Computational experience with a parallel implementation of an interior-point algorithm for multicommodity network flows

Jordi Castro [†]

Statistics and Operations Research Dept.
Universitat Politècnica de Catalunya
Pau Gargallo 5, 08028 Barcelona

Abstract: *A parallel implementation of the specialized interior-point algorithm for multicommodity network flows introduced in [6] is presented. In this algorithm, the positive definite systems of each iteration are solved through a scheme that combines direct factorizations and a preconditioned conjugate gradient (PCG) method. Although this numerical procedure works well in practice, it requires the solution of at least k systems of equations at each iteration of the PCG, k being the number of commodities to be routed through the network.*

In order to reduce the time spent by the PCG method, we propose the application of coarse-grained parallel strategies for computing the k linear systems of equations at each PCG iteration. Since the number of arithmetic operations to be performed for each commodity is the same, the load balancing between processors is guaranteed, which avoids unnecessary delays. An extensive set of computational results on a shared memory machine are presented, using problems of up to 2.5 million variables and 260,000 constraints. For the largest PDS (Patient Distribution System) problems, the efficiency of the parallel implementation developed is about 80%, which confirms that it can be a promising tool for very large and difficult multicommodity instances.

Keywords: interior-point methods, linear programming, multicommodity network flows, parallel computing.

1. Introduction

Multicommodity flows are one of the most challenging problems for linear programming solvers. This is partly due to the large size of these models in real world applications (e.g., routing in telecommunications networks). The need to solve very large multicommodity instances has led to the development of both specialized algorithms and parallel implementations. In this work we introduce a parallel implementation of a specialized multicommodity interior-point algorithm. The implementation has two main features. From the multicommodity point of view, it is not based on a decomposition approach, and thus it does not follow the master-slaves (or coordinator-subtasks) parallel scheme. From the interior-point point of view, unlike other parallel interior-point codes [4, 8, 15], the parallelization is not focused on the Cholesky factorization to be performed at each iteration —though it could be included— but on the parallel solution of smaller subsystems related to the various commodities of the problem.

The block angular structure of the multicommodity problem constraints matrix has led to a number of specialized methods. Among the earlier approaches we could mention primal partitioning, and price and resource directive decomposition (see [2, 14] for a general description). Recent variants of price directive decomposition have successfully applied bundle methods [10] and analytic centers [11]. Multicommodity problems, such as the PDS ones, were also used to test the efficiency of the early general interior-point solvers for linear programming (e.g.,

[†] This work has been supported by Spanish CICYT Project TAP96-1044 and by the European Center for Parallelism of Barcelona (CEPBA).

[1]). Attempts to develop specialized interior-point algorithms for multicommodity flows were presented in [13], [20] and [6], the latter being the most successful. This is the algorithm that will be parallelized in this work.

Parallel approaches for multicommodity problems have also been widely studied in the past. As in the sequential case, the parallel implementations make use of several decomposition strategies, such as bundle methods [7, 17], linear-quadratic penalty terms [19], and, more recently, analytic centers [12]. A discussion of these and other parallel decomposition approaches is presented in [7]. A general description of the parallelization of mathematical programming algorithms can be found in [5] and [21].

The paper is organized as follows. Section 2 presents the formulation of the problem to be solved. Section 3 outlines the specialized interior-point algorithm for multicommodity flows, including a brief description of the general path-following method. Section 4 deals with the parallelization issues of the specialized multicommodity algorithm. Finally, Section 5 gives the computational results obtained with the parallel implementation developed.

2. Problem formulation

In the most general case, the multicommodity network flow problem can be stated as how to obtain the best routing (that which involves the minimum cost) of a set of k commodities through a network of m nodes and n arcs, where the arcs have an individual capacity for each commodity, and a mutual capacity for all the commodities. The resulting problem can be written as

$$\min_{x^{(1)},\dots,x^{(k)}} \quad \sum_{i=1}^{k} {c^{(i)}}^T x^{(i)} \tag{1}$$

subject to

$$\begin{array}{|c|c|c|c|c|}\hline A_N & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & A_N & \dots & \mathbf{0} & \mathbf{0} \\ \hline \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline \mathbf{0} & \mathbf{0} & \dots & A_N & \mathbf{0} \\ \hline \mathbb{1}_n & \mathbb{1}_n & \dots & \mathbb{1}_n & \mathbb{1}_n \\ \hline \end{array} \; \begin{array}{|c|}\hline x^{(1)} \\ \hline x^{(2)} \\ \hline \vdots \\ \hline x^{(k)} \\ \hline s_{mc} \\ \hline \end{array} = \begin{array}{|c|}\hline b^{(1)} \\ \hline b^{(2)} \\ \hline \vdots \\ \hline b^{(k)} \\ \hline b_{mc} \\ \hline \end{array} \tag{2}$$

$$0 \le x^{(i)} \le \overline{x}^{(i)} \quad i = 1,\dots,k \tag{3}$$

$$0 \le s_{mc} \le b_{mc}. \tag{4}$$

Vectors $x^{(i)} \in \mathbb{R}^n$ and $c^{(i)} \in \mathbb{R}^n$ are the flow and cost arrays for each commodity $i,\ i = 1,\dots,k$. $s_{mc} \in \mathbb{R}^n$ denote the slacks of the mutual capacity constraints. $A_N \in \mathbb{R}^{m\times n}$ is the node-arc incidence matrix. We shall assume that A_N is a full row-rank matrix. This can always be guaranteed by removing any of the (redundant) node balance constraints. $b^{(i)} \in \mathbb{R}^m$ is the vector of supplies/demands for commodity i at the nodes of the network. Constraints (3) are simple bounds on the flows, $\overline{x}^{(i)} \in \mathbb{R}^n, i = 1,\dots,k$, being the upper bounds. $b_{mc} \in \mathbb{R}^n$ are the mutual capacities of the arcs for all the commodities. $\mathbb{1}_n$ denotes the $n\times n$ identity matrix.

Note that the multicommodity flow problem can be formulated as a linear programming one with $\tilde{m} = km + n$ constraints and $\tilde{n} = (k+1)n$ variables.

3. Outline of the specialized interior-point algorithm for multicommodity flows

The interior-point algorithm for multicommodity flows introduced in [6] is a specialization of the path-following algorithm for linear programming (see [23] for a thorough description).

