To appear in the *International Journal of Computer Mathematics*
Vol. 00, No. 00, February 2013, 1–12

RESEARCH ARTICLE

GPU Accelerated Pivoting Rules for Linear Optimization Problems using GPU

Nikolaos Ploskas[a] and Nikolaos Samaras[a]*

[a]*Department of Applied Informatics, School of Information Sciences, University of Macedonia, 156 Egnatia Str., 54006 Thessaloniki, Greece*

(*v1.0 released March 2013*)

Linear Programming is a significant area in the field of operations research. Pricing is the operation of the selection of an improving non-basic variable to enter the basis. Linear Programming algorithms perform successive pivot operations in order to improve the basis feasible solution. The choice of the pivot element at each iteration is one of the most-time consuming steps in Linear Programming algorithms. Graphical Processing Units (GPUs) have been applied for the solution of linear optimization algorithms and have accelerated their performance. In this paper, we propose a parallel implementation using GPU of six widely-used pivoting rules: (i) Bland's rule, (ii) Dantzig's rule, (iii) Greatest Increment Method, (iv) Least Recently Considered Method, (v) Partial Pricing Rule, and (vi) Steepest Edge Rule; and incorporate them with the revised simplex algorithm. All pivoting rules have been implemented in MATLAB and CUDA environment. Finally, a computational study on large-scale randomly generated optimal dense LPs, the Netlib (optimal, Kennington, and infeasible LPs) and the Mészáros set was conducted and showed that we can benefit from the recent hardware advances on GPUs.

Keywords: linear programming; pivoting rules; GPU; MATLAB; CUDA

AMS Subject Classification: 65F35; 90C05; 90C99

1. Introduction

Linear Programming (LP) is the process of minimizing or maximizing a linear objective function $z = \sum_{j=1}^{n} c_j x_j$ to a number of linear equality and inequality constraints. Simplex algorithm is the most widely used method for solving Linear Programming problems (LPs). Consider the following LP (LP.1) in the standard form:

$$\begin{aligned} min \quad & c^T x \\ s.t. \quad & Ax = b \\ & x \geq 0 \end{aligned} \qquad (LP.1)$$

where $A \in \mathbb{R}^{m \times n}$, $(c, x) \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and T denotes transposition. We assume that A has full rank, $rank(A) = m, m < n$. Consequently, the linear system $Ax = b$ is consistent. The simplex algorithm searches for an optimal solution by moving from one feasible

*Corresponding author. Email: samaras@uom.gr

solution to another, along the edges of the feasible set. The dual problem associated with the (LP.1) is presented in (DP.1):

$$\begin{array}{ll} min & b^T w \\ s.t. & A^T w + s = c \\ & s \geq 0 \end{array} \qquad (DP.1)$$

where $w \in \mathbb{R}^m$ and $s \in \mathbb{R}^n$.

A time-consuming step in solving an LP is the selection of the entering variable in each iteration. Good choices of the entering variable can lead to fast convergence to the optimal solution, while poor choices lead to more iterations and worst execution times or even no solutions of the LPs. The pivoting rule applied to an LP algorithm is the key factor that will determine the number of the iterations that the LP algorithm performs [16]. Many pivoting rules have been proposed in the literature. Six of these are implemented and compared in this paper; namely, (i) Bland's rule, (ii) Dantzig's rule, (iii) Greatest Increment Method, (iv) Least Recently Considered Method, (v) Partial Pricing Rule, and (vi) Steepest Edge Rule. Other well-known pivoting rules include Devex [12], Modified Devex [2], Steepest Edge approximation scheme [24], Murty's Bard type scheme [18], Edmonds-Fukuda rule [8] and its variants [6] [27] [29] [30].

