An Implementation of a Primal-Dual Interior Point Method for Block-Structured Linear Programs

Irvin Lustig
Guangye Li

CRPC-TR92194
January 1992

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Revised and renamed in June 1992. Appears in Computational Optimization and Applications, 1, 141-161, 1992.

Computational Optimization and Applications, 1, 141–161 (1992)

An Implementation of a Parallel Primal-Dual Interior Point Method for Block-Structured Linear Programs

IRVIN J. LUSTIG
Department of Civil Engineering and Operations Research, Princeton University, Princeton, NJ 08544.

GUANGYE LI
Department of Mathematical Sciences and Center for Research in Parallel Computation, Rice University, Houston, TX 77251.

Received January 10, 1992, Revised June 8, 1992.

Abstract. An implementation of the primal-dual predictor-corrector interior point method is specialized to solve block-structured linear programs with side constraints. The block structure of the constraint matrix is exploited via parallel computation. The side constraints require the Cholesky factorization of a dense matrix, where a method that exploits parallelism for the dense Cholesky factorization is used. For testing, multicommodity flow problems were used. The resulting implementation is 65%–90% efficient, depending on the problem instance. For a problem with K commodities, an approximate speedup for the interior point method of $0.8K$ is realized.

Keywords: parallel computing, interior point methods, linear programming, multicommodity flow problems.

1. Introduction

Many applications of linear programming can be modeled using the following formulation:

$$\text{minimize} \quad \sum_{k=1}^{K} (c^k)^T x^k \tag{1}$$

$$\text{subject to} \quad Ax^k = b^k, \quad k = 1, \ldots, K \tag{2}$$

$$\sum_{k=1}^{K} B^k x^k = \bar{u} \tag{3}$$

$$x^k \geq 0, \quad k = 1, \ldots, K \tag{4}$$

This formulation requires the determination of decision vectors $x^1, x^2, \ldots, x^K$ that must satisfy a set of independent block constraints (2) and a set of side constraints (3), while minimizing the objective function (1). If the constraints (3) were not present, then K independent linear programs could be solved

yielding the optical solution. The presence of the side constraints increases the computational requirements for finding a solution of the entire linear program. Often, the size of each block as well as the number of variables in each block is large, yielding a very large linear program.

The recent interest in interior point methods for linear programming has spawned implementations of both interior point methods and simplex methods that can easily solve large-scale linear programs with thousands of constraints and variables. This increase in size is also due to the improvement in computer hardware and software technology. Even with the improvement in simplex and interior point methodologies, block-structured problems with side constraints still remain among the most difficult types of linear programs to solve due to their sheer size. Interior point methods have difficulties solving these types of problems because of the fill-in within the Cholesky factor even after a suitable ordering has been performed. A possible solution to these problems is to exploit parallel processing.

For interior point methods, parallelism has been exploited in theory by computing simple decomposable preconditioners and using these preconditioners in a conjugate gradient method to solve the normal equations generated on each iteration of the interior point method. However, these methods, proposed by Vannelli ([16]), have only been shown to work on extremely well-conditioned problems with a very strong block structure. In addition, conjugate gradient methods have only been used successfully in dual affine scaling interior point algorithms. In general, numerical difficulties will most likely appear when using conjugate gradient algorithms, especially in the context of primal-dual interior point methods.

Schultz and Meyer ([14]) approach the parallel solution of block-structured problems using a shifted-barrier concept. Their method remains in the relative interior of only the side constraints by applying a shifted-barrier function to these constraints. Their method, which has been implemented on a parallel computer, does not compute fully optimal solutions nor dual optimal solutions. Their implementation, which was specialized for multicommodity flow problems where the block constraints are network constraints, takes full advantage of the network structure.

For solving systems of equations using parallel processing, iterative methods have frequently been used. With a predictor-corrector interior point method, an iterative method is not appropriate because one factorization is needed on each interior point method iteration to solve two sets of equations with different right-hand sides. Hence, a direct method seems to be more appropriate. The systems of interest are very large and sparse. Though there have been some new developments in parallel sparse factorizations (see, for example, [7]), it is difficult to obtain very high efficiencies for parallel direct matrix factorizations and triangular solves on general sparse problems.

Our approach attempts to take advantage of the problem structure in order to obtain high efficiencies for the overall method. Rather than depending on

a preconditioned conjugate gradient algorithm to solve a system of equations, the block structure of the equations is exploited directly to factor and solve the system. This method, related to work of Choi and Goldfarb ([4]), computes primal and dual optimal solutions and does not take advantage of the special structure in the blocks, but does take advantage of the special structure in the matrices B^k. The resulting implementation is 65%–90% efficient, depending on the problem instance solved.

To test our implementation, we use a test suite drawn from multicommodity flow problems. These problems are a specific well-known form of block-structured linear programs and have been extensively studied in the literature. Assad ([2]) and Kennington ([9]) provide surveys of much of the literature on these problems. Many different algorithms have been proposed to solve these problems, yet the problems still remain a challenge due to their size and complexity. The multicommodity flow model can be applied in a variety of contexts, yielding a number of different formulations. These formulations have an impact on decomposition algorithms applied to the problem as discussed by Jones et al. ([8]).

We develop our algorithm in the context of multicommodity flow problems, but it is important to stress that the algorithm is easily generalizable to any block-structured problem with side constraints of the form described by (1)–(4). The main goal of this paper is to see how parallelism can improve the performance of the basic primal-dual predictor-corrector interior point method.

Section 2 discusses the formulation of multicommodity flow problems in order to establish notation. Section 3 discusses how a primal-dual predictor-corrector interior point method can be applied to such problems. Section 4 discusses how a primal-dual interior point method can use parallelization to solve multicommodity flow problems. This is followed by computational results in Section 5 and some conclusions in Section 6.

