
ELSEVIER

Parallel Computing 26 (2000) 1889–1908

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Parallel algorithms to solve two-stage stochastic linear programs with robustness constraints ☆

P. Beraldi [a,*], L. Grandinetti [a], R. Musmanno [b], C. Triki [a]

[a] *Dipartimento di Elettronica, Informatica e Sistemistica, Università degli Studi della Calabria, 87030 Rende (CS), Italy*

[b] *Dipartimento di Ingegneria dell'Innovazione, Facoltà di Ingegneria, Università degli Studi di Lecce, 73100 Lecce, Italy*

Received 29 October 1999; received in revised form 13 March 2000; accepted 27 April 2000

Abstract

In this paper we present a parallel method for solving two-stage stochastic linear programs with restricted recourse. The mathematical model considered here can be used to represent several real-world applications, including financial and production planning problems, for which significant changes in the recourse solutions should be avoided because of their difficulty to be implemented. Our parallel method is based on a primal-dual path-following interior point algorithm, and exploits fruitfully the dual block-angular structure of the constraint matrix and the special block structure of the matrices involved in the restricted recourse model. We describe and discuss both message-passing and shared-memory implementations and we present the numerical results collected on the Origin2000. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Stochastic programming; Restricted recourse; Interior point methods; Numa multiprocessor system; PVM; OpenMP

1. Introduction

Uncertainty is pervasive in several real-world applications. For example, in financial planning, uncertainty affects the term structure of interest rates, inflation,

☆ Research partially supported through contract "HPC-Finance" (no. 951139) of the INCO '95 project funded by the Directorate General III (industry) of the European Commision.

* Corresponding author.

E-mail address: beraldi@parcolab.unical.it (P. Beraldi).

PII: S0167-8191(00)00057-0

currency exchange rates, and, thus, it has remarkable effects on the risk and the evaluation of the financial instruments of the investment portfolios. Other typical applications, where ignoring uncertainty can lead to inferior or wrong decisions, are airline scheduling, power management and so forth [5].

Stochastic programming represents a powerful tool to treat problems under uncertainty [5,8,16].

In this paper we focus our attention on a specific class of stochastic programming models: two-stage stochastic linear models with restricted recourse (RRSLP for short) [30].

Before presenting the mathematical formulation of the RRSLP model, we introduce some notation. Let $(\Omega, \Im, P)$ be a discrete probability space and consider $\Omega = \{1, \ldots, N\}$ as the set of scenarios with associated probabilities $\{p_1, \ldots, p_N\}$ such that $\sum_{l=1}^{N} p_l = 1$.

RRSLP models are defined starting from the most fundamental two-stage stochastic linear model (SLP for short) which can be posed in the following deterministic equivalent form:

$$\begin{aligned} (\text{SLP}) \min \quad & c_0^{\mathrm{T}} x + \sum_{l=1}^{N} p_l c_l^{\mathrm{T}} y_l \\ \text{s.t.} \quad & A_0 x = b_0, \\ & T_l x + W_l y_l = h_l, \quad l = 1, \ldots, N, \\ & x \geqslant 0, \quad y_l \geqslant 0, \quad l = 1, \ldots, N. \end{aligned}$$

Here $x \in \mathbb{R}^{n_1}$ is the vector of the first-stage variables, i.e. decisions made before observing the values of uncertain parameters, whereas $y_l \in \mathbb{R}^{n_2},\ l \in \Omega$, is the vector of second-stage variables, i.e. recourse actions taken once a specific realization l of the uncertain parameters has been observed. Furthermore, $A_0 \in \mathbb{R}^{m_1 \times n_1},\ b_0 \in \mathbb{R}^{m_1},\ c_0 \in \mathbb{R}^{n_1}$ represent the first-stage coefficients, while $c_l \in \mathbb{R}^{n_2},\ T_l \in \mathbb{R}^{m_2 \times n_1},\ W_l \in \mathbb{R}^{m_2 \times n_2}$, and $h_l \in \mathbb{R}^{m_2}$ represent, for each scenario $l \in \Omega$, a particular realization of the cost vector and the second-stage constraint matrices, respectively.

The RRSLP model is obtained by adding to SLP the constraint:

$$\sum_{l=1}^{N} p_l \|y_l - \bar{y}\| \leqslant \epsilon, \tag{1}$$

where $\bar{y}$ is the mean recourse vector and $\epsilon > 0$ is the user-defined level of the recourse robustness.

Constraint (1), referred to as *robustness constraint*, is used to control the dispersion of recourse decisions, in such a way to obtain solutions that are not very sensitive to the observed values of the uncertain parameters. This restriction may be required in many real-world applications where a high variance of the recourse solutions reflects difficulty of use. A nonexhaustive list of applications which can benefit from the restricted recourse formulation is given in [19,29].

It is worthwhile noting that RRSLP models are much more difficult to solve than the corresponding SLP ones because of the nonlinearity and nonseparability of the robustness constraint. Furthermore, the number of scenarios considered to fully represent uncertainty is typically very high and thus, well-tailored methods, that can take further advantage of the computational power of parallel systems, are required.

In this paper we present a parallel method specifically designed for the solution of the RRSLP model. The method is based on a primal–dual path-following interior point algorithm, and exploits fruitfully the special structure of the constraint matrix involved in the model.

This is the first proposal of a parallel method for the class of stochastic programming models incorporating robustness constraints: other existing approaches, based on the use of general purpose codes, perform poorly.

The remainder of the paper is organized as follows. In Section 2 we present the relevant aspects related to the solution of RRSLP models. In Section 3 we introduce the parallel method and discuss some implementation issues. Section 4 is devoted to the discussion of the experimental results carried out on the Origin2000. Finally, Section 5 presents concluding remarks.

2. Solution methods for restricted recourse models

As pointed out in Section 1, the main difficulty to face when solving the RRSLP model is the nonlinearity and nonseparability of the robustness constraint. Different approximation schemes have been proposed and analyzed in [30] in order to overcome these difficulties. Here we consider the most efficient scheme in terms of quality of the solution obtained. According to this scheme, we restrict the dispersion of all the second-stage decisions around a common cluster point $z \in \mathbb{R}^{n_2}$ of the recourse space. Thus, constraint (1) is replaced by its linearized counterpart defined as:

$$-\frac{\lambda}{2}d \leqslant y_l - z \leqslant \frac{\lambda}{2}d, \quad l = 1, \ldots, N.$$

Here $\lambda \in (0,1)$ is a prespecified parameter, whereas the vector $d \in \mathbb{R}^{n_2}$ contains the coordinates of the smallest hypercube including all the recourse decisions. It is computed according to the following formula:

$$d = \delta \mathbf{1} \quad \text{and} \quad \delta = \max_{i=1,\ldots,n_2} \{u_i - l_i\},$$

where $\mathbf{1}$ is the n_2-dimensional vector of ones, and u_i and l_i represent the ith component of the virtual upper and lower bounds computed as the maximum and minimum components of the recourse vector across the scenarios, respectively.

