HPC Project

Implementation of Dijkstra Algorithm using MPI parallelization technique

REPORT

Submitted By

VENKATASUBRAMANIAN SUNDARAMOORTHY (Matriculation Number: 64143)

(Email: venkatasubramanian.sundaramoorthy@student.tu-freiberg.de)

HARIKESHAVA RANGANATHAN (Matriculation Number: 63941)

(Email: harikeshava.ranganathan@student.tu-freiberg.de)

Course: COMPUTATIONAL MATERIALS SCIENCE

Table of Contents

HPC Assignment

Implementation of Dijkstra Algorithm using MPI parallelization technique

1.Introduction:

Dijkstra Algorithm is used for obtaining the shortest distance from the source defined and all other nodes present in the system. The algorithm is implemented for the un-directional graph of interest. The nodal distance values are first generated and the shortest distances for each node from the source are generated and printed. The goal of this project is to implement the algorithm and execute it using the parallelization technique. Therefore, the code is implemented in C using the MPI parallelization technique.

2.Theory:

2.1. MPI (Message Passing Interface) Parallelization:

The MPI parallelization is a technique used to minimize the execution time of the code by making use of N processors. The code is created as a copy for each processor used in the program and the communication technique is used for sharing the results obtained in each processor. The communication between the processors and flow of the results are the challenging task in implementing the MPI technique.

2.2. Workflow:

2.2.1. Serial Program:

The series code is the normal implementation of the Dijkstra algorithm as mentioned above. The sequences of the code, works as mentioned below:

1. The number of Vertices, bound of the random values and the Start point is obtained from the user.
2. The above-mentioned variables(values) are used to generated the cost matrix of the system using the rand () function.
3. The minimum distance of each node is found after the iterative technique.
4. The shortest distance of the nodes from the source is calculated using Dijkstra's algorithm and printed.
5. Our final goal of finding the shortest distance is achieved.

Serial Program using rand () function -Cost matrix

(Filename: serial_code_for_randomly_generated_values.c)

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

#define infinity 999

void create_adj_matrix(int n,int adj_matrix[n][n]/*struct Edge_Vertex Edges[]*
/){
    /* getting the vertex number and corresponding weight from the randomly ge
nerated value and
    assign it to the adj_matrix*/
    int i,j;
    for(i=0;i<n;i++){
```

```
        for(j=0;j<n;j++)
        {
            if (i<j){
                int w = rand() % 20; //creating random values
                //for undirectional graph
                adj_matrix[i][j]=w;
            }
            else {
                adj_matrix[i][j]=0;
            }
        }
    }
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            adj_matrix[j][i]=adj_matrix[i][j];
        }
    }
}

void print_matrix(int n,int adj_matrix[][n])
    /*Printing the adj_matrix*/
{
    printf("\nThe Adjacency matrix representation of the graph \n\n");
     int i,j;
     for(i=0;i<n;i++)
     {
        for(j=0;j<n;j++)
         {
            printf("%d \t",adj_matrix[i][j]);
         }
         printf("\n");
     }
}

void  dijkstra_algorithm(int n,int cost_matrix[n][n],int distance[n],int prede
sor[n],int adj_matrix[][n],int start_node)
    /* Applying the shortest path algorithm and getting the
    shortest path from the source node to the each other nodes*/
{
    int visited_node[n],count,next_node,minimum_dist,i,j;
    ////create the cost matrix and assign the weight (if there is a connection
 between edges) and infinity to other
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            if(adj_matrix[i][j]==0)
                cost_matrix[i][j]=infinity;
            else
                cost_matrix[i][j]=adj_matrix[i][j];
        }
    }
```

```
    for(i=0;i<n;i++){
        visited_node[i]=0; //assigning visited node to 0
        distance[i]=cost_matrix[start_node][i]; //assigning dist to the startn
ode row of cost matrix
        predesor[i]=start_node; //to get the path of the corresponding minimum
 dist

    }
    visited_node[start_node]=1; //visited status of the source index to 1
    distance[start_node]=0; //distance of the source is zero
    count=1;

    while(count<n-1){
        minimum_dist=infinity;
        //setting the minimum dist to infiinity