Forrest and Goldfarb (1992) presented several new steepest edge pivoting rules and compared them with Devex variants and the Dantzig rule over large LPs and concluded that the steepest-edge variants are clearly superior to the Devex variants and the Dantzig rule for solving difficult large-scale LPs. Thomadakis [25] has implemented and compared the serial implementations of five pivoting rules: (i) the Dantzig's rule, (ii) the Bland's rule, (iii) the Least-Recently Considered Method, (iv) the Greatest-Increment Method, and (v) the Steepest-Edge rule. Thomadakis examined the trade-off between the number of iterations and the execution time per iteration and concluded that: (i) Bland's rule has the shortest execution time per iteration, but it usually needs many more iterations that the other methods to converge to the optimal solution of an LP, (ii) Dantzig's rule and Least Recently Considered Method perform comparably, but the latter performs fewer iterations when degenerate pivots exist, (iii) Greatest Increment Method has the worst execution time per iterations, but it usually needs fewer iterations to converge to the optimal solution of an LP, and (iv) Steepest-Edge rule requires fewer iterations than all the other pivoting rules and its execution time per iterations is lower than Greatest Increment Method but higher than the other three methods.

Ploskas and Samaras [21] have implemented and compared the serial implementations of eight pivoting rules over small- and medium-sized Netlib LPs: (i) Bland's rule, (ii) Dantzig's rule, (iii) Greatest Increment Method, (iv) Least Recently Considered Method, (v) Partial Pricing Rule, (vi) Queue Rule, (vii) Stack Rule, and (viii) Steepest Edge Rule. In contrast to Thomadakis, Ploskas and Samaras also examined the total execution time of the LP algorithm relating to the pivoting rule that is used and concluded that: (i) with a limit of 70,000 iterations, only Dantzig's rule has solved all instances of the test set, while Bland's rule solved 45 out of 48 instances, Greatest Increment method 46 out of 48, Least Recently Considered method 45 out of 48, Partial Pricing 45 out of 48, Queue's rule 41 out of 48, Stacks' rule 43 out of 48, and Steepest Edge rule 46 out of 48, (ii) Dantzig's rule requires the shortest execution time both on average and on almost all instances, while Steepest Edge rule has the worst execution time both on average and on almost all instances, and (iii) despite its computational cost, Steepest Edge rule needs the fewest number of iterations than all the other pivoting rules, while Bland's rule is by far the worst pivoting rule in terms of the number of iterations.

The increasing size of real life LPs demands more computational power and parallel computing capabilities. The recent hardware advances have made it possible to solve large LPs in a short amount of time. Graphical Processing Units (GPUs) have gained a lot of popularity in the past decade and have been applied to scientific computing applications. Two major programming models exist for GPUs, CUDA (Compute Unified Device Architecture) and OpenCL (Open Computing Language). NVIDIA introduced CUDA in late 2006 and is only available with NVIDIA GPUs, while Apple introduced OpenCL in 2008 and is available on GPUs of different vendors and even on CPUs. In this paper, we have implemented the GPU-based pivoting rules using CUDA, because it is currently more mature. GPUs have been successfully applied for the solution of LPs (Bieling et al. [3], Lalami et al. [15], Meyer et al. [17], Spampinato and Elster [23]) and Interior Point Methods (Jung and OLeary [13], Smith et al. [22]).
To the best of our knowledge, Thomadakis' report [25] and Ploskas' and Samaras' paper [21] are the only paper that compares some of the most widely-used pivoting rules. Furthermore, there isn't any paper that attempted to implement any pivoting rule using GPUs. This paper is an extension of the work of Ploskas and Samaras [21]. The novelty of this paper is that we propose an GPU-based implementation of six pivoting rules and perform a detailed computational study on the Netlib set (optimal, Kennington and infeasible LPs)in order to establish the practical value of the GPU-based implementations.
The structure of the paper is as follows. In Section 2, six pivoting rules are presented and analyzed. Section 3 presents the GPU-based implementations of these pivoting rules. In Section 4, the computational comparison of the CPU and GPU based implementations are presented over the Netlib set (optimal, Kennington and infeasible LPs). Finally, the conclusions of this paper are outlined in section 5.

2. Pivoting Rules

Six pivoting rules are presented in this section: (i) Bland's rule, (ii) Dantzig's rule, (iii) Greatest Increment Method, (iv) Least Recently Considered Method, (v) Partial Pricing Rule, and (viii) Steepest Edge Rule. Some necessary notations should be introduced, before the presentation of the aforementioned pivoting rules. Let l be the index of the entering variable and $\overline{c_l}$ be the difference in the objective value when the non-basic variable x_l is increased by one unit and the basic variables are adjusted appropriately.