Let us consider the following linear programming problem in primal form

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & x + f = \overline{x} \\ & x, f \geq 0, \end{aligned} \tag{5}$$

where $x \in \mathbb{R}^{\tilde{n}}$ and $f \in \mathbb{R}^{\tilde{n}}$ are the primal variables, $\overline{x} \in \mathbb{R}^{\tilde{n}}$ are the upper bounds, $c \in \mathbb{R}^{\tilde{n}}$, $b \in \mathbb{R}^{\tilde{m}}$, and $A \in \mathbb{R}^{\tilde{m} \times \tilde{n}}$ is a full row-rank matrix. The dual of (5) is

$$\begin{aligned} \max \quad & b^T y - \overline{x}^T w \\ \text{subject to} \quad & A^T y + z - w = c \\ & z, w \geq 0, \end{aligned} \tag{6}$$

where $y \in \mathbb{R}^{\tilde{m}}$, $z \in \mathbb{R}^{\tilde{n}}$ and $w \in \mathbb{R}^{\tilde{n}}$ are the dual variables.

Replacing the inequalities in (5) by a logarithmic barrier in the objective function, with parameter μ, it can be seen that the KKT first order optimality conditions of this barrier problem are equivalent to the following system of nonlinear equations:

$$\begin{aligned} r_{xz} \equiv \mu e_{\tilde{n}} - XZe_{\tilde{n}} &= 0 \\ r_{fw} \equiv \mu e_{\tilde{n}} - FWe_{\tilde{n}} &= 0 \\ r_b \equiv b - Ax &= 0 \\ r_c \equiv c - (A^T y + z - w) &= 0 \\ (x, z, w) \geq 0 \quad & \overline{x} \geq x, \end{aligned} \tag{7}$$

where $e_{\tilde{n}}$ is the $\tilde{n}$-dimensional vector of 1's, X, Z, F, and W are diagonal matrices defined as $M \in \mathbb{R}^{\tilde{n} \times \tilde{n}} = \text{diag}(m_1, \ldots, m_{\tilde{n}})$, and the vectors r_* define the left-hand side terms of (7). Note that we did not include the slacks equation $x + f = \overline{x}$ in (7). Instead we replaced the slacks f by $\overline{x} - x$ (thus, $F = \overline{X} - X$ in (7)), reducing by $\tilde{n}$ the number of equations and variables. The solutions of system (7) —considering inequalities as strict inequalities— for different μ values give rise to an arc of strictly feasible points known as the central path. As μ tends to 0, the solutions of (7) converge to that of the original primal and dual problems. A path-following algorithm attempts to follow the central path. Figure 1 shows a damped version of Newton's iteration applied to the nonlinear system (7). We use it for the multicommodity specialization. Note that the matrix Θ computed at step 3 is a positive definite diagonal matrix, because of the way it is formed from positive definite diagonal matrices. A more comprehensive description of the algorithm can be found in [23].

The main computational burden of the algorithm is the solution of the positive definite system

$$(A\Theta A^T) dy = \bar{b} \tag{8}$$

at step 5 of Figure 1 ($\bar{b}$ in (8) denotes the right-hand side $r_b + A\Theta r$ of the system). General interior-point codes attempt to solve (8) through a Cholesky factorization $LL^T = P(A\Theta A^T)P^T$, where P denotes a permutation matrix obtained by some heuristic. However, even for such good permutation matrices as those obtained by the minimum degree ordering or minimum local fill-in heuristics, when A is the multicommodity constraints matrix defined in (2), the Cholesky factorization LL^T turns out to be fairly dense, making this procedure computationally expensive. This is shown in Figure 2, in which the sparsity patterns of both A and $L + L^T$ are depicted for a multicommodity problem with 64 nodes, 524 arcs and 4 commodities, using the state-of-the-art interior-point code BPMPD [16].

```
    **Algorithm path-following**$(A, b, c, \overline{x}, \xi)$
1   Initialize $\xi$, where $\xi = (x^T, f^T, y^T, z^T, w^T)^T$
2   while $\xi$ is not optimal do
3       $\Theta = (X^{-1}Z + F^{-1}W)^{-1}$
4       $r = F^{-1}r_{fw} + r_c - X^{-1}r_{xz}$
        Compute direction:
5           $(A\Theta A^T)dy = r_b + A\Theta r$
6           $dx = \Theta(A^T dy - r)$
7           $dw = F^{-1}(r_{fw} + W dx)$
8           $dz = r_c + dw - A^T dy$
9       Update $\mu$
10      Compute $\alpha$
11      $\xi \leftarrow \xi + \alpha\ d\xi$
12  end_while
```

Figure 1. Path-following algorithm.

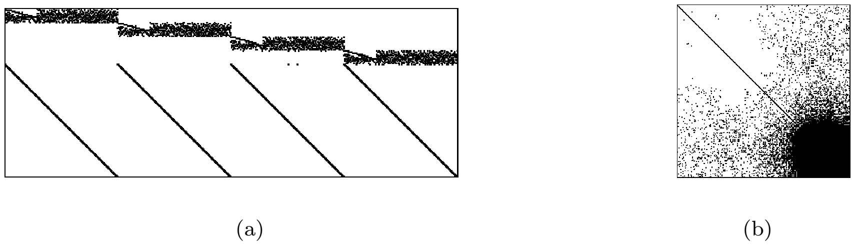


Figure 2. (a) Sparsity pattern of a multicommodity constraint matrix. (b) Sparsity pattern of the factorization of $P(A\Theta A^T)P^T$.

