A high performance dual revised simplex solver

J. A. J. Hall and Q. Huangfu*

School of Mathematics
and Maxwell Institute for Mathematical Sciences
The University of Edinburgh
Mayfield Road, Edinburgh EH9 3JZ
United Kingdom.

Technical Report ERGO-11-007†

12th May 2011

Abstract

When solving families of related linear programming (LP) problems and many classes of single LP problems, the simplex method is the preferred computational technique. Hitherto there has been no efficient parallel implementation of the simplex method that gives good speed-up on general, large sparse LP problems. This paper presents a variant of the dual simplex method and a prototype parallelisation scheme. The resulting implementation, ParISS, is efficient when run in serial and offers modest speed-up for a range of LP test problems.

Keywords: Linear programming, Dual revised simplex method, Parallel algorithms

*Email: J.A.J.Hall@ed.ac.uk

†For other papers in this series see http://www.maths.ed.ac.uk/ERGO/

1 Introduction

When solving families of related linear programming (LP) problems and many classes of single LP problems, the simplex method is the preferred computational technique in the academic and commercial worlds. There is, therefore, considerable motivation for exploring how the simplex method may exploit modern high performance computing (HPC) desktop architectures. This paper describes a variant of the dual simplex method that offers scope for exploiting such architectures. The relevant background is set out in Section 2 and a dual simplex variant, prototype parallelisation scheme and implementation as ParISS are described in Section 3. Computational results using ParISS are given in Section 4 and conclusions in Section 5.

2 Background

A linear programming (LP) problem in standard form is

$$\begin{array}{rl} \text{minimize} & \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{array} \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^m$. It may be assumed that the matrix A is of full rank.

In the simplex method, the indices of variables are partitioned into sets $\mathcal{B}$ corresponding to m basic variables $\boldsymbol{x}_B$, and $\mathcal{N}$ corresponding to $n-m$ nonbasic variables $\boldsymbol{x}_N$, such that the basis matrix B formed from the columns of A corresponding to $\mathcal{B}$ is nonsingular. The set $\mathcal{B}$ itself is conventionally referred to as the basis. The columns of A corresponding to $\mathcal{N}$ form the matrix N. The components of $\boldsymbol{c}$ corresponding to $\mathcal{B}$ and $\mathcal{N}$ are referred to as, respectively, the basic costs $\boldsymbol{c}_B$ and non-basic costs $\boldsymbol{c}_N$.

When each nonbasic variable is set to zero, the basic variables take the values $\hat{\boldsymbol{b}} = B^{-1}\boldsymbol{b}$ and if they are non-negative then the partition $\{\mathcal{B},\mathcal{N}\}$ is primal feasible. The *reduced costs* are $\hat{\boldsymbol{c}}_N^T = \boldsymbol{c}_N^T - \boldsymbol{c}_B^T B^{-1}N$ and if they are non-negative then the partition is dual feasible. Primal and dual feasibility is a necessary and sufficient condition for $\{\mathcal{B},\mathcal{N}\}$ to be optimal. The primal simplex method modifies primal feasible partitions until dual feasibility is achieved. Conversely, the dual simplex method modifies dual feasible partitions until primal feasibility is achieved.

A full discussion of the dual simplex method and modern techniques for its efficient implementation are given by Koberstein [11]. At the start of any iteration it is assumed that there is an invertible representation of B, that $\boldsymbol{x}_N = \boldsymbol{0}$, that values of $\hat{\boldsymbol{b}} = B^{-1}\boldsymbol{b}$ are known and not feasible, and that the values of $\hat{\boldsymbol{c}}_N^T$ are feasible. An outline of the computational components of a dual simplex iteration is given in Figure 1, where $\boldsymbol{e}_p$ is column p of the identity matrix. Although dual feasibility is assumed, the algorithm is readily adapted to state-of-the-art techniques for achieving dual feasibility, as presented by Koberstein and Suhl [12].