2.1 *Bland's Rule*

Bland's rule [4] selects as entering variable the first among the eligible ones, that is the leftmost among columns with negative relative cost coefficient. Although Bland's rule avoids cycling, it has been observed in practice that this pivoting rule can lead to stalling, a phenomenon where long degenerate paths are produced.

2.2 *Dantzig's Rule*

The first pivoting rule that was used in the simplex algorithm is Dantzig's rule or largest coefficient rule [7]. This pivoting rule selects the column A_l with the most negative $\overline{c_l}$. It guarantees the largest reduction in the objective value per unit of non-basic variable $\overline{c_l}$ increase. Although its worst-case complexity is exponential [14], Dantzig's rule is claimed as simple but powerful enough to guide simplex algorithm into short paths [25]. This is

the first proposed pivoting rule for the simplex algorithm and has been widely-used in simplex implementations [1] [20].

2.3 *Greatest Increment Method*

According to the Greatest Increment Method [14], the variable with the largest total objective value improvement is selected as the incoming variable. Greatest Increment Method calculates the improvement of the objective value for each non-basic variable and then selects the variable that offers the largest improvement in the objective value. Although this pivoting rule can lead to fast convergence to the optimal solution, this advantage is eliminated by the additional computational cost per iteration. Finally, Gärtner [9] constructed LPs that Greatest Increment Method showed exponential complexity.

2.4 *Least Recently Considered Method*

In the first iteration of the algorithm, the incoming variable l is selected according to Dantzig's rule, that is the leftmost among columns with negative relative cost coefficient. In the next iterations, the Least Recently Considered Method [28] starts searching for the first eligible variable with index greater than l. If $l = n$ then Least Recently Considered Method starts searching from the first column again. Least Recently Considered Method prevents stalling and it performs fairly well in practice [25]. However, its worst-case complexity has not been proved yet.

2.5 *Partial Pricing Rule*

Partial Pricing methods are variants of the standard rules that take only a part of non-basic variables into account. In the computational study presented in Section 4, we have implemented the partial pricing rule as variant for Dantzig's rule. In static partial pricing, non-basic variables are divided into equal segments with predefined size and the pricing operation is carried out segment by segment, until a reduced cost is found. In dynamical partial pricing, the segments' size is determined dynamically during the execution of the algorithm.

2.6 *Steepest Edge Rule*

Steepest Edge Rule or All-Variable Gradient Method [11] selects as entering variable the variable with the most objective value reduction per unit distance, as shown in Equation (1):

$$d_j = min \left\{ \frac{c_l}{\sqrt{1 + \sum_{i=1}^{m} x_{il}^2}} : l = 1, 2, ..., n \right\} \tag{1}$$

Although this pivoting rule can lead to fast convergence to the optimal solution, this advantage is debatable due to the additional computational cost. Approximate methods have been proposed in order to improve the computational efficiency of this method [24] [26].

3. GPU Accelerated Pivoting Rules

In this section, we present the GPU-based implementations of the aformentioned pivoting rules, taking advantage of the power that modern GPUs offer. The parallel implementations of these pivoting rules are implemented on MATLAB and CUDA. MATLAB's GPU support was added in version R2011a.

3.1 *GPU Architecture*

This section briefly describes the architecture of an NVIDIA GPU in terms of hardware and software. GPU is a multi-core processor having thousands of threads running concurrently. GPU has many cores aligned in a particular way forming a single hardware unit. Data parallel algorithms are well suited for such devices, since the hardware can be classified as SIMT (Single-Instruction, Multiple Threads). GPUs outperforms CPUs in terms of GFLOPS (Giga Floating Point Operations per Second). For example a high-end Core I7 processor with 3.46 GHz delivers up to a peak of 55.36 GFLOPs, while a high-end NVIDIA Quadro 6000 delivers up to a peak of 1030.4 GFLOPs.
NVIDIA CUDA is an architecture that manages data-parallel computations on a GPU. A CUDA program includes two portions, one that executes on the CPU and another that executes on the GPU. The code that can be parallelized is executed on the GPU, as kernels, while the rest is executed on the CPU. CPU starts the execution of each portion of code and invokes a kernel function, so the execution is moved to the GPU. The connection between CPU memory and GPU memory is through a fast PCIe 16x point to point link. Each code that is executed on the GPU is divided into many threads. Each thread executes the same code independently on different data. A thread block is a group of threads that cooperate via shared memory and synchronize their execution to coordinate their memory accesses. A grid consists of a group of thread blocks and a kernel is executed on a group of grids. A kernel is the resulting code after the compilation. NVIDIA Quadro 6000, which was used in our computational experiment, consists of 14 stream processors with 32 cores each, resulting to 448 total cores.

