

A Parallel Interior Point Method and its Application to Facility Location Problems

Amal de Silva / School of Computing and Information Technology, Griffith University, Nathan,
QLD 4111, AUSTRALIA; Email amal@cit.gu.edu.au

David Abramson / School of Computing and Information Technology, Griffith University, Nathan,
QLD 4111, AUSTRALIA; Email davida@cit.gu.edu.au

Research Report CIT:95-12

We present a parallel interior point algorithm to solve block structured linear programs. This algorithm can solve block diagonal linear programs with both side constraints (common rows) and side variables (common columns). The performance of the algorithm is investigated on uncapacitated, capacitated and stochastic facility location problems. The facility location problems are formulated as mixed integer linear programs. Each subproblem of the branch and bound phase of the MIP is solved using the parallel interior point method. We compare the total time taken by the parallel interior point method with the simplex method to solve the complete problems, as well as the various costs of re-optimisation of the non-root nodes of the branch and bound. Computational results on two parallel computers (Fujitsu AP1000 and IBM SP2) are also presented in this paper.

1 Introduction

The recent advances in parallel computers have encouraged a number of researchers to develop an efficient parallel algorithm to solve linear programs. These efforts have been reasonably successful when dense matrix representations have been used [8]. However, most practical linear programs are sparse and the development of parallel algorithms for sparse linear programs is an active area of research at present. Of the two algorithms available to solve linear programs, the interior point algorithm is favoured by most researchers for parallelisation. The reason for this is two-fold. First is that the interior point method works with matrices which have a higher number of nonzeros compared with the traditional simplex method. The second is that the proportion of time spent on communications in comparison to computations, per iteration is significantly higher for the simplex method, making it less attractive for parallelisation. The main computational step of most interior point implementations is the factorisation of a sparse symmetric positive definite matrix via Cholesky factorisation and to-date researchers has been concentrating on developing a parallel Cholesky factorisation algorithm.

Karypis et al. [15] describe a parallel interior point algorithm, which works by first ordering the matrix [12] such that the number of nonzeros created while factorising (fill-in) is minimised. Then the independent sections of the matrix for this ordering are determined with the use of elimination trees [17]. Elimination trees help to determine the columns of the matrix that can be eliminated independently in the Cholesky factorisation; thus, the shape of the elimination tree determines the degree of concurrency available and this can vary with the problem structure. Levkovitz et. al. [16] also takes a similar approach to parallelise the interior point method. In order to test these algorithms, LPs from sources such as NETLIB [11] are used without knowing anything about the underlying problems. The question then arises whether it would be possible to obtain more concurrency if independent sections of the linear program are determined from original LP model.

We take a different approach to parallelise the interior point algorithm. Instead of attempting to determine independent section of the matrix to be factorised in the solution phase, the model builder is asked to develop a block structured linear program. Thus the blocks which can be factorised independently are determined before the solution phase and a reasonably good estimate of the potential speedup can be made from the dimensions of the problem. This would help to determine the ideal amount of resources to use on the problem. Partitioning of the constraint matrix can be efficiently carried out with the use of Mathematical Programming Languages such as AMPL [10] and GAMS [5]. Thus, in our approach, partitioning of the linear program is to be done at a higher level than performed in [15].

If the constraint matrix of a linear program consists of only diagonal blocks (or blocks which are totally independent of each other), each such block can be considered to be an independent linear program that can be solved separately with perfect parallelism. Although few real world problems exhibit such ideal characteristics, many of them would exhibit a block structure with some linking between blocks either through columns or rows or both. In such situations the blocks can be still solved in parallel, with the computations of the linking rows and columns carried out sequentially. If the size of the blocks are large compared to the linking rows and columns, then good speedups can still be achieved. Some linear programs exhibit a block structure in which certain rows and columns inherently fall into blocks, such as commodities in multi-commodity network flow problems and scenarios in stochastic linear programs. Thus, the identification of the blocks should be carried out by the model builder.

Lustig and Li [19] implemented a parallel algorithm to exploit the block structure of linear programs with side constraints such as multi-commodity network flow problems. The blocks in these problems are linked through the side constraints (linking rows). Choi and Goldfarb [6] also developed a method similar to Lustig and Li to solve multicommodity networks in parallel. Stochastic linear programs also have diagonal blocks, but are linked together by columns of the first stage variables (linking columns). This property also has been exploited for parallelisation [23, 2, 14, 13]. Jessup, Yang and Zenios [14] and Yang and Zenios [13] uses the BQ algorithm [3] to solve the symmetric positive definite system of equations in parallel. Excellent performance is reported by them. In this paper we present a more general parallel algorithm for linear programs that have both linking rows and linking columns. Lustig and Li's [19] method is algebraically similar to our method when only linking rows are present. The parallel interior point method is used by us to solve facility location problems. Uncapacitated, capacitated and stochastic facility location problems are tested on a 128 node Fujitsu AP1000 as well as a 22 node IBM SP2. We also show how the constraint matrix of facility location problems can be partitioned such that it can be solved by the parallel interior point algorithm. This algorithm has also been used to solve large stochastic linear programs by the author's with excellent results [23].

Czyzyk et.al. [7] develops a parallel method to solve linear programs on the CM-2. The approach used here is different to any of the previously discussed methods of parallelisation. The previous methods were directed at parallelising the sparse matrix algebra. In this study the sparse factorisation is done sequentially by one processor (possibly a front machine) and then the dense part of the factorisation is carried out in parallel.

The facility locations problems are formulated as 1-0 mixed integer programs, and therefore a branch and bound is carried out to determine the optimal integer solution. The linear programming subproblems at each node of the branch and bound are solved by the parallel interior point algorithm. The current folklore is that the warm starting capability of the simplex method is superior to the sequential interior point method. In this paper, we also compare the efficiency of warm starting (re-optimising) linear programming subproblems by the parallel interior point algorithm and the simplex method.

To test our parallel interior point implementation, we use the facility location test set from the OR-library [1].

Section (2) develops the parallel interior point algorithm, while section (3) describes how we implement the branch and bound algorithm with the parallel interior point method. Sections (4), (5) and (6) gives the formulation and the computational results of Uncapacitated, Capacitated and Stochastic facility location problems respectively.

2 Parallel Interior Point Algorithm

In this section the parallel interior point algorithm is described.

The Linear program (LP) to be solved is,

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && Ax = b \\ & && x \geq 0. \end{aligned} \tag{1}$$

The dual of this LP is given by,

$$\begin{aligned} & \text{maximize} && b^T y \\ & \text{subject to} && A^T y + z = c \\ & && z \geq 0. \end{aligned} \tag{2}$$

The major computational step of solving a linear program by the primal dual path following interior point algorithm [18, 20, 21] and [25] is to solve the following system of equations.

$$\begin{bmatrix} -\theta^{-2} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \sigma - X^{-1}\Phi \\ \rho \end{bmatrix}. \tag{3}$$

where $\theta^{-2} = ZX^{-1}$, $\rho = b - Ax$; $\sigma = c - A^T y - z$; $\Phi = \mu e - XZe$; X and Z are diagonal matrices with elements x_j and z_j . e denotes a n vector of all ones.

Let us assume that the constraint matrix A takes the following form.

$$A = \begin{bmatrix} V & D & 0 \\ M & H & S \end{bmatrix}. \tag{4}$$

Let us also assume that D, V, H can be partitioned further as,

$$D = \begin{bmatrix} D_1 & 0 & 0 & 0 \\ 0 & D_2 & 0 & 0 \\ \vdots & \dots & \dots & \vdots \\ 0 & 0 & 0 & D_K \end{bmatrix}, \tag{5}$$

$$V = \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_K \end{bmatrix}, \tag{6}$$

$$H = [H_1 \ H_2 \ \dots \ H_K]. \tag{7}$$

In this formulation it is assumed that the linear program consists of K diagonal blocks (D_i , $i = 1 \dots K$). The matrix V consists of the columns with nonzeros in rows of more than one diagonal block (we will call them linking columns). The matrix H consists of the rows with nonzeros in columns of more than one diagonal block (referred by us as linking rows). The matrix M consists of the nonzeros of the linking columns in the linking rows. S is from the columns which have only nonzeros in the linking rows (such as slack variables of the linking rows).

