PARSMI, a parallel revised simplex algorithm incorporating minor iterations and Devex pricing

J.A.J.Hall, K.I.M.McKinnon

September 1996

MS 96-012

Supported by EPSRC research grant GR/J0842

Presented at PARA96 Copenhagen 21st August 1996: Workshop on
Applied Parallel Computing in Industrial Problems and Optimization

Department of Mathematics and Statistics
University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ
Tel. (33) 131 650 5075 E-Mail : jajhall@maths.ed.ac.uk, ken@maths.ed.ac.uk

PARSMI, a parallel revised simplex algorithm incorporating minor iterations and Devex pricing

J. A. J. Hall and K. I. M. McKinnon

Department of Mathematics and Statistics, University of Edinburgh

Abstract. When solving linear programming problems using the revised simplex method, two common variants are the incorporation of minor iterations of the standard simplex method applied to a small subset of the variables and the use of Devex pricing. Although the extra work per iteration which is required when updating Devex weights removes the advantage of using minor iterations in a serial computation, the extra work parallelises readily. An asynchronous parallel algorithm PARSMI is presented in which computational components of the revised simplex method with Devex pricing are either overlapped or parallelism is exploited within them. Minor iterations are used to achieve good load balance and tackle problems caused by limited candidate persistence. Initial computational results for a six-processor implementation on a Cray T3D indicate that the algorithm has a significantly higher iteration rate than an efficient sequential implementation.

1 Introduction

The revised simplex method for solving linear programming (LP) is one of the most widely used numerical methods. Although barrier methods are often the preferred method for solving single large LP problems, the revised simplex method is the most efficient solution procedure when families of related LP problems have to be solved. Despite its importance, relatively little progress has been reported on parallel methods based on the revised simplex algorithm.

Special techniques such as minor iterations or edge weight based pricing strategies are used by serial implementations of the revised simplex method in order to achieve faster solution times. The major computational steps in the revised simplex method with minor iterations and the Devex pricing strategy are described in Section 2. These steps are also used in the parallel algorithm, PARSMI, which is described in Section 3. PARSMI exploits parallelism within the computational steps which parallelise naturally and overlaps the remaining computational steps.

Promising computational results obtained using a prototype implementation of PARSMI on six processors of a Cray T3D are given in Section 4. These results illustrate the behaviour a major part of the algorithm. The extension of the implementation to more processors is discussed in Section 5.

1.1 Background

A linear programming problem has standard form

$$\begin{aligned} \text{minimize} \quad & f = \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ \text{where} \quad & \boldsymbol{x} \in I\!R^n \quad \text{and} \quad \boldsymbol{b} \in I\!R^m. \end{aligned} \tag{1}$$

At any stage in the simplex method the indices of the variables are partitioned into set $\mathcal{B}$ of m basic variables and set $\mathcal{N}$ of $n-m$ nonbasic variables. The nonbasic variables are given the value zero and the values of the basic variables are determined by the equations in (1). The components of $\boldsymbol{c}$ and columns of A corresponding to the index set $\mathcal{B}$ are the basic costs $\boldsymbol{c}_B$ and basis matrix B. Those corresponding to the set $\mathcal{N}$ are the non-basic costs $\boldsymbol{c}_N$ and matrix N.

A feasible point $\boldsymbol{x}$ corresponding to such a partition of the indices is a vertex of the feasible region of (1). At this vertex, the simplex method first determines the direction of an edge of the feasible region along which the objective function decreases and then finds the step to the next vertex of the feasible region along this edge. This corresponds to increasing a nonbasic variable from zero until one of the basic variables is reduced to zero. Repeating this process determines a path along edges of the feasible region which terminates at a vertex for which there is no edge along which the objective decreases.