3.2 *Implementation of GPU-Based Pivoting Rules*

Figure 1 presents the process that is performed in the GPU-based implementation of the linear programming algorithm. Firstly, CPU initialize the algorithm by reading the variables of the LP. In the second step, CPU computes a feasible solution and checks if the LP is optimal. If the LP is optimal, the algorithm terminates, while if it is not, CPU transfers the adequate variables to GPU. In step 3, GPU finds the index of the entering variable according to a pivoting rule and then transfers the index to CPU. CPU checks if the LP is unbounded in order to terminate the algorithm and if it is not, then updates the basis and the variables. Then, the algorithm continues with the next iteration until a solution is found.

3.3 *Notations*

Some necessary notations should be introduced, before the presentation of the pseudocodes of each pivoting rule. Let A be an 1xn matrix. Let $NonBasicList$ be an 1x$n-m$ vector with the indices of the non basic variables. Let $BaseInv$ be an mxm matrix with the basis inverse. Let Xb be an 1xm vector with the basic variables. Let Sn

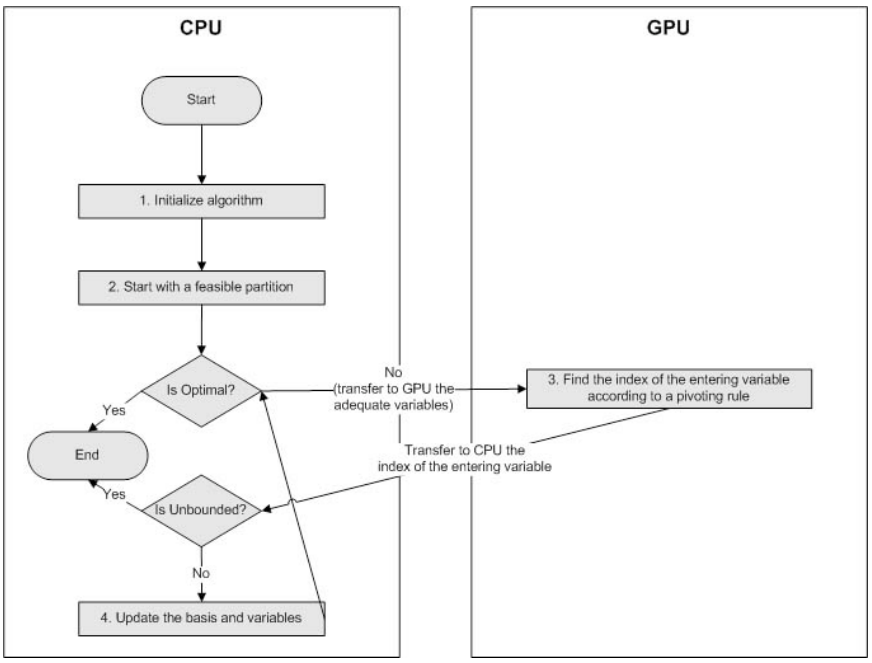

Figure 1. Flow Chart of the GPU-based Pivoting Rules

Table 1. GPU-based Bland's Rule

```
1.  transfer to GPU the vector Sn
2.  do parallel
3.      find the index of the leftmost negative element of vector Sn
4.  end parallel
5.  transfer the index to CPU
```

be an 1xn vector with dual slack variables. Let l be the index of the entering variable. Let $lrcmLast$ be the index of the last selected entering variable in the Least Recently Considered Method. Let $segmentSize$ be the fixed size of the segments and $lastSegment$ the segment that the last entering variable was found in the Partial Pricing Rule.
The pseudocode of all pivoting rules are presented in the following sub-sections of this section. Pseudocodes include "do parallel" and "end parallel" sections, in which the workload is divided into warps that are executed sequentially on a multiprocessor. The warp size in NVIDIA Quadro 6000 is 32 threads.