        //check all the values in the distance array with minium and staus of
the index
        for(i=0;i<n;i++)
            if(distance[i]<minimum_dist&&!visited_node[i]){
                minimum_dist=distance[i];
                next_node=i;
            }

            visited_node[next_node]=1;
            //calculating the minium distance
            for(i=0;i<n;i++)
                if(!visited_node[i])
                    if(minimum_dist+cost_matrix[next_node][i]<distance[i]){
                        distance[i]=minimum_dist+cost_matrix[next_node][i];
                        predesor[i]=next_node;
                    }
        count++;

    }

}

void print_cost_matrix(int n,int cost_matrix[n][n])
    /*Printing the adj_matrix*/
{
     int i,j;
     for(i=0;i<n;i++)
     {
        for(j=0;j<n;j++)
         {
            printf("%d \t",cost_matrix[i][j]);
         }
         printf("\n");
```

```
     }
}

void print_shortest_path(int n,int distance[n],int predesor[n],int start_node)
{
    //Printing the shortest path from the source to other nodes
    int i,j;
    for(i=0;i<n;i++)

        if(i!=start_node)
        {
            printf("\nDistance of Node %d = %d ",i,distance[i]);
            printf("\nCorresponding path is = %d ",i);

            j=i;
            do
            {
                j=predesor[j];
                printf("<---%d",j);
            }while(j!=start_node);
    }
    printf("\n");
}

void main(){
    int n;
    printf("Enter the size of the array "); //getting the size of the array fr
om the user
    scanf("%d",&n);
    printf("\n");
    printf("The value of the maximum unknown distance is %d\n", infinity);
    int adj_matrix[n][n];//initialization of adj matrix
    int cost_matrix[n][n],start_node,distance[n],predesor[n]; //initialization
 of cost matrix, distance and precedor

    create_adj_matrix(n,adj_matrix); //function call adj matrix with zeros and
 ones
    print_matrix(n,adj_matrix);

    printf("\nEnter the starting node "); //getting the startnode from the use
r
    scanf("%d",&start_node);

    dijkstra_algorithm(n,cost_matrix,distance,predesor,adj_matrix,start_node);
  //function call to calculate the minimum distance

    printf("\nThe cost matrix for the given graph is \n\n");
    print_cost_matrix(n,cost_matrix); //priting cost matrix

    printf("\nThe shortest path using Dijkstra Algorithm \n\n");
```

```
    print_shortest_path(n,distance,predesor,start_node);//printing shortest pa
th
}
```

Serial Program for given graph:

(Filename: serial_code_for_given_example.c)

```
c:\Users\nivas\Desktop\Masters\3 Sem\HPC\Final_Project\HPC>gcc serial_code_for_given_example.c -o serial_code_for_given_example

c:\Users\nivas\Desktop\Masters\3 Sem\HPC\Final_Project\HPC>serial_code_for_given_example

The value of the maximum unknown distance is 9999

The Adjacency matrix representation of the graph

0       4       2       0       0
0       0       3       2       3
0       1       0       4       5
0       0       0       0       0
0       0       0       1       0

Enter the starting node 0

The cost matrix for the given graph is

9999    4       2       9999    9999
9999    9999    3       2       3
9999    1       9999    4       5
9999    9999    9999    9999    9999
9999    9999    9999    1       9999

The shortest path using Dijkstra Algorithm


Distance of Node 1 = 3
Corresponding path is = 1 <---2<---0
Distance of Node 2 = 2
Corresponding path is = 2 <---0
Distance of Node 3 = 5
Corresponding path is = 3 <---1<---2<---0
Distance of Node 4 = 6
Corresponding path is = 4 <---1<---2<---0
```

Note:

Unique Random values of structure array variable was not possible due to which we generated cost matrix using random function in serial program **(Serial Program using rand () function -Cost matrix)**.

2.2.2. Parallel code:

The code is implemented using the Master-Slave technique. The root node-0 is calculating the input parameters of the code. The other ranks in the system are used to do the remaining work and the shortest distances of each node from the source is printed in the root node-0.

The MPI commands used and their roles:

