Parallelizing the dual revised simplex method

Q. Huangfu and J. A. J. Hall

August 29, 2014

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. Computational results show that the performance of PAMI is comparable with a world-leading commercial simplex solver, and that SIP complements PAMI in achieving speedup when PAMI results in slowdown.

Keywords: Revised simplex method, simplex parallelization

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 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 decomposing 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 resulted in the dual simplex algorithm being preferred.

A review of past work on parallelising the simplex method is given by Hall [8]. The standard (tableau-based) 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 is inferior to a good sparsity-exploiting sequential implementation of the revised simplex method. 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 simpled 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 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. 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 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, Sections 3 and 4 detail the design of `pami` and `sip` respectively and Section 5 presents numerical results and performance analysis. Conclusions are given in Section 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 computa-

tional 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 the key enhancements and major computational components. Section 2.3 introduces suboptimization, a relative unknown dual simplex variant which is the starting point for the `pami` parallelisation in Section 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 $B \in \mathbb{R}^{m\times m}$ which consists 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 $\widehat{\boldsymbol{c}}_N^T = \boldsymbol{c}_N^T - \boldsymbol{c}_B^TB^{-1}N$. 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 basic 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 $\widehat{\boldsymbol{c}}^T$ 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 $\widehat{\boldsymbol{c}}^T$ 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 Section 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 in the simplex method, it was identified by Hall and McKinnon [11] and is referred to as hyper-sparsity.

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 [4] 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 [11] which maintains a short list of the most attractive variables. This is more efficient for large and sparse LP problems since it avoids 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 PRICE respectively.

The dual ratio test (CHUZC) is enhanced by the Harris two-pass ratio test [12] and bound-flipping ratio test (BFRT) [6]. Details of how to apply these two techniques are set out by Koberstein [16].

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_d, 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, many 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 the dual variables $\widehat{\boldsymbol{c}}_N^T$ (UPDATE-DUAL) is achieved using

θ_d and $\widehat{\boldsymbol{a}}_p$. The update of the DSE weights is given by

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

This requires both the FTRAN result $\widehat{\boldsymbol{a}}_q$ and the solution of $\boldsymbol{\tau} = B^{-1}\widehat{\boldsymbol{e}}_p$. The latter is achieved 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 Summary and parallelisation scope

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

Table 1: Major components of 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
PRICE	Compute $\widehat{\boldsymbol{a}}_p^T = \widehat{\boldsymbol{a}}_p^T A$	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 $\boldsymbol{\tau}$	

There is immediate scope for data parallelisation within CHUZR, PRICE, CHUZC and most of the update operations since they require independent operations for each (nonzero) component of a vector. Exploiting such parallelisation has been reported by Bixby and Martin [1] who achieve speedup on a small group of LP problems with relatively expensive PRICE operations. The scope for task parallelism by overlapping FTRAN and FTRAN-DSE

was considered by Bixby and Martin but rejected as being disadvantageous computationally.

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 [19] 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 $\mathcal{P} \subset \mathcal{B}$ 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 p variable 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 $\widehat{\boldsymbol{a}}_p^T = \widehat{\boldsymbol{e}}_p^T A$ (PRICE) 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 mainly as a pivoting scheme with principal 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 [9, 10]. For dual suboptimization the major initialisation requires s BTRAN operations, where $s = |\mathcal{P}|$. Then $t \leq s$ minor iterations are performed, following which 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 Section 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 for the update method in the revised simplex method are the relatively simple product form (PF) update [18] or the efficient Forrest-Tomlin (FT) update [5]. 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 [14]. For the purpose of this report, the features of all relevant update methods are briefly summarised here.