In this paper we show how facility location problems can be formulated to take this format. The objective of formulating a linear program in this fashion is for parallelisation. It is intended to distribute the blocks to the processors such that the computations of these blocks can be done in parallel.

Equation (3) can be expressed in the following manner when the matrix A is partitioned as in (4).

$$\begin{bmatrix} -\theta_V^{-2} & 0 & 0 & V^T & M^T \\ 0 & -\theta_D^{-2} & 0 & D^T & H^T \\ 0 & 0 & -\theta_S^{-2} & 0 & S^T \\ V & D & 0 & 0 & 0 \\ M & H & S & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_D \\ \Delta x_S \\ \Delta x_V \\ \Delta y_D \\ \Delta y_H \end{bmatrix} = \begin{bmatrix} \sigma_D - X_D^{-1}\Phi_D \\ \sigma_S - X_S^{-1}\Phi_S \\ \sigma_V - X_V^{-1}\Phi_V \\ \rho_D \\ \rho_H \end{bmatrix}. \quad (8)$$

Vectors with subscripts D, V, H and S are the partitions of the corresponding vectors of equation (3), which are associated with sub-matrices D, V, H, S respectively. For example the vector x is partitioned into x_D, x_V and x_S with the elements corresponding to the columns of D, V and S . Similarly ρ is partitioned into ρ_D and ρ_H with the elements corresponding with the rows of D and H .

Using the second and third sets of equations to eliminate Δx_D and Δx_S from (2), yields the following system of equations;

$$\begin{bmatrix} -\theta_V^{-2} & V^T & M^T \\ V & D\theta_D^2 D^T & D\theta_D^2 H^T \\ M & H\theta_D^2 D^T & S\theta_S^2 S^T + H\theta_D^2 H^T \end{bmatrix} \begin{bmatrix} \Delta x_V \\ \Delta y_D \\ \Delta y_H \end{bmatrix} = \begin{bmatrix} \sigma_V - X_V^{-1}\Phi_V \\ \rho_D + D\theta_D^2(\sigma_D - X_D^{-1}\Phi_D) \\ \rho_H + H\theta_D^2(\sigma_D - X_D^{-1}\Phi_D) + S\theta_S^2(\sigma_S - X_S^{-1}\Phi_S) \end{bmatrix}; \quad (9)$$

Permuting rows and columns,

$$\begin{bmatrix} D\theta_D^2 D^T & D\theta_D^2 H^T & V \\ H\theta_D^2 D^T & S\theta_S^2 S^T + H\theta_D^2 H^T & M \\ V^T & M^T & -\theta_V^{-2} \end{bmatrix} \begin{bmatrix} \Delta y_D \\ \Delta y_H \\ \Delta x_V \end{bmatrix} = \begin{bmatrix} \rho_D + D\theta_D^2(\sigma_D - X_D^{-1}\Phi_D) \\ \rho_H + H\theta_D^2(\sigma_D - X_D^{-1}\Phi_D) + S\theta_S^2(\sigma_S - X_S^{-1}\Phi_S) \\ \sigma_V - X_V^{-1}\Phi_V \end{bmatrix}; \quad (10)$$

The $D\theta_D^2 D^T$ matrix will be made of diagonal blocks as D is a block diagonal matrix. The objective is to carry out all the computation of these blocks in parallel.

Let the system of equations to be solved be,

$$\begin{bmatrix} P & N \\ N^T & \hat{C} \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} \xi_1 \\ \xi_2 \end{bmatrix}, \quad (11)$$

where

$$P = D\theta_D^2 D^T. \quad (12)$$

$$\hat{C} = \begin{bmatrix} S\theta_S^2 S^T + H\theta_D^2 H^T & M \\ M^T & -\theta_V^{-2} \end{bmatrix}, \quad (13)$$

and

$$N = \begin{bmatrix} D\theta_D^2 H^T & V \end{bmatrix}. \quad (14)$$

System (11) is symmetric, but indefinite. Thus, there is no guarantee that the Cholesky factorisation exists (i.e. factorisation in the form $L\Lambda L^T$ such that L is lower triangular and with unit diagonal and Λ is diagonal with strictly positive entries). Vanderbei and Carpenter [25] define another class of matrices known as quasi-definite matrices. A symmetric matrix $\begin{bmatrix} F & R \\ R^T & -E \end{bmatrix}$ is quasi-definite if both E and F are positive definite. For such a system Vanderbei [24] proves that for every permutation, a factorisation $L\Lambda L^T$ will always exist, such that L is lower triangular with unit diagonal and Λ is nonsingular and diagonal.

To show the conditions at which the matrix given in the left hand side of (11) is quasi-definite, it is partitioned in the following manner.

$$F = \begin{bmatrix} D\theta_D^2 D^T & D\theta_D^2 H^T \\ H\theta_D^2 D^T & S\theta_S^2 S^T + H\theta_D^2 H^T \end{bmatrix}, \quad (15)$$

$$E = \begin{bmatrix} \theta_V^{-2} \end{bmatrix}, \quad (16)$$

and

$$R = \begin{bmatrix} V \\ M \end{bmatrix}. \quad (17)$$

Thus, for this matrix to be positive definite both E and F should be positive definite. It is apparent that E is positive definite. F would be positive definite provided that the matrix,

$$\begin{bmatrix} D & 0 \\ H & S \end{bmatrix}. \quad (18)$$

has full row rank. Let us assume that the rows of this matrix are linearly independent and thus F is positive definite. If this condition does not hold, there is no assurance that no zero pivot elements will be encountered. In our implementation we do not check for positive definiteness, but we perturb the matrix by adding a small positive value to the diagonal to force it to be positive definite. In the case of facility locations problems the matrix given by (18) will always have full row rank and in theory the factorisation can be carried out with out encountering any zero pivots. But in practice the pivot elements can take zero or very small values (especially as the optimum is approached) due to the finite precision of the floating point numbers used in computers. However, most linear programs can be solved without any numerical problems with the use of perturbation technique described previously. Other block structured linear programs (18) can be rank deficient and theoretically it would not be possible to factorise it with the algorithm given below. But in practice we have solved a number of such problems with out any numerical instability.

Assuming (18) is of full row rank, (11) will be quasi-definite. Thus, it can (theoretically) be factorised in any order without encountering a zero pivot element and we can employ the same strategy that we would if the matrix were positive definite. In particular we can apply George and Liu's implicit algorithm [12] for factoring partitioned system of equations. This algorithm proceeds as follows.

[**Factorisation**]

Step (1) Factor matrix P into $L_P L_P^T$

Step (2) Solve $L_P G = N$

Step (3) Solve $L_P^T \tilde{G} = G$

Step (4) Compute $B = N^T \tilde{G}$

Step (5) Compute $C = \hat{C} - B$

Step (6) Factor matrix C into $L_C \Lambda L_C^T$

The algorithm for triangular solution of the partitioned systems is given by,
[**Forward Solve**]

Step (1) Solve $L_P \gamma = \xi_1$

Step (2) Solve $L_P^T t = \gamma$

Step (3) Compute $\eta = N^T t$

Step (4) Compute $\tilde{\xi}_2 = \xi_2 - \eta$

Step (5) Solve $L_C \Lambda \beta = \tilde{\xi}_2$

[**Backward Solve**]

Step (6) Solve $L_C^T r_2 = \beta$

Step (7) Compute $\lambda = N r_2$

Step (8) Solve $L_P t = \lambda$

Step (9) Compute $\tilde{\gamma} = \gamma - t$

Step (10) Solve $L_P^T r_1 = \tilde{\gamma}$.