```
    //MPI initialization
    MPI_Init(&argc, &argv);
    //size and the rank of the processor
    MPI_Comm_size(MPI_COMM_WORLD, &size_Of_Cluster);
    MPI_Comm_rank(MPI_COMM_WORLD, &process_Rank);
    //broadcast the value to all other processor
    MPI_Bcast(&cost_matrix,N*N,MPI_INT,0,MPI_COMM_WORLD);
    //synchronise the flow of the parameters in all processors
    MPI_Barrier(MPI_COMM_WORLD);
```

Parallel Program using rand () function -Cost matrix using MPI:

(Filename: parallel_code_for_randomly_generated_values_v1.c)

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include "mpi.h"

#define infinity 9999 //fixing the value of infinity and to be used in cost ma
trix
#define max 20 //size of the arrays
/// The below function ia for the assignment of the cost_matrix(the weighted-
undirectional graph)
void cost_matrix_fn(int N,int min , int cost_matrix[max][max])
{
    //getting the vertex number and corresponding weight from the random and a
ssign it to the cost_matrix
    int i,j;
    int w;
    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
        {
            if (i<j)
            {
                w = rand() % min;
                //for undirectional graph
                cost_matrix[i][j]=w;
            }
            else
            {
                cost_matrix[i][j]=0;
            }
        }
    }
    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
        {
            cost_matrix[j][i]=cost_matrix[i][j];
        }
    }

    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
        {
```

```
            if(cost_matrix[i][j]==0)
            {
                cost_matrix[i][j]=infinity;
            }

        }
    }
    //Priting the cost_matrix
    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
        {
            printf("%d \t",cost_matrix[i][j]);
        }
        printf("\n");
    }
}

/// The function for calculation of the dijksthra algorithm
void  dijkstra_algorithm(int N,int cost_matrix[max][max],int shortest_distance
[max],int start, int min, int visited_status[N],int predecor[max])
    /* Applying the shortest path algorithm and getting the
    shortest path from the source node to the each other nodes*/
{
    int position,n_time,d;
    // The nodes unvisited are given a status of 0 and the adjacent node dista
nces from start point is initiallized to the shortest_distance.

     /// the distance matrixs first value should be less than this, a random a
ssumption.
        for(int j=1;j<N;j++)
        {
            if(shortest_distance[j]<min && visited_status[j]==0)
            {
                min=shortest_distance[j]; // The minimum distances is found us
ing a search logic with a<b and it should be unvisited.
                position=j;// The minimums index is stored to have the way to
move forward to the next step and to update the status.
            }
        }
        visited_status[position]=1; // Sinces the node is visited the status i
s updated, so that they are not altered again.
        for(int j=1;j<N;j++) // The minimum value found is updated in the tabl
e/array shortest_distance
        {   d=min + cost_matrix[position][j]; // min=shortest_distance[positio
n]
            if((d<shortest_distance[j]) && visited_status[j]==0)
                {
                    predecor[j] = position,
                    shortest_distance[j]=d;
```

```
                }
        }
}
/// The below function is for printing the shortest path
/// The main function for the start of the dijkstra program
int main(int argc, char** argv)
{
    srand(time(NULL));
    int size_Of_Cluster //size of the processors
        ,process_Rank, //rank of the processor
        min,N, //minimum value for random creation, and size of the array
        source; //start node
    const int root_rank=0;
    int cost_matrix[max][max], //cost_matrix with random values
        distance[max],predecor[max], //distance array
        visited_status[max]; //status of the corresponding node
    //MPI initialization
    MPI_Init(&argc, &argv);
    //size and the rank of the processor
    MPI_Comm_size(MPI_COMM_WORLD, &size_Of_Cluster);
    MPI_Comm_rank(MPI_COMM_WORLD, &process_Rank);
    //getting source, size of array and the minimum value for random creation
and broadcasting to all processors
    if(process_Rank==root_rank)
    {
        printf("Enter the Number of node\n");
        scanf("%d",&N);
    }
    MPI_Bcast(&N, 1 , MPI_INT, root_rank, MPI_COMM_WORLD);
    if(process_Rank==root_rank)
    {
        printf("Enter the max value of the range of the weights for cost matri
x\n");
        scanf("%d",&min);
    }
    MPI_Bcast(&min, 1 , MPI_INT, root_rank, MPI_COMM_WORLD);
    if(process_Rank==root_rank)
    {
        printf("Enter the source node to start the calculation of the shortest
 path\n");
        scanf("%d",&source);
    }
    MPI_Bcast(&source, 1 , MPI_INT, root_rank, MPI_COMM_WORLD);
    //calculating cost matrix in processor rank zero and broadcasting to all p
rocessors
    if(process_Rank==0)
    {
        cost_matrix_fn(N,min ,cost_matrix);
    }
```

```
    MPI_Bcast(&cost_matrix,N*N,MPI_INT,0,MPI_COMM_WORLD);
    MPI_Barrier(MPI_COMM_WORLD);
    //initializing distance, visited status and predecor initial value in all
processors
    for(int i=1;i<N;i++)
    {
        distance[i]=cost_matrix[source][i];
        visited_status[i]=0;
        predecor[i]=source;
    }
    MPI_Bcast(&distance,N,MPI_INT,0,MPI_COMM_WORLD);
    MPI_Bcast(&visited_status,N,MPI_INT,0,MPI_COMM_WORLD);
    MPI_Bcast(&predecor,N,MPI_INT,0,MPI_COMM_WORLD);
    //calculating minimum distance in all processors
    for(int i=0;i<N-2;i++)
    {
        if(process_Rank==i)
        {
            dijkstra_algorithm(N,cost_matrix,distance,source,min,visited_statu
s,predecor);
            MPI_Bcast(&distance,N,MPI_INT,i,MPI_COMM_WORLD);
            MPI_Bcast(&visited_status,N,MPI_INT,i,MPI_COMM_WORLD);
        }
    }
    //printing the minimum distance and the corresponding path
    if(process_Rank==0)
    {
        for(int i=0;i<N;i++)
        {
            if(i!=source)
            {
                printf("\nDistance of Node %d = %d ",i,distance[i]);
                printf("\nCorresponding path is = %d ",i);

                int j=i;
                do
                {
                    j=predecor[j];
                    printf("<---%d",j);
                }while(j!=source);
            }
        }
    printf("End of the program \n");
    }
    MPI_Finalize();
    return 0;
}
```