2. Multicommodity flow formulation

A linear multicommodity flow problem has K commodities with supplies and demands for each commodity distributed over a network of m nodes and n arcs. Assad ([2]) and Kennington ([9]) provide surveys of the problem. A discussion of the different formulation issues is given by Jones et al. ([8]). We consider problems that are modeled mathematically as:

$$\text{minimize} \quad \sum_{k=1}^{K} (c^k)^T x^k \tag{5}$$

$$\text{subject to} \quad Ax^k = b^k, \quad k = 1, \ldots, K \tag{6}$$

$$\sum_{k=1}^{K} x^k + t = \bar{u} \tag{7}$$

$$x^k + s^k = u^k, \quad k = 1, \ldots, K \tag{8}$$

$$x^k, s^k, t \geq 0, \quad k = 1, \ldots, K \tag{9}$$

Here, $A \in \Re^{m \times n}$ is a node-arc incidence matrix on a network with m nodes and n arcs; $c^k \in \Re^n$ is a vector of costs for commodity k; and b^k is a supply/demand vector for commodity k, where $b_i^k > 0$ indicates a supply node i for commodity k and $b_i^k < 0$ indicates a demand node i for commodity k. The vector $\bar{u} \in \Re^n$ is a vector of mutual capacities corresponding to each arc in the network, while $u^k \in \Re^n$ is a vector of individual capacities. Some of these capacities may be infinite. Note that $x^k \in \Re^n$ is a vector of arc flows for commodity k and $t \in \Re^n$ is a vector of slack variables. The vectors $s^k \in \Re^n$ are a collection of vectors of slack variables for each commodity and link in the network. If arc $j = (i_1, i_2)$ is in the network, then $A_{i_1 j} = 1$ and $A_{i_2 j} = -1$. The value K is the total number of commodities in the problem. The constraints (6) are the flow conservation constraints for the network while (7) are known as the mutual capacity constraints (or bundling constraints) that specify the total maximum flow across all commodities for each arc. The constraints (8) are individual capacity constraints on link flows for specific commodities.

The dual of (5) is

$$\begin{aligned} \text{maximize} \quad & \sum_{k=1}^{K} (b^k)^T y^k - \bar{u}^T v - (u^k)^T w^k \\ \text{subject to} \quad & A^T y^k + z^k - w^k - v = c^k, \quad k = 1, \ldots, K \\ & z^k, w^k, v \geq 0, \quad k = 1, \ldots, K \end{aligned} \tag{10}$$

Each commodity k has a vector of dual variables $y^k \in \Re^m$ associated with the commodity along with a set of reduced costs $(z^k - w^k) \in \Re^n$. The vector $v \in \Re^n$ is a set of dual variables corresponding to the bundling constraints (7). We are interested in methods that compute a pair of optimal primal solutions and optimal dual solutions.

3. Interior point methods for multicommodity flow problems

It has been shown by Lustig et al. ([10]) that a variant of Mehrotra's ([12]) predictor-corrector primal-dual interior point algorithm is the preferred interior point method for solving large-scale problems. The usual primal-dual algorithm and the predictor-corrector variant use Newton's method at each iteration to solve sets of equations. For the primal-dual algorithm, these equations are solved once, while in the predictor-corrector variant, these equations are solved twice with a varying right-hand side. These algorithms will solve linear programs

of the form

$$\begin{aligned} &\text{minimize} && \hat{c}^T\hat{x} \\ &\text{subject to} && \hat{A}\hat{x} = \hat{b} \\ &&& \hat{x}+\hat{s} = \hat{u} \\ &&& \hat{x},\hat{s} \geq 0 \end{aligned} \tag{11}$$

and the associated dual

$$\begin{aligned} &\text{maximize} && \hat{b}^T\hat{y} \\ &\text{subject to} && \hat{A}^T\hat{y}-\hat{w}+\hat{z} = \hat{c} \\ &&& \hat{z},\hat{w} \geq 0. \end{aligned} \tag{12}$$

To solve a multicommodity flow problem represented by equations (5)–(9) as a generic linear program, let

$$\hat{A} = \begin{bmatrix} A & & & & \\ & A & & & \\ & & \ddots & & \\ & & & A & \\ I & I & \cdots & I & I \end{bmatrix}, \quad \hat{b} = \begin{bmatrix} b^1 \\ \vdots \\ b^K \\ \overline{u} \end{bmatrix}, \tag{13}$$

$$\hat{c} = \begin{bmatrix} c^1 \\ \vdots \\ c^K \\ 0 \end{bmatrix}, \quad \hat{u} = \begin{bmatrix} u^1 \\ \vdots \\ u^K \end{bmatrix}$$

$$\hat{x} = \begin{bmatrix} x^1 \\ \vdots \\ x^K \\ t \end{bmatrix}, \quad \hat{s} = \begin{bmatrix} s^1 \\ \vdots \\ s^K \end{bmatrix}, \quad \hat{y} = \begin{bmatrix} y^1 \\ \vdots \\ y^K \\ -\overline{y} \end{bmatrix}, \tag{14}$$

$$\hat{z} = \begin{bmatrix} z^1 \\ \vdots \\ z^K \\ v \end{bmatrix}, \quad \text{and } \hat{w} = \begin{bmatrix} w^1 \\ \vdots \\ w^K \\ 0 \end{bmatrix}$$

where it is understood that the vector t does not appear in the constraint $\hat{x}+\hat{s}=\hat{u}$. Note that the vector $\overline{y}$ is enforcing $v \geq 0$.

In an implementation of a primal-dual interior point algorithm, the most time-consuming and crucial step is the computation of a solution to a system of equations of the form

$$\begin{aligned} \hat{A}\Delta\hat{x} &= \hat{r}_1 \\ \hat{A}^T\Delta\hat{y} - \Delta\hat{w} + \Delta\hat{z} &= \hat{r}_3 \\ \hat{X}\Delta\hat{z} + \hat{Z}\Delta\hat{x} &= \hat{r}_4 \\ \Delta\hat{x} + \Delta\hat{s} &= \hat{r}_6 \\ \hat{S}\Delta\hat{w} + \hat{W}\Delta\hat{s} &= \hat{r}_7 \end{aligned} \tag{15}$$

for a right-hand side $(\hat{r}_1, \hat{r}_3, \hat{r}_4, \hat{r}_6, \hat{r}_7)$ that varies on each iteration. Furthermore, the matrices $\hat{X}$, $\hat{Z}$, $\hat{S}$, and $\hat{W}$ are diagonal matrices with the iterates $\hat{x}$, $\hat{z}$, $\hat{s}$, and $\hat{w}$ respectively, on the diagonals.