- The *product form* (PF) update uses the FTRAN result $\widehat{\boldsymbol{a}}_q$, yielding $B_{k+1}^{-1} = E^{-1}B_k^{-1}$, where $E = I + (\widehat{\boldsymbol{a}}_q - \boldsymbol{e}_p)\boldsymbol{e}_p^T$, whose inverse is readily available.
- The *Forrest-Tomlin* (FT) update assumes $B_k = L_kU_k$ and uses both the partial FTRAN result $\tilde{\boldsymbol{a}}_q = L_k^{-1}\boldsymbol{a}_q$ and partial BTRAN result $\tilde{\boldsymbol{e}}_p^T = \boldsymbol{e}_p^TU_k^{-1}$ to modify U_k and augment L_k.
- The *alternate product form* (APF) update [14] 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 [14] 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 PF and APF updates. The value of the APF update within `pami` is indicated in Section 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 [19], incorporating (serial) algorithmic techniques and exploiting parallelism across multiple iterations.

The fundamental design of `pami` was introduced by Hall and Huangfu [7], where it was referred to as ParISS. This prototype implementation was based on the PF update and, algorithmically, was relatively unsophisticated. Subsequent revisions and refinements, incorporating the advanced algorithmic

techniques outlined in Section 2 as well as PF updates and some novel features described in this section, have yielded a very much more sophisticated and efficient implementation.

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

3.1 Overview of `pami` framework

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

3.1.1 Major optimality test

The major optimality test involves only major CHUZR operations in which up to s candidates are chosen using the DSE framework. In `pami` the value of s is the number of processors being used. It is a vector-based operation operation which can be easily parallelised, although its overall computational cost is not significant since it is only performed once per major operation. The algorithmic design of CHUZR is important and Section 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 below that of the best candidate not in $\mathcal{P}$. 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 Section 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 PRICE, CHUZC1 and CHUZC2. The PRICE operation is a sparse matrix-vector product and CHUZC1 is a

one-pass selection based on the result of PRICE. In the actual implementation, they can share one parallel initialisation. On they 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 PRICE 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 PRICE.

(c) The minor update consists of two parts, 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 in 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 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 $3\times t$ FTRAN operations (including FTRAN-DSE and FTRAN-BFRT) , updating of primal variables and DSE weights (vector based), 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 all 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) there is significant processor idleness during the serial INVERT.

3.2 Parallelising three groups of FTRAN operations

Within `pami`, there are up to $3t$ forward linear systems (for $t \leq s$) corresponding to the pivot sequence $\{p_i, q_i\}_{i=0}^{t-1}$ identified in minor iterations. There are three groups of FTRAN operations in `pami`: 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 up to t 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 i^{th} linear system (which requires B_{k+i}^{-1}) in each group, is solved by applying B_k^{-1} follower by a $i-1$ 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 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 with FTRAN-BFRT is slightly different from the other two groups of FTRAN operations. 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 result of FTRAN-BFRT is only used to update the values of the primal variables $\boldsymbol{x}_B$ by simple vector addition, which 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. 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 an APF update is used before applying B_k^{-1}, so B_{k+i}^{-1} can be expressed as

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

the primal update equation (4) can be rewritten as

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

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 FTRAN-BFRT operations into one, the number of forward linear systems is reduced to $2 \times t + 1$, or $2 \times t$ 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 solve is less than $2 \times s$, so that, on average, each of the s processors will solve 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 $t - 1$ iterations.

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

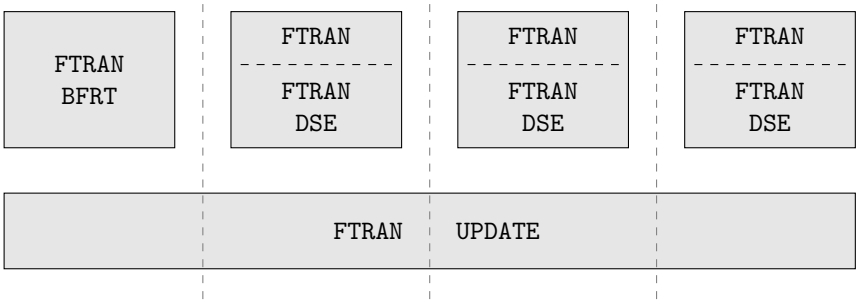

Figure 1: Task parallel scheme of all FTRAN operations in `pami`

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 serial efficiency of suboptimization schemes so merits careful discussion.