The specialized interior-point method suggested in [6] considers the structure of A presented in (2), and the following partitioning for the diagonal matrix Θ

$$\Theta = \begin{bmatrix} \boxed{\Theta^{(1)}} & & & \\ & \ddots & & \\ & & \boxed{\Theta^{(k)}} & \\ & & & \boxed{\Theta_{mc}} \end{bmatrix}, \tag{9}$$

where $\Theta^{(i)} \in \mathbb{R}^{n \times n}$ and $\Theta_{mc} \in \mathbb{R}^{n \times n}$ are related to the flows $x^{(i)}$ of commodity i and the slacks s_{mc} respectively. It is straightforward to see than the structure of $A\Theta A^T$ is

$$A\Theta A^T = \left[\begin{array}{ccc|c} A_N\Theta^{(1)}A_N^T & \dots & \mathbf{0} & A_N\Theta^{(1)} \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \dots & A_N\Theta^{(k)}A_N^T & A_N\Theta^{(k)} \\ \hline \Theta^{(1)}A_N^T & \dots & \Theta^{(k)}A_N^T & \Theta_{mc} + \sum_{i=1}^{k}\Theta^{(i)} \end{array}\right] = \left[\begin{array}{c|c} B & C \\ \hline C^T & D \end{array}\right], \tag{10}$$

where $B \in \mathbb{R}^{km \times km}$ is the block diagonal matrix

$$B = \text{diag}(A_N\Theta^{(i)}A_N^T,\ i = 1, \dots, k), \tag{11}$$

each block being a square matrix of dimension m, where $C \in \mathbb{R}^{km \times n}$ is defined as

$$C = \begin{bmatrix} \Theta^{(1)} A_N^T & \dots & \Theta^{(k)} A_N^T \end{bmatrix}^T, \tag{12}$$

and where $D \in \mathbb{R}^{n \times n}$ corresponds to the lower diagonal submatrix of $A\Theta A^T$:

$$D = \Theta_{mc} + \sum_{i=1}^{k} \Theta^{(i)}. \tag{13}$$

Since Θ is diagonal and positive definite, it follows that D is also a positive definite diagonal matrix.

Using the above structure of $A\Theta A^T$, and partitioning vectors dy and $\bar{b}$ accordingly, the solution of (8) is reduced to

$$(D - C^T B^{-1} C) dy_2 = (\bar{b}_2 - C^T B^{-1} \bar{b}_1) \equiv \beta_2 \tag{14}$$

$$B dy_1 = (\bar{b}_1 - C dy_2) \equiv \beta_1, \tag{15}$$

where β_2 and β_1 denote the right-hand sides of (14) and (15) respectively. The matrix

$$S = D - C^T B^{-1} C. \tag{16}$$

is known as the Schur complement. To solve (14) and (15) efficiently, we only need to deal with systems involving B and S. Systems with the matrix B can be decomposed into k smaller ones of dimension m with matrices $A_N \Theta^{(i)} A_N^T,\ i = 1, \dots, k$, according to (11).

The system (14) cannot be solved using a direct method (e.g., factorization of the Schur complement), since this would mean forming the matrix S, which is computationally prohibitive. Instead, we suggest using a conjugate gradient method, in virtue of the following result (see [6] for a proof).

Proposition 1. *The Schur complement matrix $S = D - C^T B^{-1} C$ defined in* (16) *is symmetric and positive definite at each iteration of the path-following algorithm.*

The main drawback of the conjugate gradient method is its slow convergence, especially when (5) and (6) are close to their solution point (the Schur complement becomes more ill-conditioned). It seems more reliable to use a preconditioned conjugate gradient (PCG) algorithm. The preconditioner that will be used consists of an approximation of the inverse of S, and it is based on Proposition 2. A proof of this result can be found in [6].

Proposition 2. *The inverse of $S = D - C^T B^{-1} C$ can be computed as*

$$S^{-1} = \Big(\sum_{i=0}^{\infty} (D^{-1} Q)^i \Big) D^{-1}, \tag{17}$$

where

$$Q = C^T B^{-1} C. \tag{18}$$

The preconditioner is then obtained by truncating the power series (17) at the term with index $i = \phi$, say. Clearly, the higher ϕ the better the preconditioning, and the fewer iterations of the PCG will be required. However, each new term in the preconditioner, after the first one, means solving one additional system with matrix B, which increases the cost of each PCG iteration. Therefore, we must balance two objectives: reducing the number of PCG iterations and the number of systems to be solved. Several numerical experiments have shown that the best results are obtained for $\phi = 0$ (in this case the preconditioner is D^{-1}, thus being diagonal) and, in some problems, for $\phi = 1$. The algorithm uses $\phi = 0$ as the default value. The extensive computational experience reported in [6] proved the efficiency of this specialized interior-point algorithm.

4. Parallelization of the algorithm

Computing the direction of the dual variables dy at step 5 of the path-following method in Figure 1 is by far the most costly procedure to be performed by the specialized multicommodity algorithm. Figure 3 summarizes the steps required to compute dy, according to (14) and (15). Looking at Figure 3 we see that all of the steps require either a factorization of B, or a backward and forward substitution with this factorization, or products of vectors with matrices C or C^T. In fact, these will be the only four procedures to be run in parallel, following a coarse-grained scheme. Considering the partitioning of B and C defined in (11) and (12), these four procedures are implemented as follows:

1. Factorization of B. Perform in parallel the k factorizations of $A_N \Theta^{(i)} A_N^T$. Note that the current implementation uses sequential Cholesky solvers. Using parallel implementations of Cholesky decompositions, such as those described in [4, 8, 15], each of the k factorizations could itself be performed in parallel, improving the efficiency of the code.
2. Solution of system $Br = s$, for any $s \in \mathbb{R}^{km}$. Solve in parallel for each diagonal block $A_N \Theta^{(i)} A_N^T$ of B.
3. Computation of $w = Cv$, for any $v \in \mathbb{R}^n$. Using (12), we compute in parallel $w^{(i)} = A_N \Theta^{(i)} v, i = 1, \ldots, k$, so $w^{(i)}$ has the components of w related to commodity i.
4. Computation of $v = C^T w$, for any $w \in \mathbb{R}^{km}$. Using (12), we compute in parallel the temporary vectors $v^{(i)} = \Theta^{(i)} A_N^T w^{(i)}, i = 1, \ldots, k$. We then add the temporary vectors sequentially ($v = \sum_{i=1}^k v^{(i)}$), obtaining v. Due to its low computational cost, the addition of the k $v^{(i)}$ vectors has not been parallelized.

```
     Procedure $A\Theta A^T dy = \bar{b}(A, \Theta, \bar{b}, dy)$
1    Factorize the $k$ blocks of $B$
2    Compute $\beta_2 = \bar{b}_2 - C^T B^{-1} \bar{b}_1$
3    PCG: Solve $(D - C^T B^{-1} C) dy_2 = \beta_2$
3.1      while $dy_2$ is not optimal do
⋮            ⋮
3.i          Compute $w = (D - C^T B^{-1} C) v$
⋮            ⋮
3.n      end while
4    Compute $\beta_1 = \bar{b}_1 - C dy_2$
5    Solve $B dy_1 = \beta_1$
6    Return: $dy = (dy_1^T \;\; dy_2^T)^T$
```

Figure 3. Procedure for computing systems (14) and (15).