The system (15) can be reduced to solving

$$(\hat{A}\hat{\Theta}\hat{A}^T)\Delta\hat{y} = r \tag{16}$$

for the vector $\Delta\hat{y}$ given the vector r, where $\hat{\Theta} = (\hat{X}^{-1}\hat{Z} + \hat{S}^{-1}\hat{W})^{-1}$ is a diagonal matrix. In the predictor-corrector algorithm, these equations are solved twice per iteration. Since the matrix $\hat{\Theta}$ changes on each iteration, a new Cholesky factorization $(\hat{A}\hat{\Theta}\hat{A}^T) = LL^T$ must be computed on each iteration. This Cholesky factorization has the property that the nonzero structure of L remains fixed over the course of the interior point method. Hence, before the first iteration a permutation matrix P is computed via an ordering heuristic (such as a minimum-degree ordering) in an attempt to reduce the number of nonzeros in L. Hence, a generic solver would actually solve the linear program

$$\begin{aligned} \text{minimize} \quad & \hat{c}^T\hat{x} \\ \text{subject to} \quad & P\hat{A}\hat{x} = P\hat{b} \\ & \hat{x} + \hat{s} = \hat{u} \\ & \hat{x}, \hat{s} \geq 0 \end{aligned} \tag{17}$$

with the consequence that the resulting solution to the dual of (17) would have to be permuted in order to obtain a solution to the dual (12) of (11).

For a multicommodity flow problem, ordering heuristics will cause the flow conservation constraints (6) and the bundling constraints (7) to be mixed, destroying the block structure in $\hat{A}$. This suggests considering techniques for solving the equations (16) that retain the block structure. It turns out that preserving this block structure allows one to take advantage of parallel processing.

It should be noted that the methods of Fourer and Mehrotra ([6]) and Vanderbei and Carpenter ([15]) can also be used to solve the Newton equations (15). In this case, the matrix that is factorized is of the form

$$\begin{bmatrix} \hat{A} & \\ \hat{\Theta}^{-1} & \hat{A}^T \end{bmatrix} \tag{18}$$

For these algorithms, a pivoting method ([6]) or an ordering heuristic ([15]) is used to solve the system of equations resulting in a permutation of both rows and columns. While these methods are designed to reduce the amount of work necessary to solve (15), there are still difficulties with respect to parallel processing because the block structure is not necessarily preserved by the pivoting/ordering methods.

4. Parallel factorization and solution of Newton equations

For a block-structured linear program with side constraints and, in particular, a multicommodity flow problem, the solution of (15) can be obtained by taking advantage of the block structure of the linear program. These equations are rewritten as

$$A\Delta x^k = r_1^k, \quad k = 1, \ldots, K \tag{19}$$

$$\sum_{k=1}^{K} \Delta x^k + \Delta t^k = r_2 \tag{20}$$

$$A^T \Delta y^k - \Delta v + \Delta z^k - \Delta w^k = r_3^k, \quad k = 1, \ldots, K \tag{21}$$

$$X^k \Delta z^k + Z^k \Delta x^k = r_4^k, \quad k = 1, \ldots, K \tag{22}$$

$$T\Delta v + V\Delta t = r_5 \tag{23}$$

$$\Delta x^k + \Delta s^k = r_6^k, \quad k = 1, \ldots, K \tag{24}$$

$$S^k \Delta w^k + W^k \Delta s^k = r_7^k, \quad k = 1, \ldots, K \tag{25}$$

and are solved with a given right-hand side $(r_1^k, r_2^k, r_3^k, r_4^k, r_5^k, r_6^k, r_7^k)$ to obtain a solution vector $(\Delta x^k, \Delta y^k, \Delta z^k, \Delta w^k, \Delta s^k, \Delta v, \Delta t)$.

The solution can be computed by the following steps:

Step 1: Let $\Theta^k = ((X^k)^{-1}Z^k + (S^k)^{-1}W^k)^{-1}$.

Step 2: Compute the Cholesky factorization $A\Theta^k A^T = L^k(L^k)^T$.

Step 3: Compute the factorization

$$\left(V^{-1}T + (\sum_{k=1}^{K} \Theta^k) - \sum_{k=1}^{K} [\Theta^k A^T((L^k)^T)^{-1}(L^k)^{-1}A\Theta^k]\right) = LL^T$$

Step 4: Compute

$$q^k = ((L^k)^T)^{-1}(L^k)^{-1}(r_1^k - A\Theta^k((X^k)^{-1}r_4^k - r_3^k + (S^k)^{-1}(W^k r_6^k - r_7^k))).$$

Step 5: Compute the vector

$$\Delta v = (L^T)^{-1}L^{-1}\left(V^{-1}r_5 - r_2 + \sum_{k=1}^{K}\Theta^k\left[(X^k)^{-1}r_4^k - r_3^k + (S^k)^{-1}(W^k r_6^k - r_7^k) + A^T q^k\right]\right)$$

Step 6: Compute $\Delta y^k = q^k + ((L^k)^T)^{-1}(L^k)^{-1}A\Theta^k\Delta v$.

Step 7: Compute $\Delta x^k = \Theta^k[(X^k)^{-1}r_4^k - r_3^k + A^T\Delta y^k - \Delta v + (S^k)^{-1}(W^k r_6^k - r_7^k)]$.

Step 8: Compute $\Delta z^k = (X^k)^{-1}(r_4^k - Z^k\Delta x^k)$.

Step 9: Compute $\Delta s^k = r_6^k - \Delta x^k$.

Step 10: Compute $\Delta w^k = (S^k)^{-1}(r_7^k - W^k\Delta s^k)$.

Step 11: Compute $\Delta t = r^2 - \sum_{k=1}^{K}\Delta x^k$.

It should be noted that the above steps do not take advantage of the network structure present in A. Furthermore, if a general block-structured program with side constraints of the form (3) is being solved, only minor changes are made to the above equations to introduce the matrices B^k. In the remainder of this section, methods of using parallel computation for the above procedure are described.