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 remains 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 this is more than offset by the amount wasted and an increase in the number of iterations required to solve the problem. Thus, for `pami`, s was chosen to be equal to 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 each α_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 unattractive ($\alpha_p < 0$) 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 using the reference set of 30 LP problems in Table 3 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.

The cutoff ratio $\psi = 1.001$ corresponds to a special situation, where 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 average 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 average speedup and larger speedup counts are relatively stable. Starting from $\psi = 0.9$, decreasing the cutoff factor results in a clear decrease in the average speedup, though the larger speedup counts remain stable until $\psi = 0.5$.

In summary, experiments suggest that the 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. However, when the LP problem exhibits hyper-sparsity, most of these scalar operations are trivial

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

and this must be exploited in efficient serial implementations [11]. When the cost of the serial operation is reduced in this way it is no longer efficient to exploit data parallelism so, when the density of the operation is below a certain threshold, `pami` reverts to serial computation.

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 PRICE, CHUZC and UPDATE-DUAL operations, having rejected the task parallelism of FTRAN and FTRAN-DSE as being computationally disadvantageous.

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 Figure 2 which also indicates the data dependency for each computational component. Note that, during CHUZC1, there is a distinction between the operations for the logical (slack) variables since they correspond to an identity matrix in A and those for the original (structural) variables. Parallelism during PRICE and CHUZC is achieved via a similar permutation and partitioning scheme to that of `pami`, as discussed in Section 3.1.3. However, for `sip`, there are $s-2$ partitions since one processor is used to perform FTRAN-DSE and another performs CHUZC1 for the logical variables. Once CHUZC2 is complete, one processor performs FTRAN in parallel with (any) FTRAN-BFRT on another and UPDATE-DUAL on a third. The scheme assumes at least four processors but with more than four only the parallelism in PRICE and CHUZC is enhanced.