The matrices P and N of equation (11) can be expressed in following manner,

$$P = \begin{bmatrix} P_1 & 0 & 0 & 0 \\ 0 & P_2 & 0 & 0 \\ \vdots & \cdots & \cdots & \vdots \\ 0 & 0 & 0 & P_K \end{bmatrix}, \quad (19)$$

and

$$N = \begin{bmatrix} N_1 \\ N_2 \\ \vdots \\ N_K \end{bmatrix}, \quad (20)$$

where

$$P_i = D_i \theta_{D_i}^2 D_i^T \quad i = 1 \cdots K \quad (21)$$

and

$$N_i = \begin{bmatrix} D_i \theta_{D_i}^2 H_i^T & V_i \end{bmatrix}, \quad (22)$$

The key observation is that P consists of diagonal blocks and thus, all the computations of each block can be done independently in parallel. The parallelisation of the factorisation is explained next.

As stated previously, the A matrix consists of K diagonal (D_i), vertical (V_i) and horizontal (H_i) blocks. The algorithm starts by distributing these blocks to processors. The blocks D_i, V_i and H_i (for a given i) are always assigned to the same CPU. These blocks are assigned such that the number of blocks per CPU is made equal as possible to balance the load and to thereby improve the parallel efficiency of the algorithm. The blocks M and S are handed over to CPU number zero. In our implementation all the sequential computations are carried out by CPU number zero (any other CPU could have done these computations if necessary). Once the blocks of the linear program are distributed among processors, the computations of each block can be carried out by its corresponding CPU. A minimum row ordering and a symbolic factorisation [12] is carried out on each block D_i and a separate sparse matrix is built for each of these blocks. In each iteration of the interior point algorithm the CPU evaluates the $D\theta_D^2 D$ matrix of its assigned blocks.

The submatrix $\hat{C}$ (which is stored using dense matrix representations) given by equation (13) is assigned to CPU number zero. As the matrices M and S are resident in this CPU, it first computes the matrix,

$$\begin{bmatrix} S\theta_S^2 S^T & M \\ M^T & -\theta_V^{-2} \end{bmatrix}. \quad (23)$$

The $H_i\theta_{D_i}^2 H_i^T$ matrices are computed by its corresponding CPUs in parallel and added across the blocks. The result is then added on to $S\theta_S^2 S^T$ of (23) to obtain $\hat{C}$, as

$$H\theta_D^2 H^T = \sum_{i=1}^K H_i\theta_{D_i}^2 H_i^T. \quad (24)$$

Once this is done the matrix $\begin{bmatrix} P & N \\ N^T & \hat{C} \end{bmatrix}$ is factorised in parallel by the algorithm given below.

[Parallel Factorisation]

Step (1) Factor matrix P_i into $L_{P_i} L_{P_i}^T$
 $\forall i$ in parallel.

Step (2) Solve $L_{P_i} G_i = N_i$
 $\forall i$ in parallel.

Step (3) Solve $L_{P_i}^T \tilde{G}_i = G_i$
 $\forall i$ in parallel.

Step (4a) Compute $B_i = N_i^T \tilde{G}_i$
 $\forall i$ in parallel.

Step (4b) Compute $B = \sum_{i=1}^K B_i$

Step (5) Compute $C = \hat{C} - B$

Step (6) Factor matrix C into $L_C \Lambda L_C^T$

Steps (1), (2) and (3) are performed on the matrix P , which contains independent blocks. Therefore these three steps can be carried out in parallel. We base steps (1), (2) and (3) on the algorithms proposed by George and Liu [12] to solve sparse positive definite systems. Step (4) of

the factorisation algorithm is divided into two sub-steps step (4a) and (4b). Step (4a) can be done in parallel while step (4b) requires the addition of K , $(m_1 + n_1) \times (m_1 + n_1)$ matrices across the blocks. Where m_1 is the number of linking rows and n_1 is the number of linking columns. For the AP1000 step (4b) is carried out by first adding all the blocks resident in each CPU (each CPU does this in parallel) and thereafter the resultant blocks are added pairwise. The final result is resident in cell number zero. From this procedure a speedup in the order of $\log(\text{nprocs})$ (where nprocs is the number of CPUs) can be achieved. For the SP2 a function provided by the Message Passing Library (MPL) is used for the parallel addition.

Steps (5) and (6) are carried sequentially by CPU number zero. The parallel efficiency of the algorithm is dependent on the proportion of time spent in the parallel computation compared to the sequential computation. For problems where the number of linking columns and rows are small compared to the size of the diagonal blocks, high speedups can be expected. The forward and backward solve can also be similarly parallelised. As previously stated, this parallel algorithm is implemented on two parallel machines, namely the Fujitsu AP1000 and the IBM SP2, which are distributed memory MIMD parallel machines.

The sequential component of the factorisation as well as the forward and backward solve is carried out with dense matrix representations while, the parallel component utilises sparse matrix representations.

A detailed description on the parallel algorithm with respect to stochastic linear programs (only linking columns are present) is given by De Silva and Abramson in [23]. The reader is referred to this paper on details of parallelisation of the other computational steps of the interior point method.

Although we carried out step (6) sequentially (a dense matrix factorisation), it is possible to parallelise this step as well. On problems with a large number of linking columns or linking rows, parallelising step (6) would improve the performance of the algorithm significantly. This method also partitions the dense and sparse parts of the factorisation. By doing this, it is possible to use the most efficient parallel algorithm to factorise the two types of data structures (sparse and dense).

In our implementation, we use the predictor-corrector variant of the primal-dual algorithm, suggested by Mehrotra [22]. Although this requires two back-solves per iteration, the number of iterations is often substantially reduced. Our implementation of this method is based on the details provided by Lustig et.al [21].

3 The Branch and Bound Algorithm

As facility location problems are mixed integer programs a branch and bound phase is required to obtain 0 or 1 values for the binary variables. Consequently, this was incorporated into our code and the linear programming subproblems encountered in each node of the algorithm were solved with the parallel interior point method described in the previous section.

Borchers and Mitchell implement the branch and bound code for integer programming in [4]. This paper, gives a number of techniques to improve the performance of the branch and bound algorithm with the interior point method. Our implementation of the branch and bound is mostly based on their algorithm. One of these techniques is the warm starting of the interior point method. Warm starting is to re-optimize a slightly different linear program, starting from a previously obtained optimal solution. This feature is used extensively in zero-one mixed integer linear programs when the binary variables are fixed at zero or one and the subproblem is re-optimised. The current folklore is that the simplex method is superior in warm starting. The experiments conducted by us show that, for the initial root node, (relaxed linear program) the interior point algorithm outperforms the state of the art simplex code of IBM's Optimisation Subroutine Library (OSL). But in all but one instance, the simplex method out-performs the sequential interior point algorithm in solving the non-root nodes due to the superior warm starting capability. However, later in the paper we show that the ability to carry out the interior point algorithm in parallel makes it competitive with the simplex method in re-optimising linear programs. Borchers and

Mitchell use the optimal solution obtained in the parent node to warm start the interior point algorithm of the current node. To implement this, it is necessary to store the optimal values of x, y, z variables for each leaf node of the search tree. This would require an enormous amount of storage for the very large problems that we have tested. Thus, we start the interior point algorithm from the usual starting point (with out any warm starting) for the large problems. In some instances we use the optimal solution of the root node to warm start all the non-root nodes. In this case we have to store only one set of primal and dual variables. For the smaller problems the warm starting procedure given by Borchers and Mitchell is used.

Another method used by Borchers and Mitchell to improve the branch and bound algorithm, with the use of the interior point method, is the early fathoming of subproblems (fathoming is to discard a subproblem if the lower bound of the subproblem is greater than a known integer solution). This method is motivated by the observation that the interior point methods tend to find feasible solutions quickly, but takes a relatively long time to obtain the accurate optimal solution. Any feasible dual objective value will be a lower bound of the optimal objective of the current subproblem, and therefore this can be used to fathom the subproblem. This feature has been implemented in our test code.