4.1. Parallel factorization

We are interested in using parallel computation to compute the solution. Assume that K processors are available. Steps 1–3 are equivalent to computing a factorization of $(\hat{A}\hat{\Theta}\hat{A}^T)$. Steps 1 and 2 can clearly be done in parallel. In Step 3, each of the matrices $D^k = (\Theta^k - \Theta^k A^T((L^k)^T)^{-1}(L^k)^{-1}A\Theta^k)$, $1 \le k \le K$, can be computed in parallel. These matrices are dense and symmetric and require the storage of $O(d^2)$ nonzeros each, where d is the number of bundled links in the network, i.e., the number of links such that $\bar{u}_j$ is finite. They can be computed efficiently on each processor by computing the vector $p_j^k = ((L^k)^T)^{-1}(L^k)^{-1}A_j\Theta_j^k$ for each arc j, followed by computing column j of D^k as $D_j^k = \Theta_j^k - \Theta^k A^T p_j^k$. Hence, storage for the matrix D^k is needed by each processor. Once the K matrices D^k are computed, their sum can be computed by partitioning the matrices into K parts and using each processor to compute the sum of its individual part. Note that this part of the computation requires that the matrices D^k be communicated to each processor. If a shared-memory multiprocessor is used, this communication does not incur any overhead.

4.2. *A dense parallel Cholesky factorization*

Step 3 also requires the factorization of a dense matrix. Here, parallel computation can again be used. In computing a Cholesky factorization, there are a number of choices of methods. These issues are discussed by Lustig et al. ([11]) with regard to using Cholesky factorizations within the context of interior point methods. An important aspect of these issues for parallel computation is the difference between a "pulling" or "leftward-looking" factorization and a "pushing" or "rightward-looking" factorization. It turns out that the pushing version is more appropriate for parallel computation.

A rightward-looking Cholesky is computed as follows, gives that $E = LL^T$ is to be computed and that $E \in \Re^{d\times d}$. The initial value of L is set to the lower triangular part of the matrix E. The elements of L are computed as follows:

1. **for** $j = 1$ to d **do**
2. $\quad L_{jj} \leftarrow \sqrt{L_{jj}}$
3. $\quad$ **for** $i = j + 1$ to d **do**
4. $\quad\quad L_{ij} \leftarrow L_{ij}/L_{jj}$
5. $\quad$ **end for**
6. $\quad$ **for** $k = j + 1$ to d **do**
7. $\quad\quad$ **for** $i = k$ to d **do**
8. $\quad\quad\quad L_{ik} \leftarrow L_{ik} - L_{jj}^2 L_{kj} L_{ij}$
9. $\quad\quad$ **end for**
10. $\quad$ **end for**
11. **end for**

The elements in column j are computed in steps 2–4 and not modified again. Once they are computed, columns $k > j$ are modified by column j in the loop in steps 7–8. These modifications can be done in parallel. If K processors are available, then each processor can be given $(d - j)/K$ columns to modify. This procedure has the majority of the computation [$O(d^3)$ arithmetic operations total] done in parallel and the computation of each column [$O(d^2)$ arithmetic operations total] done sequentially. The efficiency depends upon the relative values of K and d.

4.3. *Parallel solution*

On each iteration of the predictor-corrector algorithm, (19)–(25) must be solved twice for different right-hand sides r_ℓ, $1 \le \ell \le 7$. Hence, on each iteration, steps 5–11 are executed twice. Each of steps 4 and 6–10 can clearly be done on K processors. For step 5, the vectors

$$\hat{q}^k = \Theta^k \left[(X^k)^{-1} r_4^k - r_3^k + (S^k)^{-1} (W^k r_6^k - r_7^k) + A^T q^k \right], \tag{26}$$

$1 \le k \le K$, can be computed in parallel. The sum $\bar{q} = \sum_{k=1}^{K} \hat{q}^k$ is a vector of length $d < n$. It can be computed in parallel or sequentially depending on the length of d. In the implementation, we chose to use a sequential computation independent of the value of d. Finally, step 5 involves an FTRAN and a BTRAN step, which requires a sophisticated parallel computation to compute v with very little benefit for our application because the size of the dense matrix is relatively small. Hence, we used a straightforward sequential computation. In addition, step 11 also requires the sum of $K + 1$ vectors. Since Δt is a vector of length d, we chose to do this part of the computation sequentially.

From the above discussion, it is clear that most of the procedure to compute the factorization and solution can be done in parallel, using a coarse-grain approach. Sequential computation is used in one step of the method for computing the dense factor L as well as in the computations of the vector $\bar{q}$, Δv, and Δt. The most substantial part of the sequential computation is in the FTRAN and BTRAN steps required in step 5.

4.4. *Other parallel computations*

Each iteration of a primal-dual predictor-corrector interior point method involves the computation of the vectors r_1^k, r_3^k, r_4^k, r_5^k, $1 \le k \le K$. On K processors, these vectors can be computed in parallel. In addition, each iteration also involves a ratio test for the primal step length α_P computed as

$$\begin{aligned} \alpha_P \;=\; 0.9995 \min\Big\{ \min_{1\le k\le K} \Big\{ \min\Big\{ \min_{1\le j\le n} \Big\{ \frac{x_j^k}{-\Delta x_j^k},\ \Delta x_j^k < 0 \Big\}, \\ \min_{1\le j\le n} \Big\{ \frac{s_j^k}{-\Delta s_j^k},\ \Delta s_j^k < 0 \Big\} \Big\} \Big\},\ \min_{1\le j\le d} \Big\{ \frac{t_j}{-\Delta t_j},\ \Delta t_j^k < 0 \Big\} \Big\} \end{aligned} \tag{27}$$

For each value of k, $1 \le k \le K$, the values

$$\min\Big\{ \min_{1\le j\le n} \Big\{ \frac{x_j^k}{-\Delta x_j^k},\ \Delta x_j^k < 0 \Big\},\ \min_{1\le j\le n} \Big\{ \frac{s_j^k}{-\Delta s_j^k},\ \Delta s_j^k < 0 \Big\} \Big\} \tag{28}$$

can be computed on K separate processors with the resulting value of α_P computed on a controlling processor. Similarly, the dual step length α_D can also be computed in parallel when performing the ratio tests on z^k, w^k, and v.