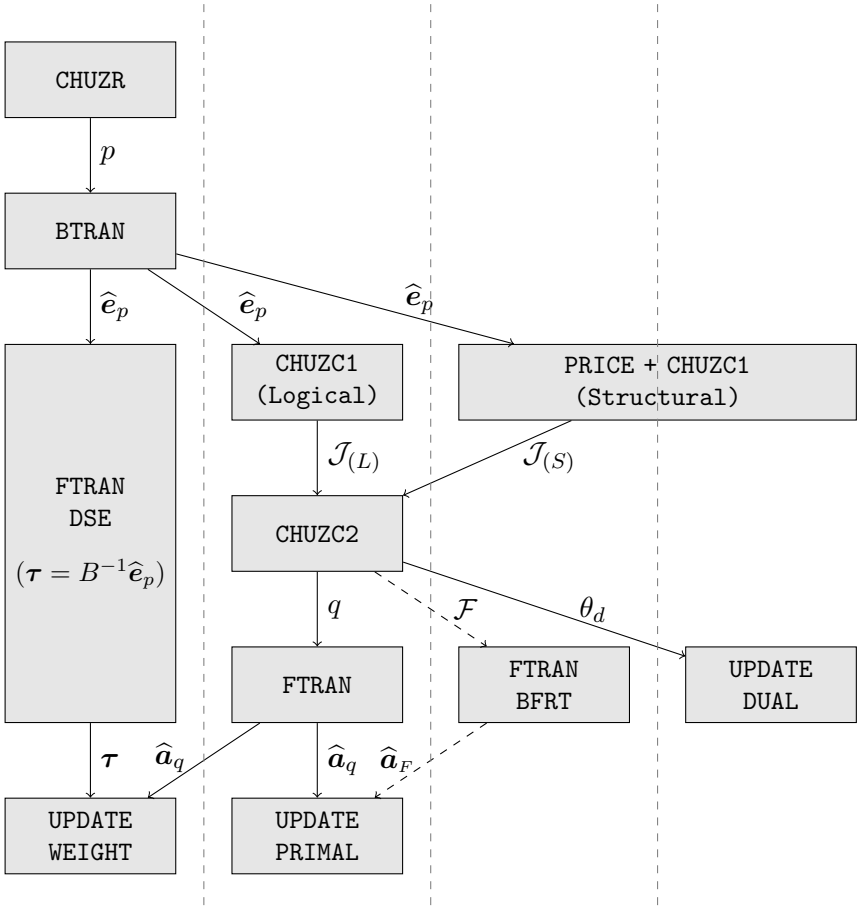

Figure 2: `sip` data dependency and parallelisation scheme

5 Computational results

5.1 Test problems

Throughout this report, the performance of the simplex solvers is assessed using a reference set consisting of 30 problems. Most of these LP problems are taken from a comprehensive list of various representative LP problems [17] maintained by Mittelmann.

Table 3: The reference set of 30 LP problems

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

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 [11] 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.

The performance of `pami` and `sip` is assessed using experiments performed on a workstation with 16 (Intel Xeon E5620, 2.4GHz) cores, using eight of the cores for the parallel calculations. Numerical results are shown in Table 5.

5.2 Performance of `pami`

The solution times for `hsol`, the serial version of `pami` and `pami` running in parallel with 8 cores are listed in columns headed `hsol`, `pami1` and `pami8` respectively in Table 5. For more than 65% of the reference set, the speedup of `pami` compared to its sequential version is more than 2, with a (geometric) mean of 2.23. However, when compared to the regular dual simplex method, the sequential version of `pami` is generally less efficient (about 30% slower). The overall speedup of `pami` is thus compromised, resulting in a mean speedup (over `hsol`) of about 1.5. When solved with `pami`, a speedup of 1.8 was achieved for 8 of 30 instances, with the best speedup obtained being 2.53.

It is worth noting that the instances of better speedup (greater than the average) correspond largely to the dense LP problems. This happens partially because `pami` frequently requires a smaller number of iterations for these problems. For example, the iteration counts when solving DFL001 with `hsol` and `pami` are 26322 and 23668 respectively. However, the performance of `pami1` is worse than `hsol` because of wasted BTRAN operations and other inefficiencies.

The performance of `pami` when solving the dense instances is inconsistent. In particular, its worst performances are associated with dense LP problems. For example, using `pami` to solve MAROS-R7 and LINF_520C, results in a slowdown in both cases, and almost no speedup is obtained for SELF. This can be explained by referring to the profiling of computational components provided in Table 4 where, for these three models, INVERT requires the highest proportion of solution time among all dense LP problems.

The performance of `pami` when solving hyper-sparse LP problems is moderate but relatively stable. For the three best instances of the hyper-sparse problems, namely CRE-B, SGPF5Y6 and STP3D, the large percentage

Table 4: Iteration time (ms) and computational components profiling (the percentages of overall solution time) when solving LP problems with an advanced dual revised simplex method implementation

		Group 1			Group 2		Group 3				Group 4	
Model	Iter. Time	CHUZR	CHUZC1	CHUZC2	PRICE	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

