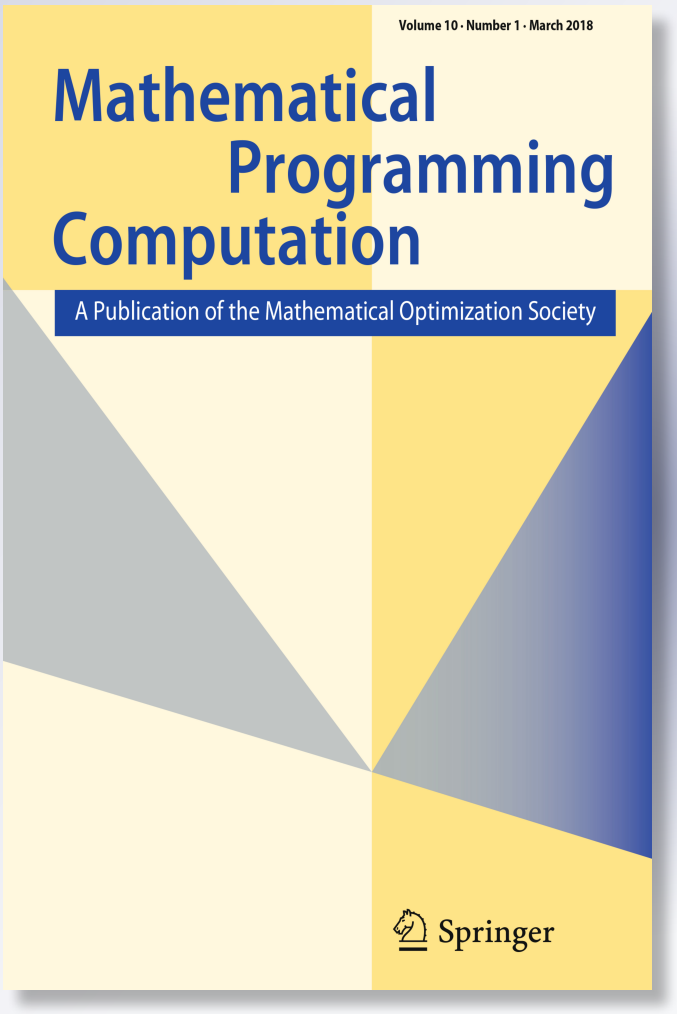
Parallelizing the dual revised simplex method

Q. Huangfu & J. A. J. Hall

Mathematical Programming Computation
A Publication of the Mathematical Optimization Society

ISSN 1867-2949
Volume 10
Number 1

Math. Prog. Comp. (2018) 10:119-142
DOI 10.1007/s12532-017-0130-5



Springer

Math. Prog. Comp. (2018) 10:119–142
https://doi.org/10.1007/s12532-017-0130-5

FULL LENGTH PAPER

Parallelizing the dual revised simplex method

Q. Huangfu[1] · **J. A. J. Hall**[2]

Received: 6 March 2015 / Accepted: 10 February 2016 / Published online: 14 December 2017

Abstract This paper introduces the design and implementation of two parallel dual simplex solvers for general large scale sparse linear programming problems. One approach, called PAMI, extends a relatively unknown pivoting strategy called suboptimization and exploits parallelism across multiple iterations. The other, called SIP, exploits purely single iteration parallelism by overlapping computational components when possible. Computational results show that the performance of PAMI is superior to that of the leading open-source simplex solver, and that SIP complements PAMI in achieving speedup when PAMI results in slowdown. One of the authors has implemented the techniques underlying PAMI within the FICO Xpress simplex solver and this paper presents computational results demonstrating their value. In developing the first parallel revised simplex solver of general utility, this work represents a significant achievement in computational optimization.

Keywords Linear programming · Revised simplex method · Parallel computing

Mathematics Subject Classification 90C05 · 90C06 · 65K05

1 Introduction

Linear programming (LP) has been used widely and successfully in many practical areas since the introduction of the simplex method in the 1950s. Although an alternative

✉ J. A. J. Hall
J.A.J.Hall@ed.ac.uk

1 FICO House, International Square, Starley Way, Birmingham B37 7GN, UK

2 School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh, James Clerk Maxwell Building, Peter Guthrie Tait Road, Edinburgh EH9 3FD, UK

solution technique, the interior point method (IPM), has become competitive and popular since the 1980s, the dual revised simplex method is frequently preferred, particularly when families of related problems are to be solved.

The standard simplex method implements the simplex algorithm via a rectangular tableau but is very inefficient when applied to sparse LP problems. For such problems the revised simplex method is preferred since it permits the (hyper-)sparsity of the problem to be exploited. This is achieved using techniques for factoring sparse matrices and solving hyper-sparse linear systems. Also important for the dual revised simplex method are advanced algorithmic variants introduced in the 1990s, particularly dual steepest-edge (DSE) pricing and the bound flipping ratio test (BFRT). These led to dramatic performance improvements and are key reasons for the dual simplex algorithm being preferred.

A review of past work on parallelising the simplex method is given by Hall [10]. The standard simplex method has been parallelised many times and generally achieves good speedup, with factors ranging from tens to up to a thousand. However, without using expensive parallel computing resources, its performance on sparse LP problems is inferior to a good sequential implementation of the revised simplex method. The standard simplex method is also unstable numerically. Parallelisation of the revised simplex method has been considered relatively little and there has been less success in terms of speedup. Indeed, since scalable speedup for general large sparse LP problems appears unachievable, the revised simplex method has been considered unsuitable for parallelisation. However, since it corresponds to the computationally efficient serial technique, any improvement in performance due to exploiting parallelism in the revised simplex method is a worthwhile goal.

Two main factors motivated the work in this paper to develop a parallelisation of the dual revised simplex method for standard desktop architectures. Firstly, although dual simplex implementations are now generally preferred, almost all the work by others on parallel simplex has been restricted to the primal algorithm, the only published work on dual simplex parallelisation known to the authors being due to Bixby and Martin [1]. Although it appeared in the early 2000s, their implementation included neither the BFRT nor hyper-sparse linear system solution techniques so there is immediate scope to extend their work. Secondly, in the past, parallel implementations generally used dedicated high performance computers to achieve the best performance. Now, when every desktop computer is a multi-core machine, any speedup is desirable in terms of solution time reduction for daily use. Thus we have used a relatively standard architecture to perform computational experiments.

A worthwhile simplex parallelisation should be based on a good sequential simplex solver. Although there are many public domain simplex implementations, they are either too complicated to be used as a foundation for a parallel solver or too inefficient for any parallelisation to be worthwhile. Thus the authors have implemented a sequential dual simplex solver (`hsol`) from scratch. It incorporates sparse LU factorization, hyper-sparse linear system solution techniques, efficient approaches to updating LU factors and sophisticated dual revised simplex pivoting rules. Based on components of this sequential solver, two dual simplex parallel solvers (`pami` and `sip`) have been designed and developed.

Section 2 introduces the necessary background, Sects. 3 and 4 detail the design of `pami` and `sip` respectively and Sect. 5 presents numerical results and performance analysis. Conclusions are given in Sect. 6.

2 Background

The simplex method has been under development for more than 60 years, during which time many important algorithmic variants have enhanced the performance of simplex implementations. As a result, for novel computational developments to be of value they must be tested within an efficient implementation or good reasons given why they are applicable in such an environment. Any development which is only effective in the context of an inefficient implementation is not worthy of attention.

This section introduces all the necessary background knowledge for developing the parallel dual simplex solvers. Section 2.1 introduces the computational form of LP problems and the concept of primal and dual feasibility. Section 2.2 describes the regular dual simplex method algorithm and then details its key enhancements and major computational components. Section 2.3 introduces suboptimization, a relatively unknown dual simplex variant which is the starting point for the `pami` parallelisation in Sect. 3. Section 2.4 briefly reviews several existing simplex update approaches which are key to the efficiency of the parallel schemes.

2.1 Linear programming problems

A linear programming (LP) problem in general computational form is

$$\text{minimize } f = \boldsymbol{c}^T\boldsymbol{x} \quad \text{subject to } A\boldsymbol{x} = \boldsymbol{0} \text{ and } \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, \tag{1}$$

where $A \in \mathbb{R}^{m\times n}$ is the coefficient matrix and $\boldsymbol{x}$, $\boldsymbol{c}$, $\boldsymbol{l}$ and $\boldsymbol{u} \in \mathbb{R}^m$ are, respectively, the variable vector, cost vector and (lower and upper) bound vectors. Bounds on the constraints are incorporated into $\boldsymbol{l}$ and $\boldsymbol{u}$ via an identity submatrix of A. Thus it may be assumed that $m < n$ and that A is of full rank.

As A is of full rank, it is always possible to identify a non-singular basis partition $B \in \mathbb{R}^{m\times m}$ consisting of m linearly independent columns of A, with the remaining columns of A forming the matrix N. The variables are partitioned accordingly into basic variables $\boldsymbol{x}_B$ and nonbasic variables $\boldsymbol{x}_N$, so $A\boldsymbol{x} = B\boldsymbol{x}_B + N\boldsymbol{x}_N = \boldsymbol{0}$, and the cost vector is partitioned into basic costs $\boldsymbol{c}_B$ and nonbasic costs $\boldsymbol{c}_N$, so $f = \boldsymbol{c}_B^T\boldsymbol{x}_B + \boldsymbol{c}_N^T\boldsymbol{x}_N$. The indices of the basic and nonbasic variables form sets $\mathcal{B}$ and $\mathcal{N}$ respectively.