In CHUZR, the immediate quality of a candidate to leave the basis is the amount by which it is negative. Efficient dual simplex implementations weight this quality by (a measure of) the magnitude of the corresponding row of B^{-1}, the most popular being the steepest edge weight of Forrest and Goldfarb [4] which, for row p, is $s_p = \|B^{-1}\boldsymbol{e}_p\|_2$. To update these weights requires the pivotal column $\hat{\boldsymbol{a}}_q$ and $\boldsymbol{\tau} = B^{-1}\hat{\boldsymbol{a}}_q$. Although calculating $\boldsymbol{\tau}$ adds significantly to the cost

`CHUZR:` Scan $\hat{\boldsymbol{b}}$ for the row p of a good candidate to leave the basis.
`BTRAN:` Form $\boldsymbol{\pi}_p^T = \boldsymbol{e}_p^T B^{-1}$.
`PRICE:` Form the pivotal row $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}_p^T N$.
`CHUZC:` Scan the ratios $\hat{c}_j/\hat{a}_{pj}$ for a good candidate q to enter the basis.
Update $\hat{\boldsymbol{c}}_N^T := \hat{\boldsymbol{c}}_N^T - \beta\hat{\boldsymbol{a}}_p^T$, where $\beta = \hat{c}_q/\hat{a}_{pq}$.
`FTRAN:` Form the pivotal column $\hat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$, where $\boldsymbol{a}_q$ is column q of A.
Update $\hat{\boldsymbol{b}} := \hat{\boldsymbol{b}} - \alpha\hat{\boldsymbol{a}}_q$, where $\alpha = \hat{b}_p/\hat{a}_{pq}$.
`If` {growth in representation of B} `then`
`INVERT:` Form a new representation of B^{-1}.
`else`
`UPDATE:` Update the representation of B^{-1} corresponding to the basis change.
`end if`

Figure 1: Operations in an iteration of the dual revised simplex method

of an iteration, the investment usually more than pays off due to the steepest edge strategy reducing the number of iterations.

For the efficient implementation of the revised simplex method, the means by which B^{-1} is represented is of fundamental importance. It is based on an LU decomposition of some basis matrix B_0, determined by a sparsity-exploiting variant of Gaussian elimination as discussed, for example, by Tomlin [18] and Suhl and Suhl [17]. Invertible representations of subsequent basis matrices are obtained by updates until it is preferable on grounds of efficiency or numerical stability to form a new representation via Gaussian elimination. There are many approaches to updating the invertible representation of the basis matrix, the simplest of which is the product form update of Dantzig and Orchard-Hays [3].

2.1 Suboptimization

When in-core memory was severely restricted, a popular variant of the primal revised simplex method incorporated minor iterations of the standard (tableau) primal simplex method restricted to a small subset of the variables. This is described by Orchard-Hays [14] and is referred to as *multiple pricing*. Rosander [15] applied the concept to the dual simplex method, using the term *suboptimization*, performing minor iterations of the standard dual simplex method restricted to a small subset of the constraints. An outline of the computational components of a dual simplex iteration with suboptimization and steepest edge pricing is given in Figure 2.

The disadvantages of using suboptimization for the dual simplex method are that, after the first minor iteration, $\mathcal{P}$ may not contain the row of the best (global) candidate to leave the basis, and for some of the rows in $\mathcal{P}$ the candidate may no longer be attractive. Thus the number of iterations required to solve the LP problem may increase, and the work of computing some of the pivotal rows may be wasted.

2.2 Parallelising the simplex method

Previous attempts to develop simplex implementations with the aim of exploiting HPC architectures are reviewed by Hall [7]. Work on exploiting parallelism in the revised simplex method has