Table 5: Performance of `pami` and `sip`

	Solution time				Speedup				Sparsity	
M odel	`hsol`	`pami1`	`pami8`	`sip`	`p1/hsol`	`p8/p1`	`p8/hsol`	`sip/hsol`	FTRAN	BTRAN
CRE-B	6.2	5.1	3.2	5.7	1.21	1.62	1.95	1.08	100	83
DANO3MIP_LP	52.6	76.9	24.8	35.4	0.68	3.10	2.12	1.49	1	6
DBIC1	78.9	141.3	60.0	65.1	0.56	2.36	1.31	1.21	100	83
DCP2	12.9	16.0	8.5	10.5	0.81	1.89	1.52	1.23	100	97
DFL001	15.7	23.7	8.6	12.2	0.66	2.74	1.81	1.28	34	57
FOME12	99.5	159.4	61.9	81.3	0.62	2.58	1.61	1.22	45	58
FOME13	256.1	371.0	168.5	207.4	0.69	2.20	1.52	1.24	100	98
KEN-18	13.5	16.0	10.4	17.1	0.84	1.54	1.30	0.79	100	100
L30	10.9	23.3	8.5	8.5	0.47	2.74	1.29	1.28	10	8
LINF_520C	3460.6	9182.9	4587.5	2644.6	0.38	2.00	0.75	**1.31**	10	11
LP22	22.0	36.4	13.2	15.1	0.61	2.75	1.67	1.45	13	22
MAROS-R7	11.3	37.4	24.0	10.1	0.30	1.56	0.47	**1.12**	5	13
MOD2	52.7	103.2	40.7	42.3	0.51	2.53	1.29	1.25	46	68
NS1688926	24.8	41.2	19.6	20.1	0.60	2.10	1.26	1.23	72	100
NUG12	125.2	197.8	70.5	108.3	0.63	2.81	1.78	1.16	1	20
PDS-40	28.4	42.2	21.0	24.6	0.67	2.00	1.35	1.16	100	98
PDS-80	66.0	109.4	57.0	63.1	0.60	1.92	1.16	1.05	100	99
PDS-100	84.3	122.8	65.5	79.7	0.69	1.88	1.29	1.06	100	99
PILOT87	6.6	11.0	4.4	5.1	0.60	2.48	1.50	1.31	10	19
QAP12	157.5	170.9	62.1	190.8	0.92	2.75	2.53	0.83	2	15
SELF	39.0	66.2	36.5	28.7	0.59	1.81	1.07	**1.36**	0	2
SGPF5Y6	169.0	213.5	88.8	252.2	0.79	2.40	1.90	0.67	100	100
STAT96V4	156.7	235.3	67.3	76.4	0.67	3.50	2.33	2.05	73	31
STORMG2-125	9.4	11.0	6.5	12.3	0.85	1.70	1.44	0.76	100	100
STORMG2-1000	427.0	576.6	256.7	510.2	0.74	2.25	1.66	0.84	100	100
STP3D	565.2	546.3	234.9	492.1	1.03	2.33	2.41	1.15	95	70
TRUSS	7.9	10.8	4.0	5.0	0.73	2.67	1.94	1.58	37	2
WATSON_1	50.2	59.4	32.4	62.6	0.85	1.83	1.55	0.80	100	100
WATSON_2	53.6	57.5	33.3	68.4	0.93	1.72	1.61	0.78	100	100
WORLD	65.1	120.7	47.5	51.3	0.54	2.54	1.37	1.27	41	61
GEOMEAN	50.8	76.0	34.1	44.8	0.67	2.23	1.49	1.13		

solution time (see Table 4 for reference) of PRICE (42.9% for CRE-B and 19.2% for STP3D) and FTRAN-DSE (80.7% for SGPF5Y6 and 27% for STP3D) explains the good speedup. In `pami`, the PRICE 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.

The overall performance of `pami` is assessed via the performance profile shown in Figure 3. As the figure indicates, the performance of `pami` is comparable with the dual simplex implementation of `cplex` 12.4 [15], a world-leading commercial dual revised simplex solver, and clearly superior to that of `clp` 1.15 [2], the world's leading open-source solver.