3.4 *GPU-based Bland's Rule*

Table 1 shows the pseudocode of the implementation of the Bland's rule on a GPU. Initially, CPU transfers the vector Sn to GPU (line 1). Then, the index of the leftmost negative element of Sn is calculated in parallel (lines 2 - 4). Finally, GPU transfers the index to CPU (line 5).

3.5 *Dantzig's Rule*

Table 2 shows the pseudocode of the implementation of the Dantzig's rule on a GPU. Initially, CPU transfers the vector Sn to GPU (line 1). Then, the index of the minimum

Table 2. GPU-based Dantzig's Rule

```
1.  transfer to GPU the vector Sn
2.  do parallel
3.      find the index of the minimum element of Sn
4.  end parallel
5.  transfer the index to CPU
```

Table 3. GPU-based Greatest Increment Method

```
1.  transfer to GPU vectors Sn, NonBasicList, Xb and A(:,l) and array BaseInv
2.  do parallel
3.      localMaxDecrease = inf
4.      for i=1:length(NonBasicList)
5.          if(Sn(i) < 0)
6.              l = NonBasicList(i)
7.              h_l = BaseInv * A(:,l)
8.              mrt = indices of the positive elements of h_l
9.              Xb_hl_div = Xb(mrt) ./ h_l(mrt)
10.             theta0 = minimum index of positive elements of vector Xb_hl_div
11.             currentDecrease = theta0 * Sn(column)
12.             if(currDecrease < localMaxDecrease)
13.                 localMaxDecrease = currDecrease
14.                 index = i
15.             end
16.         end
17.     end
18.     maxDecrease = minimum decrease of all localMaxDecrease
19.     find the index of the variable where the maxDecrease occurs
20. end parallel
21. transfer the index to CPU
```

element of Sn is calculated in parallel (lines 2 - 4). Finally, GPU transfers the index to CPU (line 5).

3.6 *Greatest Increment Method*

Table 3 shows the pseudocode of the implementation of the Greatest Increment Method on a GPU. Initially, CPU transfers the vectors Sn, $NonBasicList$, Xb and $A(:,l)$ and array $BaseInv$ to GPU (line 1). Then, for each candidate variable with negative coefficient cost, the improvement in the objective value is calculated (lines 4 - 17). The index of the variable that offers the largest improvement in the objective value is selected (lines 18 - 19). Finally, GPU transfers the index to CPU (line 21).

3.7 *GPU-based Least Recently Considered Method*

Table 4 shows the pseudocode of the implementation of the Least Recently Considered Method on a GPU. Initially, CPU transfers the value $lrcmLast$ and vectors Sn and $NonBasicList$ to GPU (line 1). In the first iteration of the algorithm, the incoming variable is selected according to the Dantzig's rule (lines 2 - 5). In the next iterations,

Table 4. GPU-based Least Recently Considered Method

```
1.  transfer to GPU the value lrcmLast and vectors Sn and NonBasicList
2.  if(iterations == 1)
3.      do parallel
4.          find the index of the minimum element of Sn
5.      end parallel
6.  else
7.      do parallel
8.          find the index of the first eligible variable with index greater than lrcmLas
9.      end parallel
10. end
11. transfer the index to CPU
```

Table 5. GPU-based Partial Pricing Rule

```
1.  transfer to GPU the values segmentSize and lastSegment and vector Sn
2.  while(index == null)
3.      do parallel
4.          find the index of the minimum element in the lastSegment of Sn
6.              lastSegment = lastSegment + 1
7.          end
8.      end parallel
9.  end
10. transfer the index to CPU
```

Least Recently Considered Method finds the index of the first eligible variable with index greater than *lrcmLast* (lines 6 - 10). Finally, GPU transfers the index to CPU (line 11).

3.8 *GPU-based Partial Pricing Rule*

Table 5 shows the pseudocode of the implementation of the Partial Pricing Rule on a GPU. Initially, CPU transfers the values *segmentSize* and *lastSegment* and vector Sn to GPU (line 1). Dantzig's rule is performed in the segment denoted by the variable *lastSegment* (line 4). If a reduced cost does not exist in the specific segment, then we search in the next segment (lines 5 - 7). Finally, GPU transfers the index to CPU (line 10).