They also provide a heuristic to detect 0-1 variables that are converging to fractional values, whereby the interior point algorithm can be terminated early. As this heuristic made the branch and bound search a large number nodes unnecessarily in few instances, it was not included in our implementation.

4 Uncapacitated Facility Location problems

4.1 Problem Formulation

The uncapacitated facility location problem is to locate facilities (e.g. warehouses) at a number of potential locations such that the demands of a set of customers are serviced and the cost is minimised. This problem can be formulated as a mixed integer linear program as shown below.

$$\begin{aligned}
& \text{minimize} && \sum_{j=1}^{n_f} F_j u_j + \sum_{i=1}^{n_c} \sum_{j=1}^{n_f} f_{ij} v_{ij} \\
& \text{subject to} && \sum_{j=1}^{n_f} v_{ij} = 1, && i = 1 \cdots n_c \\
& && -u_j + v_{ij} \leq 0, && j = 1 \cdots n_f \quad i = 1 \cdots n_c \\
& && v_{ij} \leq 1, && j = 1 \cdots n_f \quad i = 1 \cdots n_c \\
& && u_j \in \{0, 1\}, && j = 1 \cdots n_f \\
& && v_{ij} \geq 0, && j = 1 \cdots n_f \quad i = 1 \cdots n_c
\end{aligned} \tag{25}$$

n_c is the number of customers, while n_f is the number of potential facility locations. The variable v_{ij} is the proportion of customer i 's demand delivered from facility j . u_j is a binary variable which takes a value of 1 or 0 whether facility j is established or not. The fixed cost of establishing the j th facility is F_j , while the cost of delivering the total demand of customer i from facility j is given by f_{ij} .

We partition the constraint matrix of the linear program given by (25) to solve it via the parallel interior point method in the following manner. A diagonal block is created for each customer and the columns due to the variables u_j is included in the vertical blocks V . All the nonzeros associated with a given customer location are included in one diagonal block. Thus the number of blocks K is equal to the number of customers (n_c). No horizontal blocks (H) are required for uncapacitated facility location problems. The i th diagonal block of the linear program is given below,

$$D_i = \begin{bmatrix} 1 & 1 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 1 \end{bmatrix}. \tag{26}$$

$$\begin{array}{llllll}
\text{minimize} & F^T u & + f_1^T v_1 & f_2^T v_2 & \cdots & + f_K^T v_{n_f} \\
\text{subject to} & V_1 u & + D_1 v_1 & & & = h_1, \\
& V_2 u & & + D_2 v_2 & & = h_2, \\
& \vdots & \cdots & \vdots & \vdots & \cdots = \vdots, \\
& V_{n_f} u & & & + D_{n_f} v_{n_f} & = h_{n_f}, \\
& & v_1, & v_2, & \cdots & v_{n_f} \geq 0, \\
& u & & & & \in \{0, 1\}.
\end{array}$$

Figure 1: Linear program for Uncapacitated facility location problems

The total number of columns of D_i is $2 \times n_f$ and the first n_f columns are from the v_{ij} variables of the i th customer (block) ($i = 1 \cdots n_c$). The last n_f columns are the slack variables added to convert the inequality constraints $v_{ij} - u_j \leq 0$ to equality constraints. The first row is from the $\sum_{j=1}^{n_f} v_{ij} = 1$ constraint, while the last n_f rows are from the $v_{ij} - u_j \leq 0$ constraints. The $v_{ij} \leq 1$ constraint is included in the problem as a bound on the variable v_{ij} (See [20] for further details on this). The i th vertical block V_i is given by;

$$V_i = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}. \quad (27)$$

and the n_f columns of V_i are from the u_j ($j = 1 \cdots n_f$) variables. The resulting linear program is given in figure (1). Where,

$$F = \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_{n_f} \end{bmatrix}; f_i = \begin{bmatrix} f_{i1} \\ f_{i2} \\ \vdots \\ f_{in_f} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}; v_i = \begin{bmatrix} v_{i1} \\ v_{i2} \\ \vdots \\ v_{in_f} \\ s_{i1} \\ s_{i2} \\ \vdots \\ s_{in_f} \end{bmatrix}; h_i = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \quad (28)$$

s_{ij} is the slack variable added to the constraint $v_{ij} - u_j \leq 0$.

4.2 Computational Results

The performance of the parallel interior point algorithm were tested on the set of facility location problems from the OR-Library [1]. The capacitated and the uncapacitated test problems of this test suite are identical except the capacity values are ignored in the case of uncapacitated problems.

All the uncapacitated problems of this test suite was solved by the parallel interior point algorithm with out any problems or numerical instability and the correct optimal solution was obtained. The results of some the selected problems are given in Table (2). The parallel efficiencies obtained from the other problems are similar as the dimensions of these problems are similar. Thus, the results given in Table (2) are illustrative of all the problems of the test set. The dimensions of the problems tested are given in Table (1).

In all but one problem (uncapc) the solution of the root node (linear program with 0-1 requirement of the binary variables relaxed) yielded a solution with either 0 or 1 for the binary variables. Therefore, no further branching was required for these problems. The three problems uncapa, uncapb and uncapc have 101,000 constraints, 200,100 variables and 400,000 nonzeros, and can be considered extremely large compared to most LPs that are solved today. As there are 1,000 blocks ($n_c = 1,000$) the size of D is large compared to $\hat{C}$. Thus, good speedups are achieved for these large problems. Three subproblems were solved for capc to obtain the optimal mixed integer solution. Approximately 40% of the time was taken for the initial linear program, while around 30% of the time was taken for each of the two subsequent nodes. To estimate the parallel efficiency achieved by the algorithm, each problem was tested with the number CPUs taking two values (with a higher and lower number of CPUs). The time taken with the lower number of CPUs is used as the base for the calculation of the speedup (i.e. we do not always use the time taken with one CPU as the base for the speedup calculations as it is usually done). This is because the memory of one CPU was not sufficient to hold all the required data for some problems. For these problems, the time taken by the smallest number of CPUs, that would distribute the diagonal blocks equally as well as solve the problem successfully, was used as the base for the speedup calculations. Time:1 and Time:2 columns of Table (2) gives the time taken by the parallel interior point algorithm with the lower and higher number of CPUs respectively. The number of CPUs are given within brackets. All the times reported are only for the computation phase of the interior point algorithm.

Problem name	n_f	n_c	Number of integer variables	Rows	Columns	Nonzeros
uncap71	16	50	16	850	1,616	3,200
uncap81	25	50	25	1,300	2,525	5,000
uncap91	25	50	25	1,300	2,525	5,000
uncap101	25	50	25	1,300	2,525	5,000
uncap111	50	50	50	2,550	5,100	10,000
uncap121	50	50	50	2,550	5,100	10,000
uncap131	50	50	50	2,550	5,100	10,000
uncapa	100	1,000	100	101,000	200,100	400,000
uncapb	100	1,000	100	101,000	200,100	400,000
uncapc	100	1,000	100	101,000	200,100	400,000

Table 1: The dimensions of uncapacitated problems tested

Problem name	BB nodes	Time:1 (nprocs) (secs)	Time:2 (nprocs) (secs)	Speedup	Parallel efficiency
uncap71	1	10.44 (1)	0.62(50)	16.83	33.66
uncap81	1	23.19 (1)	1.23(50)	18.85	37.70
uncap91	1	23.20 (1)	1.15(50)	20.17	40.34
uncap101	1	23.46 (1)	1.17(50)	20.05	40.10
uncap111	1	101.69 (1)	4.71(50)	21.59	43.18
uncap121	1	101.55 (1)	4.88(50)	20.81	41.62
uncap131	1	101.54 (1)	4.61(50)	22.03	44.06
uncapa	1	1176.36(10)	113.77(125)	103.40	82.72
uncapb	1	1597.61(10)	149.01(125)	107.17	85.57
uncapc	3	4531.16(10)	427.00(125)	106.16	84.92