The two main variants of the simplex method correspond to different means of calculating the data required to determine the step to the new vertex. The first variant is the standard simplex method in which the directions of all edges at the current vertex and the gradient of the objective in each of these directions are maintained in a rectangular tableau. In the revised simplex method, the vector of gradients and the direction of the chosen edge are determined by solving systems involving the matrix B, using a factored form of its inverse, and forming matrix-vector products using the matrix N. For large sparse problems the standard simplex method is completely uncompetitive in terms of speed compared with the revised simplex method. It is also much less stable numerically.

A simple parallel implementation of the revised simplex method using a sparse LU decomposition is described by Shu and Wu in [7]. However they only parallelise the sparse matrix-vector products and, as a result, report little or no speed-up over their serial implementation for general problems. Bixby and Martin discuss parallelising the CPLEX dual simplex implementation in [2] where, once again, parallelism was only successfully exploited in the sparse matrix-vector products, with minimal speed-up on general problems.

Only recently have parallel algorithms been developed which give worthwhile speed-up relative to an efficient serial implementation. Hall and McKinnon have developed an algorithm, ParLP, which is described in [5] and achieve a speed-up of up to 5 on general problems. Wunderling described an algorithm in [8] which is completely parallel for 2 processors.

2 The serial revised simplex method

It is possilbe incorporate good features of the standard into the revised simplex method by using minor iterations. Within minor iterations the standard simplex method is applied to the LP sub-problem corresponding to a small subset of the variables. This is a central feature of PARSMI. There are various strategies for weighting each reduced cost. They aim to predict the likely cost reduction in an iteration and so reduce the number of iterations . Devex pricing described by Harris in [6] is such a strategy and is used in PARSMI. It is commonly the default in efficient sequential implementations. The major computational steps of the revised simplex method with minor iterations and Devex pricing are illustrated in Figure 1.

`CHUZC:` Scan $\hat{\boldsymbol{c}}_N$ for a set $\mathcal{Q}$ of good candidates to enter the basis.
`FTRAN:` Form $\hat{\boldsymbol{a}}_j = B^{-1}\boldsymbol{a}_j$, $\forall\, j \in \mathcal{Q}$, where $\boldsymbol{a}_j$ is column j of A.
`Loop` {minor iterations}
    `CHUZC_MI:` Scan $\hat{\boldsymbol{c}}_{\mathcal{Q}}$ for a good candidate q to enter the basis.
    `CHUZR:` Scan the ratios $\hat{b}_i/\hat{a}_{iq}$ for the row p of a good candidate to leave the basis, where $\hat{\boldsymbol{b}} = B^{-1}\boldsymbol{b}$ (ratio test). Let $\alpha = \hat{b}_p/\hat{a}_{pq}$.
    `UPDATE_MI:` Update $\mathcal{Q} := \mathcal{Q}\backslash\{q\}$.
    Update $\hat{\boldsymbol{b}} := \hat{\boldsymbol{b}} - \alpha\hat{\boldsymbol{a}}_q$.
    Update the columns $\hat{\boldsymbol{a}}_{\mathcal{Q}}$ and reduced costs $\hat{\boldsymbol{c}}_{\mathcal{Q}}$.
    Update the Devex weights for the candidates in $\mathcal{Q}$.
`End loop` {minor iterations}
`For` {each basis change} `do`
    `BTRAN:` Form $\boldsymbol{\pi}^T = \boldsymbol{e}_p^T B^{-1}$.
    `PRICE:` Form pivotal row $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}^T N$.
        Update reduced costs $\hat{\boldsymbol{c}}_N := \hat{\boldsymbol{c}}_N - \hat{c}_q\hat{\boldsymbol{a}}_p^T$ and Devex weights.
    `If` {growth in factors} `then INVERT:` Form factored inverse of B.
    `else UPDATE:` Update the inverse of B corresponding to the basis change.
`End do`