In the simplex algorithm, the values of the (primal) variables are defined by setting each nonbasic variable to one of its finite bounds and computing the values of the basic variables as $\boldsymbol{x}_B = -B^{-1}N\boldsymbol{x}_N$. The values of the dual variables (reduced costs) are defined as $\widehat{\boldsymbol{c}}_N^T = \boldsymbol{c}_N^T - \boldsymbol{c}_B^T B^{-1}N$. When $\boldsymbol{l}_B \leq \boldsymbol{x}_B \leq \boldsymbol{u}_B$ holds, the basis is said to be primal feasible. Otherwise, the primal infeasibility for each basic variable $i \in \mathcal{B}$ is defined as

$$\Delta x_i = \begin{cases} l_i - x_i & \text{if } x_i < l_i \\ x_i - u_i & \text{if } x_i > u_i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

If the following condition holds for all $j \in \mathcal{N}$ such that $l_j \neq u_j$

$$\widehat{c}_j \geq 0 \ (x_j = l_j), \quad \widehat{c}_j \leq 0 \ (x_j = u_j) \tag{3}$$

then the basis is said to be dual feasible. It can be proved that if a basis is both primal and dual feasible then it yields an optimal solution to the LP problem.

2.2 Dual revised simplex method

The dual simplex algorithm solves an LP problem iteratively by seeking primal feasibility while maintaining dual feasibility. Starting from a dual feasible basis, each iteration of the dual simplex algorithm can be summarised as three major operations.

1. *Optimality test*. In a component known as CHUZR, choose the index $p \in \mathcal{B}$ of a good primal infeasible variable to leave the basis. If no such variable can be chosen, the LP problem is solved to optimality.
2. *Ratio test*. In a component known as CHUZC, choose the index $q \in \mathcal{N}$ of a good nonbasic variable to enter the basis so that, within the new partition, $\widehat{c}_q$ is zeroed whilst $\widehat{c}_p$ and other nonbasic variables remain dual feasible. This is achieved via a ratio test with $\widehat{\boldsymbol{c}}_N^T$ and $\widehat{\boldsymbol{a}}_p^T$, where $\widehat{\boldsymbol{a}}_p^T$ is row p of the reduced coefficient matrix $\widehat{A} = B^{-1}A$.
3. *Updating*. The basis is updated by interchanging indices p and q between sets $\mathcal{B}$ and $\mathcal{N}$, with corresponding updates of the values of the primal variables $\boldsymbol{x}_B$ using $\widehat{\boldsymbol{a}}_q$ (being column q of $\widehat{A}$) and dual variables $\widehat{\boldsymbol{c}}_N^T$ using $\widehat{\boldsymbol{a}}_p^T$, as well as other components as discussed below.

What defines the revised simplex method is a representation of the basis inverse B^{-1} to permit rows and columns of the reduced coefficient matrix $\widehat{A} = B^{-1}A$ to be computed by solving linear systems. The operation to compute the representation of B^{-1} directly is referred to as INVERT and is generally achieved via sparsity-exploiting LU factorization. At the end of each simplex iteration the representation of B^{-1} is updated until it is computationally advantageous or numerically necessary to compute a fresh representation directly. The computational component which performs the update of B^{-1} is referred to as UPDATE-FACTOR. Efficient approaches for updating B^{-1} are summarised in Sect. 2.4.

For many sparse LP problems the matrix B^{-1} is dense, so solutions of linear systems involving B or B^T can be expected to be dense even when, as is typically the case in the revised simplex method, the RHS is sparse. However, for some classes of LP problem the solutions of such systems are typically sparse. This phenomenon, and techniques for exploiting it in the simplex method, was identified by Hall and McKinnon [13] and is referred to as hyper-sparsity. This advanced technique has been incorporated throughout the design and development of the new parallel dual simplex solvers.

The remainder of this section introduces advanced algorithmic components of the dual simplex method.

2.2.1 Optimality test

In the optimality test, a modern dual simplex implementation adopts two important enhancements. The first is the dual steepest-edge (DSE) algorithm [6] which chooses the basic variable with greatest weighted infeasibility as the leaving variable. This variable has index

$$p = \arg\max_i \frac{\Delta x_i}{||\widehat{\boldsymbol{e}}_i^T||_2}.$$

For each basic variable $i \in \mathcal{B}$, the associated DSE weight w_i is defined as the 2-norm of row i of B^{-1} so $w_i = ||\widehat{\boldsymbol{e}}_i^T||_2 = ||\boldsymbol{e}_i^T B^{-1}||_2$. The weighted infeasibility $\alpha_i = \Delta x_i / w_i$ is referred to as the attractiveness of a basic variable. The DSE weight is updated at the end of the simplex iteration.

The second enhancement of the optimality test is the hyper-sparse candidate selection technique originally proposed for column selection in the primal simplex method [13]. This maintains a short list of the most attractive variables and is more efficient for large and sparse LP problems since it avoids repeatedly searching the less attractive choices. This technique has been adapted for the dual simplex row selection component of `hsol`.

2.2.2 Ratio test

In the ratio test, the updated pivotal row $\widehat{\boldsymbol{a}}_p^T$ is obtained by computing $\widehat{\boldsymbol{e}}_p^T = \boldsymbol{e}_p^T B^{-1}$ and then forming the matrix vector product $\widehat{\boldsymbol{a}}_p^T = \widehat{\boldsymbol{e}}_p^T A$. These two computational components are referred to as BTRAN and SPMV respectively.

The dual ratio test (CHUZC) is enhanced by the Harris two-pass ratio test [14] and bound-flipping ratio test (BFRT) [8]. Details of how to apply these two techniques are set out by Koberstein [18].

For the purpose of this report, advanced CHUZC can be viewed as having two stages, an initial stage CHUZC1 which simply accumulates all candidate nonbasic variables and then a recursive selection stage CHUZC2 to choose the entering variable q from within this set of candidates using BFRT and the Harris two-pass ratio test. CHUZC also determines the primal step θ_p and dual step θ_q, being the changes to the primal basic variable p and dual variable q respectively. Following a successful BFRT, CHUZC also yields an index set $\mathcal{F}$ of any primal variables which have flipped from one bound to the other.

2.2.3 Updating

In the updating operation, besides UPDATE-FACTOR, several vectors are updated. Update of the basic primal variables $\boldsymbol{x}_B$ (UPDATE-PRIMAL) is achieved using θ_p and $\widehat{\boldsymbol{a}}_q$, where $\widehat{\boldsymbol{a}}_q$ is computed by an operation $\widehat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$ known as FTRAN. Update of

Table 1 Major components of the dual revised simplex method and their percentage of overall solution time

Components	Brief description	Percentage
INVERT	Recompute B^{-1}	13.3
UPDATE-FACTOR	Update basis inverse B_k^{-1} to B_{k+1}^{-1}	2.3
CHUZR	Choose leaving variable p	2.9
BTRAN	Solve for $\widehat{\boldsymbol{e}}_p^T = \boldsymbol{e}_p^T B^{-1}$	8.7
SPMV	Compute $\widehat{\boldsymbol{a}}_p^T = \widehat{\boldsymbol{e}}_p^T A$	18.4
CHUZC1	Collect valid ratio test candidates	7.3
CHUZC2	Search for entering variable p	1.5
FTRAN	Solve for $\widehat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$	10.8
FTRAN-BFRT	Solve for $\widehat{\boldsymbol{a}}_F = B^{-1}\boldsymbol{a}_F$	3.5
FTRAN-DSE	Solve for $\boldsymbol{\tau} = B^{-1}\widehat{\boldsymbol{e}}_p$	26.4
UPDATE-DUAL	Update $\widehat{\boldsymbol{c}}^T$ using $\widehat{\boldsymbol{a}}_p^T$	
UPDATE-PRIMAL	Update $\boldsymbol{x}_B$ using $\widehat{\boldsymbol{a}}_q$ or $\widehat{\boldsymbol{a}}_F$	4.8
UPDATE-WEIGHT	Update DSE weight using $\widehat{\boldsymbol{a}}_q$ and $\boldsymbol{\tau}$	

the dual variables $\widehat{\boldsymbol{c}}_N^T$ (UPDATE-DUAL) is achieved using θ_q and $\widehat{\boldsymbol{a}}_p^T$. The update of the DSE weights is given by

$$\begin{aligned} w_p &:= w_p/\widehat{a}_{pq}^2 \\ w_i &:= w_i - 2(\widehat{a}_{iq}/\widehat{a}_{pq})\tau_i + (\widehat{a}_{iq}/\widehat{a}_{pq})^2 w_p \, i \neq p \end{aligned}$$

This requires both the FTRAN result $\widehat{\boldsymbol{a}}_q$ and $\boldsymbol{\tau} = B^{-1}\widehat{\boldsymbol{e}}_p$. The latter is obtained by another FTRAN type operation, known as FTRAN-DSE.