Table 2: Results of the uncapacitated facility locations problems on the AP1000

Problem name	BB nodes	Time:1 (nprocs) (secs)	Time:2 (nprocs) (secs)	Speedup	Parallel efficiency
uncapa	1	907.68(1)	93.97(10)	9.66	96.59
uncapb	1	1163.04(1)	122.31(10)	9.51	95.08
uncapc	3	3826.09(1)	405.28(10)	9.44	94.40

Table 3: Results of the uncapacitated facility locations problems on the SP2

Table (3) shows the performance of the algorithm on the IBM SP2 with the three larger problems. The parallel efficiency achieved with the three larger problems is very good. The reason for this is that the amount of computation done in parallel is large compared to the sequential portion. This is because the number of rows and columns of D is large compared to the number of columns of V . Thus, there is significant potential for solving large uncapacitated facility location problems on parallel computers. A good estimate of the degree of concurrency available can be made by knowing the number of facilities and locations. It can be observed that a CPU of the SP2 is much faster than the a AP1000 CPU. We had access to only 10 of the available 22 CPUs on the SP2 and thus results shown are with a maximum of 10 CPUs. The algorithm solves the three large problems capa, capb, and capc with excellent speedups on both the AP1000 and the SP2. Parallel efficiencies around 85% are achieved with 125 CPUs on the AP1000, while efficiencies around 95% are achieved with 10 CPUs on the SP2. The AP1000 solves the smaller problems with parallel efficiencies ranging from 33% to 41% (50CPUS).

5 Capacitated Facility Location problems

5.1 Problem Formulation

In uncapacitated facility location problems the amount of goods that can be supplied by a given facility is assumed to be unlimited. For capacitated problems a limit to this value is included in the formulation. Thus, extra constraints have to be included in the model for uncapacitated problems. The new mixed integer linear programming formulation is given in (29). d_i is the demand of customer i , while C_j is the maximum capacity of facility j . The formulation of capacitated problems differ from the uncapacitated problems due to the inclusion of an extra constraint $\sum_{i=0}^{n_c} d_i v_{ij} \leq C_j u_j$ (for $j = 1 \dots n_f$). Each of these constraints ensure the demand supplied by an established facility is less than its capacity. The constraint $v_{ij} \leq u_j$ is also changed to $v_{ij} \leq \text{MIN}(1, C_j/d_i)u_j$. This prevents the demand supplied from facility j to customer i exceeding the capacity of facility j .

$$\begin{aligned}
& \text{minimize} && \sum_{j=1}^{n_f} F_j u_j + \sum_{i=1}^{n_c} \sum_{j=1}^{n_f} f_{ij} v_{ij} \\
& \text{subject to} && \sum_{j=1}^{n_f} v_{ij} = 1, \quad i = 1 \dots n_c \\
& && -\text{MIN}(1, C_j/d_i)u_j + v_{ij} \leq 0 \quad j = 1 \dots n_f \quad i = 1 \dots n_c \\
& && -C_j u_j + \sum_{i=0}^{n_c} d_i v_{ij} \leq 0 \quad i = 1 \dots n_f \\
& && u_j \in \{0, 1\}, \quad j = 1 \dots n_f \\
& && v_{ij} \geq 0, \quad j = 1 \dots n_f \quad i = 1 \dots n_c
\end{aligned} \tag{29}$$

The new constraints create nonzeros in the columns of the diagonal blocks and therefore link all the blocks together. Thus, for capacitated problems there would be horizontal blocks as well as vertical blocks. The $H_i, i = 1 \dots n_c$ blocks are given by,

$$H_i = \begin{bmatrix} d_i & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & d_i & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \\ 0 & 0 & \cdots & d_i & 0 & 0 & \cdots & 0 \end{bmatrix}, \quad (30)$$

and the M and S sub-matrices are given by,

$$M = \begin{bmatrix} -C_1 & 0 & \cdots & 0 \\ 0 & -C_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & -C_{n_f} \end{bmatrix}, \quad (31)$$

$$S = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}. \quad (32)$$

The change of the constraint $-u_j + v_{ij} \leq 0$ in the formulation for uncapacitated problems to $-MIN(1, C_j/d_i)u_j + v_{ij} \leq 0$ in capacitated problems will also change the matrix V_i to,

$$V_i = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ m_{i1} & 0 & \cdots & 0 \\ 0 & m_{i2} & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & m_{in_f} \end{bmatrix}, \quad (33)$$

where $m_{ij} = MIN(1, C_j/d_i)$.

The matrix S consists of the nonzeros from the slack variables included to convert the constraints $-MIN(1, C_j/d_i)u_j + v_{ij} \leq 0$ into standard LP form. The D blocks are identical to the D blocks of the uncapacitated problems. Once the matrix is partitioned in this manner the parallel interior point algorithm can be used to solve them. The resulting linear program is given in figure (2). Where, $h_0 = [0 \ 0 \ \cdots \ 0]^T$.

5.2 Computational Results

As in the uncapacitated case all the smaller problems (problems other than capa, capb and capc) were solved without any numerical or convergence problems with our parallel interior point code. Unlike the uncapacitated problems, a greater number of nodes had to be searched (LP subproblems) to obtain the optimal solution. It was also noted, in general (but not always), that as the capacity of a problem was decreased, the number of nodes visited increased. Each of the three large uncapacitated problems in the OR-Library (uncapa, uncaph, uncaphc) creates four capacitated facility location problems. For example the uncapacitated problem uncapa of the OR-Library is equivalent to the four capa capacitated problems in all respects except for the capacity. The capacity of the four problems capa.1, capa.2, capa.3 and capa.4 takes the values 8000, 10000, 12000 and 14000 respectively.

Table(5) gives the results of some of the smaller capacitated problems (problems excluding capa, capb, capc) on the AP1000. The parallel efficiencies achieved range from 27% to 33% with 50 CPUs for these problems. The parallel efficiencies achieved are lower compared to the

$$\begin{array}{llllllll}
\text{minimize} & F^T u & + f_1^T v_1 & f_2^T v_2 & \cdots & + f_{n_f}^T v_{n_f} & & \\
\text{subject to} & Mu & + H_1 v_1 & + H_2 v_2 & \cdots & + H_{n_f} v_{n_f} & + St & = h_0 \\
& V_1 u & + D_1 v_1 & & \cdots & & & = h_1, \\
& V_2 u & & + D_2 v_2 & \cdots & & & = h_2, \\
& \vdots & \cdots & \vdots & \vdots & \cdots & & = \vdots, \\
& V_{n_f} u & & & & + D_{n_f} v_{n_f} & & = h_{n_f}, \\
& & v_1, & v_2, & \cdots & v_{n_f} & & \geq 0, \\
& u & & & & & & \in \{0, 1\}.
\end{array}$$

Figure 2: Linear program for Capacitated facility location problems

uncapacitated problems. This can be attributed to the increase in the sequential component due to the presence of horizontal blocks in capacitated problems. Also, a small amount of time was spent sequentially in tasks such as branching and bounding. This time was smaller (or absent) in uncapacitated problems due to the fewer number of nodes searched. Table(4) shows the sizes of the problems tested.