Fig. 1. The revised simplex method with minor iterations and Devex pricing

When a single nonbasic variable x_j increases by unit amount, the basic variables must change by $-B^{-1}\boldsymbol{a}_j$ so as to maintain equation $A\boldsymbol{x} = \boldsymbol{b}$. (Here $\boldsymbol{a}_j$ is column j of A.) Thus the corresponding change in the objective function value is the *reduced cost* $\hat{c}_j = c_j - \boldsymbol{c}_B^T B^{-1}\boldsymbol{a}_j$. The traditional 'Dantzig' criterion for determining the quality of nonbasic variables is just the size of the reduced cost. However, if the vector $\hat{\boldsymbol{a}}_j = B^{-1}\boldsymbol{a}_j$ is large relative to $\hat{c}_j$ it is likely that only a small increase in x_j will be possible before one of the basic variables is reduced to zero. Thus edge weight based pricing strategies, such as Devex, weight the reduced cost by dividing it by (a measure of) the length of $\hat{\boldsymbol{a}}_j$. The Devex strategy

maintains weights $s_j \approx 1 + \|\hat{\boldsymbol{a}}_j\|_R^2$, where $\|.\|_R$ is the 2-norm taken over a set R. This set R is initially empty, with corresponding unit initial weights. Following each subsequent basis change, one index is either added to or removed from the set R and periodically R is reset to be empty. The details of this are described by Harris in [6] and the computation required in order to update the weights is outlined below.

2.1 The revised simplex method with minor iterations and Devex pricing

At the beginning of a major iteration in Figure 1, it is assumed that the vector of reduced costs $\hat{\boldsymbol{c}}_N^T = \boldsymbol{c}_N^T - \boldsymbol{c}_B^T B^{-1} N$ and the vector $\hat{\boldsymbol{b}} = B^{-1}\boldsymbol{b}$ of current values of the basic variables are known and that a factored inverse of the basis matrix B is available. The first operation is CHUZC which scans the weighted reduced costs to determine a set $\mathcal{Q}$ of good candidates to enter the basis. The inner loop then applies the standard simplex method to the LP problem corresponding to the candidates in $\mathcal{Q}$ so requires the corresponding columns $\hat{\boldsymbol{a}}_j = B^{-1}\boldsymbol{a}_j$ of the standard simplex tableau. These columns are formed by passing forwards through the factors of B^{-1}, an operation known as FTRAN. The matrix formed by the columns $\hat{\boldsymbol{a}}_j$, $j \in \mathcal{Q}$ and the corresponding vector of reduced costs for the candidates $j \in \mathcal{Q}$ are conveniently denoted by $\hat{\boldsymbol{a}}_{\mathcal{Q}}$ and $\hat{\boldsymbol{c}}_{\mathcal{Q}}$.

In each minor iteration, CHUZC_MI scans the weighted reduced costs of the candidates in $\mathcal{Q}$ and selects one, q say, to enter the basis. The variable to leave the basis is determined by CHUZR which scans the ratios $\hat{b}_i/\hat{a}_{iq}$. In the discussion below, p is used to denote the index of the row in which the leaving variable occurred and p' denotes the index of the leaving variable. The value $\alpha = \hat{b}_p/\hat{a}_{pq}$ is the new value of x_q. The vector of new values of the basic variables is given by

$$\hat{\boldsymbol{b}} := \hat{\boldsymbol{b}} - \alpha\hat{\boldsymbol{a}}_q \tag{2}$$

and clearly has a zero value in component p. Once the indices q and p' have been interchanged between the sets $\mathcal{B}$ and $\mathcal{N}$, a *basis change* is said to have occurred. The standard simplex tableau corresponding to the new basis is obtained by updating the previous tableau in an operation known as UPDATE_MI. The values of the basic variables are updated according to (2). The matrix $\hat{\boldsymbol{a}}_{\mathcal{Q}}$ and reduced costs $\hat{\boldsymbol{c}}_{\mathcal{Q}}$ are updated by a Gauss-Jordan elimination step and the Devex weights are updated using row p of the updated tableau.

