ASYNPLEX, an asynchronous parallel revised simplex algorithm

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

July 1997

MS 95-050a

Supported by EPSRC research grant GR/J08942

Presented at APMOD95 Brunel University 3rd April 1995

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

ASYNPLEX, an asynchronous parallel revised simplex algorithm

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

15^{th} July 1997

Abstract

This paper describes ASYNPLEX, an asynchronous variant of the revised simplex method which is suitable for parallel implementation on MIMD computers with fast inter-processor communication. The method overlaps simplex iterations on different processors. Candidates to enter the basis are tentatively selected using reduced costs which may be out of date. Later the up-to-date reduced costs of the tentative candidates are calculated and candidates are either discarded or accepted to enter the basis. The implementation of this algorithm on a Cray T3D is described and results demonstrating significant speed-up are presented.

1 Introduction

Linear programming (LP) is a widely applicable technique both in its own right and as a sub-problem in the solution of other optimization problems. The revised simplex method and the barrier method are the two efficient methods for general LP problems on serial machines. There have been successful parallel implementations of the barrier method but as yet little progress has been reported on parallel methods based on the revised simplex algorithm. In contexts where families of related LP problems have to be solved, such as in integer programming and decomposition methods, the revised simplex method is usually the more efficient method, so there is strong motivation to devise a parallel version of this method. If this is to be of value then it should be significantly faster than current serial simplex solvers.

The major computational steps in the revised simplex method are described later in this section and general approaches to exploiting

parallelism are discussed. The particular approach to exploiting parallelism which is considered in this paper is to overlap simplex iterations performed on a number of *iteration processors*, with an additional *invert processor* devoted to calculating a factored inverse of simplex basis matrices.

Candidates for variables to enter the basis are tentatively selected using the most recently available reduced costs, and the true reduced costs for these columns are calculated cheaply later before finally deciding whether or not they will enter the basis. The effectiveness of the method relies on there being some persistence in the values of the reduced costs in the course of a small number of iterations. The algorithm is a variant of the simplex method and as such follows a single path on the surface of the feasible region. This requires the coordination of the basis change decisions among all the processors.

The algorithm ASYNPLEX is described in Section 2, and a description of its implementation on a Cray T3D is given in Section 2.6. The properties of the four test problems selected from the Netlib set [2] are discussed in Section 2.7 and results demonstrating significant speed-up for these problems are presented in Section 2.8. Conclusions are offered in Section 3.

1.1 Background

The two main variants of the simplex method are the standard simplex method and the revised simplex method. Although early versions of the revised simplex method used an explicit form of the inverse, this was quickly replaced by methods based on a factored form of the inverse. Most important LP problems are large (some with millions of variables and constraints) and sparse (the coefficient matrix has an average of 5–10 non-zeros per column). An important point to note is that for large sparse problems the standard simplex and the explicit inverse form of the revised simplex are completely uncompetitive in speed compared with the revised simplex method when a factored form of the inverse is used.

There have been several studies such as those by Luo *et. al* in [9] and Stunkel in [11] which have implemented either the standard form of the simplex or the revised simplex with the inverse of the basis matrix stored as a full matrix. Both methods parallelise well but, as noted above, are so bad for large sparse problems, that the results are a lot slower than a good serial implementation.

A simple parallel implementation of the revised simplex method using a sparse LU decomposition is described by Shu and Wu in [10]. However they only parallelise the PRICE operation and report little or no speed-up over their serial implementation.

`BTRAN:` Form $\boldsymbol{\pi}^T = \boldsymbol{c}_B^T B^{-1}$.
`PRICE:` Calculate the *reduced costs* $\hat{\boldsymbol{c}}_N^T = \boldsymbol{c}_N^T - \boldsymbol{\pi}^T N$.
`CHUZC:` Scan $\hat{\boldsymbol{c}}_N$ for the best candidate q to enter the basis.
If no such candidate exists then exit (optimality).
`FTRAN:` Form $\hat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$, where $\boldsymbol{a}_q$ is column q of A.
`CHUZR:` Scan the ratios $\dfrac{\hat{b}_i}{\hat{a}_{iq}}$ for a good candidate to leave the basis,
where $\hat{\boldsymbol{b}} = B^{-1}\boldsymbol{b}$ (ratio test).
`If` {growth in factors} `then`
`INVERT:` Find a factored inverse of B.
`else`
`UPDATE:` Update the inverse of B corresponding to the basis change.
`endif`

Figure 1: A major iteration of the revised simplex method

1.2 The revised simplex method

A linear programming problem has the form

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

At any stage in the simplex method the variables are partitioned into two sets, basic variables $\boldsymbol{x}_B$ and nonbasic variables $\boldsymbol{x}_N$. The set of basic variables is referred to as the basis. If the problem is partitioned correspondingly then the objective function is $f = \boldsymbol{c}_B^T\boldsymbol{x}_B + \boldsymbol{c}_N^T\boldsymbol{x}_N$, the constraints are $B\boldsymbol{x}_B + N\boldsymbol{x}_N = \boldsymbol{b}$ and the basis matrix B is nonsingular. Each basic variable is identified with a particular row of the constraint matrix A, and each nonbasic variable is identified with a particular column of the matrix N. The major computational steps of the revised simplex method are illustrated in Figure 1.