After a ratio test, each of the vectors x^k, s^k, y^k, z^k, and w^k must be updated. Clearly this can be done in parallel. However, the update of the vectors t and v must be done serially.

It should be noted that the methods described above are not necessarily the best methods for achieving parallelization of a primal-dual interior point method for

block-structured linear programs with side constraints. Certainly other avenues exist for using parallel processing in other parts of the procedure. However, because the method spends the majority of time computating a dense Cholesky factorization and the block factorizations, small improvements to parallelization will probably lead to only small differences in the results.

5. Computational results

To test the above ideas, a special version of Mehrotra's primal-dual predictor-corrector algorithm ([12]) as implemented in OB1 ([10]) was created to solve multicommodity flow problems. Minor modifications would be required to allow the implementation to solve general block-structured linear programs. The algorithm is identical to the algorithm described in [10], except for the choice of starting point. For a starting point, the values

$$x_j^k = \| b^k \|_\infty, \quad 1 \le k \le K, \quad 1 \le j \le n \tag{29}$$

$$s_j^k = \| b^k \|_\infty, \quad 1 \le k \le K, \quad 1 \le j \le n \tag{30}$$

$$t = \overline{u} \tag{31}$$

$$y = 0 \tag{32}$$

$$v_j = \max_{1 \le k \le K} \| c^k \|_\infty, \quad 1 \le j \le n \tag{33}$$

$$z_j^k = \begin{cases} 2 \max_{1 \le k \le K} \| c^k \|_\infty + c_j^k & \text{if } c_j^k \ge 0; \\ 2 \max_{1 \le k \le K} \| c^k \|_\infty & \text{if } c_j^k < 0; \end{cases} \quad 1 \le k \le K, \quad 1 \le j \le n \tag{34}$$

$$w_j^k = \begin{cases} 2 \max_{1 \le k \le K} \| c^k \|_\infty - c_j^k & \text{if } c_j^k < 0; \\ 2 \max_{1 \le k \le K} \| c^k \|_\infty & \text{if } c_j^k \ge 0; \end{cases} \quad 1 \le k \le K, \quad 1 \le j \le n \tag{35}$$

were used. Note that for any link that also appears in the bundle constraints (7), it is possible to construct the initial dual solution to satisfy dual feasibility for that link. Furthermore, if u_j^k is infinite, then $s_j^k = 0$, and $w_j^k = 0$, eliminating these variables from the computation. If $\overline{u}_j$ is infinite, then $t_j = 0$ and $v_j^k = 0$, eliminating these variables as well.

This version of the interior point method implementation assumed that the problem was derived from a multicommodity flow problem and, hence, did not need to store explicit nonzeros of $\hat{A}$. To factor the matrices $(A\Theta^k A^T)$, a simple Cholesky factorization using no supernode or dense window strategies was used. Before the problem was solved, a minimum-degree ordering of AA^T was computed in order to reduce fill in the factors L^k. Hence, each of the factors L^k, $1 \le k \le K$ had the same nonzero structure.

The code was written in Fortran-77 on a Sequent Symmetry S81 with 20 Intel 80386 processors with Weitek WTL1167 floating point accelerators. The operating system was DYNIX 3.1.2 and the Sequent ATS 2.1 Fortran compiler was

Table 1. Statistics for Assad data set.

Problem name	Network Statistics				LP Statistics	
	No. of comm. (K)	No. of nodes (m)	No. of links (n)	No. of bund. links (d)	No. of rows	No. of columns
assad1.1k	3	47	98	98	239	294
assad1.2k	5	47	98	98	333	490
assad1.3k	7	47	98	98	427	686
assad1.4k	6	47	98	98	380	588
assad1.5k	10	47	98	98	568	980
assad1.6k	15	47	98	98	803	1,470
assad1.7k	3	47	98	98	239	294
assad1.8k	3	47	98	98	239	294
assad2.1k	4	28	102	102	214	408
assad2.2k	10	28	102	102	382	1,020
assad2.3k	20	28	102	102	662	2,040
assad2.4k	34	28	102	102	1,054	3,468
assad2.5k	10	28	102	102	382	1,020
assad2.6k	14	28	102	102	494	1,428
assad2.7k	20	28	102	102	662	2,040
assad2.8k	24	28	102	102	774	2,448
assad3.1k	4	28	204	204	544	816
assad3.2k	6	85	204	204	714	1,224
assad3.3k	12	85	204	204	1,224	2,448
assad3.4k	18	85	204	204	1,734	3,672
assad3.5k	6	85	204	204	714	1,224
assad3.6k	12	85	204	204	1,224	2,448
assad3.7k	18	85	204	204	1,734	3,672
assad4.1k	6	47	274	274	556	1,644
assad4.2k	10	47	274	274	744	2,740
assad4.3k	14	47	274	274	932	3,836
assad4.4k	20	47	274	274	1,214	5,480
assad4.5k	30	47	274	274	1,684	8,220

used. The parallel directives available in the compiler were used to achieve the coarse-grain parallelization described in Section 4. A maximum of 18 processors were used on any given run. Real memory on the Sequent was limited to 80 megabytes. All of the problems solved in this study were chosen so that the total amount of memory required by the implementation was less than 80 megabytes, reducing the paging activity on the system to a minimum. Times are reported as processor time of the controlling processor as measured by the Sequent function **etime**. This time was found to be very similar to the actual wall-clock time.

5.1. Test problems

The test problems used for this study come from three different sources. The first set comes from test problems of Assad ([1]) and are described in Table 1. The second set comes from test problems used by Farvolden ([5]) and are described in Table 2. The third set are problems from the patient distribution system model of the Military Airlift Command that were used by Carolan et al. ([3]) in their testing of the AT&T KORBX system. This last set is described in Table 3. For each table, the statistics about the multicommodity flow network as well as the size of the equivalent linear programming formulation are given.

Table 2. Statistics for Farvolden data set.

