Simplex Method with CUDA

Fan E / 1452890

Abstract

The simplex algorithm is one of the most commonly used optimization algorithms in the industrial world. First, most realistic problems faced in real world are linear programming problem, which has both linear objective function and linear constraints; Then, despite the effort finding a more efficient algorithm to solve linear programming problem, the simplex algorithm works the most profoundly and efficiently. Although, in the worst case, it has exponential complexity, in general it takes polynomial-time calculation.

1 Introduction to Linear Programming

A linear programming problem can be formulated in below algebraic form,

Minimize

$$z = \mathbf{cx} \tag{1a}$$

Subject to

$$\mathbf{Ax} = \mathbf{b} \tag{1b}$$

$$\mathbf{x} \geq 0 \tag{1c}$$

This matrix $\mathbf{A}$ corresponds to the coefficients on $x_1, x_2, ..., x_n$ in the constraints of a linear programming. The vector $\mathbf{x}$ is a vector of solutions to the problem, $\mathbf{b}$ is the right-hand-side vector, and $\mathbf{c}$ is the cost coefficient vector.

There are some important concepts to understand the simplex method.

Definition 1.1 *Given the system* $\mathbf{Ax} = \mathbf{b}$ *and* $\mathbf{x} \geq 0$ *where* $\boldsymbol{A}$ *is an* $m * n$ *matrix and* $\boldsymbol{b}$ *is a column vector with* m *entries. Supposes that rank(*$\boldsymbol{A}$*,*$\boldsymbol{b}$*)=rank(*$\boldsymbol{A}$*)=*m*. After possibly rearranging the columns of* $\boldsymbol{A}$*, let* $\mathbf{A} = [\mathbf{B}, \mathbf{N}]$ *where* $\boldsymbol{B}$ *is an* $m * m$ *invertible matrix and* $\boldsymbol{N}$ *is an* $m * (n - m)$ *matrix. The solution* $\mathbf{x} = \begin{bmatrix} \mathbf{x_B} \\ \mathbf{x_N} \end{bmatrix}$ *to the equations* $\mathbf{Ax} = \mathbf{b}$ *where*

$\mathbf{x_B} = \mathbf{B^{-1}b}$ *and* $\mathbf{x_N} = 0$ *is called a* ***basic solution*** *of the system. If* $\mathbf{x_B} \geq \mathbf{0}$ *then* $\boldsymbol{x}$ *is called a* ***basic feasible solution*** *of the system. Here* $\boldsymbol{B}$ *is called the* ***basic matrix*** *and* $\boldsymbol{N}$ *is called the* ***nonbasic matrix****. The components of* $\mathbf{x_B}$ *are called* ***basic variables*** *and the components of* $\mathbf{x_N}$ *are called* ***nonbasic variables****. If* $\mathbf{x_B} > \mathbf{0}$ *then* $\boldsymbol{x}$ *is a* ***nondegenerate basic solution****, but if at least one component of* $\mathbf{x_B}$ *is zero then* $\boldsymbol{x}$ *is a* ***degenerate basic solution****.*

We can imagine that the system of linear constraints forms a polyhedron in the n dimension space, and the objective function forms a hyperplane. By moving the hyperplane represented by the objective function, we will reach a extreme point where the object can not improve any more.

The simplex method calculates one extreme point at each time and compares the objective value of the current extreme point to the former. If in all directions there is no improvement, it will mean that we obtain our optimal solution.

The algorithm can be summarized as below:

Input: Matrix $\mathbf{A}$, vectors $\mathbf{b}$ and $\mathbf{c}$. Problem in canonical augmented form.
Output: Optimal solution or unbounded problem message

```
1  /* Initialize data (Range assignements with Matlab-like notation). */
2  B[m][m] ← Initialize(A);
3  c_B[m] ← c[n − m : n − 1];
4  x_B[m] ← 0;
5  Optimum ← ⊥;
6  while !Optimum do
7      /* Determine the entering variable */
8      Index p ← {j|c̃_j == min_t (c_B B^-1 A_t − c_t)};
9      x_B ← B^-1 b;
10     if c̃_p ≥ 0 then
11         Optimum ← ⊤;
12         break;
13     /* Determine the leaving variable */
14     α ← B^-1 A_p;
15     Index q ← {j|θ_j == min_t (x_Bt/α_t, α_t > 0)};
16     if α ≤ 0 then
17         exit("Problem unbounded");
18         break;
19     /*Update the basis */
20     B ← UpdateBasis(A, p, q);
21     /* Update basis cost */
22     c_Bq ← c_p;
23     /* Update basis solution */
24     x_B ← B^-1 b;
25 if Optimum then
26     exit(x_B, z ← x_B c_B);
```