At the beginning of an iteration of the revised simplex method it is assumed that a factored inverse of the basis matrix B is available, i.e. elementary row or column matrices $M_1, M_2, \ldots, M_r$ are known such that $B^{-1} = M_1 M_2 \ldots M_r$. The first operation is the calculation of the dual variables $\boldsymbol{\pi}^T = \boldsymbol{c}_B^T B^{-1}$ by passing backwards through the factors of B^{-1}, an operation known as BTRAN. This is followed by the PRICE operation, which is a sparse matrix-vector product which yields the reduced costs of the nonbasic variables. These reduced costs are scanned in the operation known

as CHUZC to find a good candidate q, say, to enter the basis. This procedure was traditionally based on the *Dantzig* criterion of selecting the candidate with the largest reduced cost. For many problems it is more efficient to use an exact or approximate steepest edge criterion. Exact steepest edge is described by Goldfarb and Reid in [5] and *Devex* approximate steepest edge is described by Harris in [7]. However a discussion of these techniques in the parallel context is beyond the scope of this paper.

In order to determine the basic variable which would be replaced by the candidate to enter the basis it is necessary to calculate the column of the standard simplex tableau corresponding to this candidate. This *pivotal column* $\hat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$, where $\boldsymbol{a}_q$ is column q of the constraint matrix A, is formed by passing forward through the factors of B^{-1}, an operation known as FTRAN. The row corresponding to the leaving variable is determined by the CHUZR operation which scans the ratios $\hat{b}_i/\hat{a}_{iq}$, where $\hat{\boldsymbol{b}} = B^{-1}\boldsymbol{b}$ is the vector of current values of the basic variables. Traditionally the leaving variable is that corresponding to the smallest non-negative ratio since this ensures that no variable exceeds its bounds after transformation. However it is preferable to use a selection criterion based on finding the most numerically advantageous candidate from those which lead to bound violations which are sufficiently small. Such 'thick pencil' techniques are described by Harris in [7] and by Gill *et al* in [4].

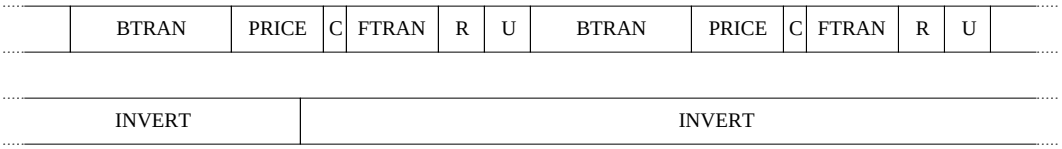
Once a basis change has occurred, the inverse of the current basis matrix is updated unless it is either more efficient or necessary on the grounds of numerical stability to find a new factored inverse. Various methods are possible for the UPDATE operation. Let the basis invert produced by INVERT be denoted by B_0 and the inverse k basis changes later by B_k. The simplest form of update is the product form update [1], which represents B_k^{-1} as

$$S_k^{-1} \ldots S_1^{-1} B_0^{-1}. \tag{1}$$

The matrix S_j is an elementary column matrix whose only non unit column is simply derived from the vector $\hat{\boldsymbol{a}}_q$ for the variable to enter the basis in that iteration and is referred to as an **eta vector**. The Bartels-Golub and Forrest-Tomlin updates modify the factors of B_0 in order to reduce the size of the factored inverse of B_k. Update procedures based on the use of a Schur complement are described by Gill *et al* in [3].

1.3 General approaches to parallelising the revised simplex method

With the exception of INVERT, the computational steps of the revised simplex method, as presented in Figure 1, cannot be performed independently. The immediate scope for parallelism is, therefore, to dedicate one processor to performing the INVERT operations and another to performing simplex iterations, as illustrated by the following Gantt chart.



C: CHUZC R: CHUZR U: UPDATE

Note that this approach also results in an increased frequency of INVERT, leading to a reduction in the size of the factors of the inverse and, hence, a reduction in the time required for BTRAN and FTRAN. A further consequence is that the basis matrix which is inverted is no longer current when INVERT is complete since basis changes have occurred as a result of simplex iterations performed on the other processor. This issue is discussed in Section 2.2 in the context of the ASYNPLEX algorithm. The efficiency of this approach of simply overlapping INVERT with simplex iterations is severely limited due to the fact that INVERT usually accounts for less than 10% of execution time for a serial LP code. Set against this meagre saving is the overhead due to communicating the factored inverse and bringing it up-to-date.

Any worthwhile parallel algorithm for the revised simplex method must either exploit parallelism within computational steps of the revised simplex method or overlap computational steps. For sparse problems, although it is easy to parallelise PRICE, CHUZC and CHUZR, the parallelism which may be exploited within BTRAN and FTRAN is limited, very fine grained and hard to achieve. For this reason the algorithm described in Section 2 exploits parallelism by using several processors to overlap the computational steps required to perform simplex iterations, and using one further processor to perform inverts.