Since the variable to enter the basis is removed from the set $\mathcal{Q}$, the number of minor iterations which are performed is limited by the initial cardinality of $\mathcal{Q}$. However, due to the changes in the reduced costs, $\mathcal{Q}$ may consist of purely unattractive candidates before it becomes empty. In this case the work done in forming and updating the corresponding standard simplex columns is wasted. The number of minor iterations which are actually performed depends on the extent to which candidates remain attractive. This is an example of the property of *candidate persistence* which is highly problem-dependent and has a major influence on the effectiveness of the parallel algorithms described by Hall and McKinnon in [5] and Wunderling in [8].

Before the next major iteration can be performed it is necessary to update the reduced costs and Devex weights and obtain a factored inverse of the new basis. Updating the reduced costs and Devex weights following each basis change requires the corresponding pivotal row $\hat{\boldsymbol{a}}_p^T = \boldsymbol{e}_p^T B^{-1} N$ of the (full) standard simplex tableau. This is obtained in two steps. First the vector $\boldsymbol{\pi}^T = \boldsymbol{e}_p^T B^{-1}$ is formed by passing backwards through the factors of B^{-1}, an operation known as BTRAN, and then the vector $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}^T N$ of values in the pivotal row is formed. This sparse matrix-vector product with N is referred to as PRICE. Once the reduced costs and Devex weights have been updated, the factored inverse of the new basis is obtained by updating the current factored inverse (the UPDATE operation). Note that, eventually it will be either more efficient, or necessary for numerical stability, to find a new factored inverse using the INVERT operation.

2.2 FTRAN and BTRAN

In order to describe the parallel algorithm PARSMI, it is necessary to consider the FTRAN and BTRAN operations in further detail. Let the basis matrix whose inverse is factored by INVERT be denoted by B_0 and the basis matrix after a further U basis changes be denoted by $B_U = B_0 E_U$ so $B_U = E_U^{-1} B_0^{-1}$. Let $\mathcal{P}$ be the set of rows in which basis changes have occurred. Note that for large LP problems $P = |\mathcal{P}| \ll m$. The most convenient representation of the matrix E_U^{-1} is as a product of Gauss-Jordan elimination matrices. In FTRAN the vector $\hat{\boldsymbol{a}}_j = B_U^{-1} \boldsymbol{a}_j$ is formed by the operations $\tilde{\boldsymbol{a}}_q = B_0^{-1} \boldsymbol{a}_q$ and $\hat{\boldsymbol{a}}_q = E_U^{-1} \tilde{\boldsymbol{a}}_q$. In BTRAN the vector $\boldsymbol{\pi}^T = \boldsymbol{e}_p^T B_U^{-1}$ is formed by the operations $\tilde{\boldsymbol{\pi}}^T = \boldsymbol{e}_p^T E_U^{-1}$ and $\boldsymbol{\pi}^T = \tilde{\boldsymbol{\pi}}^T B_0^{-1}$. Note that forming $\tilde{\boldsymbol{\pi}}$ only requires rows $i \in \mathcal{P}$ of the elimination matrices.

3 PARSMI, a parallel revised simplex algorithm

The concept of PARSMI is to dedicate some processors to performing minor iterations in parallel, while simultaneously using other processors to select and form the vectors $\tilde{\boldsymbol{a}}_{q'}$ for new good candidates to be added to $\mathcal{Q}$. Other processors are used to form the $\boldsymbol{\pi}$ vectors and perform the PRICE operations required to bring the reduced costs and Devex weights up-to-date.

One processor, referred to as the MI_CT processor, is dedicated to controlling the minor iterations and performing the computation which uses the data in rows $i \in \mathcal{P}$ of the elimination matrices which represent E_U^{-1}. When it the vector $\tilde{\boldsymbol{a}}_{q'}$ has been formed for a good candidate, q' is added to $\mathcal{Q}$ and its current reduced cost is calculated (and subsequently updated) using just the data in rows $i \in \mathcal{P}$ of the elimination matrices. Only when a candidate is actually chosen as the variable x_q to enter the basis is the full vector $\hat{\boldsymbol{a}}_q = E_U^{-1} \tilde{\boldsymbol{a}}_q$ actually computed with the consequent advantage that the cost of forming $\hat{\boldsymbol{a}}_j$ for candidates which are discarded without entering the basis is avoided. This operation, together with CHUZR and operation (2), called *update RHS*, can be distributed by rows on the minor iteration (MI) processors.