Following a BFRT ratio test, if $\mathcal{F}$ is not empty, then all the variables with indices in $\mathcal{F}$ are flipped, and the primal basic solution $\boldsymbol{x}_B$ is further updated (another UPDATE-PRIMAL) by the result of the FTRAN-BFRT operation $\widehat{\boldsymbol{a}}_F = B^{-1}\boldsymbol{a}_F$, where $\boldsymbol{a}_F$ is a linear combination of the constraint columns for the variables in $\mathcal{F}$.

2.2.4 Scope for parallelisation

The computational components identified above are summarised in Table 1. This also gives the average contribution to solution time for the LP test set used in Sect. 5.

There is immediate scope for data parallelisation within CHUZR, SPMV, CHUZC and most of the update operations since they require independent operations for each (nonzero) component of a vector. Exploiting such parallelisation in SPMV and CHUZC has been reported by Bixby and Martin [1] who achieve speedup on a small group of LP problems with relatively expensive SPMV operations. The scope for task parallelism by overlapping FTRAN and FTRAN-DSE was considered by Bixby and Martin but rejected as being disadvantageous computationally. Another notable development extending the ideas of Bixby and Martin [1] is `Aboca`, which was introduced by Forrest at a conference in 2012 [4]. Other than making the source code available [5], the authors believe that there is no published reference to this work.

2.3 Dual suboptimization

Suboptimization is one of the oldest variants of the revised simplex method and consists of a major-minor iteration scheme. Within the primal revised simplex method, suboptimization performs minor iterations of the standard primal simplex method using small subsets of columns from the reduced coefficient matrix $\widehat{A} = B^{-1}A$. Suboptimization for the dual simplex method was first set out by Rosander [21] but no practical implementation has been reported. It performs minor operations of the standard dual simplex method, applied to small subsets of rows from $\widehat{A}$.

1. *Major optimality test*. Choose index set $\mathcal{P} \subseteq \mathcal{B}$ of primal infeasible basic variables as potential leaving variables. If no such indices can be chosen, the LP problem has been solved to optimality.
2. *Minor initialisation*. For each $p \in \mathcal{P}$, compute $\widehat{\boldsymbol{e}}_p^T = \boldsymbol{e}_p^T B^{-1}$.
3. *Minor iterations*.
   (a) *Minor optimality test*. Choose and remove a primal infeasible variable p from $\mathcal{P}$. If no such variable can be chosen, the minor iterations are terminated.
   (b) *Minor ratio test*. As in the regular ratio test, compute $\widehat{\boldsymbol{a}}_p^T = \widehat{\boldsymbol{e}}_p^T A$ (SPMV) then identify an entering variable q.
   (c) *Minor update*. Update primal variables for the remaining candidates in set $\mathcal{P}$ only ($\boldsymbol{x}_\mathcal{P}$) and update all dual variables $\widehat{\boldsymbol{c}}_N$.
4. *Major update*. For the pivotal sequence identified during the minor iterations, update the primal basic variables, DSE weights and representation of B^{-1}.

Originally, suboptimization was proposed as a pivoting scheme with the aim of achieving better pivot choices and advantageous data affinity. In modern revised simplex implementations, the DSE and BFRT are together regarded as the best pivotal rules and the idea of suboptimization has been largely forgotten.

However, in terms of parallelisation, suboptimization is attractive because it provides more scope for parallelisation. For the primal simplex algorithm, suboptimization underpinned the work of Hall and McKinnon [11,12]. As discussed by Hall [10], Wunderling [22] also experimented with suboptimization for the primal simplex method. For dual suboptimization the major initialisation requires s BTRAN operations, where $s = |\mathcal{P}|$. Following $t \leq s$ minor iterations, the major update requires t FTRAN operations, t FTRAN-DSE operations and up to t FTRAN-BFRT operations. The detailed design of the parallelisation scheme based on suboptimization is discussed in Sect. 3.

2.4 Simplex update techniques

Updating the basis inverse B_k^{-1} to B_{k+1}^{-1} after the basis change $B_{k+1} = B_k + (\boldsymbol{a}_q - B\boldsymbol{e}_p)\boldsymbol{e}_p^T$ is a crucial component of revised simplex method implementations. The standard choices are the relatively simple product form (PF) update [20] or the efficient Forrest–Tomlin (FT) update [7]. A comprehensive report on simplex update techniques is given by Elble and Sahinidis [3] and novel techniques, some motivated by the design and development of `pami`, are described by Huangfu and Hall [16]. For the purpose of this report, the features of all relevant update methods are summarised as follows.

- The *product form* (PF) update uses the FTRAN result $\widehat{\boldsymbol{a}}_q$, yielding $B_{k+1}^{-1} = E^{-1}B_k^{-1}$, where the inverse of $E = I + (\widehat{\boldsymbol{a}}_q - \boldsymbol{e}_p)\boldsymbol{e}_p^T$, is readily available.
- The *Forrest–Tomlin* (FT) update assumes $B_k = L_k U_k$ and uses both the partial FTRAN result $\tilde{\boldsymbol{a}}_q = L_k^{-1}\boldsymbol{a}_q$ and partial BTRAN result $\tilde{\boldsymbol{e}}_p^T = \boldsymbol{e}_p^T U_k^{-1}$ to modify U_k and augment L_k.
- The *alternate product form* (APF) update [16] uses the BTRAN result $\widehat{\boldsymbol{e}}_p^T$ so that $B_{k+1}^{-1} = B_k^{-1}T^{-1}$, where $T = I + (\boldsymbol{a}_q - \boldsymbol{a}_{p'})\widehat{\boldsymbol{e}}_p^T$ and $\boldsymbol{a}_{p'}$ is column p of B. Again, T is readily inverted.
- Following suboptimization, the *collective Forrest–Tomlin* (CFT) update [16] updates B_k^{-1} to B_{k+t}^{-1} directly, using partial results obtained with B_k^{-1} which are required for simplex iterations.

Although the direct update of the basis inverse from B_k^{-1} to B_{k+t}^{-1} can be achieved easily via the PF or APF update, in terms of efficiency for future simplex iterations, the collective FT update is preferred to the PF and APF updates. The value of the APF update within `pami` is indicated in Sect. 3.

3 Parallelism across multiple iterations

This section introduces the design and implementation of the parallel dual simplex scheme, `pami`. It extends the suboptimization scheme of Rosander [21], incorporating (serial) algorithmic techniques and exploiting parallelism across multiple iterations.

The concept of `pami` was introduced by Hall and Huangfu [9], where it was referred to as ParISS. This prototype implementation was based on the PF update and was relatively unsophisticated, both algorithmically and computationally. Subsequent revisions and refinements, incorporating the advanced algorithmic techniques outlined in Sect. 2 as well as FT updates and some novel features introduced in this section, have yielded a very much more sophisticated and efficient implementation. Specifically, our implementation of `pami` out-performs ParISS by almost an order of magnitude in serial and to achieve the speed-up demonstrated in Sect. 5 has required new levels of task parallelism and parallel algorithmic control techniques described in Sects. 3.2 and 3.3, in addition to the linear algebra techniques introduced by Huangfu and Hall in [16].

Section 3.1 provides an overview of the parallelisation scheme of `pami` and Sect. 3.2 details the task parallel FTRAN operations in the major update stage and how to simplify it. A novel candidate quality control scheme for the minor optimality test is discussed in Sect. 3.3.

3.1 Overview of the `pami` framework

This section details the general `pami` parallelisation scheme with reference to the suboptimization framework introduced in Sect. 2.3.

3.1.1 Major optimality test

The major optimality test involves only major CHUZR operations in which s candidates are chosen (if possible) using the DSE framework. In `pami` the value of s is the number of processors being used. It is a vector-based operation which can be easily parallelised, although its overall computational cost is not significant since it is only performed once per major operation. However, the algorithmic design of CHUZR is important and Sect. 3.3 discusses it in detail.

3.1.2 Minor initialisation

The minor initialisation step computes the BTRAN results for (up to s) potential candidates to leave the basis. This is the first of the task parallelisation opportunities provided by the suboptimization framework.

3.1.3 Minor iterations

There are three main operations in the minor iterations.

(a) Minor CHUZR simply chooses the best candidates from the set $\mathcal{P}$. Since this is computationally trivial, exploitation of parallelism is not considered. However, consideration must be given to the likelihood that the attractiveness of the best remaining candidate in $\mathcal{P}$ has dropped significantly. In such circumstances, it may not be desirable to allow this variable to leave the basis. This consideration leads to a candidate quality control scheme introduced in Sect. 3.3.

(b) The minor ratio test is a major source of parallelisation and performance improvement. Since the BTRAN result is known (see below), the minor ratio test consists of SPMV, CHUZC1 and CHUZC2. The SPMV operation is a sparse matrix-vector product and CHUZC1 is a one-pass selection based on the result of SPMV. In the actual implementation, they can share one parallel initialisation. On the other hand, CHUZC2 often involves multiple iterations of recursive selection which, if exploiting parallelism, requires many synchronisation operations. According to the component profiling in Table 1, CHUZC2 is a relative cheap operation thus, in `pami`, it is not parallelised. Data parallelism is exploited in SPMV and CHUZC1 by partitioning the variables across the processors before any simplex iterations are performed. This is done randomly with the aim of achieving load balance in SPMV.