2 ASYNPLEX, an asynchronous parallel algorithm

In the simplex method there is normally considerable flexibility in the choice of variable to enter the basis and, as mentioned earlier, different schemes have been proposed for this. Provided the variable has a negative reduced cost, a valid simplex iteration will occur. However, except in the case of (near) degeneracy, or when certain special techniques are used to obtain a feasible basis, there is no choice in the variable to leave the basis.

The parallel algorithm proposed in this section exploits parallelism by overlapping computational steps which are executed sequentially rather than by exploiting parallelism within any individual computational step. Whenever a processor is idle it is given a tentative candidate to enter the basis using the most up-to-date reduced costs which are available. Once the pivotal column is up-to-date with respect to all basis changes determined by other processors, the reduced cost for the variable, q say, can be calculated as $c_q - \boldsymbol{c}_B^T \hat{\boldsymbol{a}}_q$. If this reduced cost is negative then the iteration can continue, otherwise the column is rejected and the work done in performing the FTRAN is wasted. Whenever a processor determines a basis change, BTRAN and PRICE are performed in order to obtain a new set of reduced costs.

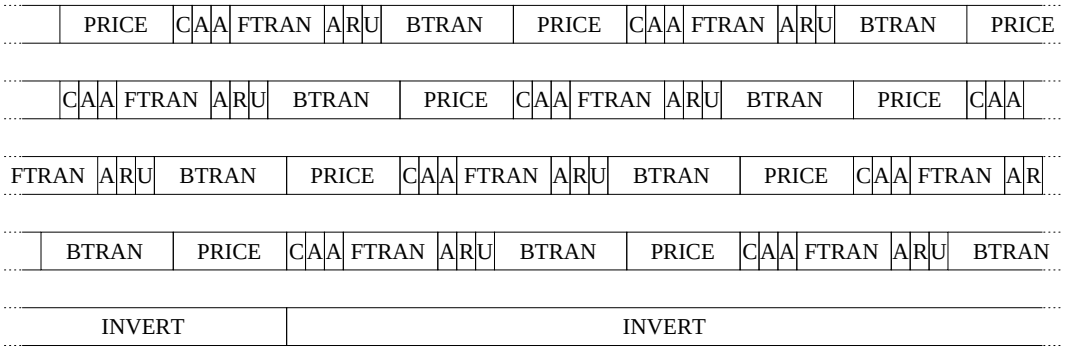
2.1 Overlapping simplex iterations

In the ASYNPLEX algorithm, one invert processor performs INVERT and p iteration processors perform the computational steps required to determine iterations of the revised simplex method. When an iteration processor receives a tentative candidate to enter the basis it performs FTRAN for the current basis. The pivotal column (and the values of the basic variables) are brought up-to-date with respect to basis changes which have been determined on other processors by a number of operations, one for each basis change, which are referred to in this paper as APPLY. Once the entering variable, pivotal column, step length and leaving variable have been received from the processor which determined the basis change, the APPLY operation performs the following: update the pivotal column (if APPLY follows FTRAN), update the values of the basic variables, update the indexing of basic and non-basic variables then update the factored inverse (UPDATE).

Following CHUZR, the basis change and the pivotal column are broadcast to the other iteration processors so that they can bring themselves up-to-date. The basis change is then sent to the invert processor where it will be incorporated into the next basis matrix to be inverted. The processor which

has just completed CHUZR now calculates reduced costs with respect to the new basis by performing BTRAN and PRICE.

In the ideal case where all simplex operations of the same type take the same time and all tentative candidates remain attractive once the pivotal column is up-to-date, the iterations overlap in a regular manner, as illustrated in the following Gantt chart for the case when $p = 4$.



C: CHUZC R: CHUZR U: UPDATE A: APPLY

Iteration processor i determines the basis change associated with iterations i, $p + i$, $2p + i$, ... As a result, the reduced cost of a chosen column is $p - 1$ basis changes out-of-date when that column enters the basis.

However, if the candidate is no longer attractive once the pivotal column is up-to-date then the column has to be rejected and the work done in performing the FTRAN is wasted. If the regular allocation of basis changes to processors is adhered to, then the processor would be idle for the time when it would otherwise be performing CHUZR, BTRAN and PRICE. If there are a significant number of unattractive candidates then this idleness will severely limit the extent to which the solution time can be reduced by exploiting parallelism. On finding an unattractive candidate, it would be preferable to start FTRAN immediately for a new candidate and allow the next processor (the opportunity) to determine the basis change at the current vertex.

2.2 Maintaining the factored inverse

Since each iteration processor must maintain a factored inverse of the current basis, there is duplication of the work required to update it, both after each basis change and when recovering an up-to-date factored inverse following reinversion. Thus these operations must be performed efficiently lest they become a serious limitation on the speed-up which may be obtained.

The issue of recovering an up-to-date factored inverse following reinversion is discussed by Hall and McKinnon in [6] where it is concluded

that the use of Bartels-Golub or Forrest-Tomlin updates is inappropriate. However, when product form or Schur complement update procedures are used, the inverse can be brought up-to-date in an efficient manner.

