https://drive.google.com/file/d/10IpoTHV4HMhR8LMNzlQRQqnNAx4ppSM_/view?usp=sharing

VIT®
UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
VELLORE ■ CHENNAI
www.vit.ac.in

CSE4001-Parallel and Distributed Computing

Slot: D2+TD2

Parallelization of Shortest Path Finder between two nodes: Floyd-Warshall Algorithm

Venkatasai V 15BCE0797

Jaya Krishna Naidu 15BCE0865

Vinay Aditya 15BCE0829

Project Report

Under the Guidance of

Prof.Manoov.R

School of Computer Science & Engineering

VIT University – India

FALL SEMESTER 2017

CERTIFICATE

This is to certify that the Project work entitled **"Parallelization of Shortest Path Finder between two nodes: Floyd-Warshall Algorithm**"that is being submitted by **Venkatasai, Jaya krishna and Vinay Aditya** in B.Tech for **CSE4001 Parallel and Distributed Computing** is a record of bonafide work done under my supervision.The contents of this Projectwork, in full or in parts, have neither been taken from any other source nor have been submitted for any other course.

Signature of faculty:

PROF. MANOOV R

AKNOWLEDGEMENT

We are thankful to the Department because of whom, we have gained confidence in Innovative Thinking and it also enhanced our professional skills as to become competent in this field.

In performing our project, we had to take the help and guide line of some respected persons, who deserve our greatest gratitude. The completion of this project gives us much Pleasure. We would like to show our gratitude to **Prof.Manoov, SCOPE VITUniversity** for giving us a good guide line for project throughout numerous consultations. We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us in this project.

Thankyou,

15BCE0797 -Venkatasai

15BCE0829-Vinay Aditya

15BCE0865-Jaya Krishna Naidu

Contents

1. Abstract:

The project deals with implementation of Floyd Warshall Algorithm i.e. All Pair Shortest Path. This algorithm is implemented using parallel programming concept for faster solution. Floyd Warshall algorithm has overcome the drawbacks of Dijkstra's and Bellman Ford Algorithm. For parallel programming, this project is implemented using MPI for which C programming is used. The purpose of developing this project is to find the shortest path between all the present nodes in a graph.

This project can be implemented for Airline Systems, Transportation services, Courier Services, Networking etc.

From the results it is observed that parallel algorithm is considerably effective for large graph sizes and MPI implementation is better in terms of performance over serial implementation of Floyd warshall algorithm

2. Introduction:

Finding the shortest path between two objects or all objects in a graph is a common task in solving many day to day and scientific problems. The algorithms for finding shortest path find their application in many fields such as social networks, bioinformatics, aviation, routing protocols, Google maps etc. Shortest path algorithms can be classified into two types: single source shortest paths and all pair shortest paths. There are many different algorithms for finding the all pair shortest paths. Some of them are Floyd-Warshall algorithm and Johnson's algorithm.

All pair shortest path algorithm which is also known as Floyd-Warshall algorithm was developed in 1962 by Robert Floyd. This algorithm follows the methodology of the dynamic programming. The algorithm is used for graph analysis and finds the shortest paths (lengths) between all pair of vertices or nodes in a graph. The graph is a weighted directed graph with negative or positive edges. The algorithm is limited to only returning the shortest path lengths and does not return the actual shortest paths with names of nodes.

3. Literature Review:

The computational complexity of graph algorithms is a function of the number of vertices/edges or both the vertices and edges of a given graph (West 2000). With the increasing number of either vertices or edges, the computational complexity increases. The research community has attempted to devise several efficient algorithms to reduce such computational complexity.

In general two approaches are dominant:

(1) Improving the serial implementations of the algorithms; and

(2) Formulating parallel implementations of the algorithms. In the first approach, researchers have adopted approaches, such as graph partitioning, hierarchical network decomposition, and different data structures, to improve the speed of the Dijkstra algorithm (Gilbert and Zmijewski 1987; Hendrickson and Leland 1995; Romeijin and Smith 1999; Möhringet al. 2005). In the graph partitioning approach, a given graph is divided into multiple pieces of the

same size in such a way that there are few connections between the pieces (Hendrickson and Leland 1995; Möhring et al. 2005).
For example, to perform graph partitioning, Möhring et al. (2005) used different data structures, such as a grid, k-dimensional (k-d) trees, and quadtrees to perform graph partitioning. A speedup, up to 500, was achieved by using a suitable data structure on certain graphs. Unfortunately, the graph partitioning itself is an nondeterministic polynomial-complete problem (West 2000).
Thus, one has to rely on certain heuristics that cannot be guaranteed to work in every situation. Romeijn and Smith (1999) developed a hierarchical network decomposition algorithm to partition a network into multiple pieces, which could be independently solved to yield approximate shortest paths for a large-scale network. Their approach a logðnÞ savings in computation time compared to a traditional implementation of the Dijkstra algorithm, where n represents the number of vertices in a given network.

