Parallelizing A Linear Programming Solver using OpenMP
A 15418 Project Proposal By:

[Raghav Gupta, Charlie Phillips]
[raghavgu, cgphilli]
[15-418]

Our project website is currently a readme in a public Git repository https://github.com/raggs1561/15418-ProjectPublic

---

Summary:

We plan on implementing a parallelized version of the Simplex Linear Programming solver using OpenMP on the GHC Clusters and PSC supercomputers. We want to show if parallelizing this algorithm will speed up computation compared to a sequential version. If any speedup exists, we want to determine how it varies depending on the number of cores/threads and how running it on PSC changes performance over the running it on the GHC clusters. We plan on using graphs to visually convey our findings.

Background:

Linear Programs are a way to represent multiple algorithms, such as maximum flow or scheduling. Linear programs can be used to represent optimization problems and are an important topic in the field of operations research and logistics. As a result, an efficient linear programming solver is highly useful for this field. There exist multiple open-source solvers (e.g. GNU Linear Programming Kit, HiGHS) and closed-source solvers from companies such as Gurobi.

A linear program is specified with inputs:

- n real-valued variables $x_1, \ldots, x_n$
- A linear objective function using these variables as inputs
- m linear inequalities (not strict) in these variables

With the goal of maximizing the objective function while satisfying the constraints. We can represent the feasible region of valid variables $x_1, \ldots, x_n$ that satisfy our constraints as a n-dimensional polytope.

Algorithms for solving Linear Programs include the Simplex algorithm (Hill Climbing), Khachiyan's Ellipsoid algorithm, and Karmarkar's Algorithm (a kind of interior point/barrier method). While the latter two algorithms run in polynomial time, the Simplex algorithm is not necessarily slower than the others depending on the problem.

The simplex algorithm starts at a vertex in the polytope and iteratively moves to the best neighboring vertex (the neighboring vertex with the best objective value) until it reaches the maximum. This works since there are no local maxima since the feasible region is convex.

There are exponentially many possible vertices the simplex algorithm can explore. There are multiple axes of parallelism we can explore to speed up this problem.

We seek to exploit axes of parallelism such as splitting the potential search space among different parallel execution elements. Additionally, linear programs can be converted into their dual form, which may be easier to solve. We can have threads for the primal problem and the dual problem,

competing to reach the answer first. We could also have different parallel execution units start at different vertices, as starting position is a significant predictor of overall runtime.

Challenges Involved:

The biggest challenge involves the large amount of ways to potentially parallelize and optimize an implementation of the simplex algorithm. Effective performance debugging will be a large challenge here.

The simplex algorithm involves different potential axes of parallelism. With multiple different ways to parallelize the simplex algorithm, we will be implementing and testing multiple approaches to determine whether we can speed up this algorithm with OpenMP and if so, how much we can speed it up.

Communicating the representation of a problem uses a significant amount of data, so communicating and splitting the work between workers efficiently is a challenge. Since we are using OpenMP, we want to minimize the amount of data moved between each core's own L2 cache to reduce communication.

Finding which axes of parallelism work best for simplex algorithm is hard. Simplex can have an exponnential runtime since there can be an exponentially large number of vertices explored. Deciding whether to synchronize access to a shared data structure or to split the problems and combine results is another challenge.

Memory accesses can have high locality, since each execution unit will repeatedly use the same inputs to our problem, such as constraints.

Properties of the system that make mapping the workload to a GHC cluster machine or PSC Blacklight machine challenging include dealing with limited memory, namely the size of a cache. Our working set of information (especially constraints) can be large. For instance, if we our number of constraints is in the thousands, representing our constraints as a matrix means potentially using megabytes of storage. The constraints have large spatial locality but not much temporal locality within a run.

The reference implementations for the Simplex algorithm involve matrix multiplication. Since constraints usually take the form of a sparse matrix, we can either find a way to represent sparse matrices and perform matrix operations on them or we will need to use an parallelism model that has sufficiently large memory. For example, if we aren't using an efficient sparse matrix representation, then we can't store all our data in a block's memory in a GPU.

Additionally, we may want to make use of SIMD parallelism when finding argmax, or matrix mult, etc. This will be a target to explore once we have completed an OpenMP implementation.

Resources To Use:

We choose to use the C++ programming language, OpenMP, the GHC machines, and the PSC.

Our codebase starts from sample sequential Python code for the Simplex algorithm we found online. Sample Python Implementations of Simplex in blog posts we have found:

- https://medium.com/@jacob.d.moore1/coding-the-simplex-algorithm-from-scratch-using-python-and-numpy-93e3813e6e70
- https://github.com/kshitijl/linear-programming-interior-point

- https://radzion.com/blog/operations/simplex

References we used to learn more about this algorithm include:

- A 15-451 lecture that introduced linear programming. Both members of this group are currently taking 15-451. https://www.cs.cmu.edu/ 15451-f22/lectures/lec15-lp1.pdf
- A paper by one of the creators of HiGHS, a high performance linear programming solver library titled "Towards a practical parallelisation of the simplex method": https://link.springer.com/content/p 008-0080-5.pdf on optimizing simplex
- An online seminar by Gurobi: https://www.gurobi.com/events/how-to-exploit-parallelism-in-linear-and-mixed-integer-programming/

We have found a set of linear programming test cases at https://netlib.org/lp/data/index.html . We plan on using this if possible to avoid having to write our own test case generator.

We don't believe that we need to use special machines, but would benefit from using a shared-memory model machine similar to the one used in the aforementioned paper.

Goals and Deliverables:

Goals we plan on achieving:

- Sequential Simplex algorithm implementation
- Parallel OpenMP Simplex algorithm implementation
- Speedup graphs for OpenMP with Simplex

Goals we hope to achieve:

- Revised Simplex method, sequential and parallel, utilizing sparse matrix multiplication methods.
- Karmarkar's method, sequential and parallel, suitable for different problem types.

Platform Choice:

We choose to use the C++ programming languages, OpenMP, the GHC machines, and the PSC. We are targeting the shared-memory parallelism model. We plan on applying the skills gained from project 3 to our final project. We feel that we are most familiar with this setup.

C++ and OpenMP have well-constructed parallelism infrastructure and excellent documentation. Additionally, we don't have to manually split and marshall data between workers, as we can use arrays in shared memory. This is an advantage over CUDA and OpenMPI.

Schedule:

Week 1 (November 7 to November 11)	Finalize Project Idea, Write Project Proposal, Set up Github Repository, Implement Serial Version of Simplex Algorithm, Establish Benchmark for Testing Serial Version, Get Baseline Visuals
Week 2 (November 14 to November 18)	Parallelize Simplex Algorithm
Week 3 (November 21 to November 25)	Finish Parallelizing Simplex Algorithm, Implement Serial Karmarkar's Algorithm
Week 4 (November 28 to December 2)	Optimize Parallel Simplex Code, Get Simplex Visuals Implement Parallel Karmarkar's Algorithm
Week 5 (December 5 to December 9)	Finish Final Report + Poster