CS 378 – Project #3

Due Monday, Nov 10, Midnight

You may, but don't have to, work in groups up to 3. Turn in a single project for all members of the group with all names listed.

1 Parallel SSSP

Implement a parallel SSSP. You should implement the asynchronous level-correcting algorithm. Multiple threads could try to update the same node simultaniously, so you do need some form of synchronization.

Use the CSR storage format for the graph.

Implement and test the four schedulers below. Obviously you will need some synchronization.

- Global LIFO
- Global FIFO
- Global priority queue
- Priority queue per thread, work is push locally, threads steal when out of work

You may use either pthreads or openmp or c++11 threads.

Validate that your implementation produces the correct answer.

You should at least use (and report times for) the Dimacs full USA road network (travel time) graph: `http://www.dis.uniroma1.it/challenge9/download.shtml` You may use other graphs if you wish.

2 Notes

Syncronization at the node can be done with a mutex or the update can be done with a compare and swap.

If you are using C++, by all means use standard containers for the scheduler. This isn't an excercise in writting a priority queue (or equivalent).

Validation can be done by inspecting every node in the graph and ensuring that each neighbor is labeled no further than the node's label plus the edge length.

You will need to support edge data in your graph representation, unlike in homeworks.

Example code, modified from Wikipedia:

```
function Dijkstra(Graph, source):
      for each vertex v in Graph:           // Initializations
          dist[v]  := infinity ;            // Unknown distance function from
                                            // source to v
      end for

      dist[source]  := 0 ;                  // Distance from source to source
      Q.push(source);

      while Q is not empty:                 // The main loop
```

```
        u := Q.pop();
        for each neighbor v of u:      // where v has not yet been
            alt := dist[u] + dist_between(u, v) ;
            if alt < dist[v]:          // Relax (u,v,a)
                dist[v]  := alt ;
                Q.push(v);
            end if
        end for
    end while
    return dist;
endfunction
```

here dist is the data on the nodes, dist_between is the data on the edges. Q is the scheduler (in a sequential dijkstra implementation, it is simply a priority queue).

3 Deliverables

Submit your code and include in your writeup performance. Plot strong scaling for each scheduler. Run on stampede.

4 Turning in

Turn in by email an archive (.zip or .tgz) to the TA.

5 Bonus

Implement a better scheduler. The writup whould explain the scheduler, why it is better, and back this up with numbers.