Bereket Molla(bam2243)

Linear Programming and the Simplex Method

Linear Programming

Context: Linear programming (LP) is a method to achieve the best outcome in a mathematical model whose requirements are represented by linear relationships. The simplex algorithm is a popular method used to solve LP problems. However, its sequential nature often limits the speed of computation, especially for large-scale problems.

Project Goal: This project aims to develop a parallelized version of the simplex algorithm using Haskell. Haskell's functional nature and robust support for concurrent and parallel programming make it an ideal choice for this project. The goal is to improve the algorithm's efficiency by parallelizing its computational steps, thereby reducing the time complexity for large LP problems. I will focus on adapting the simplex algorithm to utilize Haskell's parallel processing capabilities effectively.

Simplex Method

The simplex method is a cornerstone algorithm for solving linear programming problems. The method works by moving along the edges of the feasible region defined by the constraints to find the optimal solution. At each step, it evaluates corner points (vertices) of the feasible region, searching for the one that optimizes the objective function.

Assumptions of the Simplex Implementation

This simplex implementation is based on several assumptions that simplify the structure of the linear programming problem:

1. Linear Inequalities and Equations: The constraints of the problem are expressed as linear inequalities or equations. Inequalities are of the form $a^T x \leq b$, and equations are of the form $a^T x \leq b$, where a and x are vectors, and b is a scalar. Each constraint represents a linear relationship that the solution must satisfy.

2. Non-negative Variables: All variables in the problem (elements of the vector x) are assumed to be non-negative. This non-negativity constraint is crucial for the geometric interpretation of the feasible region and for the simplex algorithm's operation.

3. Maximization or Minimization: The simplex method can be applied to both maximization and minimization problems. In standard form, it's common to formulate the problem as a maximization; however, minimization problems can be converted to maximization problems through a transformation of the objective function.

Simplex Algorithm

The simplex algorithm is a step-by-step procedure to solve linear programming problems. Here's a high-level overview of the algorithm:

1. Initialization: Convert the linear programming problem into standard form and construct an initial simplex tableau.

2. Finding the Pivot: Identify the entering variable (column) that will improve the objective function most significantly. This is usually the variable with the most negative coefficient in the objective function row of the tableau.

3. Determine the Leaving Variable: For the chosen entering variable, determine the row that limits its increase (the pivot row). This is done by calculating the ratio of the right-hand side to the coefficient of the entering variable in each row and selecting the smallest positive ratio.

4. Pivot Operation: Perform row operations to make the pivot element equal to 1 and all other elements in the pivot column zero. This step effectively moves the solution to an adjacent vertex of the feasible region.

5. Iterate: Repeat the process of finding the pivot column and row, and performing the pivot operation, until there are no more negative coefficients in the objective function row (indicating the optimal solution is found) or until no pivot row can be found (indicating the problem is unbounded).

6. Solution Extraction: Once the optimal tableau is achieved, extract the solution. The values of the decision variables are found in the right-hand side of the tableau.

Parallelization

The idea behind parallelizing the simplex method is to distribute the computation involved in the pivot operation across multiple processor cores. The pivot operation involves updating each row of the tableau based on the pivot row and pivot column. This operation is inherently parallelizable as the update of each row can be performed independently of others.

Benchmark Results:

The benchmarks were conducted to evaluate the performance of the parallelized simplex algorithm against its non-parallelized counterpart. The tests were run on a multi-core system, and the execution times were recorded. Below are the results:

- Test 1: Smaller Problem Size
  - Parallelized Execution (-N8): 2.38s user time, 0.24s system time, 142% CPU, total time 1.833s.
  - Non-Parallelized Execution: 2.38s user time, 0.25s system time, 142% CPU, total time 1.841s.

For smaller problem sizes, there is no significant difference in execution time between the parallelized and non-parallelized versions. This indicates that for less computationally intensive problems, the benefit of parallelization is not pronounced, likely due to the overhead of managing parallel threads.

- Test 2: Larger Problem Size (First Run)
  - Parallelized Execution (-N8): 154.55s user time, 11.90s system time, 115% CPU, total time 2:24.32
  - Non-Parallelized Execution: 144.16s user time, 11.71s system time, 118% CPU, total time 2:11.63.

- Test 3: Larger Problem Size (Second Run)
  - Parallelized Execution (-N8): 137.63s user time, 12.67s system time, 119% CPU, total time 2:05.66.
  - Non-Parallelized Execution: 135.88s user time, 12.48s system time, 119% CPU, total time 2:04.03.

Observation

In both tests with larger problem sizes, the parallelized version did not consistently outperform the non-parallelized one. In the first run, the parallelized version was slower,

while in the second run, the performance was comparable. This suggests that while parallelization has the potential to improve performance, its effectiveness can vary depending on the specific characteristics of the problem and the computational environment.

In parallel computing, there's inherent overhead associated with managing parallel processes. This includes the time and resources required to create and synchronize threads, handle communication between them, and manage shared resources. For the simplex method, each pivot operation involves multiple row operations that are relatively quick to compute. The overhead of distributing these operations across multiple threads and then synchronizing the results can be significant in comparison to the actual computation time. If the overhead is larger than the computational savings gained from parallel processing, the overall performance may be worse than a sequential implementation.

Moreover, the simplex method, by its nature, involves a series of pivot operations where each pivot is dependent on the outcome of the previous one. This sequential dependency limits the opportunities for parallel execution. In each iteration, the algorithm identifies a pivot element and then adjusts the tableau based on this pivot. The next pivot cannot be determined until the current pivot operation is complete and the tableau is updated. Therefore, despite parallelizing certain computations within each pivot operation, the fundamental sequential nature of the simplex algorithm imposes a limit on the overall parallelizability of the method. This inherent sequential dependency can significantly constrain the performance gains from parallelization, as each step must wait for the previous step to complete before proceeding.

Conclusion

In conclusion, while parallelization has the potential to enhance computational efficiency, its effectiveness in the simplex algorithm is constrained by the significant overhead associated with managing parallel processes and the inherently sequential nature of the algorithm's pivot operations. Therefore, careful consideration and analysis are necessary to determine if parallelization offers tangible benefits for a specific implementation of the simplex method.