`CHUZR:` Scan $\hat{b}_p/s_p$ for $p = 1, \ldots, m$ to identify a set $\mathcal{P}$ of rows of good candidates to leave the basis.
`BTRAN:` Form $\boldsymbol{\pi}_p^T = \boldsymbol{e}_p^T B^{-1}$, $\forall\, p \in \mathcal{P}$.
`PRICE:` Form the pivotal row $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}_p^T N$, $\forall\, p \in \mathcal{P}$.
`Loop` {minor iterations}
    `CHUZR_MI:` Scan $\hat{\boldsymbol{b}}$ for the row $p \in \mathcal{P}$ of a good candidate to leave the basis.
        If p is not defined `End loop` {minor iterations}.
    `CHUZC:` Scan the ratios $\hat{c}_j/\hat{a}_{pj}$ for a good candidate q to enter the basis.
        Update $\hat{\boldsymbol{c}}_N^T := \hat{\boldsymbol{c}}_N^T - \beta\hat{\boldsymbol{a}}_p^T$, where $\beta = \hat{c}_q/\hat{a}_{pq}$.
    `UPDATE_MI:` Update $\mathcal{P} := \mathcal{P}\backslash\{p\}$ and $\hat{\boldsymbol{c}}_N^T := \hat{\boldsymbol{c}}_N^T - \beta\hat{\boldsymbol{a}}_p^T$, where $\beta = \hat{c}_q/\hat{a}_{pq}$.
        Update the rows $\hat{\boldsymbol{a}}_P^T$ and $\hat{\boldsymbol{b}}_P$.
`End loop` {minor iterations}
`For` {each basis change} `do`
    `FTRAN1:` Form $\hat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$, where $\boldsymbol{a}_q$ is column q of A.
        Update $\hat{\boldsymbol{b}} := \hat{\boldsymbol{b}} - \alpha\hat{\boldsymbol{a}}_q$, where $\alpha = \hat{b}_p/\hat{a}_{pq}$.
    `FTRAN2:` Form $\boldsymbol{\tau} = B^{-1}\hat{\boldsymbol{a}}_q$.
        Update s_p for $p = 1, \ldots, m$.
    `If` {growth in representation of B} `then`
       `INVERT:` Form a new representation of B^{-1}.
    `else`
       `UPDATE:` Update the representation of B^{-1} corresponding to the basis change.
    `end if`
`End do`

Figure 2: Operations in an iteration of the dual revised simplex method with suboptimization and steepest edge pricing

mainly been restricted to the primal simplex method, notably Forrest and Tomlin [5], Shu [16], Hall and McKinnon [8, 9] and Wunderling [19, 20]. Only Bixby and Martin [1] considered the dual simplex method, parallelising little more than the matrix-vector product $\boldsymbol{\pi}_p^T N$. This focus on the primal simplex method is understandable since it is the more natural variant to study and only since most of the work on parallel simplex was done has the dual simplex method become the preferred variant.

No parallel implementation of the simplex method has yet offered significantly better performance relative to an efficient serial simplex solver for general large scale sparse LP problems [7]. In two of the more successful implementations [8, 9], the limited speed-up came at the cost of unreliability due to numerical instability. Thus any performance improvement in the revised simplex method resulting from an efficient and reliable parallel implementation would be a significant achievement.

Several parallel implementations [8, 9, 19, 20] exploited multiple pricing within the primal revised simplex method. This had the attractive properties of independent computational components that could be overlapped on multiple processors, and minor iterations of the standard simplex method that offered immediate data parallelism. This paper considers the scope for parallelising the dual simplex method with suboptimization.

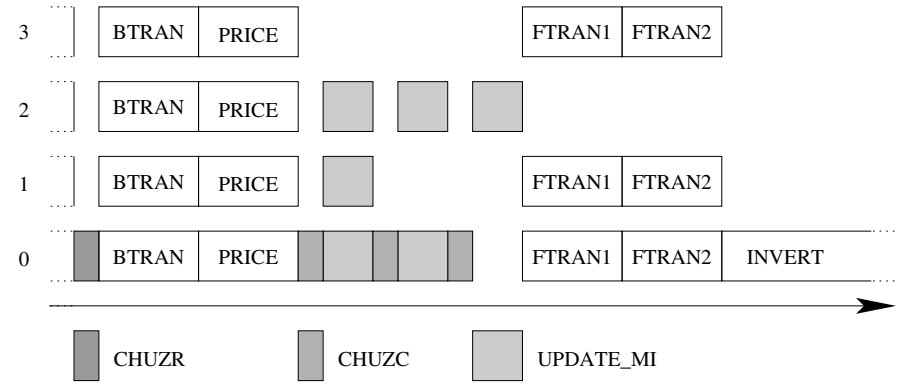

Figure 3: ParISS: a prototype parallel implementation of the dual revised simplex with suboptimization