The solution of the RRSLP problem can be carried out by using an iterative scheme. Given a specified tolerance $\epsilon > 0$, the linearized model is iteratively solved for different values of d. At each iteration, one of the two cases can occur:

- the current RRSLP problem is infeasible, that is the degree of restriction is too aggressive and we stop;

- an optimal solution is found, and tighter degree of robustness restriction can be imposed; in this case we check if constraint (1) is satisfied by the current setting; if so we stop, otherwise we compute the vector d and the process is repeated.

Fig. 1 shows the effect of the restricted recourse procedure on a typical test problem (see Table 1). The point corresponding to the highest dispersion value (without any restriction imposed) represents the solution of the SLP model. As we impose further restriction on the solution, an increase of the objective function value is recorded (up to 7.7% over the starting value). The figure shows, however, that it is possible to achieve considerable decrease in the dispersion of the recourse decisions without any increase in the objective value during the first iterations. The effect of restriction on other test problems has been analyzed and discussed in [3].

All the methods proposed in literature for the solution of SLP problems exploit the special block angular structure of the constraint matrix. In the RRSLP model this structure cannot be recognized anymore unless a specific decomposition scheme is applied. By properly reorganizing the constraint matrix, once introduced slack variables, and by setting

$$\bar{A}_0 = (0A_0),$$

and, for each scenario $l \in \Omega$,

$$\bar{T}_l = \begin{pmatrix} 0 & T_l \\ -I & 0 \end{pmatrix} \quad \text{and} \quad \bar{W}_l = \begin{pmatrix} W_l & 0 \\ I & -I \end{pmatrix},$$

we regain the dual block angular structure of the constraint matrix $\bar{A}$ of the RRSLP model.

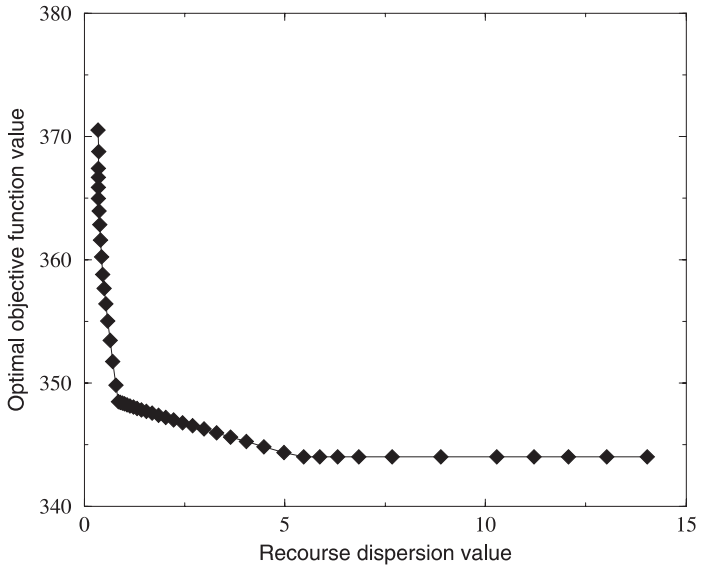

Fig. 1. Effect of the restricted recourse on problem `sctapl` (64 scenarios).

Table 1
Characteristics of the test problems

Problem	SLP model				RRSLP model			
	First stage		Second stage		First stage		Second stage	
	Rows	Columns	Rows	Columns	Rows	Columns	Rows	Columns
scagr7	15	20	38	40	15	60	78	80
scsd8	10	70	20	140	10	210	160	280
sctap1	30	48	60	96	30	144	156	192
scrs8	28	37	28	38	28	75	66	76

To solve the linearized RRSLP model, we have designed and implemented a primal–dual path-following interior point algorithm (PF). The computation of the dual search direction Δt is performed through the solution of a symmetric linearized system called *Newton equation*:

$$M\Delta t = \psi. \tag{2}$$

Here ψ is a given coefficient vector, and $M = \bar{A}D\bar{A}^{\mathrm{T}}$ ($D \in \mathbb{R}^{\bar{n}\times\bar{n}}$ is a diagonal positive definite matrix depending on the current recourse solution).

The computation of the dual step represents, as in any interior point algorithm, the bottleneck of the entire solution procedure (up to 90% of the total time). Furthermore, matrix M is much denser than the original matrix $\bar{A}$ and thus specific approaches have to be applied in order to overcome these difficulties (interested readers are referred, for example, to [2,4]).

We have implemented the strategy proposed by Birge and Qi [6] which is based on the generalized Shermann–Morrison–Woodbury formula. The idea is to decompose the matrix M into the sum of a matrix S and the product of two similar matrices $\bar{U}$ and $\bar{V}$. More specifically, the matrix $S = \mathrm{diag}\{S_0, S_1, \ldots, S_N\}$, where $S_0 = I_{m_1}$ and, for each scenario $l \in \Omega$, $S_l = \bar{W}_l D_l \bar{W}_l^{\mathrm{T}} \in \mathbb{R}^{(m_2+n_2)\times(m_2+n_2)}$, whereas the matrices $\bar{U}$ and $\bar{V}$ are defined as:

$$\bar{U} = \begin{pmatrix} \bar{A}_0 D_0 & I \\ \bar{T}_1 D_0 & 0 \\ \vdots & \vdots \\ \bar{T}_N D_0 & 0 \end{pmatrix} \quad \text{and} \quad \bar{V} = \begin{pmatrix} \bar{A}_0 D_0 & -I \\ \bar{T}_1 D_0 & 0 \\ \vdots & \vdots \\ \bar{T}_N D_0 & 0. \end{pmatrix}.$$

By using the Shermann–Morrison–Woodbury formula and the above matrices, the inverse of M can be computed as:

$$M^{-1} = (S + \bar{U}\bar{V}^{\mathrm{T}})^{-1} = S^{-1} - S^{-1}\bar{U}G^{-1}\bar{V}^{\mathrm{T}}S^{-1},$$

where

$$G = \begin{pmatrix} H & \bar{A}_0^{\mathrm{T}} \\ -\bar{A}_0 & 0 \end{pmatrix} \quad \text{and} \quad H = D_0^{-2} + \bar{A}_0^{\mathrm{T}}\bar{A}_0 + \sum_{l=1}^{N} \bar{T}_l^{\mathrm{T}} S_l^{-1} \bar{T}_l.$$

In what follows we describe how the computation of the dual step can be efficiently carried out in the case of the RRSLP model. A more detailed description is reported in [3]. Here, for the sake of completeness, we present only the basic steps that will be useful in the description of the parallel implementation.