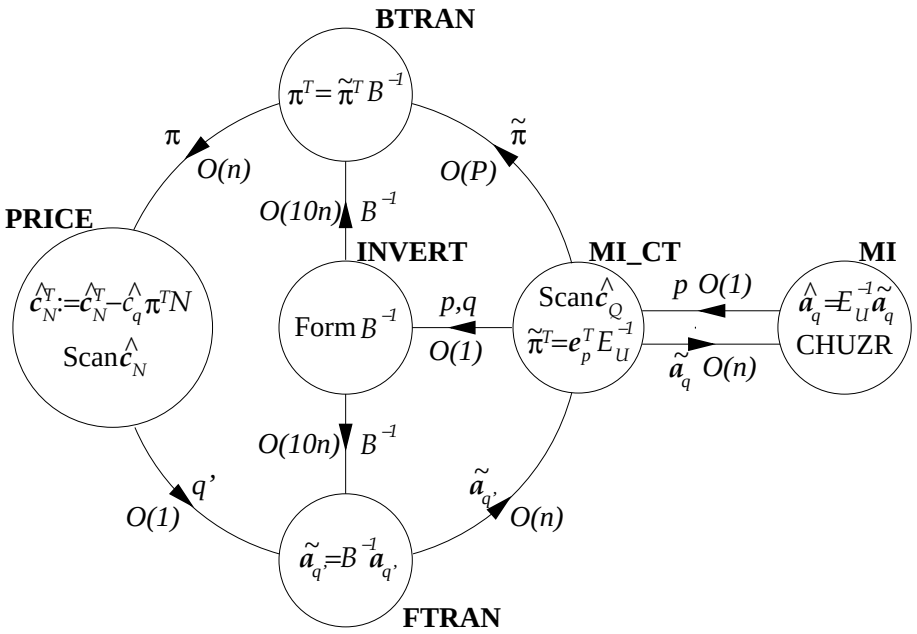

Fig. 2. Six-processor implementation of PARSMI

Once a basis change has been determined, the vector $\tilde{\boldsymbol{\pi}}^T = \boldsymbol{e}_p^T E_U^{-1}$ is formed on the MI_CT processor and one of several BTRAN processors completes the calculation of $\boldsymbol{\pi}^T = \tilde{\boldsymbol{\pi}}^T B_0^{-1}$. The subsequent calculation of $\boldsymbol{\pi}^T N$, update of the reduced costs and Devex weights and the CHUZC operation are distributed over a number of PRICE processors and yields new good candidates for which vectors $\tilde{\boldsymbol{a}}_{q'}$ are formed on the FTRAN processors. One or more INVERT processors are dedicated to keeping the basis B_0 as up-to-date as possible in order to reduce the operations required to form $\hat{\boldsymbol{a}}_q$ from $\tilde{\boldsymbol{a}}_q$.

3.1 Implementation

The algorithm could be implemented on a shared memory multiprocessor or a distributed memory machine with high ratio of communication speed to computation speed. The minimum number of processors required to implement the algorithm is six, and such an implementation is currently being developed on the Edinburgh Cray T3D. The implementation shares its fundamental computational components with a highly efficient sequential implementation of the revised simplex method. This is competitive with commercial simplex solvers and is used to assess the performance of the parallel implementation. Most message passing, in particular all short messages, is done via the Cray-specific shared memory SHMEM [1] subroutine library, with the remaining message-passing being done via the Parallel Virtual Machine (PVM) subroutine library [4].

On a distributed memory machine there is an additional benefit of splitting the FTRAN and the BTRAN operations since the data required to form $\boldsymbol{a}_q$ from $\tilde{\boldsymbol{a}}_q$ is already available on the processor on which it is used.