Problem name	Network Statistics				LP Statistics	
	No. of comm. (K)	No. of nodes (m)	No. of links (n)	No. of bund. links (d)	No. of rows	No. of columns
5term.100	5	95	225	46	521	1,125
5term.50	5	95	227	48	523	1,135
5term.0	5	95	229	50	525	1,145
5term	5	95	267	88	563	1,335
10term.100	10	190	491	127	2,027	4,910
10term.50	10	190	498	134	2,034	4,980
10term.0	10	190	507	143	2,043	5,070
10term	10	190	507	146	2,046	5,070
15term.0	15	285	745	202	4,477	11,175
15term	15	285	796	253	4,528	11,940
20term.0	20	380	1,150	428	8,028	23,000

Table 3. Statistics for PDS data set.

Problem name	Network Statistics				LP Statistics	
	No. of comm. (K)	No. of nodes (m)	No. of links (n)	No. of bund. links (d)	No. of rows	No. of columns
pds02	11	252	685	181	2,953	7,535
pds06	11	835	2,605	696	9,881	28,655
pds10	11	1,399	4,433	1,169	16,558	48,763

In a multicommodity flow problem, a commodity can be defined as one of three types. The product specific problem (PSP) corresponds to a product to be shipped (such as different types of freight). The destination specific problem (DSP) corresponds to a specific product from many origins to a single destination (such as freight in a hub-and-spoke network from the spokes to the hub). The origin-destination problem (ODP) corresponds to a specific product with a single origin and a single destination (such as a telephone call). These issues are discussed by Jones et al. ([8]). The pds problems described in Table 3 are PSP problems. The term problems described in Table 2 are DSP problems while the assad problems in Table 1 are ODP problems. The assad problems can be converted to DSP problems by aggregating flows into a destination, reducing the total number of commodities while reducing the opportunity to take full advantage of extra parallel processors. Problems of this type were created by modifying the problem assad3.4k. This problem, entitled DSP.assad3.4k, has six commodities and the same network structure as the problem assad3.4k. The equivalent linear programming formulation has 715 rows and 1,230 columns, which is smaller than the ODP formulation of assad3.4k.

5.2. *Efficiency and speedups*

The effectiveness of a parallel implementation can be evaluated in a number of ways. The relative speedup of an algorithm that uses parallel computation measures how much faster the parallel implementation is as compared to a single-processor implementation. The efficiency of an implementation measures how well the parallel implementation uses the available processors. Ideally, an algorithm using p processors would have a speed up close to p and a corresponding efficiency of close to 100%. To achieve "good parallelism," one desires to balance the computational load among the p processors so that the computation is equally distributed across the multiple processors.

Timings for solving a problem with K commodities were recorded for the code using one processor and K processors. If $K > 18$, then $K/2$ processors were

Table 4. Results for Assad data set.

Problem name	No. of comm. (K)	1-processor time (secs)	No. of proc. (p)	p-processor time (secs)	Speedup	Eff.
assad1.1k	3	29.95	3	12.73	2.4	0.78
assad1.2k	5	40.57	5	11.02	3.7	0.74
assad1.3k	7	50.93	7	10.20	5.0	0.71
assad1.4k	6	41.97	6	9.47	4.4	0.74
assad1.5k	10	73.72	10	10.73	6.9	0.69
assad1.6k	15	94.25	15	9.57	9.9	0.66
assad1.7k	3	30.13	3	12.77	2.4	0.79
assad1.8k	3	29.98	3	12.72	2.4	0.79
assad2.1k	4	28.05	4	9.70	2.9	0.72
assad2.2k	10	54.35	10	8.72	6.2	0.62
assad2.3k	20	103.47	10	14.38	7.2	0.72
assad2.4k	34	182.27	17	16.57	11.0	0.65
assad2.5k	10	49.65	10	7.82	6.4	0.64
assad2.6k	14	63.55	14	7.65	8.3	0.59
assad2.7k	20	94.27	10	13.10	7.2	0.72
assad2.8k	24	132.22	12	15.63	8.5	0.70
assad3.1k	4	265.45	4	80.03	3.3	0.83
assad3.2k	6	292.73	6	61.97	4.7	0.79
assad3.3k	12	485.63	12	54.55	8.9	0.74
assad3.4k	18	702.33	18	55.55	12.6	0.70
assad3.5k	6	268.37	6	56.55	4.7	0.79
assad3.6k	12	483.92	12	54.62	8.9	0.74
assad3.7k	18	809.28	18	63.18	12.8	0.71
assad4.1k	6	413.63	6	87.48	4.7	0.79
assad4.2k	10	502.68	10	68.77	7.3	0.73
assad4.3k	14	591.75	14	61.58	9.6	0.69
assad4.4k	20	727.25	10	93.47	7.8	0.78
assad4.5k	30	1,153.32	15	102.55	11.2	0.75

used in an attempt to achieve effective load balancing. Computational results for the assad, term, and pds problems are presented in Tables 4, 5, and 6, respectively. Note that the problem pds10 was estimate to take over 50 hours of time on one processor. A comparison for this problem is given in Section 5.3.

All times in the tables represent wall-clock times. The speedup is the relative speedup of the algorithm using p processors versus one processor. The efficiencies indicate how well the p processors were utilized by the procedure when using multiple processors.

Table 5. Results for Farvolden data set.

Problem name	No. of comm. (K)	1-processor time (secs)	No. of proc. (p)	p-processor time (secs)	Speedup	Eff.
5term.100	5	40.47	5	9.92	4.1	0.82
5term.50	5	42.53	5	10.40	4.1	0.82
5term.0	5	47.45	5	11.63	4.1	0.82
5term	5	98.52	5	24.32	4.1	0.81
10term.100	10	661.38	10	79.23	8.3	0.83
10term.50	10	712.60	10	82.25	8.7	0.87
10term.0	10	757.08	10	91.33	8.3	0.83
10term	10	852.62	10	102.52	8.3	0.83
15term.0	15	2,645.50	15	206.88	12.8	0.85
15term	15	4,329.00	15	351.32	12.3	0.82
20term.0	20	15,333.98	10	1,791.72	8.6	0.86

Table 6. Results for PDS data set.