4. Objective:

The primary goal of this project is to implement the Floyd-Warshall Algorithm in Parallel with the help of MPI Programming to find the shortest path between two nodes.

Floyd Warshall:

Floyd Warshall algorithm is All Pair Shortest Path finder. It is mainly used to overcome the drawbacks of Dijkstra's and Bellman Ford Algorithm. It considers negative weight present in the graph. In Floyd Warshall algorithm every node of the graph is visited and the shortest path is computed.

The Floyd–Warshall algorithm is an example of dynamic programming.

The algorithm is also known as **Floyd's algorithm**, the **Roy–Warshall algorithm**, the **Roy–Floyd algorithm**, or the **WFI algorithm**.

5. Existing Model:

Sequential algorithm

```
for k = 0 to  N-1
        fori = 0 to N-1
            for j = 0 to N-1

                Iij (k+1) = min (Iij(k), Iik(k) + Ikj(k))
            Endfor
        Endfor
Endfor
```

The Floyd-Warshall Sequential algorithm

Sequential pseudo-code of this algorithm is given above requires N^3 comparisons. For each value of k that is the count of inter mediatory nodes between node i and j the algorithm computes the distances between node i and j and for all k nodes between them and compares it with the distance between i and j with no inter mediatory nodes between them. It then considers the minimum distance among the two distances calculated above. This distance is the shortest distance between node i and j. The time complexity of the above algorithm is $\Theta(N^3)$. The space complexity of the algorithm is $\Theta(N^2)$. This algorithm requires the adjacency matrix as the input. Algorithm also incrementally improves an estimate on the shortest path between two nodes, until the path length is minimum.

In this project I have implemented the parallel version of all pair shortest path algorithm in MPI. From the results I found that parallel version gave speedup benefits over sequential one, but these benefits are more observable for large datasets.

6. Proposed system:

Parallel algorithm:

```
for k = 0 to  N-1
        fori = local_i_start to local_i_end
                for j = 0 to N-1
                        Iij (k+1) = min (Iij(k), Iik(k) + Ikj(k))
                Endfor
        Endfor
Endfor
```

The Floyd-Warshall Parallel algorithm

The Floyd-Warshall algorithm compares all possible paths through the graph between each pair of vertices. Consider a graph G (V, E) where V is no. of vertices and E is no. of edges. For computing minimum path between each pair of node Wk. is computed where k ranges from 1 to V. For computing shortest from i to j, W [i, j] = W [i, k] +W [k, j]

INPUT: A graph G (V, E) where V is a set of vertices and E is set of weighted edges between these vertices. A source vertex form V.

OUTPUT: The distance of shortest paths between the source vertex and every vertex in G.

7. Components Used:

Hardware Interfaces:

* One dual-core INTEL(R) core i5-5200U CPU @ 2.6 GHz processor.

*8GB RAM.

*2GB Nvidia 920M Graphics

Software Interfaces:

*Languages Used: C programming, Dynamic programming, MPI

*Linux distro ElementryOS Operating System

8. Code Snippet:

MPI Implementation of Floyd-warshall Algorithm:

```
/* File:      mpi_floyd.c
 * Purpose:   Implement a parallel version of Floyd's algorithm for finding
 *            the least cost path between each pair of vertices in a labelled
 *            digraph.
 *
 * Compile:   mpicc -g -Wall -o mpi_floyd mpi_floyd.c
 * Run:       mpiexec -n <number of processes> ./mpi_floyd
 *
 * Input:     n, the number of vertices
 *            mat, the adjacency matrix
 * Output:    mat, after being updated by floyd so that it contains the
 *            costs of the cheapest paths between all pairs of vertices.
 *
 * Notes:
 * 1.  n, the number of vertices should be evenly divisible by p, the
 *     number of processes.
 * 2.  The entries in the matrix should be nonnegative ints:  0 on the
```

```
 *     diagonal, positive off the diagonal.  Infinity should be indicated
 *     by the constant INFINITY.  (See below.)
 * 3.  The matrix is distributed by block rows.
 */
#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 */

/*--------------------------------------------------------------------
 * Function:  Read_matrix
 * Purpose:   Read in the local_matrix on process 0 and scatter it using a
 *            block row distribution among the processes.
 * In args:   All except local_mat
```

```
 * Out arg:   local_mat
 */
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 */

/*--------------------------------------------------------------------
 * Function:  Print_row
 * Purpose:   Convert a row of the matrix to a string and then print
 *            the string.  Primarily for debugging:  the single string
 *            is less likely to be corrupted when multiple processes
 *            are attempting to print.
 * In args:   All
 */
```

```
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 */

/*--------------------------------------------------------------------
 * Function:  Print_matrix
 * Purpose:   Gather the distributed matrix onto process 0 and print it.
 * In args:   All
 */
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);
```

