SHANGHAI JIAO TONG UNIVERSITY, SCHOOL OF ELECTRONIC INFORMATION AND ELECTRICAL ENGINEERING

Parallel Implementation Of Dijkstra's Algorithm Using MPI Library

Zhao Zijian, Zhang Haoxiang

October 19, 2016

1 THE SINGLE-SOURCE SHORTEST PATH PROBLEM

The single-source shortest path problem is defined as follows:

- Given : A directed graph, G = (V, E). Cardinalities |V| = n, |E| = m.
- S(source) : distinguished vertex of the graph. (0 for this report)
- W : weight of each edge, typically, the distance between the two vertexes.
- Single source shortest path: The single source shortest path (SSSP) problem is that of computing, for a given source vertex s and a destination vertex t, the weight of a path that obtains the minimum weight among all the possible paths.

2 DIJKSTRA'S ALGORITHM

2.1 INTRODUCTION

The Dijkstra's algorithm uses four main data structures:

1. mat[v][w] : adjacency matrix of the graph recording the weight of edges;
2. dist[v] : current estimate of the distance from source to v;
3. pred[v] : the predecessor of vertex v on the current path 0->v;
4. known[v] : whether the shortest path from source to v is known.

Our vertices are 0, 1, 2, ... , n-1 and vertex 0 is the source. The algorithm is divided into two phases: initialization and iterated path-finding. The core idea is to find the vertex of shortest distance from source among the unknown vertices and update distances and paths of all the vertices connected to it each step until all vertices are known.

2.2 Serial Implementation

Create a list $list[V]$
Given a source vertex s
** O(V) **
While (there exists a vertex that is not in the $list[V]$)
    ** O(V) **
    FOR (all the vertices outside the $list[V]$)
        Calculate the distance from nonmember vertex to s through the $list[V]$
    END
    ** O(V) **
    Select the vertex with the shortest path and add it to the $list[V]$
END

The Running time is $O(V^2)$.

2.3 Parallel Implementation

Create a list $list[V]$
Given a source vertex s
Use $MPI\_Scatter$ to scatter the weight matrix to different processes.
Each process handles a subgroup of V/P vertices
** O(V) **
While (there exists a vertex that is not in the $list[V]$)
    ** Each processor work in parallel O(V/P) **
    FOR (vertices in my subgroup but outside the $list[V]$)
        Calculate the distance from non-member vertex to s through the $list[V]$
        Select the vertex with the shortest path as the local closest vertex
    END
    ** The $MPI\_AllReduce$ takes log(P) **
    Use $MPI\_AllReduce$ to find the global closest vertex among all the local closest vertices from each process and add it to the $list[V]$.
END
Use MPI_Gather to gather the distance and path vector intto one processes.

The running time is $O(\frac{V^2}{P} + V\log(P))$.
But the subtle point here is that the data scatter and gather procedure takes some time that we can't ignore. And it proves to be the primary time overhead when the data is not big enough.

3 EXPERIMENT

The following is the testing results on some samples:

$comm\_sz$	Order of Matrix									
	100	300	400	500	800	1000	2000	3000	4000	5000
1	0.000175	0.001198	0.0018	0.0029	0.0066	0.0093	0.0368	0.0807	0.157	0.236
2	0.000161	0.000800	0.0011	0.0016	0.0032	0.0046	0.0207	0.0448	0.089	0.138
4	0.000350	0.000990	0.0012	0.0016	0.0033	0.0048	0.0177	0.0372	0.071	0.111

Table 3.1: Elapsed of Parallel Dijkstra's Algorithm

$comm\_sz$	Order of Matrix									
	100	300	400	500	800	1000	2000	3000	4000	5000
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	1.08	1.50	1.63	1.81	2.06	2.02	1.78	1.80	1.76	1.71
4	0.50	1.33	1.50	1.81	2.00	1.94	2.08	2.17	2.21	2.13

Table 3.2: Speedups of Parallel Dijkstra's Algorithm

Note: The elapsed time in this table doesn't include the $MPI\_Scatter$ and MPI_Gather overhead. We just want to justify the speedup of parallel programs according to the time complexity analysied above.
And we can summarize some meaningful conclusions from the table:

1. The parallel program does speed up the process. And as the data get bigger and bigger, the speedup is much greater.

2. The speedup ratio is not linear. This is due to the overhead arising from the communications between the processes.

3. There are some samples getting super-linear speedup in this table. This is not possible in theory. And after multiple experiments, it turns out that the result is unstable. So it may be something about the operating system affair. I leave it as a reminder of this problem.

4. For this experiment, the data is too small. According to my experiment, if the elapsed time includes the $MPI\_Scatter$ and MPI_Gather overhead, then the overhead will be dominated by these two functions. And it turns out to be that the elapsed time get longer and longer as the $comm\_sz$ get bigger and bigger.

Appendix: Code Details

Input: A file contains the order of matrix and matrix in the following format.
2
0 1
2 0
Output: Two files contains the distance and path information.
"*paral_dist.txt*"
v dist 0->v
- - - - - - - - - - - - - - - -
1 1

"*paral_path.txt*"
v Path 0->v
- - - - - - - - - - - - - - - -
1: 0 1
Complie: mpicc -g -Wall -o paral.out *paral_dijkstra.c*
Run: mpiexec -n <p> ./paral.out filename