank, p, comm);

 if (my_rank == 0)
    printf("We got\n");

 Print_matrix(local_mat, n, my_rank, p, comm);
```

```
 if (my_rank == 0)
    printf("\n");

 Floyd(local_mat, n, my_rank, p, comm);

 if (my_rank == 0)
    printf("The solution is:\n");

 Print_matrix(local_mat, n, my_rank, p, comm);

 free(local_mat);
 MPI_Finalize();
 return 0;

}


 /* main */
void Read_matrix(int local_mat[], int n, int my_rank, int p, MPI_Comm comm)
{
 int i, j;
 int* temp_mat = NULL;
 if (my_rank == 0)
 {
 temp_mat = malloc(n*n*sizeof(int));

 for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        scanf("%d", &temp_mat[i*n+j]);

 MPI_Scatter(temp_mat, n*n/p, MPI_INT, local_mat, n*n/p, MPI_INT, 0, comm);

 free(temp_mat);

 }
 else
 {
 MPI_Scatter(temp_mat, n*n/p, MPI_INT,
 local_mat, n*n/p, MPI_INT, 0, comm);
 }
}

/* Read_matrix */

void Print_row(int local_mat[], int n, int my_rank, int i)
{
 char char_int[100];
 char char_row[1000];
 int j, offset = 0;
```

```
 for (j = 0; j < n; j++)
 {
 if (local_mat[i*n + j] == INFINITY)
    sprintf(char_int, "i ");
 else
    sprintf(char_int, "%d ", local_mat[i*n + j]);
    sprintf(char_row + offset, "%s", char_int);
    offset += strlen(char_int);
 }

 printf("Proc %d > row %d = %s\n", my_rank, i, char_row);

}
/* Print_row */

void Print_matrix(int local_mat[], int n, int my_rank, int p, MPI_Comm comm)
{
 int i, j;
 int* temp_mat = NULL;

 if (my_rank == 0)
 {
 temp_mat = malloc(n*n*sizeof(int));
 MPI_Gather(local_mat, n*n/p, MPI_INT,
 temp_mat, n*n/p, MPI_INT, 0, comm);
 for (i = 0; i < n; i++)
 {
 for (j = 0; j < n; j++)
 if (temp_mat[i*n+j] == INFINITY)
 printf("i ");
 else
 printf("%d ", temp_mat[i*n+j]);
 printf("\n");
 }
 free(temp_mat);
 }
 else
 {
 MPI_Gather(local_mat, n*n/p, MPI_INT,
 temp_mat, n*n/p, MPI_INT, 0, comm);
 }
}

 /* Print_matrix */


void Floyd(int local_mat[], int n, int my_rank, int p, MPI_Comm comm)
{
 int global_k, local_i, global_j, temp;
```

```
 int root;
 int* row_k = malloc(n*sizeof(int));
 for (global_k = 0; global_k < n; global_k++)
 {
 root = Owner(global_k, p, n);
 if (my_rank == root)
 Copy_row(local_mat, n, p, row_k, global_k);
 MPI_Bcast(row_k, n, MPI_INT, root, comm);
 for (local_i = 0; local_i < n/p; local_i++)
 for (global_j = 0; global_j < n; global_j++)
 {
 temp = local_mat[local_i*n + global_k] + row_k[global_j];
 if (temp < local_mat[local_i*n+global_j])
 local_mat[local_i*n + global_j] = temp;
 }
 }
 free(row_k);
}

/* Floyd */


int Owner(int k, int p, int n)
{
 return k/(n/p);
} /* Owner */


void Copy_row(int local_mat[], int n, int p, int row_k[], int k)
{
 int j;
 int local_k = k % (n/p);
 for (j = 0; j < n; j++)
 row_k[j] = local_mat[local_k*n + j];
} /* Copy_row */
```

```
bing@LAPTOP-1DJLU00J:/mnt/c/Users/Gomathinayagam/Desktop$ mpirun -np 4 ./a.out
How many vertices?
4
Enter the local_matrix
0 1 5 0
3 0 0 2
9 0 0 1
0 4 3 0
We got
0 1 5 0
3 0 0 2
9 0 0 1
0 4 3 0

The solution is:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
bing@LAPTOP-1DJLU00J:/mnt/c/Users/Gomathinayagam/Desktop$
```