The product form update following each basis change can be implemented particularly efficiently when exploiting parallelism as outlined above. If the pivotal column is communicated so that it is stored at the end of the eta file, then all that is required to update the factored inverse is to calculate the reciprocal of the pivot, store the index of the row in which it occurs and increase the eta count by one.

In order to discuss the recovery of an up-to-date factored inverse following reinversion, suppose that basis r is current when INVERT is started and that basis k is current on an iteration processor when it receives the new factored inverse. The factored inverse is brought up-to-date by discarding the product form etas corresponding to basis changes 1 to r inclusive and identifying the start of the product form update etas with the eta corresponding to basis change $r + 1$. Note that this requires the order of the list of basic variables to be unchanged by the INVERT procedure. Any interchanges are accumulated into a permutation which is applied to the indices of the factors upon completion of INVERT, the right-hand-side vector before FTRAN and the solution following BTRAN. However this is a relatively small overhead.

The efficiency of using the product form update comes at a cost in terms of numerical stability. Each matrix S_j in the representation (1) of B_k^{-1} corresponds to a Gauss-Jordan elimination step using as the pivot the entry $\hat{a}_{pq}$ of the pivotal column for the corresponding iteration. If this pivot is small relative to the other entries in the pivotal column then growth may occur so that the computed representation of B_k^{-1} corresponds to a large perturbation of the inverse. In other words, the inverse is represented in an unstable manner so that the errors in subsequent pivotal columns may be larger than would be expected if the inverse were represented in a stable manner. This has three major consequences. Firstly, in CHUZR it is more likely that a perturbed zero may be used as a pivot, leading to further growth and the likelihood of a singular basis being detected during reinversion. Secondly, since the product form eta vector is obtained directly from the pivotal column, the accuracy of the updated representation of B_k^{-1} may be further undermined. Finally, when the pivotal column is used to update the current values of the basic variables, significant residual errors may be introduced leading to a possible loss of feasibility.

During reinversion, the ability to select pivots on numerical grounds means that a stable representation of the basis being factorised can be obtained. For a serial implementation, if the values of the basic variables and reduced costs are also recalculated then subsequent simplex iterations

may be viewed as having a numerical 'fresh start'.

However, in the parallel case, the fact that some of the previous etas are retained has more serious implications for the accuracy of the representation of B_k^{-1}. As observed above, if growth occurs then subsequent etas may be relatively inaccurate. Although the basis change which causes the growth will ultimately be incorporated into a basis which is reinverted—immediately if it is one of basis changes $1, \ldots, r$—some inaccurate etas may be retained. Thus the numerical cleansing resulting from reinversion is not so readily achieved in the parallel case. The 'thick pencil' row selection techniques described by Harris in [7] and by Gill *et al* in [4], may reduce the incidence of small pivots and so are particularly valuable in the context of parallel algorithms in which INVERT and simplex iterations are performed concurrently.

2.3 Limit on speedup by overlapping iterations

There are a number of factors which limit the speedup which can be achieved by overlapping iterations. These are duplicated work, wasted work, communication time and an increase in the number of iterations with increasing number of processors.

Let the average time required to perform one of each of the simplex operations BTRAN, PRICE, CHUZC, FTRAN, CHUZR and UPDATE be denoted by S, the time for FTRAN by F, the time for CHUZR by R, and the time for APPLY by A. The number of simplex iterations required to solve a problem depends on the number of processors p and will be denoted by N_p. Since the reduced cost of a column is not up-to-date when the variable enters the basis, this column may well not be the best candidate to enter the basis in the particular iteration, so the use of out-of-date costs is expected to increase the number of iterations required to solve the LP problem. It is also possible that the column no longer has a negative reduced cost, in which case it must be discarded, so that FTRAN is wasted. The number of wasted FTRANs depends on p and will be denoted by n_p. The more out of date the costs are the more wasted FTRANs will occur. The variation of N_p and n_p with p is highly problem-dependent.

Neglecting communication time (which is reasonable for the intended platform, the Cray T3D), and assuming all operations can be overlapped, the time taken to perform these simplex iterations using p iteration processors is

$$\hat{T}(p) = \frac{N_p}{p}(S + (p-1)A) + \frac{n_p}{p}F$$

So long as N_p and n_p increase less than linearly with p, this tends to the limit A as $p \to \infty$. A is the time of an APPLY operation, and this is the only

significant duplicated work in the algorithm. Note that the APPLY operation is a very small proportion of the work of an iteration.

Since CHUZR cannot be started on one processor until the previous CHUZR has been performed on another processor and the resulting basis change has been communicated, received and applied, the time for these operations limits the maximum rate at which simplex iterations can be performed. The makespan of the algorithm is therefore given by

$$T(p) = \max(\hat{T}(p), N_p(A + R)),$$

and, as the number of processors is increased, $N_p(A+R)$ becomes the limiting time to solve solve the LP problem.