First of all, we point out that the computation of Δt is not carried out by explicity computing M^{-1}, but as the difference of two vectors $\Delta t = p - r$. Here p is the solution of

$$Sp = \psi, \tag{3}$$

whereas r is obtained by solving

$$Gq = \bar{V}^{\mathrm{T}} p, \quad \text{and then } Sr = \bar{U}q.$$

By exploiting the block-diagonal structure of S, we can explicitly rewrite system (3) for each scenario and solve for $p'_l \in \mathbb{R}^{m_2}$ and $p''_l \in \mathbb{R}^{n_2}$, subvectors of p, the following set of systems:

$$S'_l p'_l = \psi'_l - S''_l \psi''_l, \tag{4}$$

where the matrices $S'_l \in \mathbb{R}^{m_2 \times m_2}$ and $S''_l \in \mathbb{R}^{m_2 \times n_2}$ are defined as:

$$S'_l = W_l D'_l W_l^{\mathrm{T}} - W_l D'_l (D'_l + D''_l)^{-1} D'_l W_l^{\mathrm{T}}, \tag{5}$$

$$S''_l = W_l D'_l (D'_l + D''_l)^{-1}, \tag{6}$$

and

$$p''_l = (D'_l + D''_l)^{-1} \psi''_l - S'''_l p'_l \tag{7}$$

with $S'''_l \in \mathbb{R}^{n_2 \times m_2}$ given by:

$$S'''_l = (D'_l + D''_l)^{-1} D'_l W_l^{\mathrm{T}}, \tag{8}$$

where

$$\psi_l = \begin{pmatrix} \psi'_l \\ \psi''_l \end{pmatrix}$$

and $D'_l, D''_l \in \mathbb{R}^{n_2 \times n_2}$ are submatrices of D_l such that $D_l = \mathrm{diag}\{D'_l, D''_l\}$.

Submatrices S'_l, S''_l, and S'''_l are also used to compute $\bar{T}_l^{\mathrm{T}} S_l^{-1} \bar{T}_l$. These components are then summed up, for each scenario $l \in \Omega$, and added to $D_0^{-2} + \bar{A}_0^{\mathrm{T}} \bar{A}_0$ to form H.

The solution of $Sr = \bar{U}q$ can be efficiently carried out by exploiting the block structure of G. Let

$$V^{\mathrm{T}} p = \begin{pmatrix} \pi' \\ \pi'' \end{pmatrix},$$

then $q_1 \in \mathbb{R}^{(n_1+n_2)}$ and $q_2 \in \mathbb{R}^{m_1}$, subvectors of q, can be determined as:

$$q_1 = H^{-1}(\pi' - \bar{A}_0^{\mathrm{T}} q_2), \tag{9}$$

$$q_2 = -L^{-1}(\pi'' + \bar{A}_0 H^{-1} \pi'), \tag{10}$$

where $L = -\bar{A}_0 H^{-1} \bar{A}_0^{\mathrm{T}}$.

Once q is known, r is computed component-wise by solving the system $S_l r_l = \bar{T}_l q_1$. Here, the special structure of S_l can be again fruitfully used to find $r'_l \in \mathbb{R}^{m_2}$ and $r''_l \in \mathbb{R}^{n_2}$, subvectors of r_l, as follows:

$$r'_l = S'^{-1}_l (T_l q''_1 + S''_l q'_1), \tag{11}$$

$$r''_l = -(D'_l + D''_l) q''_1 - S'''_l r'_l, \tag{12}$$

where $q'_1 \in \mathbb{R}^{n_2}$, and $q''_1 \in \mathbb{R}^{n_1}$ are subvectors of q_1.

To summarize, we notice that the special structure of the matrices involved in the RRSLP model, can be efficiently exploited in the computation of the dual step Δt. In effect, instead of factorizing the matrix $S_l \in \mathbb{R}^{(m_2+n_2)\times(m_2+n_2)}$ for each scenario $l \in \Omega$, we consider the smaller matrix $S'_l \in \mathbb{R}^{m_2 \times m_2}$. The achievable advantage is particularly remarkable for problems involving a high number of second-stage variables.

3. Parallel implementations of the path following interior point method

As mentioned in Section 2, the major computational cost of the PF algorithm is due to the factorization of M which has to be executed at each iteration in order to determine the dual step. With the aim of reducing the computational time, we propose a parallel implementation of such factorization. The procedure seems to be particularly well-suited to be implemented on parallel computers since the relevant matrix computation can be executed upon independent blocks.

The parallel implementation proposed here is based on the computational model defined by the message-passing paradigm. Our approach implements the *stand-alone* technique: each process executes the same set of instructions, working on different subset of the data domain (SPMD mode), without the supervision of any master process.

Data distribution and load balancing represent two main issues to address in order to design efficient parallel implementations.

Data distribution. Input data for each process are represented by a subset of second-stage coefficients and a copy of first-stage problem data. This distribution allows each process to perform (concurrently) computations involving second-stage data and (redundantly) calculations concerning the first-stage problem. The replication of first-stage data does not require a remarkable extra memory space since they are typically little; on the other hand, it avoids frequent communication required if data were available on a unique process.

Load balancing. Our parallel implementation is scenarios-oriented, i.e. it is performed by exploiting independent blocks of the constraint matrix. A good balancing of the workload could be obtained simply by splitting equally scenarios among the available processors: remaining scenarios could be then assigned arbitrarily to some of the processors. Since the total number of scenarios is typically very high, the slight difference in the number of the scenarios assigned to each processors cannot cause any relevant unbalance of the workload.

For the sake of simplicity, we assume in the sequel that the number of available processors coincides with the number of scenarios.

The parallel scheme of the proposed technique is as follows:

Parallel scheme for the dual step computation

Step 0. The first-stage problem data A_0, D_0, S_0 and ψ_0 are available to all the generated processes. The second-stage problem data W_l, T_l, D_l and ψ_l, corresponding to scenario $l \in \Omega$, are assigned to process l.

Step 1. (Parallel computation of $Sp = \psi$)

All the processes solve $S_0 p_0 = \psi_0$. Process l:

(a) Forms $S'_l, S''_l e S'''_l$, (Eqs. (5), (6) and (8)).

(b) Solves $S'_l p'_l = \psi'_l - S''_l \psi''_l$ and determines p''_l (Eq. (7)).

Step 2. (Parallel computation of $Gq = \bar{V}^{\mathrm{T}} p$)

(a) Process l solves $S'_l (u_l)_j = (S''_l)_j$ to find $(u_l)_j$, $j = 1, \ldots, n_2$, and solves $S'_l (\hat{u}_l)_i = (T_l)_i$ for $(\hat{u}_l)_i$, $i = 1, \ldots, n_1$. Determine the columns of the matrix $S_l^{-1} \bar{T}_l$.