3 A parallel scheme and its implementation

This section introduces a prototype parallelisation scheme for the dual revised simplex method with suboptimization and steepest edge pricing, together with the computational techniques underlying its implementation as the solver ParISS. The inherent strengths and weaknesses of both the parallel scheme and its implementation are discussed.

A parallelisation scheme

The following scheme exploits some, but by no means all, of the available task and data parallelism in the dual revised simplex method with suboptimization. It is discussed in detail below, taking the operations in Figure 2 in order. Its implementation, ParISS, assigns one tableau row to each available core and Figure 3 illustrates an example of the distribution of operations in the case of four cores.

Firstly, the relatively cheap CHUZR operation of selecting a set $\mathcal{P}$ of good candidates to leave the basis is executed on just one core. Following this, the multiple BTRAN ($\boldsymbol{\pi}_p^T = \boldsymbol{e}_p^T B^{-1}$), and PRICE ($\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}_p^T N$) operations for $p \in \mathcal{P}$ are distributed over all cores. Since minor iterations are performed with only a very small subset of rows, the CHUZR_MI operation is trivial so is performed by one core and not illustrated. The selection of the entering column performed by CHUZC is relatively cheap so is also performed on one core. The minor iteration is completed by UPDATE_MI, in which the data parallel update of the dual simplex tableau rows (that remain candidates) and the reduced costs $\hat{\boldsymbol{c}}_N^T$ is distributed over all cores. The trivial update of $\hat{\boldsymbol{b}}_P$ is performed by the core that will perform CHUZR_MI. Following the minor iterations, FTRAN operations $\hat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$ and $\boldsymbol{\tau} = B^{-1}\hat{\boldsymbol{a}}_q$ for each basis change are distributed over all cores. If INVERT is performed, it is executed serially, without overlapping any other computation.

Serial efficiency

In many earlier parallel implementations of the revised simplex method [7], the serial inefficiency of the computational components yielded greater scope for parallel efficiency but the resulting

implementation was wholly inferior to a good serial solver. For a parallel simplex solver to be of greatest practical value, it is important that its components are computationally efficient. Not only must they exploit the sparsity of the constraint matrix, but they should consider the more advanced techniques for exploiting hyper-sparsity identified by Hall and McKinnon [10] that have led to huge performance improvements for important classes of LP problems.

Implementation of the scheme and its parallel efficiency

The aim of the prototype implementation, ParISS, is to identify the immediate scope for parallelism when using efficient serial components. Like the underlying scheme, ParISS is not fully parallel efficient. In particular, for convenience, it identifies a "master" core that is used for computation when an alternative core might be preferable on grounds of data locality.

Figure 3 clearly illustrates the major parallel inefficiency in the scheme, that of INVERT being performed serially with no other overlapping computation. There is a similar, but less serious, source of parallel inefficiency when performing the very much cheaper operations CHUZR and CHUZC. Both of these perform comparison operations on each component of a full-length vector, so it may be significantly more efficient to distribute the data over multiple cores and accumulate the results from each. In ParISS, CHUZC is performed on the master core rather than the core where the pivotal row is likely to be available in cache.

Another source of parallel inefficiency in the scheme occurs if not all the candidates in $\mathcal{P}$ yield a basis change, in which case the number of FTRAN operations may not be a multiple of the number of available cores. In ParISS, a core performs both FTRAN operations if and only if its (single) tableau row is pivotal. Thus, if the candidate variable in this row does not leave the basis, the core performs no FTRAN operations. Figure 3 illustrates this in the case where the last candidate in $\mathcal{P}$ does not leave the basis. With hindsight, this tableau row is computed and updated unnecessarily but this is a serial consequence of suboptimization.