Problem name	n_f	n_c	Number of integer variables	Rows	Columns	Nonzeros
cap41	16	50	16	866	1,632	4,032
cap43	16	50	16	866	1,632	4,032
cap51	16	50	16	866	1,632	4,032
cap61	16	50	16	866	1,632	4,032
cap71	16	50	16	866	1,632	4,032
cap81	25	50	25	1,325	2,550	6,300
cap91	25	50	25	1,325	2,550	6,300
cap101	25	50	25	1,325	2,550	6,300
cap111	50	50	50	2,600	5,150	12,600
cap122	50	50	50	2,600	5,150	12,600
cap131	50	50	50	2,600	5,150	12,600
capa	100	1,000	100	101,100	200,200	500,200
capb	100	1,000	100	101,100	200,200	500,200
capc	100	1,000	100	101,100	200,200	500,200

Table 4: The dimensions of capacitated problems tested

Most of the three larger sets of problems (capa, capb, capc) could not be solved with our parallel interior point code as a very large number of nodes had to be searched. And also the time taken for each node was high due to the size of the problem. We were unable to find enough time on the computer to keep the code running until the optimal solution was found (even in parallel). Table(6) gives the times taken by the parallel interior point code to solve the root node of problems capa.4, capb.4 and capc.4. The solution of the root node of capa.4 yielded the optimal integer solution and no further branching was required. The parallel efficiencies achieved range from 80% to 88% with 10 CPUs for these problems. The results indicate that good speedups are achieved in the solution of the root node (although it is smaller than the speedups achieved for the uncapacitated problems). As almost all the time taken in branch and bound is to solve the LP subproblems, the parallel interior point algorithm should still achieve good efficiency if the code was allowed to run until the solution was found. In fact if a larger number of CPU's were at our disposal we may have been able to solve these problems in a reasonable amount of time.

Problem name	BB nodes	Time:1 (nprocs) (secs)	Time:2 (nprocs) (secs)	Speedup	Parallel efficiency
cap41	1	21.62 (1)	1.36(50)	15.89	31.79
cap43	7	120.06 (1)	7.09(50)	16.93	33.86
cap51	10	161.47 (1)	9.78(50)	16.51	33.02
cap61	1	17.06 (1)	0.98(50)	17.48	34.96
cap71	1	27.45 (1)	1.58(50)	17.37	34.74
cap81	3	144.03 (1)	8.69(50)	16.57	33.14
cap91	1	47.83 (1)	2.76(50)	20.17	40.34
cap101	1	67.30 (1)	3.90(50)	17.32	34.64
cap111	5	812.13 (1)	61.02(50)	13.31	26.62
cap122	7	1554.72(1)	109.36(50)	14.22	28.44
cap131	1	242.34 (1)	16.99(50)	14.26	28.52

Table 5: Results of the capacitated facility locations problems on the AP1000

Problem name	BB nodes	Time:1 (nprocs) (secs)	Time:2 (nprocs) (secs)	Speedup	Parallel efficiency
capa.4	1	966.07(2)	239.14(10)	8.08	80.80
capb.4*	1	4349.84(2)	984.59(10)	8.84	88.84
capc.4*	1	2454.49(2)	553.60(10)	8.86	88.60

* not solved till optimality

Table 6: Results of the capacitated facility locations problems on the SP2

Next, we evaluate the performance of the parallel interior point algorithm in solving only the non-root subproblems. As stated previously, we compare the parallel interior point method with the simplex method in re-optimising non-root nodes. The simplex method is considered to be superior in warm starting linear programs and the parallel interior point method may not provide significant real speedup. That is, it may be advantageous to switch to the simplex method to solve the non-root nodes. To investigate if it is beneficial to continue solving the non-root nodes with the parallel interior point algorithm rather than switching to a sequential simplex code, we compare the average time taken in solving only the non-root nodes by the simplex method with our parallel interior point code. For this we use the mixed integer programming code from IBM's Optimisation Subroutine Library (OSL-EKKMSLV). The root node was solved with the plain simplex routine (EKKSSLV) and the subsequent nodes were solved with EKKMSLV. EKKMSLV is called such that the optimal basis of the root node is used to warm start the simplex code in solving the of the subsequent subproblem. The time taken on one processor (RS 6000/590) of the SP2 was observed for both the root node and the non-root nodes. Since it was not possible to solve any of the larger problems with either OSL or our code due to the lack of memory and the lack of computer time on the SP2, the size of the problems capa, capb and capc was reduced. The number of facilities and customers were divided by four such that the number of facilities is 25 and the number of customers is 250. Only the parameters of the first 25 facility locations and 250 customers were considered and the rest was ignored. We will refer to these problems as capa25.250, capab.25.250 and capac.25.250. The capacity of the facilities was varied and the performance of the parallel interior point method was compared with the simplex routine of OSL. Table (8) summarises the results of these investigations. In most instances OSL searched fewer nodes than our branch and bound implementation. In two cases our parallel branch and bound solved a smaller number of nodes compared to OSL. The number of nodes searched in the branch and bound is not an issue in our investigation. Any algorithm or heuristic used to reduce the number of nodes searched

is equally applicable, whether the subproblems are solved with the simplex or the interior point method. Our main concern is the time taken to solve each subproblem. Every attempt was made to make both the implementations solve an equal number of nodes and in most cases the number of nodes searched are almost equal for both codes. Table (9) shows the overall speedups achieved by the parallel interior point algorithm with 10 CPUs compared with the the OSL simplex code. The results show that our parallel interior point implementation is 2 to 214 times faster than OSL with 10 CPUs. This speedup calculations include the time taken to solve the root as well as the non-root nodes.

Problem name	n_f	n_c	Number of integer variables	Rows	Columns	Nonzeros
uncapa.25.250	25	250	25	6,525	12,550	31,300
uncapb.25.250	25	250	25	6,525	12,550	31,300
uncapc.25.250	25	250	25	6,525	12,550	31,300

Table 7: The dimensions of capacitated problems tested (with the size reduced)

Problem name	Capacity	OSL Simplex			Interior Point				
		Number of non-root nodes	Time		Number of non-root nodes	1 CPU		10 CPU	
			Time root node	Time non-root nodes		Time root node	Time non-root nodes	Time Root nodes	Time non-root nodes
capa.25.250	14,000	0	289.87	0.00	0	9.52	0.00	1.35	0.00
capa.25.250	12,000	17	260.80	95.44	17	18.94	322.29	3.14	51.76
capa.25.250	10,000	0	236.06	0.00	0	12.46	0.0	1.75	0.00
capa.25.250	8,000	0	353.01	0.00	0	10.48	0.0	1.89	0.00
capa.25.250	6,000	264	201.66	4837.84	273	20.49	4278.5	3.12	685.88
capa.25.250	1,500	89	177.12	580.26	123	17.23	1881.53	2.65	295.26
capb.25.250	8,000	0	328.85	0.00	0	16.47	0.00	2.67	0.00
capb.25.250	7,000	0	272.28	0.00	0	15.41	0.00	2.14	0.00
capb.25.250	6,000	0	212.93	0.00	0	14.41	0.00	2.01	0.00
capb.25.250	5,000	2	323.33	5.50	2	15.01	29.27	2.18	4.06
capb.25.250	2,000	254	160.98	1732.80	241	16.67	5388.34	2.85	839.75
capb.25.250	1,500	1635	153.38	12210.18	499	21.36	11618.12	3.34	1736.96
capc.25.250	7,250	2	267.29	2.66	3	17.23	49.89	2.71	7.54
capc.25.250	6,500	2	236.11	1.89	3	17.60	49.81	2.76	7.67
capc.25.250	5,750	2	208.41	2.12	3	16.15	48.45	2.23	6.71
capc.25.250	5,000	4	230.27	23.96	5	22.23	109.6	3.21	15.17
capc.25.250	2,000	43	192.53	285.97	44	23.53	1301.48	4.72	190.28
capc.25.250	1,500	71	190.27	422.57	33	22.54	753.308	3.53	113.49

Table 8: A comparison of the parallel interior point algorithm with the OSL's Simplex

Table(10) shows the speedups of the parallel interior point for the solution of non-root nodes. The speedups calculations are based on the average time taken to solve non-root nodes with 1 CPU as well as the average time taken by OSL Simplex. This analysis was carried out to investigate, if it is advantageous to switch over to the simplex method to solve the non-root nodes.