From our computational experience, it can be stated that for large problems the above four procedures represent more than 97% of the execution time (see Figure 6 in Section 5). This guarantees that the fraction of the sequential region will be small enough for it not to be a major bottleneck. It should also be noted that, in each of the four parallelized procedures, the number of floating point operations for each commodity (and thus for each processor) will be the same, which guarantees the load balancing between processors and avoids unnecessary delays.

4.1. Parallel programming environment

The parallel implementation of the multicommodity interior-point algorithm was developed on a Silicon Graphics Origin2000 (SGI O2000) server. The SGI O2000 is a shared memory machine, main memory being physically distributed across several processors. In addition, each processor has a first level cache memory of 64Kb (32Kb for instructions, 32Kb for data), and a secondary data cache memory of 4Mb (for both instructions and data).

The main advantage of using a shared memory machine such as the SGI O2000 is the ease with which an existing sequential code can be ported, in comparison with distributed parallel environments. The latter require the use of one of the message passing communication standards (e.g., MPI or PVM), whereas the SGI O2000 provides a more user-friendly system based on including special directives in sequential C or Fortran codes [22]. These directives, usually located at the beginning of loops, create different threads of execution that will run in parallel different sections of the iterative region. Moreover, unlike distributed systems, which force the programmer to allocate data structures between processors and to keep communication low, the parallel environment of the SGI O2000 automatically attempts to perform these tasks. The default data distribution provided, however, can result in an excessive number of cache misses and page faults from the local memory of each processor, the performance of the parallel executions thus being severely impaired. Although advanced directives enable this feature to be controlled, the computational results presented in Section 5 were obtained with the default data distribution across processors provided by the system. This default distribution was also used in [4]. Further details about the use of the parallel directives of the SGI O2000 can be found in [22].

4.2. Performance measures

The performance measures presented below will be considered in Section 5 when reporting the computational results obtained. All of these performance measures are widely used in the field of parallel computing [5]. Considering a particular parallel implementation of an algorithm, we will denote the execution time obtained with p processors by T_p. The speedup S_p obtained with p processors can thus be defined as

$$S_p = \frac{T_1}{T_p}. \tag{19}$$

The fraction of the total execution time consumed in the sequential version by the parallel region will be denoted by f. Values of f close to 1 guarantee good theoretical speedups, whereas the bottleneck represented by the sequential region increases with $1 - f$. This is summarized by Amdahl's law, which provides a theoretical upper bound $\overline{S_p}$ for the best possible speedup

$$\overline{S_p} = \frac{1}{f/p + (1-f)} \leq \frac{1}{(1-f)}. \tag{20}$$

Finally, we can define the efficiency with p processors as

$$E_p = \frac{S_p}{p} \leq \overline{E_p} = \frac{\overline{S_p}}{p}. \tag{21}$$

The efficiency represents the fraction that a particular processor (of the p available) is usefully employed during the execution of the algorithm. Note than when $f = 1$, we have $\overline{S_p} = p$ and $\overline{E_p} = 1$.

5. Computational results

The sequential code of the algorithm outlined in Section 3 was implemented and named IPM in [6]. The parallel version developed in this work will be denoted as pIPM. It is written mainly in C, with only the Cholesky factorization routines (devised by E. Ng and B. Peyton [18]) coded in Fortran. Both the sequential and parallel versions can be freely obtained for academic purposes from `http://www-eio.upc.es/~jcastro/software.html`. All the runs were carried out on the SGI Origin2000 server located at the European Center for Parallelism of Barcelona (CEPBA), running an IRIX64 6.5 Unix operating system. The main characteristics of the server are shown in Figure 4, as reported by the `hinv` (hardware inventory) command. This computer appears at position 275 of the TOP500 supercomputer sites list [9].

64 250 MHZ IP27 Processors
CPU: MIPS R10000 Processor Chip Revision: 3.4
FPU: MIPS R10010 Floating Point Chip Revision: 0.0
Main memory size: 8192 Mbytes
Instruction cache size: 32 Kbytes
Data cache size: 32 Kbytes
Secondary unified instruction/data cache size: 4 Mbytes

Figure 4. Characteristics of the SGI Origin2000 server used for the executions.

Two sets of multicommodity instances were used for the computational experiments. The first is made up of 18 problems obtained with Ali and Kennington's Mnetgen generator [3]. Table 1 shows the dimensions and optimal solutions of the Mnetgen problems. The parameters used to generate the instances can be found in [10], and can be retrieved from `http://www.di.unipi.it/di/groups/optimize/Data/MMCF.html#MNetGen`. Columns "m", "n", and "k" show the number of nodes, arcs, and commodities. Columns "$\tilde{n}$" and "$\tilde{m}$" give the number of variables and constraints of the linear problem (where $\tilde{n} = (k+1)n$ and $\tilde{m} = km + n$). Finally, column "$c^T x^*$" gives the exact optimal objective function value. For the last two problems no exact objective value has been computed (an approximate solution obtained with IPM is reported in Table 3).

Table 1. Dimensions and optimal solutions of the Mnetgen problems.