The major computation and communication requirements of the six-processor implementation of PARSMI are illustrated in Figure 2. An alternative view of the algorithm is provided by the Gantt chart in Figure 3. Each box represents the time spent performing a particular operation on a given processor, with SI corresponding to sending a new factored inverse, C being CHUZC, F being FTRAN and R being CHUZR. The unlabelled box following CHUZR corresponds to updating the RHS.

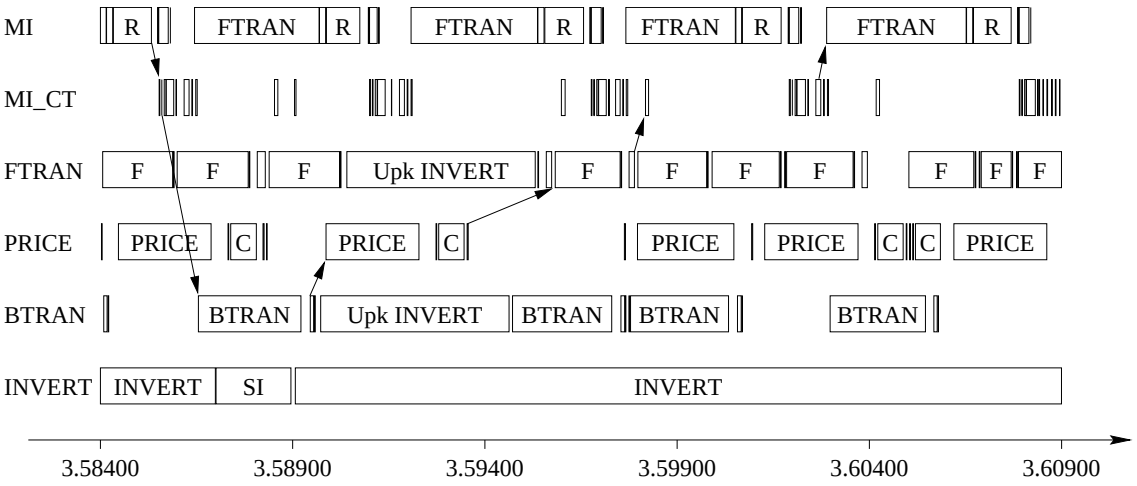

Fig. 3. Gantt chart of processor activity for 25FV47

4 Computational results

The results presented in this section are for the six processor implementation on the Edinburgh Cray T3D. They demonstrate the effectiveness of a key part of the algorithm and motivate the discussion of implementations using a larger number of processors in Section 5. The current implementation is only for feasible starting bases and only permits a few tens of iterations to be performed. However this is long enough to get a good assessment of the steady state time per iteration when started from a feasible basis. Thus the decrease in the time per iteration of the six-processor implementation over that for a comparable serial implementation may be determined. The effect on the total number of iterations due to using out-of-date reduced costs cannot yet be assessed.

A number of experiments were performed using the classic Netlib [3] test problem 25FV47 for which the serial revised simplex implementation obtains