The interior point algorithm with 1 CPU is approximately 7 to 30 times faster than the simplex method in solving the root node. When the number of CPUs is increased the superiority of the interior point algorithm in solving the root node is even more pronounced. However, the simplex algorithm out-performs the interior point method with one CPU, in solving non-root nodes in all but one instance. This is not surprising because the simplex method has superior warm starting

Problem name	Capacity	Speedup (10 CPU) based on OSL Simplex
capa.25.250	14,000	214.71
capa.25.250	12,000	6.49
capa.25.250	10,000	134.89
capa.25.250	8,000	186.78
capa.25.250	6,000	7.31
capa.25.250	1,500	2.54
capb.25.250	8,000	123.16
capb.25.250	7,000	127.23
capb.25.250	6,000	105.94
capb.25.250	5,000	52.70
capb.25.250	2,000	2.25
capb.25.250	1,500	7.10
capc.25.250	7,250	26.33
capc.25.250	6,500	22.82
capc.25.250	5,750	23.55
capc.25.250	5,000	13.83
capc.25.250	2,000	2.45
capc.25.250	1,500	5.24

Table 9: The overall speedups based on the OSL Simplex with 10 CPUs on the SP2

Problem name	Capacity	Average time per non-root node			Speedup of 10 CPUs	
		OSL	Interior point 1CPU	Interior Point 10CPU	Based on 1 CPU Interior point	Based on OSL
capa.25.250	12,000	5.61	18.95	3.04	6.23	1.85
capa.25.250	6,000	18.33	15.67	2.51	6.24	7.30
capa.25.250	1,500	6.51	15.30	2.40	6.37	2.71
capb.25.250	5,000	2.75	14.63	2.03	7.21	1.36
capb.25.250	2,000	6.82	22.35	3.48	6.42	1.96
capb.25.250	1,500	7.47	23.28	3.48	6.69	2.14
capc.25.250	7,250	1.33	16.63	2.51	6.63	0.53
capc.25.250	6,500	0.95	16.60	2.56	6.48	0.37
capc.25.250	5,750	1.06	16.15	2.23	7.26	0.48
capc.25.250	5,000	5.99	21.92	3.03	7.23	1.98
capc.25.250	2,000	6.65	29.57	4.32	6.85	1.54
capc.25.250	1,500	5.95	22.82	3.43	6.65	1.73

Table 10: The efficiency of solving non-root nodes by the parallel interior point method compared with OSL simplex

capability. When the number of CPUs are increased the performance of the parallel interior point method gets more competitive and a peak speedup of 7.3 with 10 CPUs (based on OSL simplex) is achieved. Thus, from the results obtained, it is apparent that in some cases the parallel interior point method is competitive with the simplex method for solving non-root nodes of a branch bound implementation. In 9 out of 12 test problems the parallel interior point method with 10 CPUs was faster than the sequential simplex code. As OSL is a commercial code which has been highly optimised to the RS6000 architecture, we believe that the performance of our algorithm could be improved further by fine tuning the code.

6 Stochastic Facility Location problems

6.1 Problem Formulation

In the previous formulations of facility location problems, the parameters of the problem (such as the demand of each customer or the cost of transporting from the facilities to customers) were assumed to be deterministic. That is, the exact values of these parameters were known before the model is solved. In practice this rarely happens, and most often these parameters are random. Stochastic programming was developed to overcome this inadequacy of deterministic linear programming. The reader is referred to [9] for more details on stochastic programming. One of the commonly used methods to include randomness into the parameters of linear programs is the two stage stochastic programming with recourse model. Also most often the stochasticity is considered in the form of scenarios assuming that the random parameters have a discrete probability distribution. This provides a robust method to model stochastic programs.

The problem of random demands of customers in facility location problems is addressed in this section. In a majority of real world facility location problems the demand of the customers are not known before the model is solved, and generally an approximation of the demand is used to solve the model. But in real world problems it is difficult to accurately predict the actual demand before the model is solved.

We will consider capacitated problems in our implementation of stochastic facility location problems. The facilities that are to be established have to be decided before knowing what scenario is going to occur (referred to as “Here and now”) and therefore the u_j variables of figure (29) are first stage variables, while v_{ij} variables are second stage variables with a set of v_{ij} variables for each scenario. Thus, the capacitated facility location problem given by figure (29) can be reformulated as shown below; $\text{minimize } \sum_{j=1}^{n_f} F_j u_j + \sum_{k=1}^{n_s} p^k \{ \sum_{i=1}^{n_c} \sum_{j=1}^{n_f} f_{ij} v_{ij}^k \}$

subject to

$$\begin{aligned}
 & \sum_{j=1}^{n_f} v_{ij}^k = 1, & i = 1 \cdots n_c & & & \\
 -u_j & + v_{ij}^k \leq 0, & j = 1 \cdots n_f & i = 1 \cdots n_c & k = 1 \cdots n_s & \\
 -C_j u_j & + \sum_{i=0}^{n_c} d v_{ij}^k \leq 0 & i = 1 \cdots n_f & k = 1 \cdots n_s & & (34) \\
 -MIN(1, C_j/d_i^k) u_j & + v_{ij}^k \leq 0 & j = 1 \cdots n_f & i = 1 \cdots n_c & k = 1 \cdots n_s & \\
 u_j & \in \{0, 1\}, & j = 1 \cdots n_f & & & \\
 & v_{ij}^k \geq 0, & j = 1 \cdots n_f & i = 1 \cdots n_c & k = 1 \cdots n_s &
 \end{aligned}$$

n_s is the number of scenarios, while v_{ij}^k is the proportion of i th customers demand supplied by facility j if scenario k occurs.

As the number of scenarios are quite large for most stochastic linear programs, we will include all the columns of v_{ij} from one scenario in a diagonal block D_s ($k = 1 \cdots n_s$). Thus, all the second stage variables of a scenario are included in a diagonal block. The columns due to u_j are included in the vertical blocks V_s . There will be no H blocks in this formulation and it can be observed that stochastic linear programs also can be formulated in the block structured format given in equation (4). Our parallel interior point algorithm can again be used, to solve stochastic facility location problems.

6.2 Computational Results

The problem cap43 from the OR-Library was modified to carry out the computational testing of the parallel interior point algorithm on stochastic facility location problems. Stochasticity was introduced into this problem by assuming that the demand of a given number of customers demands (selected randomly) were stochastic. The number of customers with stochastic demand was varied to obtain problems with scenarios ranging from 8 to 128. The stochastic demand of each selected customer was assumed to have two realisations, from a uniform distribution $\pm 50\%$ around the deterministic demand. If the number of stochastic customers selected are β , and since there are two realizations per selected customer, the total number of scenarios is 2^β . Thus the number of scenarios increase rapidly. The results of the parallel algorithm on the IBM SP2, when the number of random customers varies from 3 to 7, is given in Table (12). The number of customers with random demands can not be increased beyond 7 due to a lack of memory. Table(6.2) shows the sizes of the stochastic linear programs that were tested. The results of solving the equivalent large mixed integer with OSL simplex is shown in Table (14). The speedups achieved by the parallel interior point method on the AP1000 is given in Table (13). The super-linear speed exhibited by some problems can be attributed to the performance of the cache in each cell of the AP1000. The problems which were tested with smaller number of CPU's had a lower cache hit ratio than with the ones with a larger number of CPU's. Therefore the execution time when the number of CPU's was small was proportionately longer than when the number of CPU's was large, leading to super-linear speedup. Whilst, this result can be misleading it does highlight an advantage of parallel processing of large problems, namely, improved cache and memory behaviour when the amount of data per processor is reduced. The results show that the parallel efficiency increases as the number of scenarios increase. This is as expected as the amount of parallel computations compared to the sequential computations increase as the number of scenarios goes up.