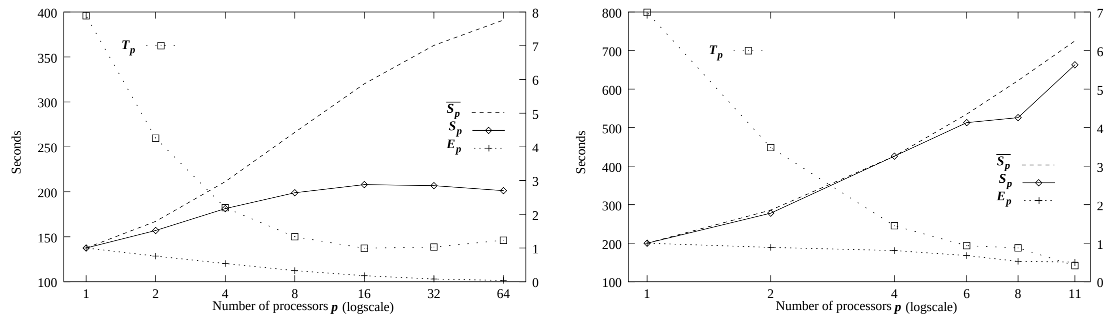
Problem	m	n	k	$\tilde{n}$	$\tilde{m}$	$c^T x^*$
M$_{128-8}$	128	1089	8	9801	2113	1924133.9
M$_{128-16}$	128	1114	16	18938	3162	4145079.4
M$_{128-32}$	128	1141	32	37653	5237	9785961.1
M$_{128-64}$	128	1171	64	76115	9363	19269824.2
M$_{128-128}$	128	1204	128	155316	17588	40143200.8
M$_{256-8}$	256	2165	8	19485	4213	9919483.2
M$_{256-16}$	256	2308	16	39236	6404	20692883.7
M$_{256-32}$	256	2314	32	76362	10506	45671076.1
M$_{256-64}$	256	2320	64	150800	18704	92249381.1
M$_{256-128}$	256	2358	128	304182	35126	190137259.9
M$_{256-256}$	256	2204	256	566428	67740	397882591.3
M$_{512-8}$	512	4373	8	39357	8469	46339269.9
M$_{512-16}$	512	4620	16	78540	12812	96992237.2
M$_{512-32}$	512	4646	32	153318	21030	192941834.8
M$_{512-64}$	512	4768	64	309920	37536	412943158.7
M$_{512-128}$	512	4786	128	617394	70322	828013599.8
M$_{512-256}$	512	4810	256	1236170	135882	—
M$_{512-512}$	512	4786	512	2455218	266930	—

The second set consists of ten of the PDS (Patient Distribution System) problems. These problems arise from a logistic model for evacuating patients from a place of military conflict. They can be retrieved from `http://www.di.unipi.it/di/groups/optimize/Data/MMCF.html-#Pds`. Their dimensions and optimal objective functions can be found in Table 2. The meaning of the columns is the same as in Table 1.

Table 2. Dimensions and optimal solutions of the PDS problems.

Problem	m	n	k	$\tilde{n}$	$\tilde{m}$	$c^T x^*$
PDS1	126	372	11	4464	1758	29083930523.0
PDS10	1399	4792	11	57504	20181	26727094976.0
PDS20	2857	10858	11	130296	42285	23821658640.0
PDS30	4223	16148	11	193776	62601	21385445736.0
PDS40	5652	22059	11	264708	84231	18855198824.0
PDS50	7031	27668	11	332016	105009	16603525724.0
PDS60	8423	33388	11	400656	126041	14265904407.0
PDS70	9750	38396	11	460752	145646	12241162812.0
PDS80	10989	42472	11	509664	163351	11469077462.0
PDS90	12186	46161	11	553932	180207	11087561635.0

Before performing all the executions, we studied in detail the performance of pIPM in two particular instances, $\mathrm{M}_{128-128}$ for the Mnetgen and PDS30 for the PDS problems. The behavior of the code with these two instances turned out to be fairly representative of the general behavior for each data set. Figure 5 shows the results obtained. Each plot gives the execution time T_p (left-hand vertical scale) and the theoretical best speedup $\overline{S_p}$, observed speedup S_p, and observed efficiency E_p (right-hand vertical scale) for different numbers of processors (horizontal axis). Although both problems have a similar f value (0.88 for $\mathrm{M}_{128-128}$, 0.92 for PDS30), pIPM behaved very differently in each case. For $\mathrm{M}_{128-128}$, the gap between S_p and $\overline{S}_p$ increases with the number of processors, whereas for PDS30 the best theoretical speedup is almost always achieved. This fact, together with the different maximum number of processors used in the two problems (64 vs. 11), gives rise to efficiencies of $E_{64} = 0.04$ for $\mathrm{M}_{128-128}$ (the best possible value was $\overline{E_{64}} = 0.12$) and $E_{11} = 0.51$ for PDS30 ($\overline{E_{11}} = 0.56$). It can also be observed that the execution time T_p decreases for PDS30 with p, whereas for $\mathrm{M}_{128-128}$ it remains almost the same for 8, 16, and 32, and slightly increases for 64 processors. This lack of scalability of a shared memory machine when using a large number of processors was also stated in [4]. However, we believe that these results can be improved by exploiting the data distribution between processors, as suggested in Subsection 4.1. This additional work remains to be done.

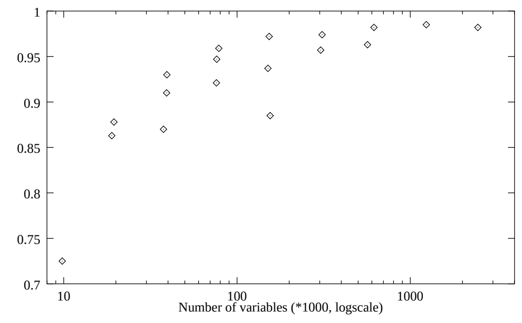


(a) (b)

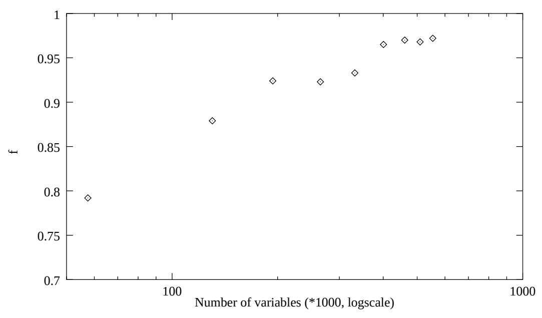
Figure 5. Behavior of the algorithm with the (a) $\mathrm{M}_{128-128}$ problem.
(b) PDS30 problem.