(b) Process l multiplies $(T_l^{\mathrm{T}})(u_l)_i$, $i = 1, \ldots, n_1$ and $(T_l^{\mathrm{T}})(\hat{u}_l)_j$, $j = 1, \ldots, n_2$ to find $\bar{T}_l^{\mathrm{T}} S_l^{-1} \bar{T}_l$. All-to-all communication to determine H and to compute π' and π''.

(c) Process l solves $Hu = \pi'$ to find u and sets $\hat{u} = \pi'' + \bar{A}_0 u$.

(d) Process l forms L by solving $(H) w_i = (\bar{A}_0^{\mathrm{T}})_i$ for w_i, $i = 1, \ldots, m_1$ and by setting $L = -\bar{A}_0 [w_1, \ldots, w_{m_1}]$.

(e) Process l solves $Lq_2 = -\hat{u}$ for q_2, and finds q_1 (Eq. (9)).

Step 3. (Parallel solution of $Sr = \bar{U} q$)

All the processes set $r_0 = \bar{A}_0 q_1 + q_2$. Process l:

(a) Solves $S'_l r'_l = T_l q''_1 + S''_l q'_1$ to find r'_l (Eq. (11)).

(b) Determines r''_l (Eq. (12)).

Step 4. (Parallel solution of $M \Delta t = \psi$)

All the processes compute $\Delta t_0 = p_0 - r_0$.

Process l computes $\Delta t_l = p_l - r_l$.

The proposed algorithm also exploits parallelism for the computation of the components of ψ and the primal direction of the PF algorithm.

Communication issues. In the parallel scheme introduced above, the only communication occurs at Step 2(b) in order to exchange the terms $\bar{T}_l^{\mathrm{T}} S_l^{-1} \bar{T}_l$ to form the matrix H and to compute π' and π''. At the end of the procedure, an additional data communication is needed in order to collect the components of the dual direction vector on one processor.

An alternative parallel scheme for the computation of the dual step based on a master–slave paradigm has been proposed in [15]. In this case, computations involving dense scenarios-coupled matrices are carried out by the master, whereas independent tasks corresponding to scenarios are performed with full parallelism. Within this scheme, communication occurs in three different points. An all-to-one communication at Step 2(b) in order to collect the terms $\bar{T}_l^{\mathrm{T}} S_l^{-1} \bar{T}_l$; a one-to-all communication to broadcast the matrix H and π' and π'' computed by the master

and needed to the slaves to perform the remaining calculation in parallel (Step 2(c)); a final communication to collect the dual step components on the master process.

The proposed stand-alone scheme seems to be more suitable because of the limited communication required compared to the master–slave scheme. Moreover, the redundant calculation on first-stage data (actually not very computationally demanding) does not cause any additional cost since slaves should remain anyway in an idle state waiting for information from master to be able to continue their work.

The parallel scheme introduced above, is also suitable to be implemented by using the standard shared-memory interface OpenMP [21,22]. Indeed, OpenMP is based on a task distribution and communication is replaced by access to the common memory.

In Section 4 we will present the computational results for both message-passing and shared-memory implementations.

4. Computational experiments

This section is devoted to the presentation and the discussion of the parallel path-following method developed for the solution of the RRSLP model. The core of the solution process, that is the computation of the dual step, requires the factorization of both dense (H and L) and sparse matrices ($S'_l,\ l \in \Omega$). To this aim, we have used LAPACK library and Cholesky factorization, respectively. More specifically, we have used a sequential version of the supernodal Cholesky factorization of Ng and Peyton [20] since our code relies on a scenarios-oriented parallelization rather than a functional one.

Roughly speaking, our code (RRSP for short) can be viewed as an iterative process based on outer iterations that progressively reduce the dimension of the hypercube containing the recourse solutions. Within each outer iteration, we solve the restricted recourse linear program by using the PF algorithm (inner iterations). The number of outer iterations depends not only on the considered problem, but also on the choice of the parameter ϵ that traces the cost-robustness tradeoff. For this reason, in order to evaluate the performance of RRSP, we will not consider the number of outer iterations, but the execution time per outer iteration (t_o), the average number of inner iterations (n_i), and the average execution time per inner iteration (t_i).

4.1. Numerical results

The performance of parallel RRSP have been evaluated on a set of test problems selected from the collection of Holmes [14]. The characteristics of the test problems are shown in Tables 1 and 2, where we report, respectively, the size of the first- and second-stage problems and the size of the deterministic equivalent form for an increasing number of scenarios.

All the computational results have been collected on the Origin2000, a SGI supercomputer composed of four nodes, each with 128 Mb of memory. Each node is

Table 2
Characteristics of the deterministic equivalent problems

Problem	Scenarios	SLP model		RRSLP model	
		Constraints	Variables	Constraints	Variables
scagr7.8	8	319	340	639	700
scagr7.16	16	623	660	1263	1340
scagr7.32	32	1231	1300	2511	2620
scagr7.64	64	2447	2580	5007	5180
scagr7.108	108	4119	4340	8439	8700
scagr7.216	216	8223	8660	16863	17340
scagr7.432	432	16431	17300	33711	34620
scagr7.864	864	32847	34580	67407	69180
scsd8.8	8	170	1190	1290	2450
scsd8.16	16	330	2310	2570	4690
scsd8.32	32	650	4550	5130	9170
scsd8.64	64	1290	9030	10250	18130
scsd8.108	108	2170	15190	17290	30450
scsd8.216	216	4330	30310	34570	60690
scsd8.432	432	8650	60550	69130	121170
sctap1.8	8	510	816	1278	1680
sctap1.16	16	990	1584	2526	3216
sctap1.32	32	1950	3120	5022	6288
sctap1.64	64	3870	6192	10014	12432
sctap1.108	108	6510	10416	16878	20880
sctap1.216	216	12990	20784	33726	41616
sctap1.480	480	28830	46128	74910	92304
scrs8.8	8	252	341	556	683
scrs8.16	16	476	645	1084	1291
scrs8.32	32	924	1253	2140	2507
scrs8.64	64	1820	2469	4252	4939
scrs8.128	128	3612	4901	8476	9803
scrs8.256	256	7196	9765	16924	19531
scrs8.512	512	14364	19493	33820	38987

characterized by two R10000 processors clocked at 195 MHz with a cache memory of 4 Mb. The nodes are connected via two routers that control the data flow exchange between nodes, and, in addition, allow to address to the distribute memory as a unique logically shared memory (Numa machine). The operating system is IRIX 6.5 and the compilers used are `cc` and `f77` with the default optimization level. Moreover, we have used the following options `-mips4`, `-rl0000`, `-OPT:roundoff=3`, `-OPT:IEEE_arithmetic=3` in the compilation and linkage phases.