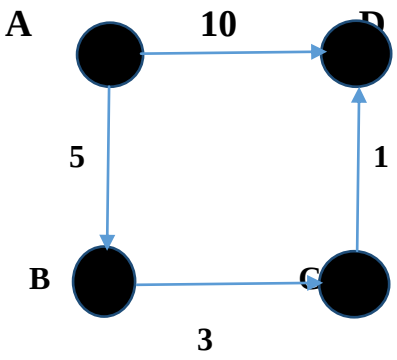
```
      for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
          if (temp_mat[i*n+j] == INFINITY)
            printf("i ");
          else
            printf("%d ", temp_mat[i*n+j]);
        printf("\n");
      }
      free(temp_mat);
    } else {
      MPI_Gather(local_mat, n*n/p, MPI_INT,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

/*--------------------------------------------------------------------
 * Function:    Floyd
 * Purpose:     Implement a distributed version of Floyd's algorithm for
 *              finding the shortest path between all pairs of vertices.
 *              The adjacency matrix is distributed by block rows.
 * In args:     All except local_mat
 * In/out arg:  local_mat:  on input the adjacency matrix.  On output
 *              the matrix of lowests costs between all pairs of
 *              vertices
 */
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 */

/*--------------------------------------------------------------------
 * Function:  Owner
 * Purpose:   Return rank of process that owns global row k
 * In args:   All
 */
int Owner(int k, int p, int n) {
   return k/(n/p);
}  /* Owner */

/*--------------------------------------------------------------------
 * Function:  Copy_row
 * Purpose:   Copy the row with *global* subscript k into row_k
 * In args:   All except row_k
 * Out arg:   row_k
 */
```

```
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 */
```

Input Matrix:

The following Directed graph consists of 4 vertices with respective weights.



9. Result Screenshots:

```
harsha@harsha-Vostro-3558:~/Downloads$ mpicc -g -Wall -o mpi_floyd mpi_floyd.c
harsha@harsha-Vostro-3558:~/Downloads$
harsha@harsha-Vostro-3558:~/Downloads$ mpicc -g -Wall -o mpi_floyd mpi_floyd.c
harsha@harsha-Vostro-3558:~/Downloads$ mpiexec -n 4 ./mpi_floyd
How many vertices?
4
Enter the local_matrix
0
5
1000000
10
0
1000000
0
3
1000000
1000000
1000000
0
1
1000000
1000000
```

```
0
1
1000000
1000000
1000000
0
We got
0 5 i 10
i 0 3 i
i i 0 1
i i i 0

The solution is:
0 5 8 9
i 0 3 4
i i 0 1
i i i 0
```

10. Applications of Floyd Warshall Algorithm:

The Floyd–Warshall algorithm can be used to solve the following problems, among others:

- Shortest paths in directed graphs (Floyd's algorithm).
- Finding a regular expression denoting the regular language accepted by a finite automaton (Kleene's algorithm, a closely related generalization of the Floyd–Warshall algorithm)[11]
- Inversion of real matrices (Gauss–Jordan algorithm
- Optimal routing between two vertices such as distance between two cities.
- Fast computation of Pathfinder networks.
- Widest paths/Maximum bandwidth paths
- Computing canonical form of difference bound matrices (DBMs)

11. Conclusion:

In this project, we developed parallel implementations of shortest path graph algorithm known as Floyd-Warshall, which were used to find the all-pairs shortest path in transportation networks. Both the sequential and parallel implementation is done for the Floyd-Warshall algorithm.The implemented Floyd-Warshall algorithm had theoretical computational complexity, which wasverified experimentally. Here we implemented Floyd-Warshall parallelly with the help of MPI Thus, the system implements Parallelization of shortest path finder algorithm: Floyd-Warshall. This system can be implemented in the areas where the operation is to be performed on large dataset and the computation time should be less hence to reduce the time of computation parallelization can be implemented using GPU.

12. References:

[1].Parallelization of Shortest Path Finder on GPU: Floyd-Warshall by DhananjayKulkarni ,Neha Sharma ,PrithvirajShinde , Vaishali Varma

[2]. Jian Ma ; Sch. of Transp. Eng., Tongji Univ., Shanghai, China ; Ke-Ping Li ; Li-yan Zhang, A Parallel Floyd-Warshall algorithm based on TBB, IEEE.

[3] John D.owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Kruger, Aaron E.Lefohn, Timothy. Purcell,A survey of general-purpose computation on graphics hardware, Computer Graphics Forum 34(March) (2007)80-113.

[4] Olaf Schenk,MatthiasChristen,HelmarBurkhart"J. Parallel Distrib. Comput".

[5] http://www.nvidia.com/object/what-is-gpu-computing.html

[6] Efficient multi GPU algorithm for all pair shortest path.

[7] G. Venkataraman, S. Sahni, and S. Mukhopadhyaya, A blocked all-pairs shortest-paths algorithm, J. Exp. Algorithmics.