Title: Parallel Spectral Clustering on Undirected Graphs

Noor Mostafa, Rohan Shenoy

URL: https://parallelgraphs.github.io.

Summary:

We are going to be clustering nodes in a graph by projecting them into a lower dimensional space using spectral decomposition and then applying a K means++ clustering algorithm; this overall strategy is known as Spectral Clustering. We will be performing a detailed performance analysis with implementation using OpenMP (shared address space) and MPI as well as static versus dynamic assignment of nodes.

Background:

Spectral clustering is a data partitioning algorithm in spectral graph theory that's main goal is to identify some number of clusters of nodes each with similar or nearby values. Spectral clustering has many useful applications in areas such as image segmentation, educational data mining, entity resolution, speech separation, spectral clustering of protein sequences, text image segmentation, etc. Spectral Clustering is a version of a general clustering algorithm that uses the connectivity between data points to compute the optimal clusters. It uses eigenvalues and eigenvectors of the Laplacian matrix to forecast the data into lower dimensions space to cluster the data points. The Laplacian matrix is used because it conceptually captures the connectivity and structure of the graph. Specifically, it represents the relationship between nodes based on the presence of edges between them.

The K means++ clustering algorithm is a version of the K means clustering algorithm which is one of the most well known unsupervised machine learning algorithms for clustering. The algorithm iteratively adjusts the classification of the data points to minimize the sum of the squared distances between each data point and the centroid of its classification. The K++ means algorithm differs in that during the initial iteration, rather than randomly assigning k (user-defined) data points to be the initial centroids, it maximizes the distance between centroids to prevent suboptimal solutions.

Because the K means++ clustering algorithm is iterative and is usually computed on a large number of points, parallelizing the K means++ algorithm would likely improve the computation time of spectral clustering as this seems like the main bottleneck compared to spectral decomposition which is just matrix operations. Hence, we are planning to implement both a shared address space model using OpenMP and message passing model using MPI to

investigate the communication overhead and dependencies involved with parallelizing the model across processors. We also plan on using static and dynamic assignments with different task granularities to assign data points (originally the vertices of the graph) to different processors when computing classification.

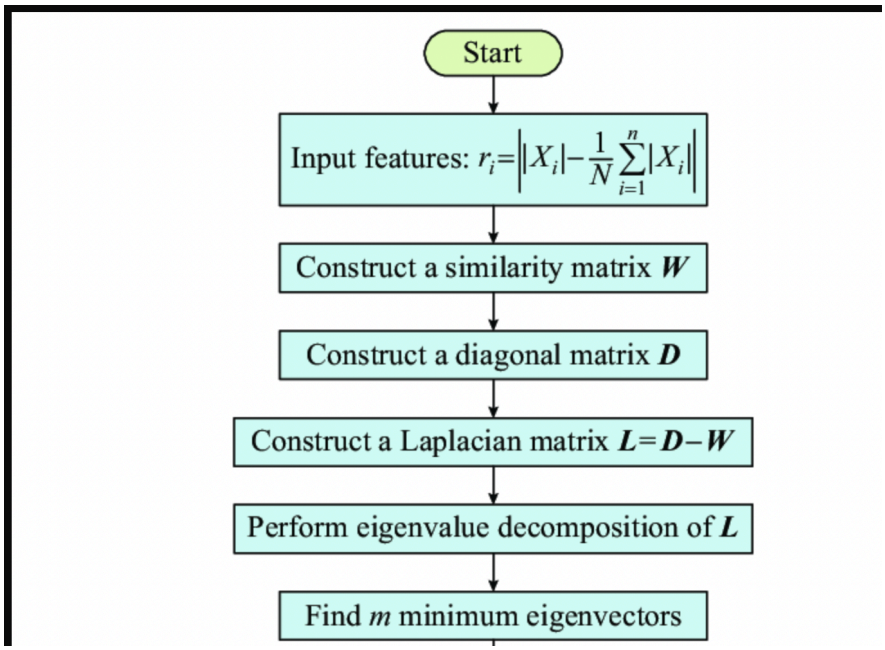

Figure 1: Flow Chart for Spectral Decomposition, transitioning from adjacency (similarity) matrix representation of original graph to lower dimensional space basis vectors