3.9 *GPU-based Steepest Edge*

Table 6 shows the pseudocode of the implementation of the Steepest Edge on a GPU. Initially, CPU transfers the the vectors Sn, $NonBasicList$ and arrays A and $BaseInv$ to GPU (line 1). Then, the index of the incoming variable is calculated according to the Equation (1) (lines 3 - 6). Finally, GPU transfers the index to CPU (line 8).

4. Computational Results

Computational studies have been widely used in order to examine the practical efficiency of an algorithm or even compare algorithms. The computational comparison of the

Table 6. GPU-based Steepest Edge

```
1.  transfer to GPU the vectors Sn, NonBasicList and arrays A and BaseInv
2.  do parallel
3.      Y = BaseInv * A(:,NonBasicList)
4.      dj = sqrt(1 + diag(Y' * Y))
5.      rj = Sn' ./ dj
6.      find the index of the minimum element of the vector rj
7.  end parallel
8.  transfer the index to CPU
```

Table 7. Statistics of the Netlib (optimal, Kennington and infeasible LPs) and Mészáros LPs

Name	Constraints	Variables	Nonzeros A	Objective value
BNL1	644	1,175	6,129	1.98E+03
CRE-A	3,517	4,067	19,054	2.35E+07
CRE-C	3,069	3,678	16,922	2.52E+07
KLEIN1	55	54	696	Infeasible
KLEIN2	478	54	4,585	Infeasible
SCFXM3	991	1,371	7,846	5.49E+04
SCTAP3	1,481	2,480	1,0734	1.42E+03
STOCFOR2	2,158	2,031	9,492	-3.90E+04

aforementioned updating schemes has been performed on a quad-processor Intel Core i7 3.4 GHz with 32 Gbyte of main memory and 8 cores, a clock of 3700 MHz, an L1 code cache of 32 KB per core, an L1 data cache of 32 KB per core, an L2 cache of 256 KB per core, an L3 cache of 8 MB and a memory bandwidth of 21 GB/s, running under Microsoft Windows 7 64-bit and on a NVIDIA Quadro 6000 with 6 GB GDDR5 384-bit memory, a core clock of 574 MHz, a memory clock of 750 MHz and a memory bandwidth of 144 GB/s. It consists of 14 stream processors with 32 cores each, resulting to 448 total cores. The graphics card driver installed in our system is NVIDIA 64 kernel module 306.23. The serial algorithms have been implemented using MATLAB Professional R2012b. MATLAB (MATrix LABoratory) is a powerful programming environment and is especially designed for matrix computations in general.
Serial pivoting rules automatically execute on multiple computational threads, in order to take advantage of the multiple cores of the CPU. The execution time of all serial pivoting rules already includes the performance benefit of the inherent multithreading in MATLAB. Execution times both on CPU and GPU-based pivoting rules have been measured using tic and toc MATLAB's built-in functions.
In the computational study, two test beds were used: (i) randomly generated dense optimal LPs (problem instances have the same number of constraints and variables and the largest problem tested has 3000 constraints and 3000 variables), and (ii) ten large-scale LPs from the Netlib set (optimal, Kennington and infeasible LPs) (Gay 1985, Carolan et al. 1990) and Misc section of Mészáros collection [18]. The Netlib library is a well known suite containing many real world LPs. Ordóñez and Freund [19] have shown that 71% of the Netlib LPs are ill-conditioned. Hence, numerical difficulties may occur. Table 7 presents some useful information about the test bed, which was used in the computational study. The first column includes the name of the problem, the second the number of constraints, the third the number of variables, the fourth the nonzero elements of matrix A and the fifth the objective value. The test bed includes 4 optimal and 2 infeasible LPs from Netlib, 2 Kennington and 2 LPs from Mészáros collection that do not have ranges and bounds sections in their mps files.