On the basis of the results in Figure 5 we decided to execute the Mnetgen problems with 8, 16, and 32 processors (always guaranteeing $k \geq p$), whereas 6 and 11 where used for the PDS ones. Tables 3 and 4 show the results obtained for the two sets of problems. Column $c^T x^*_{\text{pIPM}}$ gives the optimal solution computed by pIPM. The relative error with respect to the exact optimal solutions of Tables 1 and 2 ranges between 10^{-5} and 10^{-8} for all the cases. Column f gives the f value (fraction represented by the parallel region in the sequential version). Column p is the number of processors used in the execution. T_p denotes the execution time. For $p = 1$, this time means CPU time, as reported by the `times` Unix command, and was obtained by executing the instances on a single processor to reduce context switches. For $p > 1$, T_p denotes wall-clock time, and was obtained by executing pIPM alone on the server to improve the accuracy of the time measures. Columns S_p and E_p give the observed speedups and efficiencies, and, enclosed in parentheses, their best theoretical values $\overline{S_p}$ and $\overline{E_p}$ respectively. Note that, for some of the PDS problems, the observed speedups and efficiencies are greater than their theoretical upper bounds. These superlinear speedups can be explained by: firstly, a lack of accuracy in the measures of both T_1 and T_p, $p > 1$; and secondly, as suggested in [7], a reduction of the number of cache misses in the parallel execution with respect to the sequential one due to the data distribution between processors.

Some of the information in Tables 3 and 4 is summarized in Figures 6, 7, and 8. Figure 6 shows the evolution of the f value with the number of variables of the problem, for both the Mnetgen and PDS instances. Clearly, this value increases with the size of the problem, and for the largest ones it is greater than 0.97, as stated in Section 4 above. Accordingly, the bottleneck associated with the sequential version is consistently reduced for larger and larger instances, which results in a (theoretical) good behavior of the parallel implementation.



(a)



(b)

Figure 6. Fraction f represented by the parallel section in (a) the Mnetgen problems. (b) the PDS problems.

Figures 7 and 8 show the execution times and efficiencies for the M_{512-*} and PDS problems, for different number of processors. For the M_{512-*} problems the best improvements are clearly obtained when moving from 1 to 8 processors. It is also clear that efficiencies tend to decrease with the number of processors, and that, for any p, they remain stable with the size of the problem. These results for the Mnetgen problems are not so good as those observed by parallel implementations of decomposition approaches for multicommodity flows (e.g., [7]). pIPM has a better behavior for the PDS problems. For instance, speedups of about 5 are obtained for the largest problems with $p = 6$, and execution times are almost reduced to half when moving from 6 to 11 processors. Figure 8(b) shows that, unlike in the M_{512-*} problems, efficiencies for 6 and 11 processors are almost the same, and that they become better with the dimension of the problem. The scalability of the code with the PDS problems is not observed, in general, with other parallel implementations of interior-point algorithms using a similar number of processors (e.g., [4], [8], [12]).

Table 3. Results obtained for the Mnetgen problems.

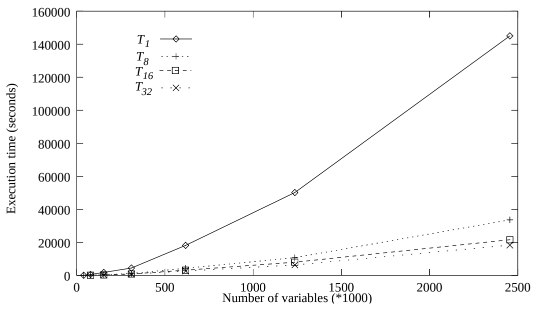
Problem	$c^T x^*_{\mathrm{pIPM}}$	f	p	T_p	S_p $(\overline{S_p})$	E_p $(\overline{E_p})$
M_{128-8}	1924113.4	0.72	1	3.1	1.0 (1.0)	1.00 (1.00)
			8	1.7	1.8 (2.7)	0.22 (0.34)
M_{128-16}	4145089.5	0.86	1	16.2	1.0 (1.0)	1.00 (1.00)
			8	7.3	2.2 (4.1)	0.27 (0.51)
			16	7.7	2.1 (5.2)	0.13 (0.32)
M_{128-32}	9785902.8	0.87	1	36.4	1.0 (1.0)	1.00 (1.00)
			8	15.8	2.3 (4.2)	0.28 (0.52)
			16	14.8	2.5 (5.4)	0.15 (0.33)
			32	18.8	1.9 (6.4)	0.06 (0.19)
M_{128-64}	19269830.9	0.92	1	195.4	1.0 (1.0)	1.00 (1.00)
			8	79.9	2.5 (5.2)	0.30 (0.64)
			16	73.9	2.6 (7.3)	0.16 (0.45)
			32	71.9	2.7 (9.3)	0.08 (0.29)
$\mathrm{M}_{128-128}$	40143266.0	0.89	1	395.9	1.0 (1.0)	1.00 (1.00)
			8	150.1	2.6 (4.4)	0.33 (0.55)
			16	137.3	2.9 (5.9)	0.18 (0.36)
			32	138.6	2.9 (7.0)	0.08 (0.21)
M_{256-8}	9919478.8	0.88	1	19.4	1.0 (1.0)	1.00 (1.00)
			8	7.7	2.5 (4.3)	0.31 (0.54)
M_{256-16}	20692714.5	0.91	1	69.0	1.0 (1.0)	1.00 (1.00)
			8	25.4	2.7 (4.9)	0.33 (0.61)
			16	23.5	2.9 (6.8)	0.18 (0.42)
M_{256-32}	45671345.2	0.95	1	342.0	1.0 (1.0)	1.00 (1.00)
			8	123.5	2.8 (5.8)	0.34 (0.73)
			16	91.1	3.8 (8.9)	0.23 (0.55)
			32	98.0	3.5 (12.1)	0.10 (0.37)
M_{256-64}	92249411.9	0.94	1	586.2	1.0 (1.0)	1.00 (1.00)
			8	199.0	3.0 (5.5)	0.36 (0.69)
			16	146.5	4.0 (8.2)	0.25 (0.51)
			32	143.8	4.1 (10.8)	0.12 (0.33)
$\mathrm{M}_{256-128}$	190138392.4	0.96	1	3352.8	1.0 (1.0)	1.00 (1.00)
			8	627.6	5.3 (6.2)	0.66 (0.76)
			16	558.9	6.0 (9.7)	0.37 (0.60)
			32	511.9	6.5 (13.7)	0.20 (0.42)
$\mathrm{M}_{256-256}$	397883691.5	0.96	1	7486.0	1.0 (1.0)	1.00 (1.00)
			8	1597.0	4.7 (6.3)	0.58 (0.79)
			16	1494.0	5.0 (10.3)	0.31 (0.64)
			32	1274.5	5.9 (14.9)	0.18 (0.46)
M_{512-8}	46338411.5	0.93	1	109.5	1.0 (1.0)	1.00 (1.00)
			8	31.9	3.4 (5.4)	0.42 (0.67)
M_{512-16}	96992142.3	0.96	1	550.9	1.0 (1.0)	1.00 (1.00)
			8	140.9	3.9 (6.2)	0.48 (0.77)
			16	111.0	5.0 (9.9)	0.31 (0.61)
M_{512-32}	192941650.0	0.97	1	1820.3	1.0 (1.0)	1.00 (1.00)
			8	395.1	4.6 (6.7)	0.57 (0.83)
			16	318.4	5.7 (11.3)	0.35 (0.70)
			32	313.7	5.8 (17.1)	0.18 (0.53)