(c) The minor update consists of the update of dual variables and the update of BTRAN results. The former is performed in the minor update because the dual variables are required in the ratio test of the next minor iteration. It is simply a vector addition and represents immediate data parallelism. The updated BTRAN result $\boldsymbol{e}_i^T B_{k+1}^{-1}$ is obtained by observing that it is given by the APF update as $\boldsymbol{e}_i^T B_k^{-1} T^{-1} = \widehat{\boldsymbol{e}}_i^T T^{-1}$. Exploiting the structure of T^{-1} yields a vector operation which may be parallelised. After the BTRAN results have been updated, the DSE weights of the remaining candidates are recomputed directly at little cost.

3.1.4 Major update

Following t minor iterations, the major update step concludes the major iteration. It consists of three types of operation: up to $3t$ FTRAN operations (including FTRAN-DSE and FTRAN-BFRT), the vector-based update of primal variables and DSE weights, and update of the basis inverse representation.

The number of FTRAN operations cannot be fixed *a priori* since it depends on the number of minor iterations and the number involving a non-trivial BFRT. A simplification of the group of FTRANs is introduced in 3.2.

The updates of all primal variables and DSE weights (given the particular vector $\boldsymbol{\tau} = B^{-1}\widehat{\boldsymbol{e}}_p$) are vector-based data parallel operations.

The update of the invertible representation of B is performed using the collective FT update unless it is desirable or necessary to perform INVERT to reinvert B. Note that both of these operations are performed serially. Although the (collective) FT update is relatively cheap (see Table 1), so has little impact on performance, there is significant processor idleness during the serial INVERT.

3.2 Parallelising three groups of FTRAN operations

Within `pami`, the pivot sequence $\{p_i, q_i\}_{i=0}^{t-1}$ identified in minor iterations yields up to $3t$ forward linear systems (where $t \leq s$). Computationally, there are three groups of FTRAN operations, being t regular FTRANs for obtaining updated tableau columns $\widehat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$ associated with the entering variable identified during minor iterations; t additional FTRAN-DSE operations to obtain the DSE update vector $\boldsymbol{\tau} = B^{-1}\widehat{\boldsymbol{e}}_p$ and FTRAN-BFRT calculations to update the primal solution resulting from bound flips identified in the BFRT. Each system in a group is associated with a different basis matrix, $B_k, B_{k+1}, \ldots, B_{k+t-1}$. For example the t regular forward systems for obtaining updated tableau columns are $\widehat{\boldsymbol{a}}_{q_0} = B_k^{-1}\boldsymbol{a}_{q_0}, \widehat{\boldsymbol{a}}_{q_1} = B_{k+1}^{-1}\boldsymbol{a}_{q_1}, \ldots, \widehat{\boldsymbol{a}}_{q_{t-1}} = B_{k+t-1}^{-1}\boldsymbol{a}_{q_{t-1}}$.

For the regular FTRAN and FTRAN-DSE operations, the ith linear system (which requires B_{k+i}^{-1}) in each group, is solved by applying B_k^{-1} followed by $i-1$ PF transformations given by $\widehat{\boldsymbol{a}}_{q_j}$, $j < i$ to bring the result up to date. The operations with B_k^{-1} and PF transformations are referred to as the inverse and update parts respectively. The multiple inverse parts are easily arranged as a task parallel computation. The update part of the regular FTRAN operations requires results of other forward systems in the same group and thus cannot be performed as task parallel calculations. However, it is possible and valuable to exploit data parallelism when applying individual PF updates when $\widehat{\boldsymbol{a}}_{q_i}$ is large and dense. For the FTRAN-DSE group it is possible to exploit task parallelism fully if this group of computations is performed after the regular FTRAN. However, when implementing `pami`, both FTRAN-DSE and regular FTRAN are performed together to increase the number of independent inverse parts in the interests of load balancing.

The group of up to t linear systems associated with BFRT is slightly different from the other two groups of systems. Firstly, there may be anything between none and t linear systems depending how many minor iterations are associated with actual bound

flips. More importantly, the results are only used to update the values of the primal variables $\boldsymbol{x}_B$ by simple vector addition. This can be expressed as a single operation

$$\boldsymbol{x}_B := \boldsymbol{x}_B + \sum_{i=0}^{t-1} B_{k+i}^{-1} \boldsymbol{a}_{Fi} = \boldsymbol{x}_B + \sum_{i=0}^{t-1} \left(\prod_{j=i-1}^{0} E_j^{-1} B_k^{-1} \boldsymbol{a}_{Fi} \right) \tag{4}$$

where one or more of $\boldsymbol{a}_{Fi}$ may be a zero vector. If implemented using the regular PF update, each FTRAN-BFRT operation starts from the same basis inverse B_k^{-1} but finishes with different numbers of PF update operations. Although these operations are closely related, they cannot be combined. However, if the APF update is used, so B_{k+i}^{-1} can be expressed as

$$B_{k+i}^{-1} = B_k^{-1} T_0^{-1} \cdots T_{i-1}^{-1},$$

the primal update Eq. (4) can be rewritten as

$$\boldsymbol{x}_B := \boldsymbol{x}_B + \sum_{i=0}^{t-1} \left(B_k^{-1} \prod_{j=0}^{i-1} T_j^{-1} \boldsymbol{a}_{Fi} \right) = \boldsymbol{x}_B + B_k^{-1} \left(\sum_{i=0}^{t-1} \prod_{j=0}^{i-1} T_j^{-1} \boldsymbol{a}_{Fi} \right) \tag{5}$$

where the t linear systems start with a cheap APF update part and finish with a *single* B_k^{-1} operation applied to the combined result. This approach greatly reduces the total serial cost of solving the forward linear systems associated with BFRT. An additional benefit of this combination is that the UPDATE-PRIMAL operation is also reduced to a single operation after the combined FTRAN-BFRT.

By combining several potential FTRAN-BFRT operations into one, the number of forward linear systems to be solved is reduced to $2t + 1$, or $2t$ when no bound flips are performed. An additional benefit of this reduction is that, when $t \leq s - 1$, the total number of forward linear systems to be solved is less than $2s$, so that each of the s processors will solve at most two linear systems. However, when $t = s$ and FTRAN-BFRT is nontrivial, one of the s processors is required to solve three linear systems, while the other processors are assigned only two, resulting in an "orphan task". To avoid this situation, the number of minor iterations is limited to $t = s - 1$ if bound flips have been performed in the previous $s - 2$ iterations.

The arrangement of the task parallel FTRAN operations discussed above is illustrated in Fig. 1. In the actual implementation, the $2t + 1$ FTRAN operations are all started the same time as parallel tasks, and the processors are left to decide which ones to perform.

3.3 Candidate persistence and quality control in CHUZR

Major CHUZR forms the set $\mathcal{P}$ and minor CHUZR chooses candidates from it. The design of CHUZR contributes significantly to the serial efficiency of suboptimization schemes so merits careful discussion.

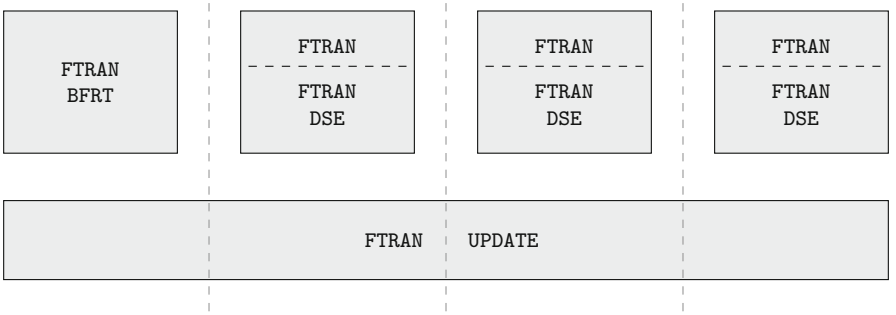


Fig. 1 Task parallel scheme of all FTRAN operations in `pami`

When suboptimization is performed, the candidate chosen to leave the basis in the first minor iteration is the same as would have been chosen without suboptimization. Thereafter, the candidates remaining in $\mathcal{P}$ may be less attractive than the most attractive of the candidates not in $\mathcal{P}$ due to the former becoming less attractive and/or the latter becoming more attractive. Indeed, some candidates in $\mathcal{P}$ may become unattractive. If candidates in the original $\mathcal{P}$ do not enter the basis then the work of their BTRAN operations (and any subsequent updates) is wasted. However, if minor iterations choose less attractive candidates to leave the basis the number of simplex iterations required to solve a given LP problem can be expected to increase. Addressing this issue of *candidate persistence* is the key algorithmic challenge when implementing suboptimization. The number of candidates in the initial set $\mathcal{P}$ must be decided, and a strategy determined for assessing whether a particular candidate should remain in $\mathcal{P}$.