The Challenge:

The first challenge comes from implementing the algorithm sequentially as we have to first implement the spatial decomposition, deciding what value of m eigenvectors we need in order to preserve the data while not making the workspace too complicated for computation during the K means ++ stage. We have to decide on optimal hyperparameters for iterations and number of clusters to ensure there isn't overfitting in the model. Lastly, we need to figure out what version of "distance" we want to use in our K-means ++ algorithm (euclidian, manhattan, etc.) that will lead to optimal results, as at higher dimensions this value may not be so straightforward.

In terms of challenges implementing parallelism, because we plan on using a shared address space and message passing, each has its own challenges to ensure correctness and efficiency. For the message passing model, we plan on dividing up the number of vertices (either statically or dynamically using a distributed work queue) and sending it to each processor. The challenges will be communication of the centroids across processors so that

each processor can classify their points independently, and also finding a way to calculate the true updated centroid for each cluster after the reclassifications. Another challenge will be ensuring that data points (nodes) on different processors that are close to each other (similar coordinates) end up getting classified together almost trivially which can be tricky as one processor cannot singly classify them together, and rather the communication consistently must ensure this. For the shared address space model, the challenges will be atomically updating the centroid locations in the lower dimension space and synchronizing threads to ensure all have been classified before moving on to the next iteration.

The workload for each processor is the number of points (vertices from the graph) that it needs to classify based on finding the closest centroid and classifying it accordingly. We plan on transverse through the number of centroids iteratively which can cause workload imbalance for large values of cluster number; potentially we can optimize this to lower the imbalance. Each point will be represented by a number of coordinates depending on the dimension space. There is locality between points that are assigned to the same processor that are in similar positions, this should allow for quick access to neighboring nodes upon classification.
Resources:

Goals and Deliverables:

PLAN TO ACHIEVE

Implement a successful sequential version of spectral clustering algorithm to ensure correctness, an idea of hyperparameters to tune, and a benchmark to test again the parallel implementation

Implement a shared address parallel implementation that potentially uses locks and/or atomic operations such that significant speedup is achieved and the overhead with synchronization and potential stalling doesn't severely limit performance

Implement a message passing model with the parallel implementation that has a high computation to communication ratio to allow for sufficient potential speedup.

Investigate different types of task assignment and potentially locks on shared address space and compare speedup, cache misses, stalling time, sync time,

HOPE TO ACHIEVE

Use our code, knowledge, to implement parallelism to hierarchical graph clustering which is basically nested clusters such that at each level down to single nodes there are clusters, usually of two elements. The resulting structure is a dendrogram.

Platform Choice:

We plan on using the GHC machine primarily to run our parallelized code, since we are utilizing OpenMP and MPI via C++. We may extend to PSC for scalability testing.

Schedule:

April 1st to April 5th: Write the sequential implementation of our algorithm, finalize plan and logistics for running in parallel (there will probably be many fined tuned and rethough ideas), write pseudocode for parallel code, hypothesize performance, bottlenecks, optimization techniques.

April 8th to April 12th: Write and finalize the parallelized version for shared address space, implement any optimization techniques, begin looking at performance, fine tuning hyperparameters, investigating different lock mechanisms, assignments

April 15th to April 19th: Write and finalize the parallelized version for the message passing model, implement any optimization techniques, begin looking at performance, fine tuning hyperparameters, investigating different lock mechanisms, assignments. Compare the results with shared address space, discussion and hypothesis.

April 22th to April 26th: Begin implementing hierarchical graph clustering algorithm, plan to finish debugging and optimizing by end of week if not beginning of following week. Do similar analysis on it comparing it to K means in our original algorithm

April 29th to May 3rd: Buffer time for anything that needs to be finalized in terms of code, gathering results, analysis, etc. Put together poster presentations and plan out demonstrations and scripts.