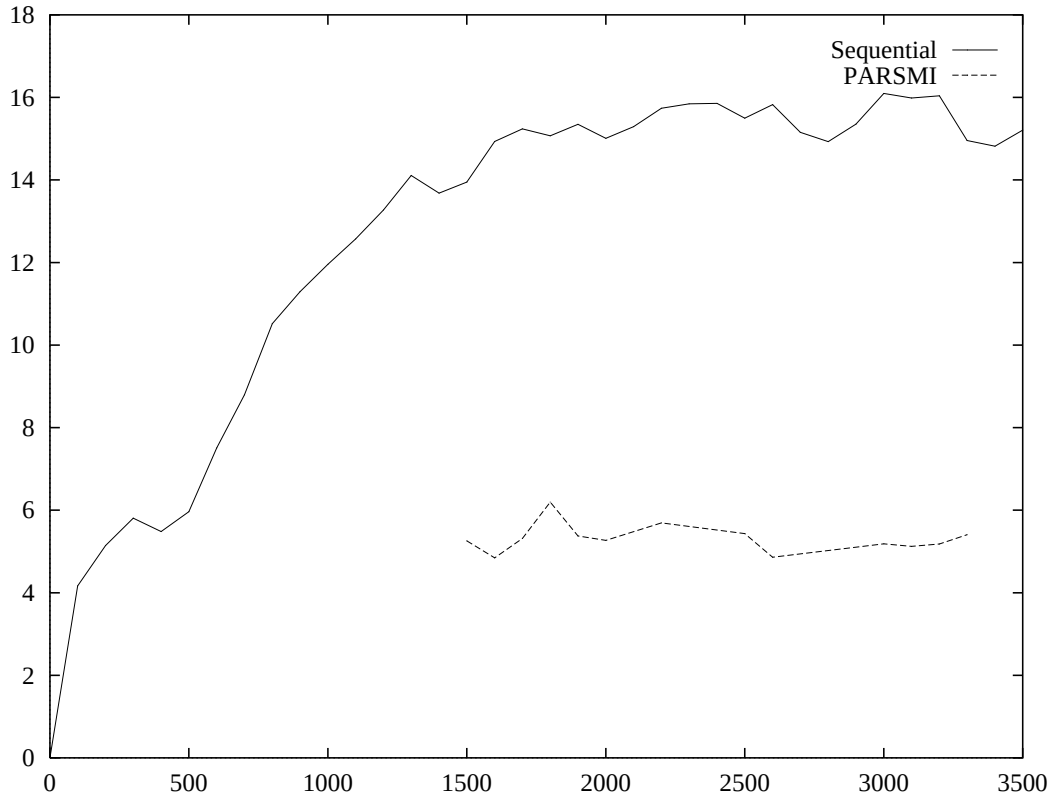

Fig. 4. Time per iteration for 25FV47

feasibility after 1500 iterations. Every hundredth basis from this point was stored and the time per iteration (averaged over 5 iterations) at each of these bases was recorded. This is represented by the solid line on the graph illustrated in Figure 4 and is approximately $16ms$ from the point at which feasibility is obtained. The six-processor implementation of PARSMI was started from each of the stored feasible bases and run until a steady state average time per iteration was reached. This time per iteration is represented by the dotted line in Figure 4 and is approximately $5ms$, corresponding to a speed-up of about 3. Given that six processors are being used the efficiency is about 50%. Analysis of the Gantt chart in Figure 3 shows that the bottleneck in the parallel implementation is the time taken to complete FTRAN and then perform CHUZR and update RHS. For the serial implementation these have average computation times of $1.8ms$, $1.0ms$ and $0.3ms$ respectively, a total of $3.1ms$, an iteration speed which, if achieved by the parallel implementation would correspond to a best possible speed-up of 5. With the parallel implementation the average time taken to complete FTRAN is $2.7ms$ so the total time for the bottleneck operations is $4.0ms$, an iteration speed which would correspond to a speed-up of 4. There remains scope for improvement

in the efficiency of the current parallel implementation which will allow the best possible speed-up to be approached.

The reason for the greater average time taken to complete FTRAN within the parallel implementation is readily explained. The serial implementation reinverts the basis every 25 iterations leading to an average number of 12 updates which must be applied. Within the parallel implementation approximately 12 basis changes occur during INVERT thus the number of updates which must be applied when completing FTRAN ranges between 12 and 24: the average of 18 corresponds to the 50% increase in the average time.