Problem name	No. of comm. (K)	1-processor time (secs)	No. of proc. (p)	p-processor time (secs)	Speedup	Eff.
pds02	11	1,206.70	11	139.85	8.6	0.78
pds06	11	40,915.07	11	4,317.93	9.5	0.86
pds10	11	222,469.20[a]	11	22,027.75	10.1[a]	0.92[a]

[a] Estimated as 1.2 times for OB1 time

5.3. *Comparison to other solvers*

To compare the performance of the parallel implementation to another solver, the problems `assad3.4k`, `pds-02`, `pds-06`, and `pds-10` were solved by the primal-dual predictor-corrector interior point method as implemented in OB1 ([10]). This code is a general-purpose linear programming code and takes no specific advantage of the block structure. However, as mentioned in Section 3, the general purpose implementation orders the entire matrix $\hat{A}$, thus, mixing the flow-conservation constraints (6) and the bundling constraints (7). Hence, the amount of work per interior point method iteration is reduced for OB1 as compared to the one-processor version of the parallel code. This is because the factorization $(\hat{A}\hat{\Theta}\hat{A}^T) = \hat{L}\hat{L}^T$ computed in the full interior point method is sparser than the combination of the matrices L^k, P^k and L computed by the method as specialized to block angular problems. Even so, the comparative K processor results for the parallel code and one processor results for OB1 as shown in Table 7 exhibit that the parallel version is still quite efficient as compared to an excellent implementation of an interior point method. It should be noted that the number of interior point iterations for OB1 and the specialized code differed slightly due to differences in the starting point.

Table 7. Results for OB1 on selected problems—1 processor Sequent times.

Problem name	OB1 CPU time (secs)
`assad3.4k`	1,074.68
`pds02`	523.33
`pds06`	28,187.80
`pds10`	185,391.00

Schultz and Meyer ([14]) tested a larger set of the `pds` problems and obtained impressive results for running a fixed number (50) of iterations of their decomposition/barrier method. Results for their algorithm running on a similar Sequent computer using 11 processors are given in Table 8. The performance of their method is clearly better than the parallel implementation of the parallel primal-dual interior point method. Their code takes full advantage of the network structure in the problem but only generates approximate primal optimal solutions as compared to our method that generates primal and dual optimal solutions that agree to eight digits in objective value. Furthermore, they reported that their method is only 35%–40% efficient, while our method makes better use of the parallel processors. For block angular problems where the blocks do

not have network structure, it is not clear how well their method will work. On the other hand, we feel that the performance of our method on general block angular problems will be similar to that reported here and that our method is well suited for supercomputers such as the CRAY-Y/MP.

Table 8. Results for method of Schultz and Meyer on selected problems—Wall-clock time on 11 processors of Sequent.

Problem name	Wall-clock time (secs)
pds02	129
pds06	524
pds10	999

Pinar and Zenios ([13]) have also solved some of the pds problems with their parallel decomposition code. When comparing to a version of OB1 that did not use the predictor-corrector enhancements, they found their parallel and vectorized code to be approximately 15.8 times faster for solving problem pds10 than OB1 running on a single processor of a CRAY-Y/MP. Improvements for OB1 since the comparison made by Pinar and Zenios have increased the speed of OB1 by a factor of 2, so that our method should be competitive with their method. (at least for this one problem instance). In addition, Pinar and Zenois report that load-balancing problems occur in their method because the subproblems solved by their method may not be of an equal degree of difficulty. The method described in this paper does not suffer from such effects.

It should be mentioned that other researchers ([17], [18]) have been successful in solving nonlinear multicommodity flow problems, and that their methods applied to the problem instances in this paper would probably outperform the parallel implementation of the primal-dual interior point method.

5.4. Comparisons of formulations and number of processors

The problems assad3.4k and DSP.assad3.4k are the same multicommodity flow problem except for the definition of a commodity. The problem assad3.4k has 18 commodities while the problem DSP.assad3.4k has six commodities. Both problems were solved using the parallel implementation on p processors, $1 \leq p \leq 18$. These results are presented graphically in Fig. 1. Note that the problem assad3.4k attains additional efficiencies due to load balancing as the number of processors increases from 8 to 9 and 17 to 18. However, more than six

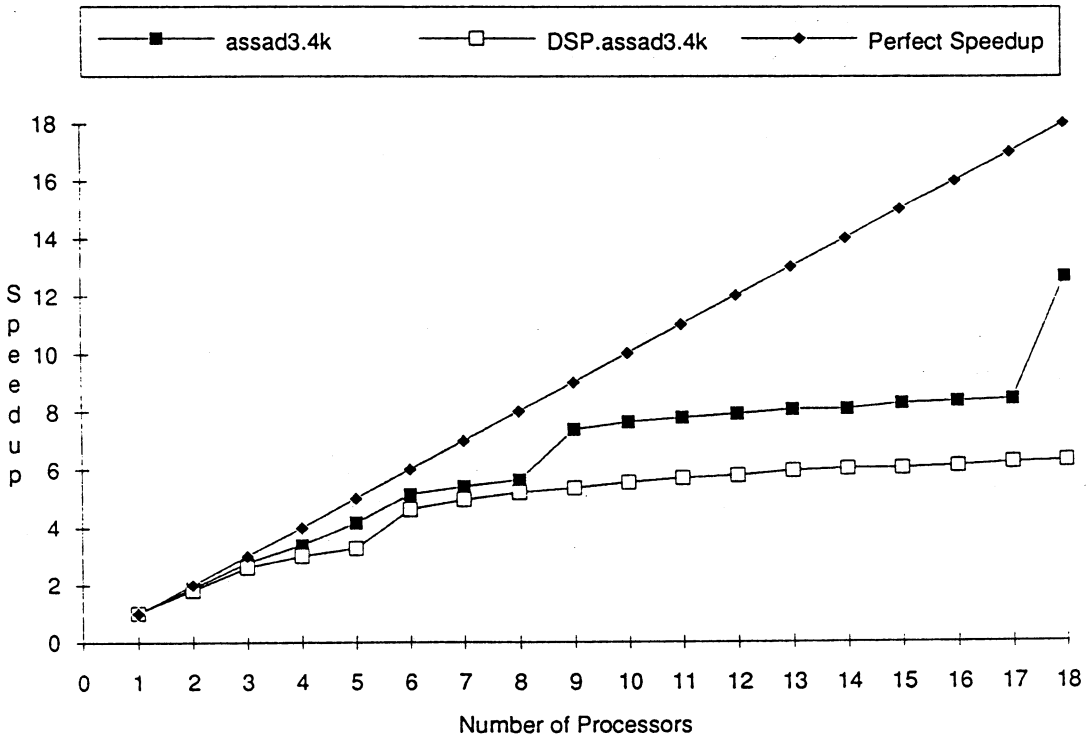

Fig. 1. Speedups for `assad3.4k` (18 commodities) and `DSP.assad3.4k` (6 commodities).