Tables **??** and **??** present the results from the execution of the serial implementations of pivoting rules over the randomly generated optimal dense LPs and over the Netlib and Mészáros set of LPs, respectively. Tables **??** and **??** present the results from the

execution of the GPU-based implementations of pivoting rules over the randomly generated optimal dense LPs and over the Netlib and Mészáros set of LPs, respectively. Figure 2 presents the average speedup for each pivoting rule. In Tables **??** - **??**, the following abbreviations are used: (i) Bland's rule - BR, (ii) Dantzig's rule - DR, (iii) Greatest Increment method - GIM, (iv) Least Recently Considered method - LRCM, (v) Partial Pricing rule - PPR, and (vi) Steepest Edge rule - SER. For each instance, we averaged times over 10 runs. All times in Tables **??** - **??** are measured in seconds. Finally, the results of the GPU are very accurate, because NVIDIA Quadro 6000 is fully IEEE 764-2008 compliant 32- and 64-bit fast double-precision.

Tables 8 - 11 and Figure 2.

The results show that ..... Figure 3 presents the average portion of time that each serial pivoting rule needs to calculate the entering variable against the total time. These findings lead us to conclude that only the steepest edge rule can be parallelized. Many implementations for CPU parallel implementations of the steepest edge rule have been proposed [3] [12] [26], while Meyer et. al. [17] proposed a multi-GPU implementation for the standard simplex algorithm using steepest edge as the pivoting rule.
Table **??** and **??** present the time to perform the selection of the entering variable both on the CPU and GPU implementation of the steepest edge rule over the randomly generated optimal dense LPs and over the Netlib and Mészáros set of LPs, respectively. Figure 4 presents the speedup of the total and the pivoting time of the GPU based implementation of the steepest edge rule over the CPU implementation. The speedup obtained from the pivoting is not depicted exactly in the speedup of the total time, because the communication cost is still expensive. We plan to port all the steps of the algorithm to GPU in order to fully parallelize the revised simplex method for GPUs.

Figure 3, Tables 12 - 13 and Figure 4.

5. Conclusions

A good selection of the pivoting rule for linear optimization solvers can lead to the reduction of the iterations and the solution time of an LP. GPUs have been already applied for the solution of linear optimization algorithms, but GPU-based implementations of the pivoting rules have not yet studied. In this paper, we reviewed and implemented six widely-used pivoting ruled and proposed GPU-based implementations for them using MATLAB and CUDA. We performed a computational study on large-scale randomly generated optimal dense LPs, the Netlib (optimal, Kennington, and infeasible LPs) and the Mészáros set and found that only steepest edge rule is suitable for GPUs. The speedup gained from the pivoting operation of steepest edge rule is ... on average.
The results are very promising and allow to hope fast GPU-based implementations for linear programming algorithms. In future work, we plan to port all steps of the revised simplex algorithm in a GPU-based implementation in order to avoid communication costs in each step and fully exploit parallelism.

Acknowledgements

The authors thank NVIDIA for their support through the Academic Partnership Program.