2.4 ASYNPLEX, an asynchronous parallel algorithm

The algorithm ASYNPLEX presented below in Figures 2-5 allows simplex iterations to be overlapped using p iteration processors. The algorithm ensures that candidates are chosen from the most up-to-date reduced costs available and maintains processor activity in the event of candidates being found to be unattractive.

Clearly no two processors should simultaneously be forming the standard simplex tableau column for the same candidate variable. Initially it is possible to distribute the p most attractive candidates over all the iteration processors. However, as soon as one basis change has been determined, it is no longer straightforward to decide which candidate to choose according to the new set of reduced costs. It is quite likely that the most attractive candidate is currently being processed by one of the other processors. What is required is some mechanism to filter out the attractive candidates which have been chosen by other processors but which have not yet entered the basis. In the algorithm presented below, a set of the most attractive columns not currently chosen as candidates to enter the basis is determined after each basis change. Not only does this provide the next chosen column for the processor on which the corresponding reduced costs were calculated but the most up-to-date set provides a pool of immediately-available attractive candidates in the event of a candidate chosen previously proving to be unattractive.

Once a basis change has been determined, the indices of the variables entering and leaving the basis, together with the corresponding pivotal column, are communicated to all other iteration processors. Since the time required to perform FTRAN for different candidates can vary significantly, it is quite possible for two processors to have received and applied all basis

```
Repeat
   Repeat
      Update list of basic variables
   until {All basis changes received have been applied}
   INVERT
   Send new factored inverse
until {Simplex algorithm terminates}
```

Figure 2: ASYNPLEX: Algorithm for the invert processor

changes which have been determined. A management scheme is necessary to determine which of these processors performs CHUZR and which waits to receive the basis change from the other. This is the only situation in which a processor is idle for a significant time.

Although it is logically possible to manage column selection and basis changes by passing short fast messages between the iteration processors, it is convenient to describe (and implement) an algorithm which uses two additional processors in order to perform these management tasks. These processors are referred to as the row selection manager and column selection manager and they operate according to the algorithms given in Figures 4 and 5 respectively.

With the exception of the invert processor, each processor must keep track of the index of the basis which is current on that processor. This is denoted by k_i in Figure 3 for iteration processor i, by k_c for the column selection manager in Figure 4, and by k_r for the row selection manager in Figure 5.

2.5 Communication

In the ASYNPLEX algorithm there are four main communication requirements. In only two of these is the volume of communication related to the dimension of the problem. However these messages can be overlapped with computation. The other communications involve only a few tens of bytes but are very frequent and not all can be overlapped with computation. As a result, the algorithm must be implemented using message-passing routines with very low latency.

Each factored basis must be broadcast from the invert processor to each of the iteration processors. This is by far the largest single communication but, as with INVERT itself, it can be overlapped with simplex iterations. The second most significant communication is the broadcast of the pivotal column from the iteration processor which determines the corresponding basis change

```
   k_i = 0
   BTRAN
   PRICE
   Let q be the i^th most attractive candidate
   Repeat
      If {Received a new factored inverse} Install new factored inverse
      Repeat
         APPLY; k_i := k_i + 1
      until {All basis changes received have been applied}
      FTRAN
1     Continue
      Repeat
         APPLY; k_i := k_i + 1
      until {All basis changes received have been applied}
      If {ĉ_q > 0}  then
         Send a message to the column selection manager that the candidate
         is unattractive
      else
         Send an offer to perform CHUZR
         Wait for a reply to offer
         If {Offer accepted} then
            CHUZR
            Send basis change and pivotal column to other iteration
            processors
            Send basis change to the invert processor and column selection
            manager
            UPDATE; k_i := k_i + 1
            BTRAN
            PRICE
            Choose a set of the most attractive candidates
            Send the most attractive candidates
         else
            Wait to receive next basis change
            goto 1
         endif
      endif
      Wait for a new candidate column
   until {Simplex algorithm terminates}
```

Figure 3: ASYNPLEX: Algorithm for iteration processor i

```
k_c = 0
Mark all nonbasic variables as un-selected
Repeat
    If {Received basis change} then
        mark the variables which has left the basis as un-selected
    else if {Received from processor i a set of candidates corresponding
              to basis k_i} then
        If {k_i > k_c} then
            Filter out the candidates already selected
            k_c = k_i
        endif
        Send to processor i a candidate to enter the basis and mark the
        candidate as selected
    else if {Received from processor i a message that its current
              candidate is now unattractive} then
        Send to processor i a candidate to enter the basis and mark the
        candidate as selected
    endif
until {Simplex algorithm terminates}
```

Figure 4: ASYNPLEX: Algorithm for the column selection manager

```
k_r = 0
Repeat
    If {Received from processor i an offer to perform CHUZR corresponding
        to basis k_i} then
        If {k_i > k_r} then
            Send an acceptance of the offer
            k_r = k_i
        else
            Send a refusal of the offer
        endif
    endif
until {Simplex algorithm terminates}
```

Figure 5: ASYNPLEX: Algorithm for the row selection manager

to the other iteration processors. This is overlapped by computation, except when the processor is idle because all previous updates have already taken place.