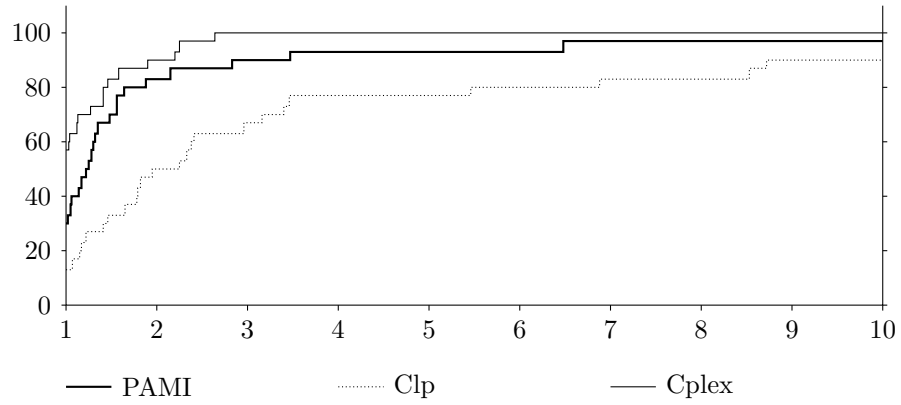

Figure 3: Performance profile of `pami`, `clp` and `cplex`

5.3 Performance of `sip`

Clearly the overall performance and average speedup (1.13) 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, in most cases, applying `sip` results in a slowdown. A typical example is SGPF5Y6, where the proportion of FTRAN-DSE is more than 80% and the total proportion of PRICE, 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 relative 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`, as a straightforward parallelisation approach 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.

6 Conclusions

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

One relatively complicated parallelisation (`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 an average speedup of 1.49. Though not extremely impressive, the resulting performance of `pami` is comparable to the dual simplex implementation of `cplex`, which is a world-leading commercial simplex solver.

The other parallelisation scheme (`sip`) exploits purely single iteration parallelism. Although its average speedup of 1.13 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 to the FICO dual revised simplex solver made by Huangfu which have been considered noteworthy enough to be reported [13]. Indeed, FICO has presented results indicating that their dual revised simplex solver is currently marginally faster than that of `cplex` or gurobi.

References

[1] R. E. Bixby and A. Martin. Parallelizing the dual simplex method. *INFORMS Journal on Computing*, 12(1):45–56, 2000.

[2] COIN-OR. Clp. `http://www.coin-or.org/projects/Clp.xml`, 2014. Accessed: 30/07/2014.

[3] J. M. Elble and N. V. Sahinidis. A review of the LU update in the simplex algorithm. *International Journal of Mathematics in Operational Research*, 4(4):366–399, 2012.

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

[5] J. J. H. Forrest and J. A. Tomlin. Updated triangular factors of the basis to maintain sparsity in the product form simplex method. *Mathematical Programming*, 2:263–278, 1972.

[6] R. Fourer. Notes on the dual simplex method. Technical report, Department of Industrial Engineering and Management Sciences Northwestern University, 1994. Unpublished.

[7] J. Hall and Q. Huangfu. 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, pages 143–151, Berlin, Heidelberg, 2012. Springer-Verlag.

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

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

[10] J. A. J. Hall and K. I. M. McKinnon. ASYNPLEX, an asynchronous parallel revised simplex method algorithm. *Annals of Operations Research*, 81:27–49, 1998.

[11] J. A. J. Hall and K. I. M. McKinnon. Hyper-sparsity in the revised simplex method and how to exploit it. *Computational Optimization and Applications*, 32(3):259–283, December 2005.

[12] P. M. J. Harris. Pivot selection methods of the Devex LP code. *Mathematical Programming*, 5:1–28, 1973.

[13] Q. Huangfu. The Algorithm that Runs the World. `http://ficolabsblog.fico.com/2014/05/the-algorithm-that-runs-the-world-.html`, 2014. Accessed: 30/07/2014.

[14] Q. Huangfu and J. A. J. Hall. Novel update techniques for the revised simplex method. Technical Report ERGO-13-001, School of Mathematics, University of Edinburgh, 2013. Accepted for publication in Computational Optimization and Applications.

[15] IBM. ILOG CPLEX Optimizer. http://www.ibm.com/software/integration/optimization/cplex-optimizer/, 2014. Accessed: 30/07/2014.

[16] 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(2):185–204, November 2008.

[17] H. D. Mittelmann. Benchmarks for optimization software. http://plato.la.asu.edu/bench.html, 2014. Accessed: 30/07/2014.

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

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