Problem name	Number of Scenarios	Number of integer variables	Rows	Columns	Nonzeros
cap43	8	16	6,928	12,944	32,256
cap43	16	16	13,856	25,872	64,512
cap43	32	16	27,712	51,728	129,024
cap43	64	16	55,424	103,440	258,048
cap43	128	16	110,848	206,864	516,096

Table 11: The dimensions of uncapacitated problems tested

Problem Name	Number of Scenarios	BB nodes	Time:1 (nprocs) secs	Time:2 (nprocs) secs	Speedup	Parallel efficiency (%)
cap43	8	7	43.72(1)	11.69(8)	3.73	46.6
cap43	16	7	68.73(1)	13.33(8)	5.16	64.5
cap43	32	5	135.85(1)	22.16(8)	6.13	76.6
cap43	64	5	319.94(1)	47.02(8)	6.84	85.5
cap43	128	17	2341.06(1)	291.93(8)	8.02	100.2

Table 12: Results of the stochastic facility locations problems on the SP2

7 Conclusion

In this paper, we have developed an algorithm to solve block structured linear programs on parallel computers. This algorithm can solve linear programs which have diagonal blocks with common

Problem Name	Number of Scenarios	BB nodes	Time:1 (nprocs) secs	Time:2 (nprocs) secs	Speedup	Parallel efficiency (%)
cap43	8	7	718.5(1)	89.42(8)	8.04	100.4
cap43	16	7	1110.67(1)	69.34(16)	16.02	100.1
cap43	32	5	1118.08(2)	69.95(32)	31.97	99.9
cap43	64	5	1311.34(4)	83.38(64)	62.90	98.28
cap43	128	17	4695.05(8)	294.22(128)	127.66	99.73

Table 13: Results of the stochastic facility locations problems on the AP1000

Problem Name	Number of Scenarios	Number of non root nodes	Time root node (secs)	Time non-root nodes (secs)
cap43	8	7	138.2	29.6
cap43	16	5	573.20	51.90
cap43	32	†	†	†
cap43	64	†	†	†
cap43	128	†	†	†

† Cannot be solved by OSL due to the lack of memory

Table 14: Results of the stochastic facility locations problems with OSL Simplex on the SP2

rows and common columns. In our approach, we determine the independent sections of computations, which are carried out in parallel, from the block structure of the linear program. The blocks are specified by the model builder and the work load is distributed to the processors before the solution phase. The parallel interior point algorithm, carries out the computations associated with the diagonal blocks in parallel using sparse matrix linear algebra and the computations associated with common rows and columns are carried out sequentially with dense matrix linear algebra. Good speedups can be achieved with this algorithms provided the sequential component is small compared to the parallel component. That is the number of common rows and common columns should be small compared to the total number columns and rows. This method will also partition the sparse and dense components of the factorisation. Thus, it is possible to use the most efficient parallel algorithms to factorise the sparse and dense components separately. Although only the sparse component of the factorisation is parallelised in our implementation, we believe that even better results can be achieved by parallelising both the sparse and dense components especially when the number of linking rows and columns are large.

We used capacitated, uncapacitated and stochastic linear programs to demonstrate the performance of the parallel algorithm. We also investigate the possibility of re-optimising non-root nodes of the branch and bound algorithm with the parallel interior point method. It is shown that in some cases, the parallel interior point method is competitive with the simplex code from OSL in re-optimising linear programs. Developing the warm start capability for the interior point method is active area of research at present. Any improvements in this capability will have a impact on using the parallel interior point algorithm to solve mixed integer programs.

All the problems used to test the algorithm had blocks of equal size. Parallelisation of problems having unequal block sizes will make it difficult in balancing the work load among processors. We expect to carry out further research into this aspect in the future. We also intend to combine the parallel interior point algorithm with an Mathematical Modelling Language, which will generate the block structure required by the parallel interior point algorithm, from the model of the problem.

References

- [1] J.E. Beasley. OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:11:1069–1072, 1990.
- [2] J.R. Birge and D. Holmes. Efficient solution of two-stage stochastic linear programs using interior point methods. *Computational Optimization and Applications*, 1:245–276, 1992.
- [3] J.R. Birge and L. Qi. Computing block-angular karmarkar projections with applications to stochastic programming. *Management Science*, 34:12:1472–1479, 1988.
- [4] B. Borchers and J. Mitchell. Using the interior point method in a branch and bound algorithm for integer programming. Mathematics Department Technical Report 195, Rensselaer Polytechnic Institute, 1992.
- [5] A. Brook, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide*. The Scientific Press, 1988.
- [6] I Choi and D. Goldfarb. Solving multicommodity network flow problems by an interior point method. In T. Coleman and Y. Li, editors, *Large Scale Numerical Optimization*, pages 58–69. SIAM, 1990.
- [7] J. Czyzyk, R. Fourer, and Sanjay Mehrotra. Using a massively parallel processor to solve large sparse linear programs by an interior point method. Report, Department of Industrial Engineering and Management Sciences, NorthWestern University, Evanston, Illinois, USA, 1994.
- [8] J. Eckstein, R. Qi, V. Ragulin, and S.A. Zenios. Data parallel implementation of dense linear programming algorithms. Technical Report TMC-230, Thinking Machines Corporation, 1992.
- [9] Y. Ermoliev and R.G.B. Wets. *Numerical Techniques for Stochastic Optimization*. Springer, Berlin, Germany, 1988.
- [10] R.M. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modelling Language for Mathematical Programming*. The Scientific Press, 1993.
- [11] D. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Society, COAL Newsletter*, 13:10–12, 1985.
- [12] A. George and J.W. Liu. *Computer Solutions to Large Sparse Positive Definite Systems*. Prentice-Hall Inc, Englewood Cliffs, New Jersey, USA, 1981.
- [13] D. Yang, and S.A. Zenios. A scalable parallel interior point algorithm for stochastic linear programming and robust optimization. Report 95-07, Department of Public and Business Administration, University of Cyprus, Nicosia, CYPRUS, ; To appear in *Computational Optimization and Applications*, 1995.
- [14] E.R. Jessup, D. Yang, and S.A. Zenios. Parallel factorization of structured matrices arising in stochastic programming. *SIAM journal on Optimization*, 4(4):833–846, 1994.
- [15] G. Karypis, A. Gupta, and V. Kumar. A parallel formulation of interior point algorithms. Technical Report TR 94-20, Department of Computer Science, University of Minnesota, Minneapolis, Maine, USA, 1994.
- [16] R. Levkowitz, J. Anderson, and G. Mitra. Interior point method for LP on parallel computers. In *System modelling and Optimization, Proceedings of the 15th IFIP conference, Zurich, Switzerland*, 1991.
- [17] J.W. Liu. Modification to the minimum degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, 11:1:141–153, 1985.

- [18] I.J. Lustig. Feasibility issues in a primal-dual interior point method for linear programming. *Mathematical Programming*, 49:145–162, 1991.
- [19] I.J. Lustig and G. Li. An implementation of a parallel primal-dual interior point method for block structured linear programs. *Computational Optimization and Applications*, 1:141–161, 1992.
- [20] I.J. Lustig, R.E. Marsten, and D.F. Shanno. Computational experience with a primal-dual interior point method for linear programming. *Linear Algebra and Its Applications*, 151:191–222, 1992.
- [21] I.J. Lustig, R.E. Marsten, and D.F. Shanno. On implementing mehrotra’s predictor-corrector interior point method for linear programming. *SIAM Journal of Optimization*, 2:3:435–449, 1992.
- [22] S. Mehrotra. On implementation of a (primal-dual) interior point method. *SIAM Journal of Optimization*, 2:4:575–601, 1992.
- [23] A. De Silva and D.A. Abramson. A parallel interior point method for stochastic linear program. Research report CIT-94-4, Griffith University, Australia, 1994.
- [24] R.J. Vanderbei. Symmetric quasi-definite matrices. Technical Report SOR 91-10, Department of Civil Engineering and Operations Research, Princeton University, New Jersey, USA, 1991.
- [25] R.J. Vanderbei and T.J. Carpenter. Symmetric indefinite systems for interior point methods. *Mathematical Programming*, 58:1–32, 1993.