Note:

No of processors = No of vertices – 2

(No of processor should satisfy the equation e.g if No of processor = 5, then No of vertices to be entered is 7)

3. Results:

1. The results for the given weighted graph given as the example was executed with 2 processor and the results obtained are mentioned below.

   Serial Program for given graph:

   (Filename: serial_code_for_given_example.c)

```
c:\Users\nivas\Desktop\Masters\3 Sem\HPC\Final_Project\HPC>gcc serial_code_for_given_example.c -o serial_code_for_given_example

c:\Users\nivas\Desktop\Masters\3 Sem\HPC\Final_Project\HPC>serial_code_for_given_example

The value of the maximum unknown distance is 9999

The Adjacency matrix representation of the graph

0       4       2       0       0
0       0       3       2       3
0       1       0       4       5
0       0       0       0       0
0       0       0       1       0

Enter the starting node 0

The cost matrix for the given graph is

9999    4       2       9999    9999
9999    9999    3       2       3
9999    1       9999    4       5
9999    9999    9999    9999    9999
9999    9999    9999    1       9999

The shortest path using Dijkstra Algorithm


Distance of Node 1 = 3
Corresponding path is = 1 <---2<---0
Distance of Node 2 = 2
Corresponding path is = 2 <---0
Distance of Node 3 = 5
Corresponding path is = 3 <---1<---2<---0
Distance of Node 4 = 6
Corresponding path is = 4 <---1<---2<---0
```

   Note:

   The above condition is not satisfied for No of processor is 2, so we executed the given example in the serial program.

2. The results of the random graph using MPI with 18 processors.

   Note:

   Eg of Costmatrix Output for Vertices 20:
   9999 17 18 15 10 9999 16 11 12 10 7
   8 11 16 14 7 9 11 7 8 -→ Row 0