Table 3 (cont.). Results obtained for the Mnetgen problems.

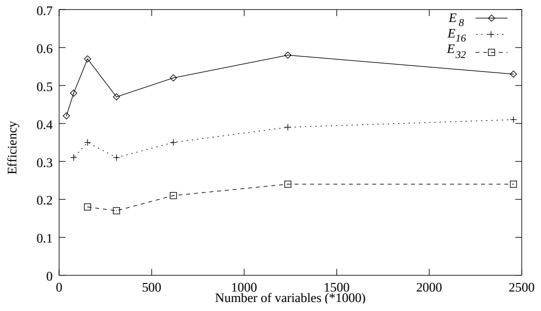
Problem	$c^T x^*_{\mathrm{pIPM}}$	f	p	T_p	S_p $(\overline{S_p})$	E_p $(\overline{E_p})$
M_{512-64}	412943655.4	0.97	1	4473.1	1.0 (1.0)	1.00 (1.00)
			8	1190.8	3.8 (6.8)	0.47 (0.84)
			16	896.2	5.0 (11.5)	0.31 (0.71)
			32	783.5	5.7 (17.7)	0.17 (0.55)
$\mathrm{M}_{512-128}$	828014985.2	0.98	1	18156.2	1.0 (1.0)	1.00 (1.00)
			8	4302.4	4.2 (7.1)	0.52 (0.88)
			16	3168.2	5.7 (12.6)	0.35 (0.78)
			32	2667.0	6.8 (20.5)	0.21 (0.64)
$\mathrm{M}_{512-256}$	1649358223.7	0.98	1	50178.7	1.0 (1.0)	1.00 (1.00)
			8	10745.5	4.7 (7.2)	0.58 (0.90)
			16	7982.6	6.3 (13.1)	0.39 (0.81)
			32	6346.4	7.9 (21.8)	0.24 (0.68)
$\mathrm{M}_{512-512}$	3487594874.0	0.98	1	145018.9	1.0 (1.0)	1.00 (1.00)
			8	33655.5	4.3 (7.1)	0.53 (0.88)
			16	21630.5	6.7 (12.6)	0.41 (0.78)
			32	18312.8	7.9 (20.5)	0.24 (0.64)

Table 4. Results obtained for the PDS problems.

Problem	$c^T x^*_{\mathrm{pIPM}}$	f	p	T_p	S_p $(\overline{S_p})$	E_p $(\overline{E_p})$
PDS1	29083850483.5	0.57	1	0.7	1.0 (1.0)	1.00 (1.00)
			6	0.5	1.4 (1.9)	0.23 (0.31)
			11	0.5	1.5 (2.1)	0.13 (0.18)
PDS10	26726869329.4	0.79	1	46.2	1.0 (1.0)	1.00 (1.00)
			6	19.3	2.4 (2.9)	0.40 (0.49)
			11	16.6	2.8 (3.6)	0.25 (0.32)
PDS20	23820311896.6	0.88	1	234.4	1.0 (1.0)	1.00 (1.00)
			6	70.1	3.3 (3.7)	0.55 (0.62)
			11	55.4	4.2 (5.0)	0.38 (0.45)
PDS30	21385482088.8	0.92	1	799.1	1.0 (1.0)	1.00 (1.00)
			6	193.7	4.1 (4.3)	0.68 (0.72)
			11	141.9	5.6 (6.2)	0.51 (0.56)
PDS40	18852465159.0	0.92	1	1017.7	1.0 (1.0)	1.00 (1.00)
			6	213.4	4.8 (4.3)	0.79 (0.72)
			11	141.3	7.2 (6.2)	0.65 (0.56)
PDS50	16601676244.4	0.93	1	2003.1	1.0 (1.0)	1.00 (1.00)
			6	390.4	5.1 (4.5)	0.85 (0.74)
			11	254.1	7.9 (6.6)	0.71 (0.59)
PDS60	14265869776.2	0.96	1	4924.1	1.0 (1.0)	1.00 (1.00)
			6	917.5	5.4 (5.1)	0.89 (0.85)
			11	551.5	8.9 (8.2)	0.81 (0.74)
PDS70	12240890481.6	0.97	1	7055.5	1.0 (1.0)	1.00 (1.00)
			6	1574.4	4.5 (5.2)	0.74 (0.87)
			11	881.5	8.0 (8.5)	0.72 (0.76)
PDS80	11468486724.2	0.97	1	7737.3	1.0 (1.0)	1.00 (1.00)
			6	1628.6	4.8 (5.2)	0.79 (0.86)
			11	928.4	8.3 (8.3)	0.75 (0.75)
PDS90	11087270971.3	0.97	1	12059.8	1.0 (1.0)	1.00 (1.00)
			6	2293.7	5.3 (5.3)	0.87 (0.87)
			11	1355.9	8.9 (8.6)	0.80 (0.78)

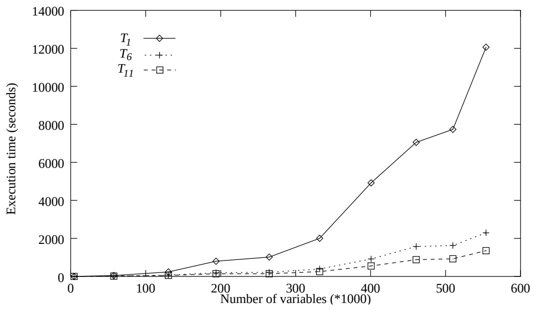


(a)

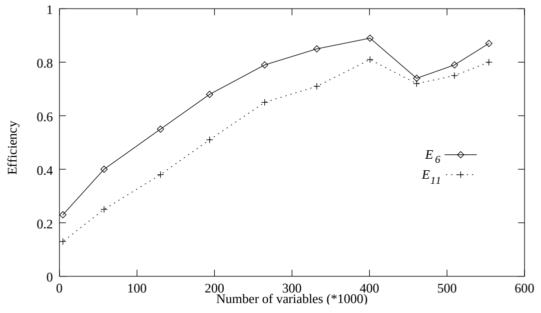


(b)

Figure 7. (a) Execution times for the M_{512-*} problems.
(b) Efficiencies for the M_{512-*} problems.