For load balancing during the minor initialisation, the initial number of candidates $s = |\mathcal{P}|$ should be an integer multiple of the number of processors used. Multiples larger than one yield better load balance due to the greater amount of work to be parallelised, particularly before and after the minor iterations, but practical experience with `pami` prototypes demonstrated clearly that this is more than offset by the amount of wasted computation and an increase in the number of iterations required to solve the problem. Thus, for `pami`, s was chosen to be eight, whatever the number of processors.

During minor iterations, after updating the primal activities of the variables given by the current set $\mathcal{P}$, the attractiveness of α_p for each $p \in \mathcal{P}$ is assessed relative to its initial value α_p^i by means of a *cutoff* factor $\psi > 0$. Specifically, if

$$\alpha_p < \psi \alpha_p^i,$$

then index p is removed from $\mathcal{P}$. Clearly if the variable becomes feasible or unattractive ($\alpha_p \leq 0$) then it is dropped whatever the value of ψ.

To determine the value of ψ to use in `pami`, a series of experiments was carried out using a reference set of 30 LP problems given in Table 3 of Sect. 5.1, with cutoff ratios ranging from 1.001 to 0.01. Computational results are presented in Table 2 which gives the (geometric) mean speedup factor and the number of problems for which the speedup factor is respectively 1.6, 1.8 and 2.0.

Table 2 Experiments with different cutoff factor for controlling candidate quality in `pami`

cutoff (ψ)	speedup	#1.6 speedup	#1.8 speedup	#2.0 speedup
1.001	1.12	1	1	0
0.999	1.52	11	7	5
0.99	1.54	13	6	4
0.98	1.53	15	8	5
0.97	1.48	11	6	5
0.96	1.52	12	8	6
0.95	1.49	13	8	4
0.94	1.56	13	8	4
0.93	1.47	13	9	4
0.92	1.52	14	7	4
0.91	1.52	14	5	3
0.9	1.50	12	9	4
0.8	1.46	13	9	3
0.7	1.46	15	9	4
0.6	1.44	11	8	6
0.5	1.42	13	5	3
0.2	1.36	10	6	4
0.1	1.29	10	7	3
0.05	1.16	9	4	2
0.02	1.28	10	6	2
0.01	1.22	8	5	3

The cutoff ratio $\psi = 1.001$ corresponds to a special situation, in which only candidates associated with improved attractiveness are chosen. As might be expected, the speedup with this value of ψ is poor. The cutoff ratio $\psi = 0.999$ corresponds to a boundary situation where candidates whose attractiveness decreases are dropped. An mean speedup of 1.52 is achieved.

For various cutoff ratios in the range $0.9 \leq \psi \leq 0.999$, there is no really difference in the performance of `pami`: the mean speedup and larger speedup counts are relatively stable. Starting from $\psi = 0.9$, decreasing the cutoff factor results in a clear decrease in the mean speedup, although the larger speedup counts remain stable until $\psi = 0.5$.

In summary, experiments suggest that any value in interval $[0.9, 0.999]$ can be chosen as the cutoff ratio, with `pami` using the median value $\psi = 0.95$.

3.4 Hyper-sparse LP problems

In the discussions above, when exploiting data parallelism in vector operations it is assumed that one independent scalar calculation must be performed for most of the components of the vector. For example, in UPDATE-DUAL and UPDATE-PRIMAL a multiple of the component is added to the corresponding component of another vector. In CHUZR and CHUZC1 the component (if nonzero) is used to compute and then compare a ratio. Since these scalar calculations need not be performed for zero components of the vector, when the LP problem exhibits hyper-sparsity this is exploited by efficient serial implementations [13]. When the cost of the serial vector operation is reduced in this way it is no longer efficient to exploit data parallelism so, when the density of the vector is below a certain threshold, `pami` reverts to serial computation. The performance of `pami` is not sensitive to the thresholds of 5–10% which are used.

4 Single iteration parallelism

This section introduces a relative simple approach to exploiting parallelism within a single iteration of the dual revised simplex method, yielding the parallel scheme `sip`. Our approach is a significant development of the work of Bixby and Martin [1] who parallelised only the SPMV, CHUZC and UPDATE-DUAL operations, having rejected the task parallelism of FTRAN and FTRAN-DSE as being computationally disadvantageous. It also extends the work of Forrest's `Aboca` code [4,5]. Based on the little evidence available, `Aboca` needs an additional (partial) BTRAN operation which is not necessary in `sip`, which incorporates the FTRAN-BFRT and a combined SPMV and CHUZC1 not exploited by `Aboca`.

Our serial simplex solver `hsol` has an additional FTRAN-BFRT component for the bound-flipping ratio test. However, naively exploiting task parallelism by simply overlapping this with FTRAN and FTRAN-DSE is inefficient since the latter is seen in Table 1 to be relatively expensive. This is due to the RHS of FTRAN-DSE being $\widehat{\boldsymbol{e}}_p$, which is dense relative to the RHS vectors $\boldsymbol{a}_q$ of FTRAN and $\boldsymbol{a}_F$ of FTRAN-BFRT. There is also no guarantee in a particular iteration that FTRAN-BFRT will be required.

The mixed parallelisation scheme of `sip` is illustrated in Fig. 2, which also indicates the data dependency for each computational component. Note that during CHUZC1 there is a distinction between the operations for the original (structural) variables and those for the logical (slack) variables, since the latter correspond to an identity matrix in A. Thereafter, one processor performs FTRAN in parallel with (any) FTRAN-BFRT on another processor and UPDATE-DUAL on a third. The scheme assumes at least four processors, but with more than four only the parallelism in SPMV and CHUZC is enhanced.

5 Computational results

5.1 Test problems

Throughout this report, the performance of the simplex solvers is assessed using a reference set of 30 LP problems listed in Table 3. Most of these are taken from a comprehensive list of representative LP problems [19] maintained by Mittelmann.

The problems in this reference set reflect the wide spread of LP properties and revised simplex characteristics, including the dimension of the linear systems (number of rows), the density of the coefficient matrix (average number of non-zeros per column), and the extent to which they exhibit hyper-sparsity (indicated by the last two columns). These columns, headed FTRAN and BTRAN, give the proportion of the results of FTRAN and BTRAN with a density below 10%, the criterion used to measure hyper-sparsity by Hall and McKinnon [13] who consider an LP problem to be hyper-sparse if the occurrence of such hyper-sparse results is greater than 60%. According to this measurement, half of the reference set are hyper-sparse. Since all problems are sparse, it is convenient to use the term "dense" to refer to those which are not hyper-sparse (Table 3).

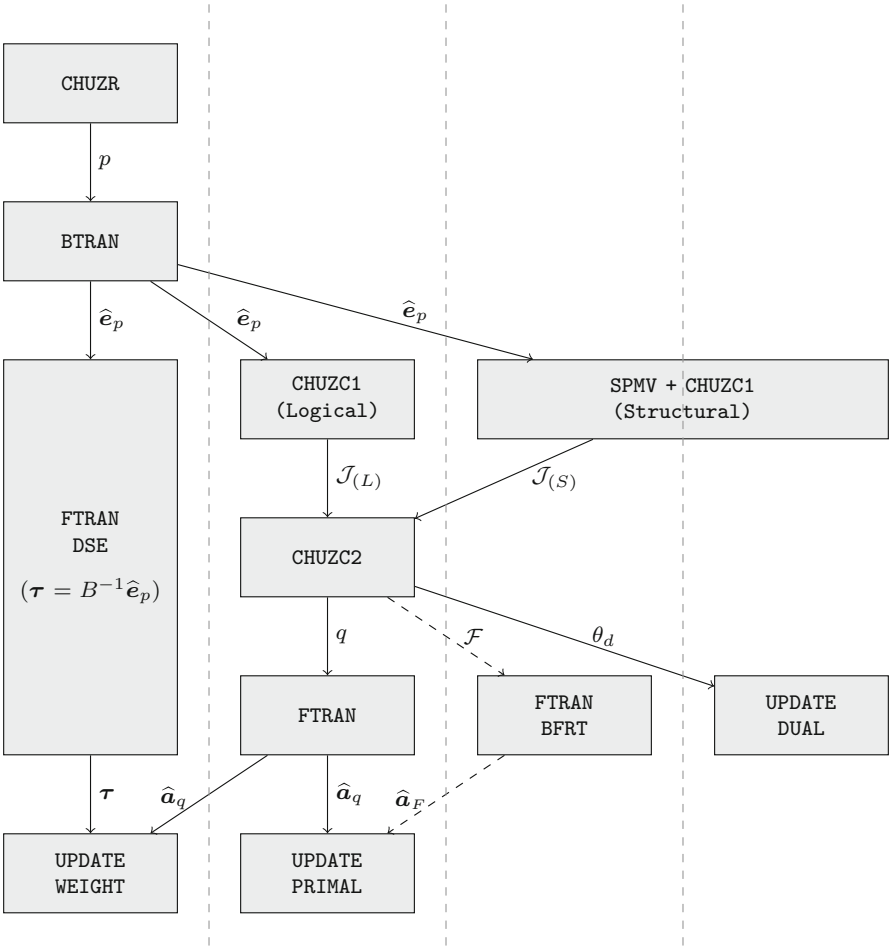


Fig. 2 `sip` data dependency and parallelisation scheme

The performance of `pami` and `sip` is assessed using experiments performed on a workstation with two Intel Xeon E5-2670s, 2.6 GHz (16 cores, 16 threads in total), using eight threads for the parallel calculations. Numerical results are given in Tables 4 and 5, where mean values of speedup or other relative performance measures are computed geometrically. The relative performance of solvers is also well illustrated using the performance profiles in Figs. 3, 4 and 5.