OUTPUT:

```
[vs12xosy@node135 pbs.1086900.hpcc1.x8z]$ mpicc -O ../parallel_code_for_randomly_
generated_values_v1.c -o parallel_code_v1
[vs12xosy@node135 pbs.1086900.hpcc1.x8z]$ mpirun -np 18 parallel_code_v1
Enter the Number of node
20
Enter the max value of the range of the weights for cost matrix
20
Enter the source node to start the calculation of the shortest path
0
9999 17 18 15 10 9999 16 11 12 10 7
8 11 16 14 7 9 11 7 8
17 9999 19 9 10 1 9999 5 9 7 2
4 16 9 1 7 16 4 7 12
18 19 9999 15 9999 2 14 9999 5 3 7
12 4 18 11 13 9 9999 15 10
15 9 15 9999 1 12 9999 9999 14 4 1
6 15 17 15 11 1 3 3 9 15
10 10 9999 1 9999 18 15 7 3 18 1
4 16 3 4 7 8 13 8 15 4
9999 1 2 12 18 9999 1 7 4 1 1
3 9999 9 9 9 5 9999 3 9999 16
16 9999 14 9999 15 1 9999 12 15 14 7
14 17 18 9 5 13 5 5 1
11 5 9999 9999 7 7 12 9999 19 5 1
6 15 6 15 11 7 8 3 8 17
12 9 5 14 3 4 15 19 9999 12 1
3 10 15 13 18 19 9999 12 19 15
10 7 3 4 18 1 14 5 12 9999 1
9 16 19 2 1 4 3 12 9
7 2 7 16 14 13 7 16 13 1 9
999 11 19 15 6 10 14 6 5 2
8 4 12 15 16 9999 14 15 10 9 1
1 9999 16 18 8 6 5 13 4 17
11 16 4 17 3 9 17 6 15 16 1
9 16 9999 6 8 16 1 9 5 9
16 9 18 15 4 9 18 15 13 19 1
5 18 6 9999 9999 7 2 4 2 7
14 1 11 11 7 9 9 11 18 2 6
8 8 9999 9999 13 5 6 9999 3
7 7 13 1 8 5 5 7 19 1 1
0 6 16 7 13 9999 9 6 9 14
9 16 9 3 13 9999 13 8 9999 4 1
4 5 1 2 5 9 9999 1 5 4
11 4 9999 3 8 3 5 3 12 3 6
13 9 4 6 6 1 9999 9 3
```

7 7 15 9 15 9999 5 8 19 12 5
4 5 2 9999 9 5 9 9999 2
8 12 10 15 4 16 1 17 15 9 2
17 9 7 3 14 4 3 2 9999

Distance of Node 1 = 9
Corresponding path is = 1 <---10<---0
Distance of Node 2 = 14
Corresponding path is = 2 <---10<---0
Distance of Node 3 = 15
Corresponding path is = 3 <---0
Distance of Node 4 = 10
Corresponding path is = 4 <---0
Distance of Node 5 = 20
Corresponding path is = 5 <---10<---0
Distance of Node 6 = 14
Corresponding path is = 6 <---10<---0
Distance of Node 7 = 11
Corresponding path is = 7 <---0
Distance of Node 8 = 12
Corresponding path is = 8 <---0
Distance of Node 9 = 8
Corresponding path is = 9 <---10<---0
Distance of Node 10 = 7
Corresponding path is = 10 <---0
Distance of Node 11 = 8
Corresponding path is = 11 <---0
Distance of Node 12 = 11
Corresponding path is = 12 <---0
Distance of Node 13 = 16
Corresponding path is = 13 <---0
Distance of Node 14 = 13
Corresponding path is = 14 <---10<---0
Distance of Node 15 = 7
Corresponding path is = 15 <---0
Distance of Node 16 = 9
Corresponding path is = 16 <---0
Distance of Node 17 = 11
Corresponding path is = 17 <---0
Distance of Node 18 = 7
Corresponding path is = 18 <---0
Distance of Node 19 = 8
Corresponding path is = 19 <---0End of the program
The source-0
The node value N-20
The min value-20

3. The below table is about the scaling of the parallel code and serial code with different number of vertices:

No. of Vertices	Serial (seconds)	Parallel (seconds)
10	0.04	0.000575
20	0.12	0.053160
30	0.24	0.072242
40	0.38	0.1016
50	0.56	0.131

4. Weak and strong scaling:

Strong Scaling = $\frac{\mathbf{1}}{s+\frac{p}{N}}$

Weak Scaling = $\boldsymbol{s + p * N}$

s = $\frac{T_s^p}{T^s}$ $(T_s^p = Time\ for\ serial\ part\ of\ the\ parallel\ program,$

$T^s = Total\ time\ of\ the\ serial\ program\ using\ single\ processor)$

p=1-s

N=No of processors

No of Processors	Strong scaling (Amdahl's law)	Weak scaling (Gustafson's law)
5	4.716	4.95
10	8.810	9.865
15	12.39	14.79
20	15.56	19.715
25	18.38	24.64
30	20.90	29.565
35	23.17	34.49
40	25.30	39.415
45	27.13	44.34
50	28.81	49.265