5 Conclusions

The computational results given in the previous section for the six-processor implementation of PARSMI demonstrate the effectiveness of part of the algorithm: the increase in the iteration speed approaches the best that can be achieved. The bottleneck in the parallel implementation which limits the decrease in the time per iteration is the computation required to complete FTRAN and then perform CHUZR and update RHS. The computation required by each of these operations could be distributed over two or more processors, without significant duplication of work, allowing the best possible time per iteration to be reduced by a corresponding factor. Note that as the time per iteration decreases, there is a consequent increase in the number of basis changes by which the factored inverse is out-of-date when INVERT is completed, and hence the number of updates which must be applied also increases. Thus the time per iteration cannot be scaled down indefinitely without additional processors being employed to increase the frequency with which fresh factored inverses are available.

However, a more critical limitation of the iteration speed once the minor iterations are parallelised is likely to be the rate at which the pivotal columns of candidates to enter the basis can be generated. The average time taken to perform FTRAN (with B_0^{-1}) is $1.8ms$ so, even if all candidates were attractive, this would represent a limit on the time per iteration unless more than one processor were devoted to FTRAN.

If the BTRAN and PRICE operations are not performed with sufficient frequency, candidates will be chosen with respect to increasingly out-of-date reduced costs. As a result, a decreasing proportion of these will still be attractive when their reduced cost is brought up-to-date. This will not only decrease the frequency of basis changes but is likely to lead to a significant increase in the number of iterations required to solve the problem since the candidate which actually enters the basis may be far from being the best with respect to the Devex criterion. Progress may even temporarily stall even though the current basis is not optimal because the most up-to-date reduced cost available indicate no candidates remain. Parallelising PRICE and overlapping more than one BTRAN would therefore be attractive.

For a given time per iteration t, the number of basis changes by which the reduced cost of a candidate is out-of-date when it enters the basis is at least T/t,

where T is the total time required to perform a BTRAN, PRICE and start FTRAN (form $\tilde{\boldsymbol{a}}_q = B_0^{-1}\boldsymbol{a}_q$). The sequence of arrows in Figure 3 show the communication and computation which is required between a basis change being determined and a candidate, chosen with respect to the reduced costs for that basis, being ready to enter a future basis. In this example the reduced cost is three basis changes out-of-date. Even if parallelism is fully exploited within a single PRICE operation, T is at least the serial time to perform BTRAN and start FTRAN, unless parallelism is exploited within BTRAN and FTRAN a task which is beyond the scope of this project.

With a larger number of processors available, the optimal distribution of the processors to activities will vary both during the solution procedure (as PRICE becomes relatively cheaper than FTRAN and BTRAN) and from one LP problem to another. LPs with a large ratio of columns to rows will benefit from a relatively more PRICE processors. Problems with proportionally more rows, and those for which the vectors $\hat{\boldsymbol{a}}_j$ are relatively dense, will benefit from relatively more processors being used to perform minor iterations. If strategies can be developed to prevent the number of iteration required to solve a problem increasing significantly, the potential fast iteration speed and adaptability of the algorithm to the full range of LP problems can be expected to lead to a parallel implementation which gives a significantly improvement over the sequential implementation of the revised simplex method.

References

1. R. Barriuso and A. Knies. *SHMEM User's guide for Fortran.* Cray Research inc.
2. R. E. Bixby and A. Martin. Parallelizing the dual simplex method. Technical Report SC-95-45, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1995.
3. D. M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter*, 13:10–12, 1985.
4. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing.* MIT Press.
5. J. A. J. Hall and K. I. M. McKinnon. An asynchronous parallel revised simplex method algorithm. Technical Report MS 95-50, Department of Mathematics and Statistics, University of Edinburgh, 1995.
6. P. M. J. Harris. Pivot selection methods of the Devex LP code. *Math. Prog.*, 5:1–28, 1973.
7. W. Shu and M. Wu. Sparse implementation of revised simplex algorithms on parallel computers. In *Proceedings of* 6^{th} *SIAM Conference on Parallel Processing for Scientific Computing*, pages 501–509, 1993.
8. R. Wunderling. Parallelizing the simplex algorithm. ILAY Workshop on Linear Algebra in Optimzation, Albi, April 1996.

This article was processed using the LaTeX macro package with LLNCS style