processors for `DSP.assad3.4k` attain a much smaller effect since the additional processors only assist in computing the dense factorization. Furthermore, using 18 processors on the 18-commodity problem `assad3.4k` is 70% efficient while using 18 processors on the six commodity problem `DSP.assad3.4k` is only 35% efficient. Clearly the formulation of the problem has an effect on interpreting the computational results.

6. Conclusions

It is clear that interior point methods can be adapted to specialized problems for parallelization. As compared to using decomposition-type ideas, which suffer from load-balancing problems (see [13]), an interior point method can ignore the combinatorial complexities of the different networks correspondings to the different commodities. As compared to the methods of Schultz and Meyer ([14]), the primal-dual interior point method computes primal and dual optimal solutions to these problems and is easily extended to problems without embedded networks. Furthermore, similar parallel techniques do not seem to apply to implementations of the simplex method.

The computational results presented here indicate future promise for using parallel processing within a primal-dual interior point method. Future research will investigate the possibility of using parallelism to solve other types of problems as well as investigating the possibility of using distributed processing.

Acknowledgments

The authors would like to thank Richard Tapia and Yin Zhang for their useful comments and encouragement during this research. Use of the Sequent Symmetry S81 was provided by the Department of Computer Science at Rice University under NSF grant CDA-8619393. The authors were supported in part by NSF Coop. Agr. No. CCR-8809615 (the first author as a visiting member of the Center for Research in Parallel Computation, Rice University, Houston, TX).

References

1. A.A. Assad, "*Solution techniques for the multicommodity flow problem,*" Master's thesis, Massachusetts Inst. of Tech., Cambridge, MA, 1976.
2. A.A. Assad, "*Multicommodity network flows – A survey,*" *Networks*, vol. 8, pp. 37–91, 1978.
3. W. Carolan, J. Hill, J. Kennington, S. Niemi, and S. Wichmann, "*An empirical evaluation of the KORBX algorithms for military airlift applications,*" *Oper. Res.*, vol. 38, pp. 240–248, 1990.
4. I.C. Choi and D. Goldfarb, "*Solving multicommodity network flow problems by an interior point method,*" in *Large-Scale Numerical Optimization,* (T.F. Coleman and Y. Li, eds.), Society of Industrial and Applied Mathematics (SIAM): Philadelphia, PA, pp. 58–69, 1990.
5. J.M. Farvolden, "*A primal partitioning solution for multicommodity network flow problems,*" PhD thesis, Princeton University, Department of Civil Engineering and Operations Research, Princeton, NJ, 1989.
6. R. Fourer and S. Mehrotra, "*Performance of an augmented system approach for solving least-squares problems in an interior-point method for linear programming,*" in *Committee on Algorithms Newsletter,* Mathematical Programming Society, pp. 26–31, 1991.
7. M.T. Heath, E. Ng, and B.W. Peyton, "*Parallel algorithms for sparse linear systems,*" *SIAM Review*, vol. 33, pp. 420–460, 1991.
8. K.L. Jones, I.J. Lustig, J.M. Farvolden, and W.B. Powell, "*Multicommodity network flows: The impact of formulation on decomposition,*" Princeton University, Department of Civil Engineering and Operations Research, Princeton, NJ, Tech. Report SOR 91-23, 1991.
9. J.L. Kennington, "*A survey of linear cost multicommodity network flows,*" *Oper. Res.*, vol. 26, pp. 209–236, 1978.
10. I.J. Lustig, R.E. Marsten, and D.F. Shanno, "*On implementing Mehrotra's predictor-corrector interior point method for linear programming,*" vol. 2, pp. 435–449, 1992.
11. I.J. Lustig, R.E. Marsten, and D.F. Shanno, "*The interaction of algorithms and architectures for interior point methods,*" in *Advances in Optimization and Parallel Computing,* (P.M. Pardalos, ed.), North-Holland: Amsterdam, pp. 190–205, 1992.
12. S. Mehrotra, "*On the implementation of a (primal-dual) interior point method,*" Northwestern University, Department of Industrial Engineering and Management Sciences, Evanston, IL, Tech. Report 90-03, 1990.
13. M.C. Pinar and S.A. Zenios, "*Parallel decomposition of multicommodity network flows using smooth penalty functions,*" University of Pennsylvania, Decision Sciences Department, The Wharton School,

Philadelphia, PA, Report 90-12-06, 1990.
14. G.L. Schultz and R.R. Meyer, "*An interior point method for block angular optimization,*" *SIAM J. on Optimization*, vol. 1, pp. 583–602, 1991.
15. R.J. Vanderbei and T. Carpenter, "*Symmetric indefinite systems for interior point methods,*" Princeton University, Department of Civil Engineering and Operations Research, Princeton, NJ, Tech. Report SOR 91-7, 1991.
16. A. Vannelli, "*A parallel implementation of an interior point method for linear programming,*" Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario Canada, 1991.
17. H. Nagamochi, M. Fukushima, and T. Ibaraki, "Relaxation methods for the strictly convex multicommodity flow problem with capacity constraints on individual commodities," *Networks,* vol. 20, pp. 409–426, 1990.
18. S.A. Zenios, "On the fine-grain decomposition of multicommodity transportation problems," *SIAM Journal on Optimization,* vol. 1, pp. 643–669, 1991.