The numerical results of the message-passing version of the parallel code are summarized in Table 3. The library used is PVM 3.3.10, developed and tuned for the Origin2000 at the Oak Ridge National Laboratory. In particular, we have reported the average wall clock time per inner iteration and the relative speed-up values

Table 3
Speed-ups on the Origin2000 – PVM implementation

Problem	Execution time t_i (s)				Speed-up		
	1 proc	2 proc	4 proc	8 proc	2 proc	4 proc	8 proc
scagr7.8	0.24	0.14	0.09	0.07	1.71	2.66	3.42
scagr7.16	f	f	f	f	f	f	f
scagr7.32	0.90	0.52	0.33	0.24	1.73	2.73	3.75
scagr7.64	1.80	1.00	0.63	0.41	1.80	2.86	4.39
scagr7.108	3.22	1.75	1.02	0.70	1.84	3.16	4.60
scagr7.216	7.20	3.32	1.94	1.18	1.87	3.20	5.25
scagr7.432	12.90	6.57	3.88	2.35	1.89	3.22	5.32
scagr7.864	25.23	13.20	7.81	4.38	1.91	3.23	5.76
scsd8.8	1.08	0.69	0.50	0.46	1.57	2.16	2.35
scsd8.16	1.99	1.12	0.90	0.62	1.78	2.21	3.21
scsd8.32	4.02	2.17	1.53	1.01	1.90	2.63	3.98
scsd8.64	7.80	4.15	2.60	1.70	1.88	3.00	4.59
scsd8.108	13.10	7.28	4.35	2.78	1.80	3.01	4.71
scsd8.216	25.90	14.17	8.48	4.66	1.83	3.05	5.56
scsd8.432	52.90	28.30	14.99	7.94	1.87	3.53	6.66
sctap1.8	1.16	0.65	0.44	0.34	1.78	2.63	3.41
sctap1.16	2.30	1.20	0.71	0.51	1.92	3.24	4.51
sctap1.32	4.48	2.33	1.36	0.83	1.92	3.29	5.40
sctap1.64	8.70	4.53	2.63	1.48	1.92	3.34	5.92
sctap1.108	14.80	7.65	4.30	2.36	1.93	3.44	6.19
sctap1.216	29.42	15.22	8.24	4.00	1.93	3.57	7.36
sctap1.480	66.25	34.20	18.35	8.66	1.94	3.61	7.65
scrs8.8	0.23	0.16	0.11	0.08	1.44	2.10	2.88
scrs8.16	0.43	0.27	0.20	0.12	1.59	2.15	3.58
scrs8.32	0.82	0.48	0.35	0.20	1.71	2.34	4.10
scrs8.64	1.60	0.80	0.60	0.36	1.78	2.77	4.44
scrs8.128	3.30	1.75	1.09	0.68	1.89	3.03	4.79
scrs8.256	6.67	3.43	2.10	1.35	1.94	3.18	4.94
scrs8.512	12.94	6.60	4.00	2.52	1.95	3.24	5.13

('f'' indicates failure because of numerical difficulties). The speed-ups for each test problem are also depicted in Figs. 2–5.

In Table 4 we report the numerical results of the OpenMP version of the code. When using SPMD mode in a shared-memory environment, computation can be carried out by exploiting either orphaning or loop-level OpenMP directives. In the first case, N threads are spawned, one for each scenario; once created, they continue working similarly to PVM processes. Loop-level directives seem to be more obvious-to-use since parallelization can be performed across scenarios loops. However, in our case, it is not enough to rely on the automatic parallelization because the compiler fails in performing some loops in parallel and usually successes only with very simple loops. Hence, it is required to check the independencies in each loop and force ''by hand'' the parallelization of those for which the compiler fails.

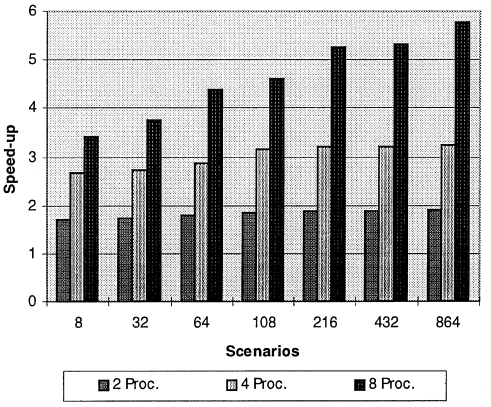

Fig. 2. Speed-ups of test problem `scagr7` on the Origin2000.

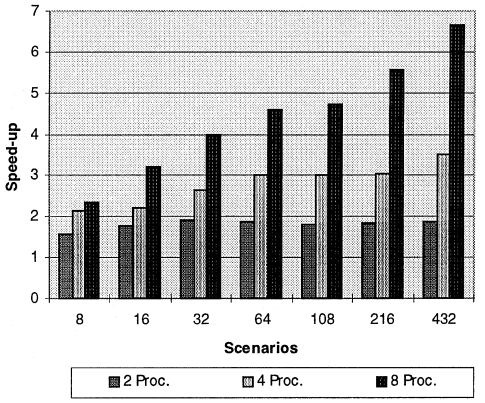

Fig. 3. Speed-ups of test problem `scsd8` on the Origin2000.

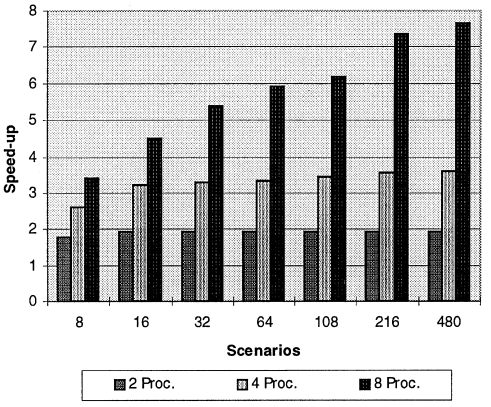

Fig. 4. Speed-ups of test problem `sctapl` on the Origin2000.

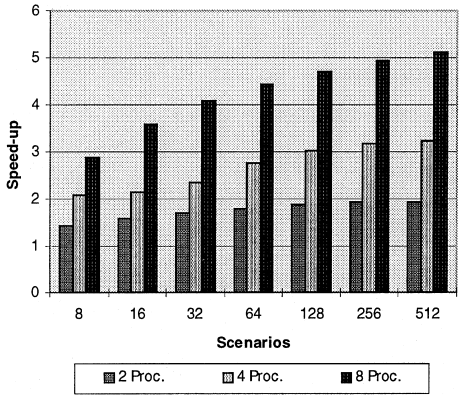

Fig. 5. Speed-ups of test problem `scrs8` on the Origin2000.