Once an iteration processor has performed BTRAN and PRICE, it chooses a set of good candidates to send to the column selection manager. In practice the number of candidates chosen is seldom more than ten so the volume of this communication is not significant.

Before the processor can start FTRAN for a new candidate, it must wait until its set of good candidates has been received by the column selection manager and a new candidate has been identified and communicated back. Once again this is a communication of insignificant volume but the iteration processor will be idle for at least twice the latency period for a single communication. A similar idle period occurs after a pivotal column has been brought up-to-date. This is due to the logical send-and-receive with the row selection manager which is required to determine whether the iteration processor can proceed with CHUZR.

2.6 Implementation

An implementation of ASYNPLEX has been written for the Cray T3D based at Edinburgh University. This machine has a very high ratio of communication to computation speed. The SHMEM suite of inter-processor communication routines allow a bandwidth of 120MB/s with a latency of fewer than ten microseconds. The processing elements are Dec Alpha chips with 30Mflop peak performance. However, this computational performance will not be approached due to the indirect addressing of arrays which is associated with large sparse problems.

The implementation of the ASYNPLEX algorithm was developed using modules of our serial code ERGOL. The INVERT module of ERGOL compromises the reduction of fill-in and numerical stability for speed. This balance of properties is important in the context of the ASYNPLEX implementation. A fast INVERT increases the frequency with which new factored bases become available and so reduces the number of update etas which must be applied, thus increasing the speed of FTRAN and BTRAN. In practice, the fill-in generated by this INVERT is not significantly greater than is obtained by procedures based on the Markowitz criterion which preserve sparsity particularly well.

2.7 Test problems

The numerical results in Section 2.8 are given for four problems from the Netlib [2] test set. The problem names and the number of rows, columns and nonzeros in the constraint matrix are given in the following table.

Problem	Rows	Columns	Nonzeros
SHELL	536	1775	3556
SCTAP3	1480	2480	8874
25FV47	821	1571	10400
GREENBEB	2382	5405	30877

Although these problems are small by modern practical standards, they are representative of the problems in the Netlib test set and have neither a common structure nor extreme relative dimensions.

SHELL is a particularly sparse problem for which the basis matrix can always be reordered into triangular form and, even at the optimal basis, pivotal columns are 2% full on average—corresponding to just ten non-zero entries. It is also a problem for which candidate persistence is known to be good. The number of iterations required to solve the problem is not affected significantly by the column selection strategy, the Dantzig, Devex and steepest edge finding the optimal solution in 774, 708 and 715 iterations respectively (using ERGOL). As a result, column selection using out-of-date reduced costs is not expected to have a significant effect on the number of iterations required to solve the problem. SCTAP3 is a larger problem which exhibits similar low fill-in but is a little more sensitive to the choice of column selection criterion.

Although of dimensions which are comparable to those of SHELL and with only three times as many nonzeros, 25FV47 is a problem for which fill-in is significant. The factored inverse of the optimal basis has 66% more nonzeros than the matrix itself and pivotal columns at the optimal basis are 58% full. Using the Dantzig, Devex and steepest edge criteria, the optimal solution is found obtained in 5003, 3279 and 1792 iterations respectively. It is therefore expected that for 25FV47 the number of iterations required to solve the problem will be adversely affected by using out-of-date reduced costs.

GREENBEB was chosen to represent the larger problems in the Netlib set. The factored inverse of the optimal basis has 23% more nonzeros than the matrix itself and pivotal columns at the optimal basis are 24% full.

2.8 Numerical results

A stated aim in the Introduction was that a parallel algorithm should be significantly faster than good serial simplex solvers. Although no comparison with a commercial solver is possible on the T3D, the following table gives a comparison of ERGOL and OSL Version 1.2 [8] for the four test problems. The solution times given are the CPU time required on a SUN SPARCstation 5, starting from a logical basis and using the Dantzig strategy in CHUZC.

Problem	Solution time (s)		Simplex iterations		Iteration frequency (s^{-1})	
	ERGOL	OSL	ERGOL	OSL	ERGOL	OSL
SHELL	3.25	3.79	774	751	238	198
SCTAP3	12.3	21.7	1630	1677	133	77
25FV47	95.8	62.7	4922	4164	51	66
GREENBEB	358.	424.	11636	11869	33	28

The results for the same four problems using ASYNPLEX on the T3D are presented in Tables 1-4. In each case the problem was scaled and then solved from a logical basis. The first column in each table gives the number of iteration processors, with zero corresponding to ERGOL running on one processor. The number of simplex iterations required to solve the problem is given in the second column. Column three gives the total number of candidates which prove to be unattractive when their pivotal column is brought up-to-date. The final two pairs of columns give the iteration frequency and solution time, together with the respective speed-up compared with the serial implementation. The total solution time is the maximum elapsed time on any processor. The number of iteration processors used was increased in unit steps up to six and then in steps of two until either no further speed-up in solution time could be achieved or a maximum total number of sixteen processors (including the two manager processors) was reached, this being the limit on the number of processors which could be used interactively.

