Parallel Floyd's Algorithm for solving All-Pairs Shortest Path Problem

Zhao Chen zhaoc2@andrew.cmu.edu
Jianlin Du jianlind@andrew.cmu.edu

URL:
https://deltadu.github.io/Parallel-Floyds/

Summary:
We are going to parallelize Floyd Algorithm to solve all-pairs shortest path problem with OpenMP & MPI and compare the performance.

Background:

Shortest Path Problem is an important part in graph theory. In contrast to single source shortest paths problem, for example Dijkstra's algorithm, all-pairs shortest path problems calculate the shortest path between all pairs of nodes in the graph. A sample pseudocode is as follows.

```
1    func Floyd_All_Pairs_SP(A) {
2        D^(0) = A;
3        for k := 1 to n do
4            for i := 1 to n do
5                for j := 1 to n do
6                    d_{i,j}^(k) := min(d_{i,j}^(k-1), d_{i,k}^(k-1) + d_{k,j}^(k-1))
7    }
```

As you can see in the pseudocode, the sequential solution is a straightforward O(V^3) time complexity algorithm(V is the number of vertices), looping through all the vertices. This leaves some possibility for parallelization. We can explore the loops and try to divide the work into processors.

Challenges:
One challenge is associated with the nature of shortest path problem. When we visit node after node to find the shortest path, the steps actually have strong sequential dependency, which means the parallelization is kind of limited. However, one of the reasons we are interested in the all pairs shortest path algorithm instead of single source shortest paths problem is that we hope

to explore more parallelization in the process of calculating different pairs and to find potential data reuse and redundancy to accelerate the algorithm.
Another challenge is a common issue in parallelizing graph problems. If we divide the vertices into several processors, when different processors update the information of vertices, the information needs to be broadcast to other processors because they are sharing the same graph. This may require a lot of communications especially using message passing models.

Resources:
For machines, we will start with GHC machines with OpenMP and MPI supported.
We will explore the sequential pseudocode of Floyd algorithm as shown in the Background section and apply OpenMP and MPI libraries to the program.

Goals and Deliverables:
Plan to Achieve: Complete the parallel version of Floyd algorithm in OpenMp and MPI and compare the performance. The parallel program is supposed to achieve significant speedup compared to sequential implementation. We will strive to minimize the time spent on synchronization and communication. As the time complexity of sequential implementation is O(n^3), the parallel version is supposed to cost O(n^3 / #processors) + O(time on sync).

Hope to Achieve:
If goes well, we will try to combine different parallel models to further boost the performance, or explore a variation of Floyd algorithm particularly for parallel implementation.
If goes slow or the speedup does meet our expectations, we will analyze the issue and present insights into limitations of our implementation(algorithms, graph representations, etc).

At the demo, the speedup graph of our multiple parallel algorithms compared to sequential algorithm will be shown. We will also present the speedup graphs with different graph sizes to show the scalability of our parallel implementation. We can also provide an interactive demo where a graph could be specified by the viewer and result generated accordingly.

Platform Choice:
C++ with OpenMP and MPI interfaces will be used. Regarding computing devices, we will use ghc machines with eight cores for starters, and test OpenMP and MPI version of the algorithm separately on them. Furthermore, since we may implement a version of code that combines the power of OpenMP and MPI, a machine with multiple nodes of multiple cores may be rented from cloud services like AWS EC2 or Blacklight.

Schedule:
10/28:
Meet with the professor to select topic and write project proposal
Set up website and project git repo
Find and read reference paper and resources

11/4:
Implement sequential version of the algorithm, which will serve as a reference for future parallel version

11/11:
Implement OpenMP version of the algorithm and benchmark it
Write checkpoint report

11/18 project checkpoint:
Implement MPI version of the algorithm and benchmark it

11/25:
Try to combine the OpenMP and MPI to achieve a faster performance on multi nodes multi cores cluster, and benchmark it

12/2:
Write final report and make poster

12/9 final report due
12/10 poster presentation