Table 4
Speed-ups on the Origin2000 – OpenMP Implementation

Problem	Execution time t_i (s)				Speed-up		
	1 proc	2 proc	4 proc	8 proc	2 proc	4 proc	8 proc
scagr7.8	0.26	0.15	0.11	0.10	1.73	2.36	2.60
scagr7.16	0.48	0.29	0.22	0.16	1.66	2.18	3.00
scagr7.32	0.93	0.54	0.36	0.29	1.72	2.58	3.21
scagr7.64	1.83	1.08	0.69	0.53	1.69	2.65	3.45
scagr7.108	3.07	1.80	1.15	0.87	1.71	2.67	3.53
scagr7.216	6.19	3.56	2.30	1.73	1.74	2.69	3.58
scagr7.432	12.54	7.28	4.67	3.50	1.72	2.69	3.58
scagr7.864	25.10	14.35	9.11	6.59	1.75	2.76	3.81
scsd8.8	1.23	0.77	0.58	0.50	1.60	2.12	2.46
scsd8.16	2.16	1.30	0.88	0.71	1.66	2.45	3.04
scsd8.32	4.03	2.34	1.49	1.13	1.72	2.70	3.57
scsd8.64	7.78	4.43	2.77	2.01	1.76	2.81	3.87
scsd8.108	12.99	7.34	4.53	3.22	1.77	2.87	4.03
scsd8.216	25.65	14.43	8.86	6.23	1.78	2.90	4.12
scsd8.432	51.62	29.31	18.00	12.44	1.76	2.87	4.15
sctap1.8	1.27	0.74	0.52	0.41	1.71	2.44	3.09
sctap1.16	2.33	1.34	0.85	0.62	1.73	2.74	3.75
sctap1.32	4.47	2.52	1.52	1.06	1.77	2.94	4.21
sctap1.64	8.76	4.85	2.90	1.93	1.80	3.02	4.53
sctap1.108	14.70	8.08	4.75	3.19	1.82	3.09	4.60
sctap1.216	29.26	16.10	9.52	6.10	1.81	3.07	4.80
sctap1.480	66.28	36.48	21.50	13.30	1.82	3.08	4.98
scrs8.8	0.25	0.17	0.13	0.12	1.47	1.93	2.08
scrs8.16	0.44	0.27	0.20	0.17	1.62	2.20	2.59
scrs8.32	0.82	0.50	0.33	0.27	1.64	2.48	3.04
scrs8.64	1.58	0.94	0.61	0.46	1.68	2.59	3.43
scrs8.128	3.19	1.82	1.16	0.85	1.75	2.75	3.75
scrs8.256	6.36	3.60	2.27	1.62	1.77	2.80	3.93
scrs8.512	12.76	7.30	4.67	3.14	1.75	2.73	4.06

4.2. Discussion

On the basis of the results reported in Tables 3 and 4, the following remarks can be drawn.

The performance of the parallel code are strongly affected by the problem dimension and the number of processors used. More specifically, better speed-up values are obtained as soon as we increase the number of scenarios. This can be explained by the fact that, as the number of scenarios increases, the sequential part of the code, mainly corresponding to the first-stage problem, becomes insignificant with respect to the computational burden due to the second-stage problem. Furthermore, the availability of more processors leads to a reduction of the time needed to solve an inner iteration, and, consequently, we obtain higher speed-up values. This confirms the efficiency of our parallel implementation and shows that a sophisticated load balancing procedure is not required, since the workload can be efficiently splited among the processors by using even a simple scenarios-based decomposition. We expect that this property is maintained even if the number of scenarios is not exactly a multiple of the number of processors.

Another factor that partially affects speed-ups is the structure of the problem. In general, better speed-up values are obtained for problems with higher number of second-stage variables with respect to the first-stage. This often happens in the case of multiperiod two-stage problems in which second-stage decisions are duplicated for each period of the time horizon. Such observation is confirmed by the speed-up values obtained in the solution of test problems `scsd8` and `sctapl`. However, `scsd8` seems to be very well structured to exploit our factorization procedure even on sequential settings. As a consequence, the execution time on a single processor is very low and the speed-up values achieved are less than those obtained for `sctapl`.

4.3. PVM vs OpenMP on the Origin2000

The analysis of the computational results clearly points out that the PVM implementation outperforms the OpenMP one at least on the Origin2000. This behavior can be explained by the following main issues:

Architecture of the machine. The Origin2000 is a distributed shared memory machine, i.e. there is no common shared memory, but harwdare and software allow the physically distributed memory of the system to be shared. This characteristic does not guarantee a uniform memory access time, but it varies depending on how far away the memory to be accessed in the system. This leads to considerable overhead in the OpenMP implementation.

On the other hand, PVM implementation takes advantage from the tightly connected architecture since communication time is reduced with respect to conventional distributed systems. Furthermore, the PVM version used is, as mentioned above, tuned for the specific architecture of Origin2000 ensuring more efficiency of the message-passing implementation.

Conflicts in memory access. Due to the shared data distributed among the nodes, there exist access memory conflicts in the OpenMP implementation. This is confirmed by a reduction in the efficiency as the number of processors increases. For example, for the test problem `scsd8.432`, the efficiency of the OpenMP implementation varies from 88% on two processors to 51.8% on eight processors with a loss of efficiency of 36.2%. This loss is reduced to only 10.23% in the PVM implementation.

Data placement in OpenMP. Data placement control guarantees that data are located in such a way that local memory accesses are always favored over remote accesses. The lack of data placement directives in the standard OpenMP makes the task of controlling and tuning the memory access painful. The "first touch" scheme cannot guarantee good performance especially if the operating system swaps the tasks among processors during the program execution. Indeed, there is no way to ensure that, after executing parallel loops, the spawner process will be executed on the same processor. This causes additional migration of data among local memories that could be avoided if data placement directives were available.

PVM implementation does not suffer from this problem since task-processor assignements are scheduled at the beginning and not altered during the overall execution.

Ease of implementation. At first glance, message-passing paradigm seems to be more difficult to use: inserting shared-memory directives is much easier than dealing explicitly with data exchange and barriers. However, this is not true if one is looking for good performance. Indeed, tuning and optimizing an OpenMP code is much more difficult than optimizing a PVM code, and, in addition, there are more parameters to tune up.

4.4. Scalability on the Origin2000

In this section we discuss the scalability of parallel RRSP, that is, its ability to maintain the same efficiency when we progressively increase both the dimension of the problem *and* the number of processors. The efficiency of a parallel algorithm on a parallel system is defined as the ratio between the speed-up value and the number of processors used. Analyzing again the results reported in Table 3 (we concentrate here on the PVM results), we can note that parallel RRSP seems to scale well when both the number of scenarios and the number of processors are increased. This observation is made even more clear by examinating the curves plotted in Figs. 6 and 7. They represent the values of the efficiency that parallel RRSP achieves when solving two significant test problems (namely, `scagr7` and `scrs8`). From these figures, it is easy to verify that if we fix the efficiency to a constant value, it is possible to solve different instances of the problem with increasing number of scenarios by using more processors. This means that parallel RRSP is able to solve problems with higher number of scenarios, whenever massively parallel system is available.