5.2 Performance of `pami`

The efficiency of `pami` is appropriately assessed in terms of parallel speedup and performance relative to the sequential dual simplex solver (`hsol`) from which it was developed. The former indicates the efficiency of the parallel implementation and the

Table 3 The reference set of 30 LP problems with hyper-sparsity measures

Model	#row	#col	#nnz	ftran	btran
cre-b	9648	72447	256095	100	83
dano3mip_lp	3202	13873	79655	1	6
dbic1	43200	183235	1038761	100	83
dcp2	32388	21087	559390	100	97
dfl001	6071	12230	35632	34	57
fome12	24284	48920	142528	45	58
fome13	48568	97840	285056	100	98
ken-18	105127	154699	358171	100	100
l30	2701	15380	51169	10	8
Linf_520c	93326	69004	566193	10	11
lp22	2958	13434	65560	13	22
maros-r7	3136	9408	144848	5	13
mod2	35664	31728	198250	46	68
ns1688926	32768	16587	1712128	72	100
nug12	3192	8856	38304	1	20
pds-40	66844	212859	462128	100	98
pds-80	129181	426278	919524	100	99
pds-100	156243	505360	1086785	100	99
pilot87	2030	4883	73152	10	19
qap12	3192	8856	38304	2	15
self	960	7364	1148845	0	2
sgpf5y6	246077	308634	828070	100	100
stat96v4	3174	62212	490473	73	31
stormG2-125	66185	157496	418321	100	100
stormG2-1000	528185	1259121	3341696	100	100
stp3d	159488	204880	662128	95	70
truss	1000	8806	27836	37	2
watson_1	201155	383927	1052028	100	100
watson_2	352013	671861	1841028	100	100
world	35510	32734	198793	41	61

latter measures the impact of suboptimization on serial performance. A high degree of parallel efficiency would be of little value if it came at the cost of severe serial inefficiency. The solution times for `hsol` and `pami` running in serial, together with `pami` running in parallel with 8 cores, are listed in columns headed `hsol`, `pami1` and `pami8` respectively in Table 4. These results are also illustrated via a performance profile in Fig. 3 which, to put the results in a broader context, also includes `Clp` 1.15 [2], the world's leading open-source solver. Note that since `hsol` and `pami` have no preprocessing or crash facility, these are not used in the runs with `Clp`.

The number of iterations required to solve a given LP problem can vary significantly depending on the solver used and/or the algorithmic variant used. Thus, using solution times as the sole measure of computational efficiency is misleading if there is a significant difference in iteration counts for algorithmic reasons. However, this is not the case for `hsol` and `pami`. Observing that `pami` identifies the same sequence of basis changes whether it is run in serial or parallel, relative to `hsol`, the number of iterations required by `pami` is similar, with the mean relative iteration count of 0.96 being marginally in favour of `pami`. Individual relative iteration counts lie in [0.85, 1.15] with the exception of those for qap12, stp3d and dano3mip_lp which, being

Table 4 Solution time and iteration counts for `hsol`, `pami`, `sip`, `Clp` and `Cplex`

	Solution time						Iteration counts				
Model	hsol	pami1	pami8	sip	Clp	Cplex	hsol	pami	sip	Clp	Cplex
cre-b	4.62	3.82	2.37	3.78	12.78	1.44	11599	10641	11632	26734	10912
dano3mip_lp	38.21	55.86	17.47	22.93	43.92	10.64	60161	47774	62581	64773	27438
dbic1	52.43	111.22	39.24	44.43	542.62	27.64	35884	36373	37909	330315	46685
dcp2	9.34	11.18	6.07	7.77	23.78	3.93	25360	24844	25360	43305	24036
dfl001	11.74	17.80	6.31	8.47	13.13	7.89	26322	23668	26417	26866	21534
fome12	71.74	116.92	42.26	56.50	54.22	50.58	103005	97646	101406	95142	85492
fome13	186.35	271.72	113.39	148.27	122.58	156.90	209722	193928	204705	189503	177456
ken-18	10.23	12.34	8.49	12.85	14.91	5.37	107471	106646	107467	106812	81952
l30	7.93	17.48	6.24	6.04	7.14	5.60	10290	11433	10389	8934	10793
Linf_520c	2329.49	6402.00	2514.32	1699.63	6869.00	11922.00	132244	127468	132244	226319	153027
lp22	15.74	26.54	9.64	10.97	14.90	8.54	25080	25778	24888	22401	18474
maros-r7	7.91	27.49	16.08	6.47	8.60	2.73	6025	6258	6025	5643	6585
mod2	38.90	73.57	29.78	32.39	25.77	19.83	43386	43100	42944	39552	48134
ns1688926	17.75	28.13	10.16	12.96	2802.23	15.38	13849	15455	13849	193565	7228
nug12	88.37	142.20	50.05	76.70	288.70	58.61	108152	102429	118370	211658	92368
pds-40	20.39	31.28	15.04	18.08	155.53	16.26	94914	92992	92888	147122	58578
pds-80	46.54	85.58	39.57	45.01	583.12	39.58	197461	200694	195658	409923	124097
pds-100	59.21	94.67	46.32	55.06	719.33	51.88	234184	231758	231570	554434	143383
pilot87	4.93	7.92	3.28	3.73	5.66	5.61	7240	7390	7130	8918	12069
qap12	111.93	123.70	43.46	134.40	168.50	58.43	128131	86418	205278	134570	90736
self	28.02	47.44	22.35	16.28	20.43	29.07	4738	5429	4738	4659	12073
sgpf5y6	111.75	153.94	53.18	174.71	188.91	5.00	348115	346042	347978	347526	59716
stat96v4	101.35	161.92	44.24	51.10	131.66	50.62	72531	65440	72531	119002	87056
stormG2-125	7.02	8.95	5.58	10.00	18.01	3.98	81869	82965	81869	92149	86526
stormG2-1000	290.35	397.72	185.34	352.44	1018.35	105.44	658534	658338	658534	738319	783176
stp3d	355.98	443.99	152.47	305.96	254.71	163.98	130689	97680	130276	126346	98914
truss	5.69	7.93	3.24	3.63	3.68	2.80	18929	15987	18929	17561	19693
watson_1	35.70	43.89	25.82	47.30	133.56	21.34	238973	239301	239819	466774	208888
watson_2	37.96	44.21	26.95	50.65	1118.00	35.88	334733	331607	334494	498797	305197
world	47.97	86.49	34.29	38.69	33.83	26.19	47104	44722	46742	46283	54656

Table 5 Speedup of `pami` and `sip` with hyper-sparsity measures

	Speedup				Hyper-sparsity	
Model	p1/hsol	p8/p1	p8/hsol	sip/hsol	FTRAN	BTRAN
cre-b	1.21	1.61	1.95	1.22	100	83
dano3mip_lp	0.68	3.20	2.19	1.67	1	6
dbic1	0.47	2.83	1.34	1.18	100	83
dcp2	0.84	1.84	1.54	1.20	100	97
dfl001	0.66	2.82	1.86	1.39	34	57
fome12	0.61	2.77	1.70	1.27	45	58
fome13	0.69	2.40	1.64	1.26	100	98
ken-18	0.83	1.45	1.20	0.80	100	100
l30	0.45	2.80	1.27	1.31	10	8
Linf_520c	0.36	2.55	0.93	1.37	10	11
lp22	0.59	2.75	1.63	1.43	13	22
maros-r7	0.29	1.71	0.49	1.22	5	13
mod2	0.53	2.47	1.31	1.20	46	68
ns1688926	0.63	2.77	1.75	1.37	72	100
nug12	0.62	2.84	1.77	1.15	1	20
pds-40	0.65	2.08	1.36	1.13	100	98
pds-80	0.54	2.16	1.18	1.03	100	99
pds-100	0.63	2.04	1.28	1.08	100	99
pilot87	0.62	2.41	1.50	1.32	10	19
qap12	0.90	2.85	2.58	0.83	2	15
self	0.59	2.12	1.25	1.72	0	2
sgpf5y6	0.73	2.89	2.10	0.64	100	100
stat96v4	0.63	3.66	2.29	1.98	73	31
stormG2-125	0.78	1.60	1.26	0.70	100	100
stormG2-1000	0.73	2.15	1.57	0.82	100	100
stp3d	0.80	2.91	2.33	1.16	95	70
truss	0.72	2.45	1.76	1.57	37	2
watson_1	0.81	1.70	1.38	0.75	100	100
watson_2	0.86	1.64	1.41	0.75	100	100
world	0.55	2.52	1.40	1.24	41	61
Mean	0.64	2.34	1.51	1.15		

0.67, 0.75 and 0.79 respectively, are significantly in favour of `pami`. Thus, with the candidate quality control scheme discussed in Sect. 3.3, suboptimization is seen not compromise the number of iterations required to solve LP problems. Relative to `Clp`, `hsol` typically takes fewer iterations, with the mean relative iteration count being 0.70 and extreme values of 0.07 for ns1688926 and 0.11 for dbic1.