The idealised Gantt chart assumes that all BTRAN, PRICE and FTRAN operations have the same computational cost but this is not so in practice when they are performed efficiently. Immediate variance results from the fact that for FTRAN1 ($\hat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$) vector $\boldsymbol{a}_q$ is a column of the (sparse) constraint matrix whereas, for FTRAN2 ($\boldsymbol{\tau} = B^{-1}\hat{\boldsymbol{a}}_q$), $\hat{\boldsymbol{a}}_q$ may be a full vector. When exploiting hyper-sparsity, the variance increases further. In ParISS, hyper-sparsity is exploited partially during BTRAN and FTRAN, and fully during PRICE, with A being held row-wise and combined according to the nonzeros in $\boldsymbol{\pi}_p$. Thus ParISS can also be expected to suffer from parallel inefficiency due to load imbalance.

4 Results

ParISS, a prototype implementation of the dual revised simplex with suboptimization has been developed in C++ with OpenMP using the techniques discussed in Section 3. Using the Intel C++ compiler, the code was tested on a dual quad-core AMD Opteron 2378 system.

Experiments were performed using problems from the standard Netlib [6] and Kennington [2] test sets of LP problems. Results are given in Table 1 for a subset of problems chosen as follows. Of the original 114 problems, many are unrealistically small so the 84 that ParISS solved in less

				Speed-up				
				Relative to 1 core			Relative to `Clp`	
Problem	Rows	Columns	Entries	2 cores	4 cores	8 cores	1 cores	8 cores
25fv47	822	1571	11127	1.18	1.07	0.53	0.51	0.27
80bau3b	2263	9799	29063	0.94	1.03	0.85	0.21	0.18
cre-b	9649	72447	328542	0.86	1.27	1.07	1.13	1.21
cre-d	8927	69980	312626	1.16	1.33	1.68	0.90	1.51
degen3	1504	1818	26230	1.28	1.19	1.12	0.31	0.35
fit2p	3001	13525	60784	0.96	0.92	0.94	0.44	0.41
osa-14	2338	52460	367220	0.93	0.84	1.01	0.17	0.17
osa-30	4351	100024	700160	1.06	1.00	0.90	0.20	0.18
pds-06	9882	28655	82269	1.62	2.19	3.06	0.47	1.43
pds-10	16559	48763	140063	1.27	1.83	1.97	0.51	1.00
qap8	913	1632	8304	1.37	1.24	1.72	0.31	0.54
stocfor3	16676	15695	74004	1.80	2.57	3.39	0.13	0.46
truss	1001	8806	36642	0.98	1.08	1.10	0.46	0.50
Mean				1.16	1.28	1.30	0.37	0.48

Table 1: Speed-up of ParISS on up to 8 cores and performance relative to `Clp`

than one second were discounted. Of the remaining 30, ParISS failed to solve 14 in one or more of the runs using 1, 2, 4 and 8 cores. To ensure that results are given for problems that ParISS solves efficiently, observe that when a single core is used only one tableau row is computed so ParISS executes the standard serial revised dual simplex algorithm. Thus its efficiency was assessed by comparing ParISS on one core with version 1.06 of the COIN-OR [13] (serial) dual simplex solver `Clp`. For the remaining 16 problems ParISS was more than ten times slower than CLP on three, so these problems were also discounted. Thus the results in Table 1 are for the 13 LP test problems for which ParISS required at least one second of CPU on one core but were solved efficiently on one core and successfully on 2, 4 and 8 cores.

As the results in columns 5–7 of Table 1 show, some speed-up was obtained for all but two of the problems and the (geometric) mean speed-up was about 30% on 4 and 8 cores. Column 8 indicates the speed of ParISS on one core relative to `Clp`: ParISS was faster for only one problem and was slower by a factor of 2.7 on average. However, using 8 cores, ParISS was at least as fast as `Clp` for four problems, and 2.1 times slower on average.