The difference in the number of simplex iterations when ERGOL is used (zero iteration processors) and the case where just one processor performs simplex iterations is explained by the fact that different basis matrices are inverted, leading to different rounding and hence differences in both column selection (when reduced costs would be equal using exact arithmetic) and row selection when the vertex is degenerate.

For SHELL and SCTAP3 there is no gain in efficiency by performing INVERT in parallel with just one simplex iteration processor. However there is some speed-up for 25FV47 and GREENBEB. This is partly because INVERTs are relatively more expensive for these problems but mainly due to

Iteration processors	Simplex iterations	Unattractive candidates	Iteration frequency (s^{-1})	Speed-up	Solution time (s)	Speed-up
0	774	-	260	-	3.0	-
1	759	0	210	0.8	3.6	0.8
2	772	198	370	1.4	2.1	1.4
3	766	312	510	2.0	1.5	2.0
4	742	341	620	2.4	1.2	2.5
5	762	469	820	3.2	0.93	3.2
6	803	421	940	3.7	0.85	3.5
8	814	779	1000	4.0	0.79	3.8
10	761	601	1200	4.5	0.66	4.5
12	783	977	1300	4.9	0.62	4.8

Table 1: Results for SHELL using ASYNPLEX

Iteration processors	Simplex iterations	Unattractive candidates	Iteration frequency (s^{-1})	Speed-up	Solution time (s)	Speed-up
0	1681	-	110	-	15.8	-
1	1589	0	110	1.0	15.0	1.1
2	1949	520	190	1.8	10.4	1.5
3	2021	1005	260	2.5	7.7	2.1
4	2119	1387	320	3.0	6.6	2.4
5	2299	1410	360	3.4	6.4	2.5
6	2390	1876	400	3.7	6.0	2.6
8	2333	2418	470	4.4	5.0	3.2
10	2453	2843	500	4.7	4.9	3.2
12	2363	2771	530	4.9	4.5	3.5

Table 2: Results for SCTAP3 using ASYNPLEX

the significant reduction in the average time for FTRAN and BTRAN when new factored inverses are available more frequently.

In each case the iteration frequency increases steadily with the number of iteration processors and a speed-up of between 4.0 and 4.5 is achieved when eight processors are used. This indicates that the time per iteration on each iteration processor has increased by a factor of up to two. This is partly explained by the increasing time spent bringing unattractive pivotal columns up-to date and applying basis changes determined on other processors, and partly by an increase in the time required to perform BTRAN and FTRAN as the number of simplex iterations between reinversion increases. Finally, note that the speed-up in the solution time will be less than the speed-up

Iteration processors	Simplex iterations	Unattractive candidates	Iteration frequency (s^{-1})	Speed-up	Solution time (s)	Speed-up
0	5003	-	48	-	107	-
1	5263	0	71	1.5	74	1.4
2	6561	3247	110	2.3	62	1.7
3	7111	5542	140	3.0	51	2.1
4	8171	8825	160	3.4	52	2.1
5	8245	10710	180	3.8	46	2.3
6	7678	12082	190	4.1	40	2.7
8	9279	16760	210	4.5	44	2.4

Table 3: Results for 25FV47 using ASYNPLEX

Iteration processors	Simplex iterations	Unattractive candidates	Iteration frequency (s^{-1})	Speed-up	Solution time (s)	Speed-up
0	11821	-	31	-	384	-
1	11914	0	37	1.2	319	1.2
2	14679	5852	64	2.1	231	1.7
3	15883	10560	82	2.7	193	2.0
4	16848	14966	98	3.2	171	2.2
5	15444	16898	110	3.7	137	2.8
6	17692	21057	120	4.0	144	2.7
8	20556	28532	130	4.3	155	2.5
10	18077	29873	150	4.9	119	3.2

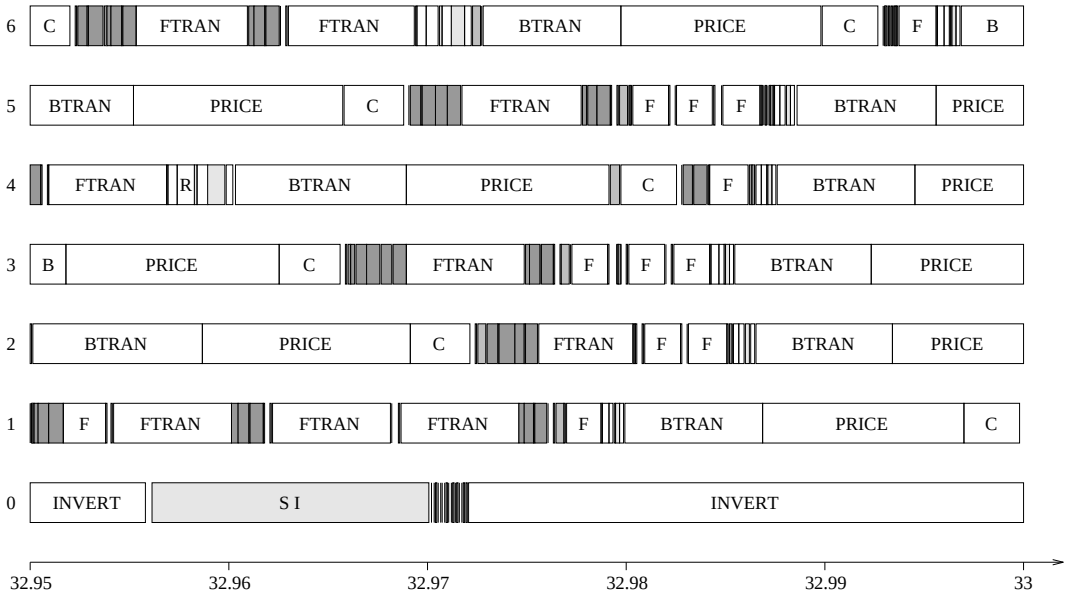
Table 4: Results for GREENBEB using ASYNPLEX