It is immediately clear from the performance profile in Fig. 3 that, when using 8 cores, `pami` is superior to `hsol` which, in turn, is generally superior to `Clp`. Observe that the superior performance of `pami` on 8 cores relative to `hsol` comes despite `pami` in serial being inferior to `hsol`. Specifically, using the mean relative solution times in Table 5, `pami` on 8 cores is 1.51 times faster than `hsol`, which is 2.29 times faster than `Clp`. Even when taking into account that `hsol` requires 0.70 times the iterations of `Clp`, the iteration speed of `hsol` is seen to be 1.60 times faster than `Clp`: `hsol` is a high quality dual revised simplex solver.

Since `hsol` and `pami` require very similar numbers of iterations, the mean value of 0.64 for the inferiority of `pami` relative to `hsol` in terms of solution time reflects the the lower iteration speed of `pami` due to wasted computation. For more than

Table 6 Iteration time (ms) and computational component profiling (the percentage of overall solution time) when solving LP problems with `hsol`

Model	Iter. Time	CHUZR	CHUZC1	CHUZC2	SPMV	UPDATE	BTRAN	FTRAN	F-DSE	F-BFRT	INVERT	OTHER
CRE-B	565	0.8	20.1	4.4	42.9	6.9	4.7	1.7	11.3	1.5	4.3	1.4
DANO3MIP_LP	885	1.8	21.2	3.0	35.5	5.3	6.4	6.9	11.7	0.3	6.2	1.7
DBIC1	2209	0.5	22.5	3.1	33.6	5.8	5.7	6.5	14.8	3.2	3.1	1.2
DCP2	509	6.5	3.9	1.7	8.7	7.3	5.4	18.1	28.4	10.4	7.4	2.2
DFL001	595	4.1	8.1	1.0	17.9	11.2	10.8	13.0	20.7	6.2	5.2	1.8
FOME12	971	7.9	5.1	0.6	12.4	6.8	12.3	14.5	24.0	7.1	7.9	1.4
FOME13	1225	10.1	4.2	0.5	10.6	5.6	11.4	13.5	26.4	6.7	9.6	1.4
KEN-18	126	5.3	2.9	0.6	5.2	2.2	7.9	11.0	24.4	3.8	32.4	4.3
L30	1081	0.8	14.1	9.9	24.0	6.3	8.6	9.0	12.9	4.1	8.5	1.8
LINF_520C	26168	1.5	2.3	0.1	11.8	4.0	16.6	19.7	23.2	0.0	19.2	1.6
LP22	888	2.0	10.9	2.0	23.3	8.4	9.4	10.4	14.9	6.8	10.0	1.9
MAROS-R7	1890	0.8	2.8	0.2	10.2	2.7	17.5	15.3	20.6	0.0	27.4	2.5
MOD2	1214	4.2	7.5	1.0	9.9	8.5	11.5	17.4	29.1	5.4	4.0	1.5
NS1688926	1806	2.0	0.1	0.0	2.9	4.8	3.3	31.4	44.1	0.0	6.5	4.9
NUG12	1157	1.6	7.4	1.1	16.3	6.9	11.6	12.4	16.7	5.8	18.1	2.1
PDS-40	302	3.4	7.5	1.9	19.2	5.1	10.8	10.3	23.2	4.4	12.0	2.2
PDS-80	337	3.7	6.6	1.8	19.8	3.9	10.5	9.1	23.7	3.9	15.0	2.0
PDS-100	360	3.5	7.0	1.8	18.6	3.7	10.4	9.0	24.1	3.8	16.0	2.1
PILOT87	918	1.2	5.1	0.8	17.9	4.4	12.0	12.9	17.4	7.6	17.9	2.8
QAP12	1229	1.5	7.5	1.0	16.2	6.6	12.1	12.3	16.7	5.9	18.4	1.8
SELF	8350	0.0	1.4	0.2	39.6	0.2	7.0	6.5	7.0	0.0	33.9	4.2
SGPF5Y6	491	1.3	0.3	0.1	0.2	0.1	5.0	2.3	80.7	0.0	8.4	1.6
STAT96V4	2160	0.4	12.4	4.9	67.6	1.7	2.4	1.7	4.3	0.6	2.2	1.8
STORMG2-125	115	5.2	0.8	0.2	1.7	0.9	4.4	8.3	48.7	0.1	26.7	3.0
STORMG2-1000	650	1.5	0.1	0.0	0.3	1.3	3.5	6.1	70.6	0.0	14.6	2.0
STP3D	4325	1.6	10.7	0.9	19.2	7.6	13.5	12.0	27.0	3.9	2.4	1.2
TRUSS	415	1.1	17.1	2.0	53.8	5.0	5.0	3.7	7.1	0.0	3.5	1.7
WATSON_1	210	4.3	0.7	0.2	1.0	1.2	5.7	6.0	54.4	3.5	19.6	3.4
WATSON_2	161	5.5	0.3	0.0	0.4	0.8	4.6	7.7	35.2	5.0	34.5	6.0
WORLD	1383	3.8	8.7	1.3	10.9	8.6	11.6	16.5	28.0	5.5	3.7	1.4
AVERAGE	867	2.9	7.3	1.5	18.4	4.8	8.7	10.8	26.4	3.5	13.3	2.3

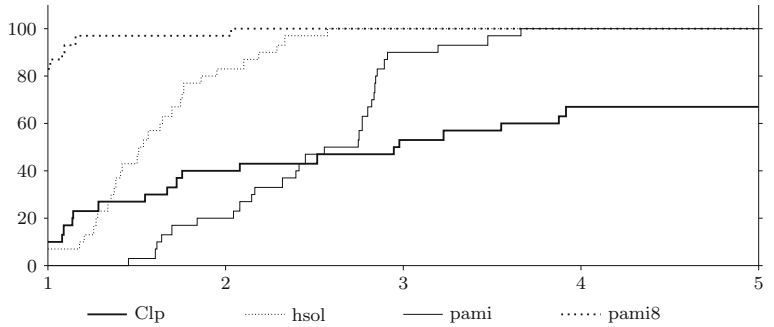


Fig. 3 Performance profile of `Clp`, `hsol`, `pami` and `pami8` without preprocessing or crash

65% of the reference set `pami` is twice as fast in parallel, with a mean speedup of 2.34. However, relative to `hsol`, some of this efficiency is lost due to overcoming the wasted computation, lowering the mean relative solution time to 1.51.

For individual problems, there is considerable variance in the speedup of `pami` over `hsol`, reflecting the variety of factors which affect performance and the wide range of test problems. For the two problems where `pami` performs best in parallel, it is flattered by requiring significantly fewer iterations than `hsol`. However, even if the speedups of 2.58 for QAP12 and 2.33 for STP3D are scaled by the relative iteration counts, the resulting relative iteration speedups are still 1.74 and 1.75 respectively. However, for other problems where `pami` performs well, this is achieved with an iteration count which is similar to that of `hsol`. Thus the greater solution efficiency due to exploiting parallelism is genuine. Parallel `pami` is not advantageous for all problems. Indeed, for MAROS-R7 and LINF_520C, `pami` is slower in parallel than `hsol`. For these two problems, serial `pami` is slower than `hsol` by factors of 3.48 and 2.75 respectively. In addition, as can be seen in Table 6, a significant proportion of the computation time for `hsol` is accounted for by INVERT, which runs in serial on one processor with no work overlapped.

Interestingly, there is no real relation between the performance of `pami` and problem hyper-sparsity: it shows almost same range of good, fair and modest performance across both classes of problems, although the more extreme performances are for dense problems. Amongst hyper-sparse problems, the three where `pami` performs best are CRE-B, SGPF5Y6 and STP3D. This is due to the large percentage of the solution time for `hsol` accounted for by SPMV (42.9% for CRE-B and 19.2% for STP3D) and FTRAN-DSE (80.7% for SGPF5Y6 and 27% for STP3D). In `pami`, the SPMV and FTRAN-DSE components can be performed efficiently as task parallel and data parallel computations respectively, and therefore the larger percentage of solution time accounted for by these components yields a natural source of speedup.

5.3 Performance of `sip`

For `sip`, the iteration counts are generally very similar to those of `hsol`, with the relative values lying in [0.98, 1.06] except for the two, highly degenerate problems

NUG12 and QAP12 where sip requires 1.09 and 1.60 times as many iterations respectively. [Note that these two problems are essentially identical, differing only by row and column permutations.] It is clear from Table 5 that the overall performance and mean speedup (1.15) of sip is inferior to that of pami. This is because sip exploits only limited parallelism.