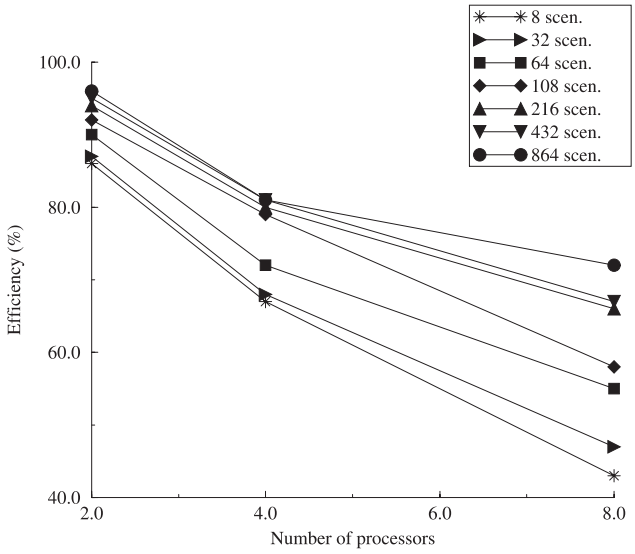

Fig. 6. Scalability for problem `scagr7` on the Origin2000.

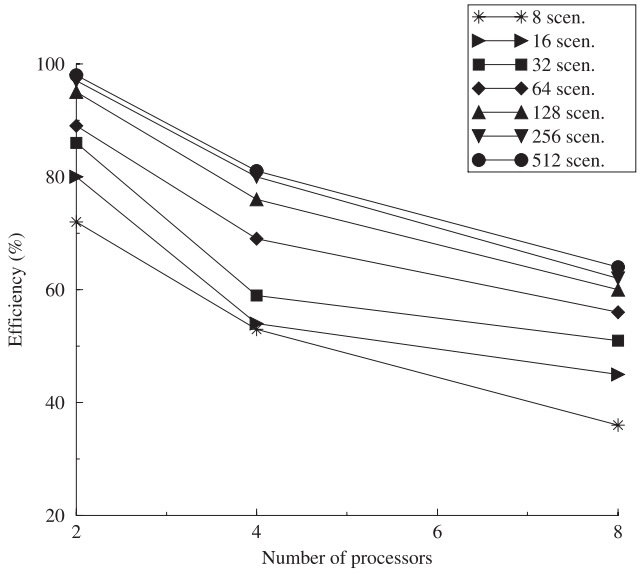

Fig. 7. Scalability for problem `scrs8` on the Origin2000.

4.5. Comparison with the state-of-the-art

On serial settings, the performance of RRSP have been compared with ROBOpT of Yang and Zenios [31] and with LOQO of Vanderbei [24]. ROBOpT is the state-of-the-art PF solver for two-stage stochastic linear programs with recourse and LOQO is the state-of-the-art of linear and quadratic programming general purpose interior point algorithms. Both methods implement the primal-dual path following algorithm with the Predictor–Corrector technique of Mehrotra [18]. They differ in the way of solving Newton equation: LOQO solves the augmented form of Newton system as described in [25–27]. ROBOpT solves the compact form by using Birge-Qi factorization, but it does not exploit the particular block structure of the matrices involved in the restricted recourse model.

Table 5
Comparison between parallel RRSP and LOQO

Problem	LOQO			Parallel RRSP		
	t_o	n_i	t_i	t_o	n_i	t_i
scagr7.8	1.65	21.0	0.07	1.60	22.0	0.07
scagr7.16	14.45	24.7	0.56	f	f	f
scagr7.32	460.14	27.5	1.57	6.50	27.0	0.24
scagr7.64	10.37	29.7	0.29	12.80	31.0	0.41
scagr7.108	21.36	35.0	0.49	23.35	33.0	0.70
scagr7.216	54.10	36.6	1.05	46.18	39.0	1.18
scagr7.432	146.37	40.3	2.27	89.99	38.2	2.35
scagr7.864	436.93	40.3	5.06	203.79	46.3	4.38
scsd8.8	3.30	15.3	0.18	6.97	15.1	0.46
scsd8.16	16.46	15.3	0.95	9.51	15.3	0.62
scsd8.32	26.50	16.5	1.30	16.70	16.5	1.01
scsd8.64	59.80	18.7	2.50	31.79	18.7	1.70
scsd8.108	130.42	24.5	4.20	68.31	24.5	2.78
scsd8.216	282.12	28.2	8.77	109.89	23.5	4.66
scsd8.432	586.58	28.3	18.45	184.00	23.1	7.94
sctap1.8	3.07	15.0	0.17	5.20	15.0	0.34
sctap1.16	28.51	16.3	1.61	8.61	16.7	0.51
sctap1.32	11.01	20.0	0.48	15.92	18.1	0.83
sctap1.64	26.45	25.0	0.90	31.47	21.2	1.48
sctap1.108	50.32	30.5	1.39	62.83	26.2	2.36
sctap1.216	124.55	33.2	2.94	106.19	26.4	4.00
sctap1.432	365.28	37.5	7.10	271.73	31.2	8.66
scrs8.8	0.73	18.0	0.04	1.48	18.0	0.08
scrs8.16	4.37	19.0	0.21	2.30	19.0	0.12
scrs8.32	2.21	20.1	0.10	4.02	20.0	0.20
scrs8.64	5.14	22.0	0.20	7.95	22.0	0.36
scrs8.128	11.75	24.0	0.40	16.36	24.0	0.68
scrs8.256	28.55	26.0	0.83	35.20	26.0	1.35
scrs8.512	75.29	27.6	1.81	69.60	27.5	2.52

The analysis of the computational results, reported in a previous paper [3], has shown that our method is remarkably faster than ROBOpT and competitive with LOQO. This behavior can be explained by the fact that ROBOpT has been designed for solving the general SLP model, and, consequently, without taking advantage from the subsequent sparsity of each sub-block of the constraint matrix of the RRSLP model. As far as LOQO is concerned, we have observed that solving the augmented Newton system leads to more efficiency on sequential machines with respect to the normal system. Consequently, even though LOQO is a general purpose method, it seems to be extremely competive with RRSP, and in some cases even superior. However, the key point is that our method, in spite of LOQO, is particularly well suited for parallel implementation and this advantage is remarkable especially when we have to solve large-scale problems.

Here we analyze empirically this issue by comparing the performance of parallel RRSP vs LOQO. In Table 5 we report, for each test problem, the average execution time of the outer iteration (t_o), the average number of inner iterations (n_i) and the average execution time of the inner iteration (t_i). The results of parallel RRSP refer to the performance of the message-passing implementation on eight processors of the Origin2000.

