15-418 Project Proposal: Parallel Low Diameter Decompositions in Graphs

Jacob Imola (jimola), Sidhanth Mohanty (smohant1)

April 26, 2017

1 Work completed so far

- We figured that a canonical graph to test our algorithm is n cliques on n vertices, where the cliques are arranged in a path and connected cliques only have a single edge going between them. A low diameter decomposition on this graph ends up being the decomposition where cliques are clusters.
- Coded up sequential algorithm and verified it's correctness on the graph mentioned above. This algorithm is our baseline, and all speedups of parallel algorithms would be measured against this sequential baseline.
- Started work on the parallel Miller-Peng-Xu algorithm and have made significant progress on its implementation.

2 Revised Schedule

- April 26-29: Parallelize the sequential algorithm. Add timing framework.
- April 30-May 3: Finish implementing Miller-Peng-Xu.
- May 3-6: Make many test cases, big and small. Test both algorithms rigorously on these graphs. Optimize both as much as we can. Produce speedup plots.
- May 7-10: Add interactive demo to our website.
- May 10-12: Try implementing low-stretch spanning trees!

3 Goals and Deliverables

We are on track on our goals and deliverables, and our goals are largely the same as what we included in our proposal. The "nice-to-have" we proposed was a parallel SDD solver that relied on our parallel LDD implementation: as of now, meeting this is somewhat unrealistic.

4 Demo Day

By the end of our project, we plan to show graphs of the speedups we attained as well as a demo to show what our algorithm does in a visually-appealing way. Our demo will show two

copies of the same input graph and will show the clusters that the two algorithms produce as they are formed. The parallel version will have many clusters being formed at once.

5 Concerns

Since we haven't parallelized our code yet; there could be a lot of troubles optimizing our parallel code. Another concern we have is that the algorithm is highly span efficient, which in theory requires a large number of processors and we aren't certain if OpenMP's parallelism is enough to observe great speedups.