2 Implementation Notes

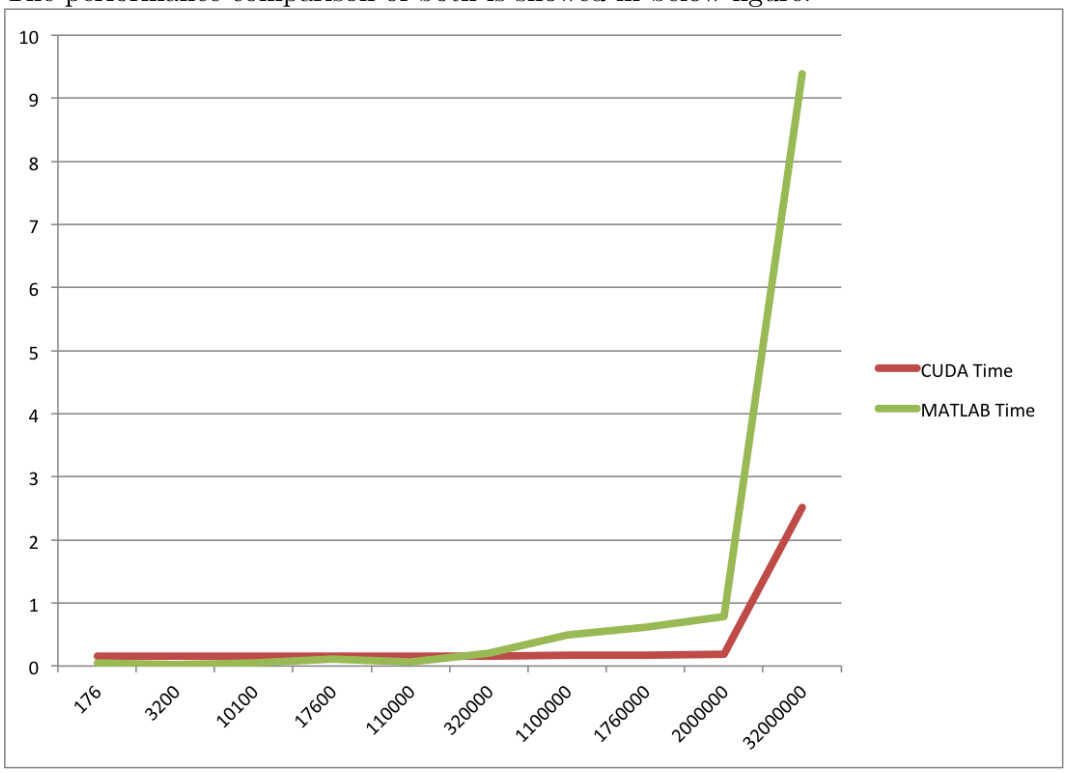
Because most calculations belong to linear algebra, both the cuBLAS library and Thrust library are used in the program. In order to check the result of each step, a few macro functions are created. In the final version of the program, the "DEBUG" macro identifier is commented, which will not affect the performance analysis.

3 Computation Experiment

The "cudaEventRecord" function is used to record the time spent on the simplex algorithm in the program. The benchmark for the CUDA program is the optimization toolbox in MATLAB. The "tic" and "toc" functions are used to record the same procedure as the CUDA program.

The test problem was generated randomly using "rand" function in C program. Here, an improvement can be made, for a random problem may not lead to an optimal solution. It would be advantageous to generate a problem with certain structure.

The performance comparison of both is showed in below figure:



There is a critical point of the problem size where the speed of CUDA program exceed the MATLAB program(on CPU). Also, the advantage of CUDA becomes more obvious as the problem size increases.

The specific data table is as below:

4 Conclusion

The CUDA implementation performs better when the problem size is large, where it has average 6 or 7 times better performance.

Problem Size M*N	**CUDA Time**	**MATLAB Time**
176	0.155564	0.047554
3200	0.15705	0.021587
10100	0.158837	0.048292
17600	0.159543	0.109366
110000	0.156384	0.065882
320000	0.160943	0.202955
1100000	0.168958	0.497276
1760000	0.168113	0.61369
2000000	0.186371	0.777857
32000000	2.511953	9.389996

The project is based on the work of Daniele Giuseppe Spampinato, "Linear Optimization with CUDA".