The worst cases when using sip are associated with the hyper-sparse LP problems where sip typically results in a slowdown. Such an example is SGPF5Y6, where the proportion of FTRAN-DSE is more than 80% and the total proportion of SPMV, CHUZC, FTRAN and UPDATE-DUAL is less than 5%. Therefore, when performing FTRAN-DSE and the rest as task parallel operations, the overall performance is not only limited by FTRAN-DSE, but the competition for memory access by the other components and the cost of setting up the parallel environment will also slow down FTRAN-DSE.

However, when applied to dense LP problems, the performance of sip is moderate and relatively stable. This is especially so for those instances where pami exhibits a slowdown: for LINF_520C, MAROS-R7, applying sip achieves speedups of 1.31 and 1.12 respectively.

In summary, sip, is a straightforward approach to parallelisation which exploits purely single iteration parallelism and achieves relatively poor speedup for general LP problems compared to pami. However, sip is frequently complementary to pami in achieving speedup when pami results in slowdown.

5.4 Performance relative to Cplex and influence on Xpress

Since commercial LP solvers are now highly developed it is, perhaps, unreasonable to compare their performance with a research code. However, this is done in Fig. 4, which illustrates the performance of Cplex 12.4 [17] relative to pami8 and sip8. Again, Cplex is run without preprocessing or crash. Figure 4 also traces the performance of the better of pami8 and sip8, clearly illustrating that sip and pami are frequently complementary in terms of achieving speedup. Indeed, the performance of the better of sip and pami is comparable with that of Cplex for

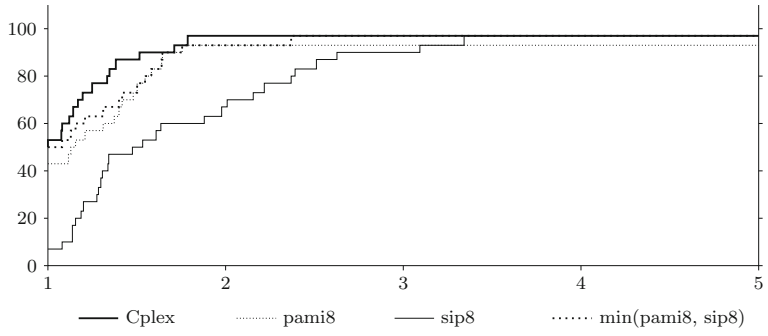


Fig. 4 Performance profile of Cplex, pami8 and sip8 without preprocessing or crash

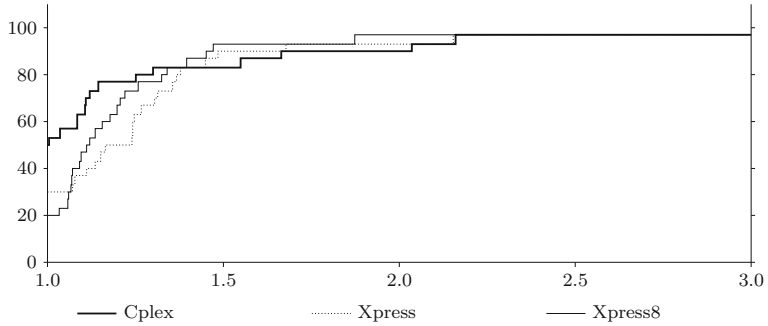


Fig. 5 Performance profile of `Cplex`, `Xpress` and `Xpress8` with preprocessing and crash

the majority of the test problems. For a research code this is a significant achievement.

Since developing and implementing the techniques described in this paper, Huangfu has implemented them within the FICO `Xpress` simplex solver [15]. The performance profile in Fig. 5 demonstrates that when it is advantageous to run `Xpress` in parallel it enables FICO's solver to match the serial performance of `Cplex` (which has no parallel simplex facility). Note that for the results in in Fig. 5, `Xpress` and `Cplex` were run with both preprocessing and crash. The newly-competitive performance of parallel `Xpress` relative to `Cplex` is also reflected in Mittelmann's independent benchmarking [19].

6 Conclusions

This report has introduced the design and development of two novel parallel implementations of the dual revised simplex method.

One relatively complicated parallel scheme (`pami`) is based on a less-known pivoting rule called suboptimization. Although it provided the scope for parallelism across multiple iterations, as a pivoting rule suboptimization is generally inferior to the regular dual steepest-edge algorithm. Thus, to control the quality of the pivots, which often declines during `pami`, a *cutoff* factor is necessary. A suitable cutoff factor of 0.95, has been found via series of experiments. For the reference set, `pami` provides a mean speedup of 1.51 which enables it to out-perform Clp, the best open-source simplex solver.

The other scheme (`sip`) exploits purely single iteration parallelism. Although its mean speedup of 1.15 is worse than that of `pami`, it is frequently complementary to `pami` in achieving speedup when `pami` results in slowdown.

Although the results in this paper are far from the linear speedup which is the hallmark of many quality parallel implementations of algorithms, to expect such results for an efficient implementation of the revised simplex method applied to general large sparse LP problems is unreasonable. The commercial value of efficient simplex implementations is such that if such linear speedup were possible then it would have been achieved years ago. A measure of the quality of the `pami` and `sip` schemes

discussed in this paper is that they have formed the basis of refinements made by Huangfu to the Xpress solver which have been considered noteworthy enough to be reported by FICO. With the techniques described in this paper, Huangfu has raised the performance of the Xpress parallel revised simplex solver to that of the worlds best commercial simplex solvers. In developing the first parallel revised simplex solver of general utility, this work represents a significant achievement in computational optimization.

References

1. Bixby, R.E., Martin, A.: Parallelizing the dual simplex method. INFORMS J. Comput. **12**(1), 45–56 (2000)
2. COIN-OR. Clp. http://www.coin-or.org/projects/Clp.xml (2014). Accessed 20 Jan 2016
3. Elble, J.M., Sahinidis, N.V.: A review of the LU update in the simplex algorithm. Int. J. Math. Oper. Res. **4**(4), 366–399 (2012)
4. Forrest, J.: Aboca: a bit of Clp accelerated. Presentation at the 21st international symposium on mathematical programming (2012)
5. Forrest, J.: Never Asked Questions. http://www.fastercoin.com/f2_files/naqs.html (2012). Accessed 20 Jan 2016
6. Forrest, J.J., Goldfarb, D.: Steepest-edge simplex algorithms for linear programming. Math. Program. **57**, 341–374 (1992)
7. Forrest, J.J.H., Tomlin, J.A.: Updated triangular factors of the basis to maintain sparsity in the product form simplex method. Math. Program. **2**, 263–278 (1972)
8. Fourer, R.: Notes on the dual simplex method. Technical report, Department of Industrial Engineering and Management Sciences Northwestern University (1994) (unpublished)
9. Hall, J., Huangfu, Q.: A high performance dual revised simplex solver. In: Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics—Volume Part I, PPAM'11, pp. 143–151. Springer, Berlin (2012)
10. Hall, J.A.J.: Towards a practical parallelisation of the simplex method. Comput. Manag. Sci. **7**, 139–170 (2010)
11. Hall, J.A.J., McKinnon, K.I.M.: PARSMI, a parallel revised simplex algorithm incorporating minor iterations and Devex pricing. In: Waśniewski, J., Dongarra, J., Madsen, K., Olesen, D. (eds.) Applied Parallel Computing. Lecture Notes in Computer Science, vol. 1184, pp. 67–76. Springer, Berlin (1996)
12. Hall, J.A.J., McKinnon, K.I.M.: ASYNPLEX, an asynchronous parallel revised simplex method algorithm. Ann. Oper. Res. **81**, 27–49 (1998)
13. Hall, J.A.J., McKinnon, K.I.M.: Hyper-sparsity in the revised simplex method and how to exploit it. Comput. Opt. Appl. **32**(3), 259–283 (2005)
14. Harris, P.M.J.: Pivot selection methods of the Devex LP code. Math. Program. **5**, 1–28 (1973)
15. Huangfu, Q.: The Algorithm that Runs the World. http://www.fico.com/en/blogs/analytics-optimization/the-algorithm-that-runs-the-world/ (2014). Accessed 20 Jan 2016
16. Huangfu, Q., Hall, J.A.J.: Novel update techniques for the revised simplex method. Comput. Opt. Appl. **60**(587–608), 1–22 (2014)
17. IBM. ILOG CPLEX Optimizer. http://www.ibm.com/software/integration/optimization/cplex-optimizer/ (2014). Accessed 20 Jan 2016
18. Koberstein, A.: Progress in the dual simplex algorithm for solving large scale LP problems: techniques for a fast and stable implementation. Comput. Opt. Appl. **41**(2), 185–204 (2008)
19. Mittelmann, H.D.: Benchmarks for optimization software. http://plato.la.asu.edu/bench.html (2015). Accessed 20 Jan 2016

20. Orchard-Hays, W.: Advanced Linear Programming Computing Techniques. McGraw-Hill, New York (1968)
21. Rosander, R.R.: Multiple pricing and suboptimization in dual linear programming algorithms. Math. Program. Study **4**, 108–117 (1975)
22. Wunderling, R.: Paralleler und objektorientierter simplex. Technical Report TR-96-09, Konrad-Zuse-Zentrum for Informationstechnik Berlin (1996)