Boruvka's Algorithm in Parallel

Dillon VanBuskirk

June 11, 2016

Contents

1 Introduction

Boruvka's Algorithm is an algorithm that is used to find the minimum spanning tree of a weighted graph. For my assignment, I began with reading the weighted graph from a text file. The graph was stored in a vector of sets which contained a struct for each Edge. These edges are stored with an integer weight and two vertices that are defined as integers. The vertices will be referenced in a vector of structs of vertices that stored with the properties of id, rank, and parent which are all integers. This way, the vector at index i will refer to the vertex whose id is equal to i. Boruvka's algorithm will be parallelized using OpenMP which sits on top of C++ and allows for easy multithreading. After finding the minimum spanning tree, the weight and the list of edges contained will be written to another file. The edges will be written in lexicographical order. I have parallelized the code in order to prove that Boruvka's algorithm is contained within Nick's Class, NC. This coding assignment took me about 19 hours.

2 Pseudocode

An outline of my iteration of Boruvka's algorithm in pseudocode is below. I apologize for the look of this pseudocode. I spent a lot of time writing the algorithm in algorithmic package and it just kept failing compilation and it's error messages are awful. This is using packageprogram and it was a breeze to use. It is more mathematical than pseudo though, hence all the camel case. Each od and fi are the ending of do's (while and for) and if's. See below:

```
begin
  readFile
  vectorOfSetsOfEdges
  vectorOfVertices
  vectorOfLightweightEdges
  setOfEdgesInMST
  for i := 0 to numV step 1 do
     MakeSetOnVertex.i
     addLightweightEdgeOfVertex.i od
  while numComponents < 1 ∧ numLightweightEdges do
        for i := 0 to numComponents step 1 do
           getLightweightEdge
           if (lightweightEdgeNotOutOfComponent)
             then
                  updateLightweightEdgeForBothVerticesOutOfThatEdge
              else addittomst fi
        od
        for i := 0 to edgeCountInMST step 1 do
           if (parentsOutOfEdgeSourceAndDestinationAreNotEqual)
             then
                  unionTheVertices
                  decrementNumComponents
                  if (vertexSrcOrDestHasNoMoreEdges)
                    then decrementNumLightweightEdges
                     else updateLightweightEdgeForBothVerticesOfThatEdge fi
           fi
        od
  od
```

3 Testing

Testing was performed with multiple inputs at interval stages of coding. By hand, I typed and digitally drew graphs and solved for their minimum spanning tree using Boruvka's algorithm. After understanding each of my inputs, I tested them to see if I would get the desired results. For all five of the graphs I designed, the actual output was equivalent to the expected output. The algorithm loops over the components of vertices in order to find lightweight edges that connect to another component. A component of vertices in this case is a collection of vertices which have a lightweight edge from one vertex to another vertex. The two vertices that share a lightweight edge between them will be in the same component. This code was tested on two of my machines, with and without OpenMP, and successfully completed. I was not able to run the code on Razor, the HPC.

4 Efficiency

When Boruvka's algorithm is serially coded, it has a runtime of $O(ElogV)$ where E is the number of edges and V is the number of vertices. Boruvka's algorithm is capable of being parallelized where it's complexity becomes $O(log^i n)$ where n is the total value of inputs - in our case, E+V. This can be achieved on a parallel computer with a polynomial number of processors, c. The i is the position in the NC hierarchy at which the algorithm is able to complete. For this to be the case, the number of processors must be equal to n.

$$T(n) = O(log^2 n)$$

The algorithm is in class NC if there exists an integer such that the runtime is bounded by $log^i n/p$.