Programming Assignment 1 Documentation
CS433
Dijkstra's Algorithm

Name: 王誉锡

Student ID: 5131309040

Name: 章一铭

Student ID: 5140219368

Contents

1 Introduction

Dijkstra's algorithm solves "the single-source shortest path problem" in a weighted, directed graph. In our implementation, input includes a matrix in which mat[v][w] is the weight of the edge from vertex v to vertex w. The algorithm can be divided into two phases: initialization and path-finding. Compared with the serial implementation, the parallel one divides work and assigns almost equal responsibility for different vertices to each process. Thus, prominently dimension of subtasks decreases, and efficiency soars.

2 Implementation of Parallelism

2.1 Initialization

Our implementation prompts the user to input the number of vertices and the graph matrix. Initialization is process 0's job. After that, our major work is to inform other processes of the submatrix that they have to process, and the number of vertices.

To send the number of vertices, we use function MPI_Send() in process 0 and MPI_Recv() in other processes. Following code is our implementation of these functions.

```
for(int i = 1; i < comm_sz; ++i )
  MPI_Send(&n, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
```

To send the submatrix, we first define loc_n, loc_dist, loc_pred and loc_mat as locality of original algorithm variables. We use function MPI_Scatter() in all processes. It is called with

```
MPI_Scatter(mat, loc_n*n, MPI_INT, loc_mat, loc_n*n, MPI_INT, 0,
    MPI_COMM_WORLD);
```

2.2 Parallel Calculation

Computational work is to find u, the vertex whose distance to 0 is minimum, and upgrade corresponding variables. There is also a main loop which traverses loc_n.

Every time each process finds their own u just as the serial algorithm does. Then we have to choose the only u from these candidates which has the minimum value globally. In order to bring into effect, each process declares two two-element arrays of int:

```
int my_min[2], glbl_min[2]
```

We get the global vertex number of each candidate, named as global_u.

```
int global_u = loc_u + my_rank * loc_n;
```

Thus, my_min[1] stores this global_u. My_min[0] stores the distance from 0 to loc_u.

Then we call the function MPI_Allreduce as

```
MPI_Allreduce(my_min, glbl_min, 1, MPI_2INT, MPI_MINLOC,
        MPI_COMM_WORLD);
```

Now glbl_min[0] has the current global minimum distance while glbl_min[1] has the global number of this chosen vertex.

We only set known[loc_u] as 1 only if the global u is in that process.

2.3 Paths upgrading

We look at all vertices assigned to the current process that are not yet elements of known. For each such vertex, we calculate the possibly new distance as

```
new_dist = glbl_min[0] + loc_mat[glbl_min[1] + v*N];
```

Only if this new distance is smaller, we upgrade the path as

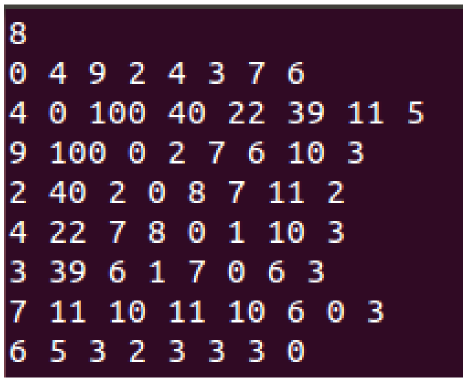
```
loc_dist[v] = new_dist;
loc_pred[v] = glbl_min[1];
```

Finally, we use following function to integrate the result for output.

```
MPI_Gather(loc_dist, loc_n, MPI_INT, dist, loc_n, MPI_INT, 0,
        MPI_COMM_WORLD);  // Form the complete dist
MPI_Gather(loc_pred, loc_n, MPI_INT, pred, loc_n, MPI_INT, 0,
        MPI_COMM_WORLD);  // Form the complete pred
```

3 Test Result

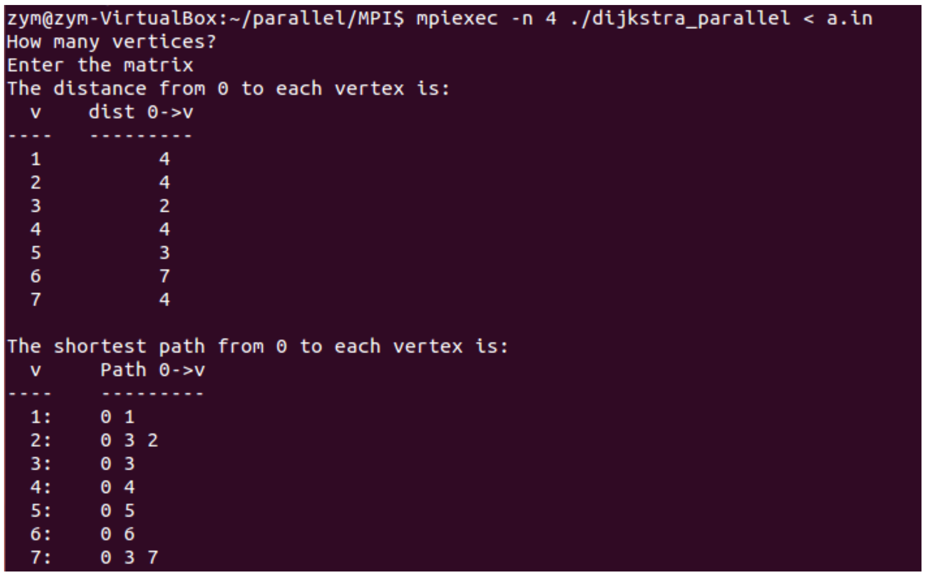
We wrote a file **a.in** as input. Figure 1 shows its content.



```
8
0 4 9 2 4 3 7 6
4 0 100 40 22 39 11 5
9 100 0 2 7 6 10 3
2 40 2 0 8 7 11 2
4 22 7 8 0 1 10 3
3 39 6 1 7 0 6 3
7 11 10 11 10 6 0 3
6 5 3 2 3 3 3 0
```

Figure 1: Test Data

Figure 2 is the output of our program. Obviously our program works perfect.



```
zym@zym-VirtualBox:~/parallel/MPI$ mpiexec -n 4 ./dijkstra_parallel < a.in
How many vertices?
Enter the matrix
The distance from 0 to each vertex is:
  v    dist 0->v
----   ---------
  1          4
  2          4
  3          2
  4          4
  5          3
  6          7
  7          4

The shortest path from 0 to each vertex is:
  v     Path 0->v
----    ---------
  1:    0 1
  2:    0 3 2
  3:    0 3
  4:    0 4
  5:    0 5
  6:    0 6
  7:    0 3 7
```

Figure 2: Test Result

4 Comparison with Serial One

The serial program finds u and updates paths in one for loop. Single CPU computing has some limits considering the cache space and performance. However, the parallel one divides the responsibility into several parts, including smaller local valuables and lighter computation. So that we can solve larger problems or accelerate the same problem.