5 Conclusions

For a prototype solver with the limited parallel efficiency recognised in Section 3, the results in Table 1 are very encouraging, both in terms of speed-up and performance relative to `Clp`. They demonstrate that it is possible to get a worthwhile performance improvement in an efficient implementation of the dual revised simplex method by exploiting the scope for parallelism offered by suboptimization when solving a range of sparse LP problems. Enhancements to the parallel efficiency of the scheme and its implementation, together with improved serial performance of its components are all the subject of current research.

References

[1] R. E. Bixby and A. Martin, *Parallelizing the dual simplex method*, INFORMS Journal

on Computing, 12 (2000), pp. 45–56.

[2] W. J. Carolan, J. E. Hill, J. L. Kennington, S. Niemi, and S. J. Wichmann, *An empirical evaluation of the KORBX algorithms for military airlift applications*, Operations Research, 38 (1990), pp. 240–248.

[3] G. B. Dantzig and W. Orchard-Hays, *The product form for the inverse in the simplex method*, Math. Comp., 8 (1954), pp. 64–67.

[4] J. J. Forrest and D. Goldfarb, *Steepest-edge simplex algorithms for linear programming*, Mathematical Programming, 57 (1992), pp. 341–374.

[5] J. J. H. Forrest and J. A. Tomlin, *Vector processing in the simplex and interior methods for linear programming*, Annals of Operations Research, 22 (1990), pp. 71–100.

[6] D. M. Gay, *Electronic mail distribution of linear programming test problems*, Mathematical Programming Society COAL Newsletter, 13 (1985), pp. 10–12.

[7] J. A. J. Hall, *Towards a practical parallelisation of the simplex method*, Computational Management Science, 7 (2010), pp. 139–170.

[8] J. A. J. Hall and K. I. M. McKinnon, *PARSMI, a parallel revised simplex algorithm incorporating minor iterations and Devex pricing*, in Applied Parallel Computing, J. Waśniewski, J. Dongarra, K. Madsen, and D. Olesen, eds., vol. 1184 of Lecture Notes in Computer Science, Springer, 1996, pp. 67–76.

[9] ——, *ASYNPLEX, an asynchronous parallel revised simplex method algorithm*, Annals of Operations Research, 81 (1998), pp. 27–49.

[10] ——, *Hyper-sparsity in the revised simplex method and how to exploit it*, Computational Optimization and Applications, 32 (2005), pp. 259–283.

[11] A. Koberstein, *Progress in the dual simplex algorithm for solving large scale LP problems: techniques for a fast and stable implementation*, Computational Optimization and Applications, 41 (2008), pp. 185–204.

[12] A. Koberstein and U. H. Suhl, *Progress in the dual simplex method for large scale LP problems: practical dual phase 1 algorithms*, Computational Optimization and Applications, 37 (2007), pp. 49–65.

[13] R. Lougee-Heimer *et al.*, *The COIN-OR initiative: Open source accelerates operations research progress*, ORMS Today, 28 (2001), pp. 20–22.

[14] W. Orchard-Hays, *Advanced Linear programming computing techniques*, McGraw-Hill, New York, 1968.

[15] R. R. Rosander, *Multiple pricing and suboptimization in dual linear programming algorithms*, Mathematical Programming Study, 4 (1975), pp. 108–117.

[16] W. Shu, *Parallel implementation of a sparse simplex algorithm on MIMD distributed memory computers*, Journal of Parallel and Distributed Computing, 31 (1995), pp. 25–40.

[17] U. H. Suhl and L. M. Suhl, *Computing sparse LU factorizations for large-scale linear programming bases*, ORSA Journal on Computing, 2 (1990), pp. 325–335.

[18] J. A. Tomlin, *Pivoting for size and sparsity in linear programming inversion routines*, J. Inst. Maths. Applics, 10 (1972), pp. 289–295.

[19] R. Wunderling, *Paralleler und objektorientierter simplex*, Tech. Rep. TR-96-09, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1996.

[20] ——, *Parallelizing the simplex algorithm.* ILAY Workshop on Linear Algebra in Optimzation, Albi, April 1996.