Parallel Algorithms for Greedy Graph Coloring - Checkpoint

Sarah Pethani (spethani) and Jeff Tan (jefftan)

April 26, 2021

1 Summary

We are going to implement different parallel algorithms for greedy graph coloring in OpenMP and possibly CUDA. We will analyze/compare their performance, speedup, and solution quality (e.g. does one algorithm use less colors than another).

2 Progress

Currently, we have implemented a baseline greedy sequential algorithm for graph coloring, as well as the Jones-Plassman (JP) and Gebremedhin-Manne (GM) algorithms which run in parallel using OpenMP. These algorithms produce valid colorings on sparse graphs, dense graphs, and other randomly generated graphs, and we are currently working on speeding them up. We have tested these algorithms on a variety of graphs to verify correctness.

Specifically, to test our graph coloring algorithms, we have written a script to generate random graphs with an arbitrary number of vertices according to the Erdos-Renyi model, where each edge is included with some fixed probability p, independently of all other edges. Depending on the value of p, Erdos-Renyi is able to produce dense graphs, fully connected sparse graphs, as well as forests with many separate connected components. In addition to using Erdos-Renyi, we may also consider testing our algorithms on existing graph algorithm benchmarks.

For input to the algorithms, we save each graph as a file where the first line contains the total number of vertices, and the remaining lines each encode an edge in the graph between two vertices. The vertices are 0-indexed, where no vertex may be higher than the number of vertices - 1. All of the algorithms rely on a graph class which we created, which parses the input file and creates an adjacency list representation of the graph in order to provide constant-time neighbor access.

3 Goals and Deliverables

We currently have correct implementations of the JP and GM algorithms in OpenMP, and our main goal is to optimize their performance to achieve good speedup on Gates and Latedays. Our progress is in line with the schedule that we provided in the proposal, and as such, we may not have

time to achieve our stretch goal of implementing these graph algorithms in CUDA, as implementing graph algorithms on GPUs is significantly more challenging.

Our planned schedule for the remaining weeks is as follows:

- Thu 04/29
  - Sarah: Profile JP implementation and try to speed up parallel implementation; try to eliminate memory contention
  - Jeff: Profile GM implementation and try to speed up parallel implementation
- Mon 05/03
  - Sarah: Try different ways of parallelizing work, in addition to parallelizing the loops suggested in the pseudocode for JP algorithm
  - Jeff: Try different ways of parallelizing work, in addition to parallelizing the loops suggested in the pseudocode for GM algorithm
- Thu 05/06
  - Sarah: Look into memory access pattern of JP to try and eliminate false sharing and improve locality
  - Jeff: Look into alternative graph representations that improve the memory access pattern of GM, and try to improve locality on large graphs
  - Both: Create a representative set of test graphs (sparse, dense, fully connected, not connected, etc.) to test speedup and execution time
- Mon 05/10
  - Sarah: Generate final performance data for JP algorithm v. baseline sequential Jeff: Generate final performance data for GM algorithm v. baseline sequential
  - Both: Start preparing the project report and presentation
- Thu 05/13
  - Both: Finalize and submit project report and presentation

4 Plan to Present

We plan to present the performance, speedup, and solution quality of both parallel algorithms on varying types of graphs, as well as qualitative conclusions about the tradeoffs between the algorithms. To determine these metrics, we plan to generate a representative set of benchmarks to test our algorithms on, and we will present charts and tables illustrating the speedup, execution time, and number of colors used for each algorithm.

5 Preliminary Results

```
spethani@ghc58:~/private/15418/parallel-graph-coloring$ time ./jp -p 1 graphs/exbig

real    0m21.630s
user    0m21.473s
sys     0m0.155s
spethani@ghc58:~/private/15418/parallel-graph-coloring$ time ./jp -p 2 graphs/exbig

real    4m27.936s
user    5m14.599s
sys     3m25.250s
spethani@ghc58:~/private/15418/parallel-graph-coloring$ time ./jp -p 8 graphs/exbig

real    1m29.401s
user    3m25.311s
sys     6m4.498s
```

For JP, the algorithm becomes much slower when introducing parallelism/using OpenMP. This can be seen from the jump in time from using 1 processor to using 2 processors instead. There is speedup when increasing the number of processors when comparing it to the 2 processor time, but the execution time is laughable compared to the sequential execution. This will have to be looked into further, to understand why introducing OpenMP creates such a large slowdown. These preliminary results are on a dense graph with 10k vertices.

```
jefftan@ghc71:~/418/project$ ./gm -p 1 graphs/ex10k
Time: 0.80657ms
jefftan@ghc71:~/418/project$ ./gm -p 2 graphs/ex10k
Time: 0.554685ms
jefftan@ghc71:~/418/project$ ./gm -p 4 graphs/ex10k
Time: 0.610954ms
jefftan@ghc71:~/418/project$ ./gm -p 8 graphs/ex10k
Time: 0.870748ms
jefftan@ghc71:~/418/project$ ./gm -p 1 graphs/ex100k
Time: 20.149ms
jefftan@ghc71:~/418/project$ ./gm -p 2 graphs/ex100k
Time: 11.6649ms
jefftan@ghc71:~/418/project$ ./gm -p 4 graphs/ex100k
Time: 6.8387ms
jefftan@ghc71:~/418/project$ ./gm -p 8 graphs/ex100k
Time: 4.85723ms
jefftan@ghc71:~/418/project$ ./gm -p 1 graphs/dense10k
Time: 251.656ms
jefftan@ghc71:~/418/project$ ./gm -p 2 graphs/dense10k
Time: 248.681ms
jefftan@ghc71:~/418/project$ ./gm -p 4 graphs/dense10k
Time: 251.384ms
jefftan@ghc71:~/418/project$ ./gm -p 8 graphs/dense10k
Time: 238.996ms
```

For GM, our speedup differs greatly based on the type of input graph we are providing. For small graphs like ex10k (a sparse graph with 10k vertices), using more processors makes the runtime slightly longer, likely because the overhead of introducing parallelism is not worth the very slight benefits that parallelism would provide. For medium-sized graphs like ex100k (a sparse graph with 100k vertices), using more processors provides some speedup. However, for very large graphs like dense10k (a fully connected graph with 10k vertices), using more processors yields no speedup over a single processor, likely because the graphs are too big to fit into memory at this point, so the main bottleneck of the code is memory access. We plan to perform additional experiments to determine how our code performs on different types of graphs.

6 Current Challenges

Currently, our parallel implementations of the JP and GM algorithms do not perform faster than the sequential baseline when parallelized.

In the JP algorithm, this could be due to the critical section around the addition of vertices to the independent set which we are coloring, but it could also be a result of the graphs we are testing on not being large enough. This is a plausible issue, as the loops which are parallelized over all vertices do not show significant speedup over their sequential counterparts, which will most likely change as we increase the number of vertices significantly.

In the GM algorithm, this could be due to our parallel work distribution method. Currently, within each iteration, we determine which colors are allowed for each vertex in parallel, and we update the remaining worklist in parallel. However, this results in a lot of synchronization between threads, as the algorithm constantly has to perform in parallel and synchronize back together. The GM algorithm also performs poorly when operating on large graphs because it keeps separate data structures for each thread, reducing locality by making the working set too large to fit in cache. We hope to improve these issues by reconsidering our work distribution between threads and possibly adjusting our graph representation to better provide spatial locality.

If we still encounter issues with the parallelization of these algorithms, we will reach out to our TA mentor.