References

[1] Bazaraa, M.S., Jarvis, J.J., Sherali, H.D.: Linear Programming and Network Flows. John Wiley & Sons, Inc. (1990)
[2] Benichou, M., Gautier, J., Hentges, G., Ribiere, G.: The efficient solution of large-scale linear programming problems. Mathematical Programming, 13, 280-322 (1977)
[3] Bieling, J., Peschlow, P., Martini, P.: An efficient GPU implementation of the revised Simplex method. In: Proceddings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2010), Atlanta, USA (2010)
[3] Bixby, R.E., Martin, A.: Parallelizing the dual simplex method. INFORMS Journal on Computing, 12(1), 45-56 (2000)
[4] Bland, R.G.: New finite pivoting rules for the simplex method. Mathematics of Operations Research, 2(2), 103-107 (1977)
[5] Carolan, W.J., Hill, J.E., Kennington, J.L., Niemi, S., Wichmann, S.J.: An Empirical Evaluation of the KORBX Algorithms for Military Airlift Applications. Operations Research, 38(2), 240-248 (1990)
[6] Clausen, J.: A note on Edmonds-Fukuda's pivoting rule for the simplex method, Eur. J. Oper. Res., 29, 378-383 (1987)
[7] Dantzig, G.B.: Linear programming and extensions. Princeton, NJ: Princeton University Press (1963)
[8] Forrest, J.J., Goldfarb, D.: Steepest-edge simplex algorithms for linear programming. Mathematical programming, 57(1-3), 341-374 (1992)
[8] Fukuda, K.: Oriented matroid programming, Ph.D. Thesis, Waterloo University, Waterloo, Ontario, Canada (1982)
[9] Gärtner, B.: Randomized optimization by Simplex-type methods. PhD thesis, Freien Universität, Berlin (1995)
[10] Gay, D.M.: Electronic mail distribution of linear programming test problems. Mathematical Programming Society COAL Newsletter, 13, 10-12 (1985)
[11] Goldfarb, D., Reid, J.K.: A Practicable Steepest-Edge Simplex Algorithm. Mathematical Programming, 12(3), 361-371 (1977)
[12] Hall, J.A.J., McKinnon, K.I.M.: An asynchronous parallel revised simplex algorithm. Tech. Rep. MS 95-50, Department of Mathematics and Statistics, University of Edinburgh (1995)
[12] Harris, P.M.J.: Pivot selection methods for the Devex LP code, Mathematical Programming, 5, 1-28 (1973)
[13] Jung, J.H., OLeary, D.P.: Implementing an interior point method for linear programs on a CPU-GPU system. Electronic Transaction on Numerical Analysis 28, 174-189 (2008)
[14] Klee, V., Minty, G.J.: How good is the simplex algorithm. In O. Shisha (Ed.), Inequalities - III. New York and London: Academic Press Inc. (1972)
[15] Lalami, M.E., Boyer, V., El-Baz, D.: Efficient Implementation of the Simplex Method on a CPU-GPU System. In: Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW 2011), pp. 1999–2006. Washington, USA (2011)
[16] Maros, I., Khaliq, M.H.: Advances in design and implementation of optimization software. European Journal of Operational Research, 140(2), 322-337 (1999)
[18] Mészáros, C.: Linear programming test problems. `http://www.sztaki.hu/~meszaros/public_ftp/lptestset/misc/`
[17] Meyer, X., Albuquerque, P., Chopard, B.: A multi-GPU implementation and performance model for the standard simplex method. In: Proceedings 1st International Symposium and 10th Balkan Conference on Operational Research, pp. 312–319. Thessaloniki, Greece (2011)
[18] Murty, K.G.: A note on a Bard type scheme for solving the complementarity problem, Opsearch, 11, 123-130 (1974)
[19] Ordóñez, F., and Freund, R.: Computational experience and the explanatory value of condition measures for linear optimization. SIAM J. Optimization, 14(2), 307-333 (2003)
[20] Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Inc. (1982)

[21] Ploskas, N., Samaras, N.: Computational Comparison of Pivoting Rules for the Revised Simplex Algorithm. In Proc. XI Balkan Conference on Operational Research, 7-10 September, Belgrade (2013)
[22] Smith, E., Gondzio, J., Hall, J.: GPU acceleration of the matrix-free interior point method. In: Parallel Processing and Applied Mathematics, pp. 681–689. Springer Berlin Heidelberg (2012)
[23] Spampinato, D.G., Elster, A.C.: Linear optimization on modern GPUs. In: Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2009), Rome, Italy (2009)
[24] Świetanowski, A.: A new steepest edge approximation for the simplex method for linear programming. Computational Optimization and and Applications, 10 (3), 271-281 (1998)
[25] Thomadakis, M.E.: Implementation and Evaluation of Primal and Dual Simplex Methods with Different Pivot-Selection Techniques in the LPBench Environment. A Research Report. Texas A&M University, Department of Computer Science (1994)
[26] Thomadakis, M.E., Liu, J.C.: An efficient steepest-edge simplex algorithm for SIMD computers. In Proc. 10th international conference on Supercomputing, 286-293 (1996)
[26] Vanderbei, R.J.: Linear Programming: Foundations and Extensions (2nd ed.). Kluwer Academic Publishers (2001)
[27] Wang, Z.: A modified version of the Edmonds-Fukuda algorithm for LP problems in the general form. Asia-Pacific J. Oper. Res. (1989)
[28] Zadeh, N.: What is the worst case behavior of the simplex algorithm? Technical report, Department of Operations Research, Stanford University (1980)
[29] Zhang, S.: On anti-cycling pivoting rules for the simplex method. Oper. Res. Lett., 10, 189-192 (1991)
[30] Ziegler, G.M.: Linear programming in oriented matroids. Technical Report No. 195, Institut für Mathematik, Universität Augsburg, Germany (1990)