(a)



(b)

Figure 8. (a) Execution times for the PDS problems.
(b) Efficiencies for the PDS problems.

6. Conclusions and future research

The parallel code pIPM introduced in this work can be an efficient and promising tool for the solution of certain types of large and difficult multicommodity problems. We have found that it is especially appropriate for those instances with large networks and few commodities, where a small number of processors is required.

However, it can be improved with many additional refinements, that form part of the further work to be done. Among these we would mention:

- The fraction f of the parallel region should be augmented to guarantee better theoretical speedups. This would mean parallelizing additional routines while keeping overhead costs low.
- It would be worth attempting to use higher order preconditioners (e.g., $\phi > 0$) for the solution of the system with the Schur complement. Although for sequential executions this would reduce the performance of the algorithm, it could augment the fraction f of the parallel region, providing better parallel executions.
- The gap between the observed and theoretical efficiencies for problems with many commodities, such as the Mnetgen ones, should be reduced. This could be attempted by considering and exploiting the data distribution across the various processors. The reduction of the number of cache misses and remote memory access could mean improvements by a factor of two.

References

[1] I. Adler, M.G.C. Resende, and G. Veiga, An implementation of Karmarkar's algorithm for linear programming, *Mathematical Programming*, 44 (1989), pp. 297–335.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[3] A. Ali, and J.L. Kennington, Mnetgen program documentation, Technical Report 77003, Dept. of Ind. Eng. and Operations Research, Southern Methodist University, Dallas, 1977.

[4] E.D. Andersen, and K.D. Andersen, A parallel interior-point algorithm for linear programming on a shared memory machine, CORE Discussion Paper 9808, CORE, Louvain-La-Neuve, Belgium, 1998.

[5] D.P. Bertsekas, and J.N. Tsitsiklis, *Parallel and Distributed Computation*, Prentice-Hall, Englewood Cliffs, 1995.

[6] J. Castro, A specialized interior-point algorithm for multicommodity network flows, *SIAM J. Optim.* (to appear). Available from `http://www-eio.upc.es/~jcastro`.

[7] P. Cappanera, and A. Frangioni, Symmetric and asymmetric parallelization of a cost-decomposition algorithm for multi-commodity flow problems, Technical Report TR-96-36, Dip. di Informatica, Università di Pisa, Italy, 1996.

[8] T.F. Coleman, J. Czyzyk, C. Sun, M. Wagner, and S.J. Wright, pPCx: parallel software for linear programming, in *Proceedings of the Eight SIAM Conference on Parallel Processing in Scientific Computing*, SIAM, March 1997.

[9] J.J. Dongarra, H.W. Meuer, and E. Strohmaier, TOP500 supercomputer sites, Technical Report UT-CS-98-404, Computer Science Dept., University of Tennessee, 1998.

[10] A. Frangioni, and G. Gallo, A bundle type dual-ascent approach to linear multicommodity min cost flow problems, to appear in *INFORMS Journal on Computing* (1999).

[11] J. Gondzio, R. Sarkissian, and J.-P. Vial, Parallel implementation of a central decomposition method for solving large scale planning problems, HEC Technical Report 98.1, 1998.

[12] J.-L. Goffin, J. Gondzio, R. Sarkissian, and J.-P. Vial, Solving nonlinear multicommodity flow problems by the analytic center cutting plane method, *Mathematical Programming*, 76 (1996), pp. 131–154.

[13] A.P. Kamath, N.K. Karmarkar, and K.G. Ramakrishnan, Computational and complexity results for an interior point algorithm on multicommodity flow problems, Technical Report TR-21/93, Dip. di Informatica, Università di Pisa, Italy, 1993, pp. 116-122. Extended abstracts of Netflow'93.

[14] J.L Kennington, and R.V. Helgason, *Algorithms for Network Programming*, Wiley, New York, 1980.

[15] I.J. Lustig and E. Rothberg, Gigaflops in linear programming, *Operations Research Letter*, 18(4) (1996), pp. 157–165.

[16] Cs. Mészáros, *The Efficient Implementation of Interior Point Methods for Linear Programming and their Applications*, Ph.D. Thesis, Eötvös Loránd University of Sciences, 1996.

[17] D. Medhi, Parallel bundle-based decomposition for large-scale structured mathematical programming problems, *Annals of Operations Research*, 22 (1990), pp. 101–127.

[18] E. Ng, and B.W. Peyton, Block sparse Cholesky algorithms on advanced uniprocessor computers, *SIAM J. Sci. Comput.*, 14 (1993), pp. 1034–1056.

[19] M.C. Pinar, and S.A. Zenios, Parallel decomposition of multicommodity network flows using a linear-quadratic penalty algorithm, *ORSA Journal on Computing*, 4(1992), pp. 235–249.

[20] L. Portugal, M.G.C. Resende, G. Veiga, G., and J. Júdice, A truncated interior-point method for the solution of minimum cost flow problems on an undirected multicommodity flow network (in Portuguese), in Proceedings of First Portuguese National Telecommunications Conference, April 1997, pp. 381–384.

[21] J.B. Rosen (editor), Supercomputers and large-scale optimization: algorithms, software, applications, *Annals of Operations Research*, 22, (1990).

[22] Silicon Graphics Inc., *C Language Reference Manual*, 1998.

[23] S. J. Wright, *Primal-Dual Interior-Point Methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.