The results show that parallel RRSP is faster than LOQO as soon as the number of scenarios becomes higher than a threshold value depending on the test problem. The threshold value needed for the outer iteration time differs from that of the inner iteration. In general, higher number of scenarios is needed in order to have an inner iteration of the parallel RRSP outperforming a LOQO inner iteration. This is due to the pre-processing phase executed by LOQO at the beginning of each outer iteration before starting the inner iterative process which requires relatively long time.

5. Conclusions

In this paper we have proposed a parallel method for solving two-stage stochastic linear programming models with restricted recourse. Several applications fall into the considered model. The method proposed in this paper is based on a primal-dual path-following interior point algorithm that takes advantage from the block structure of the constraint matrix. We proposed two different implementations: one based on the message-passing paradigm and the other one using the standard shared-memory interface. Both implementations have been tested on the Origin2000 by using a set of standard test problems. Numerical results have shown the superiority of the message-passing implementation with respect to the shared-memory counterpart. Moreoever, message-passing implementation has shown: (1) significant speed-up values, especially for problems with higher number of scenarios or/and second-stage variables; (2) the superiority of our method with respect to the state-of-the-art codes for large-scale problems; (3) good scalability skills.

These observations candidate our parallel message-passing method to be considered as an efficient tool to solve real-world large-scale applications on parallel Numa machines.

Possible developments towards the direction of parallel implementations of our method include the combination of message-passing standard (MPI) with OpenMP. We believe that such a hybrid approach could be very promising in order to take full advantage from the Numa architecture.

6. For further reading

[1,7,9–13,17,23,28].

References

[1] E.D. Andersen, J. Gondzio, C. Mészáros, X. Xu, Implementation of interior point methods for large scale linear programming, in: T. Terlaky (Ed.), Interior Point Methods in Mathematical Programming, Kluwer Academic Publishers, Dordrecht, MA, 1996, pp. 189–252 (Chapter 6).
[2] K.D. Andersen, A modified Schur complement method for handling dense columns in interior point methods for linear programming, Technical Report, Dash Associates Ltd, Quinton Lodge, Binswood Avenue, Leamington Spa, Warwickshire CV32 5TH, UK, 1995.
[3] P. Beraldi, R. Musmanno, C. Triki, Solving stochastic linear programs with restricted recourse using interior point methods, Comptational Optimization and Applications 15 (3) (2000) 215–234.
[4] J.R. Birge, D.F. Holmes, Efficient solution of two-stage stochastic linear programs using interior point methods, Comput. Optim. Appl. 1 (1992) 245–276.
[5] J.R. Birge, F.V. Louveaux, Introduction to Stochastic Programming, Springer, New York, 1997.
[6] J.R. Birge, L. Qi, Computing block-angular Karmarkar projections with applications to stochastic programming, Management Sci. 34 (12) (1990).
[7] J. Czyzyk, R. Fourer, S. Mehrotra, Parallel solutions of multi-stage stochastic linear programs by interior-point methods, Technical Report, Department of Industrial Engineering and Management Sciences, Northwerstern University, 1994 (Draft).
[8] G.B. Dantzig, Linear programming under uncertainty, Management Sci. 1 (1955) 197–206.
[9] G.B. Dantzig, Planning under uncertainty using parallel computing, Ann. Oper. Res. 14 (1988) 1–16.
[10] G.B. Dantzig, P.W. Glynn, Parallel processors for planning under uncertainty, Ann Oper. Res. 22 (1990) 1–21.
[11] A. De Silva, D. Abramson, A parallel interior point method for stochastic linear programs, Technical Report TR94-4, School of Computing and Information Technology, Griffith University, Nathan Qld 4111, Australia, 1994.
[12] A. DeSilva, D. Abramson, Parallel algorithms for solving stochastic linear programs, in: A. Zomaya (Ed.), Handbook of Parallel and Distributed Computing, McGraw-Hill, New York, 1996, pp. 1097–1115.
[13] R.M. Freund, S. Mizuno, Interior point methods: current status and future directions, Optima 51 (1996) 1–9.
[14] D. Holmes, A collection of stochastic programming problems, Technical Report 94-11, Department of Industrial and Operations Engineering, University of Michigan, 1994.
[15] E.R. Jessup, D. Yang, S.A. Zenios, Parallel factorization of structured matrices arising in stochastic programming, SIAM J. Optim. 4 (4) (1994) 833–846.
[16] P. Kall, S.W. Wallace, Stochastic Programming, Wiley, Chichester, 1994.
[17] I.J. Lustig, R.E. Marsten, D.F. Shanno, Interior point methods for linear programming, ORSA J. Comput. 6 (1994) 1–14.
[18] S. Mehrotra, On the implementation of a primal–dual interior point method, SIAM J. Optim. 2 (4) (1992) 575–601.

[19] J.M. Mulvey, R.J. Vanderbei, S. Zenios, Robust optimization of large-scale systems, Oper. Res. 43 (1995) 264–281.
[20] E.G. Ng, B.W. Peyton, Block sparse Cholesky algorithms on advanced uniprocessor computers, SIAM J. Sci. Comput. 14 (5) (1993) 1034–1056.
[21] OpenMP Fortran Application Program Interface Ver 1.0, October 1997.
[22] OpenMP C and C++ Application Program Interface Ver 1.0, October 1998.
[23] A. Ruszczyński, Interior point methods in stochastic programming, Working paper WP-93-8, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1993.
[24] R.J. Vanderbei, LOQO: An interior point code for quadratic programming, Technical Report SOR-94-15, School of Engineering and Applied Science, Department of Civil Engineering and Operations Research, Princeton University, 1994.
[25] R.J. Vanderbei, Symmetric quasidefinite matrices, SIAM J. Optim. 5 (1) (1995) 100–113.
[26] R.J. Vanderbei, T.J. Carpenter, Symmetric indefinite systems for interior point methods, Math. Programming 58 (1993) 1–32.
[27] R.J. Vanderbei, B. Yang, On the symmetric formulation of interior point method, Technical Report SOR 94-05, Statistics and Operations Research, Princeton University, 1994.
[28] H. Vladimirou, S. Zenios, Scalable parallel computations for large-scale stochastic programming, Annals of Operations Research 90 (1999) 87–129.
[29] H. Vladimirou, S. Zenios, Stochastic programs with restricted recourse: models and solution methods, Technical Report, Department of Public and Business Administration, University of Cyprus, 1998.
[30] H. Vladimirou, S.A. Zenios, Stochastic linear programs with restricted recourse, European J. Oper. Res. 101 (1997) 177–192.
[31] D. Yang, S.A. Zenios, A scalable parallel interior point algorithm for stochastic linear programming and robust optimization, Comput. Optim. Appl. 7 (1997) 143–158.