in iteration frequency if the use of out-of-date reduced costs increases the number of iterations taken.

In the results for SHELL given in Table 1, it is seen that the number of iterations does not increase with the number of iteration processors. As a result, the speed-up in the solution time is similar to that for the iteration frequency. For SCTAP3 there is some increase in the number of simplex iterations so the efficiency of using several iteration processors is not so marked. The practical performance of the parallel solver when applied to 25FV47 is less good than with the first two problems. The number of iterations and unattractive candidates increases significantly, limiting the speed-up that can be achieved. Although the candidate persistence for GREENBEB is poor, FTRAN is relatively inexpensive so significant speed-up is still achieved.

An illustration of typical processor activity when solving LP problems is

given in the following Gantt chart, the data for which come approximately three-quarters of the way through the solution of GREENBEB.



C: CHUZC F: FTRAN R: CHUZR SI: Send new factored inverse

Processor activity is seen to be almost continuous, although some of this is spent calculating pivotal columns for candidates which prove to be unattractive and may be identified by FTRANs without a subsequent BTRAN. Note that CHUZR is normally so fast that only one is actually identified in the chart. Lightly-shaded boxes correspond to time associated with communication and, with the exception of the time spent broadcasting the new factored inverse, is very low and only visible in one band for processors 4 and 6. Note that the average time required by INVERT for this problem is $0.13s$, approximately ten times the time taken to broadcast the factored inverse in this illustration. However, the work of updating with respect to simplex iterations determined on other processors is duplicated and appears in heavily-shaded boxes. Once the new factored inverse is received, it is clear that the cost of performing FTRAN is immediately reduced.

3 Conclusions

A parallel algorithm for the revised simplex method has been described and practical results demonstrating speed-up of between 2.5 and 4.8 have been given. These results could be improved by further performance optimization. However, the limitations of the algorithm should also be addressed. The

Dantzig column selection criterion is not often best and the reduced costs are calculated from scratch each time. If the reduced costs are updated then the PRICE operation can be considerably cheaper for sparse problems such as SHELL. This is because the vector for PRICE is $\boldsymbol{e}_p^T B_k^{-1}$ which is generally rather more sparse than $\boldsymbol{c}_B^T B_k^{-1}$.

An algorithmic development which leads from this observation is to maintain reduced costs by updating them on one or more processors. Steepest edge weights for column selection could be updated on dedicated processors in a similar manner, allowing the expected total number of simplex iterations to be reduced.

Although most of the communication overhead has been minimized by the use of SHMEM routines, the cost of broadcasting the new factored inverse is still significant. This factored inverse may only be applied a few times on each processor and it may prove more efficient to communicate the inverse to just one or two processors which would then communicate partially FTRANned columns and complete BTRAN for columns whose BTRAN operation has been started elsewhere.

These algorithmic developments are currently being considered and are expected to form the basis for future work.

References

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

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

[3] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Sparse matrix methods in optimization. *SIAM J. Sci. Stat. Comput.* , 5:562–589, 1984.

[4] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A practical anti-cycling procedure for linear and nonlinear programming. Technical Report SOL 88-4, Systems Optimization Laboratory, Stanford University, 1990.

[5] D. Goldfarb and J. K. Reid. A practical steepest-edge simplex algorithm. *Math. Prog.*, 12:361–371, 1977.

[6] J. A. J. Hall and K. I. M. McKinnon. Update procedures for the parallel revised simplex method. Technical Report MSR 92-13, Department of Mathematics and Statistics, University of Edinburgh, 1992.

[7] P. M. J. Harris. Pivot selection methods of the Devex LP code. *Math. Prog.*, 5:1–28, 1973.

[8] IBM. *Optimization Subroutine Library, guide and reference, release 2*, 1993.

[9] J. Luo, A. N. M. Hulsbosch, and G. L. Reijns. An MIMD work-station for large LP problems. In E. Chiricozzi and A. D'Amico, editors, *Parallel Processing and Applications*, pages 159–169. Elsevier Science Publishers B.V. (North-Holland), 1988.

[10] 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.

[11] C. B. Stunkel. Linear optimization via message-based parallel processing. In *International Conference on Parallel Processing*, volume III, pages 264–271, August 1988.