Parallel Interior Point Solver for Structured Quadratic Programs: Application to Financial Planning Problems

Jacek Gondzio Andreas Grothey

April 15th, 2003
revised December 12th, 2003 and May 27th, 2004
MS-03-001

Parallel Interior-Point Solver for Structured Quadratic Programs: Application to Financial Planning Problems*

Jacek Gondzio† Andreas Grothey‡

School of Mathematics
The University of Edinburgh
Mayfield Road, Edinburgh EH9 3JZ
United Kingdom.

April 15th, 2003, revised December 12th, 2003 and May 27th, 2004

*Supported by the Engineering and Physical Sciences Research Council of UK, EPSRC grant GR/R99683/01.
†Email: J.Gondzio@ed.ac.uk, URL: http://maths.ed.ac.uk/~gondzio/
‡Email: A.Grothey@ed.ac.uk, URL: http://maths.ed.ac.uk/~agr/

Parallel Interior-Point Solver for Structured Quadratic Programs: Application to Financial Planning Problems

Abstract

Many practical large-scale optimization problems are not only sparse, but also display some form of block-structure such as primal or dual block angular structure. Often these structures are nested: each block of the coarse top level structure is block-structured itself. Problems with these characteristics appear frequently in stochastic programming but also in other areas such as telecommunication network modelling.

We present a linear algebra library tailored for problems with such structure that is used inside an interior point solver for convex quadratic programming problems. Due to its object-oriented design it can be used to exploit virtually any nested block structure arising in practical problems, eliminating the need for highly specialised linear algebra modules needing to be written for every type of problem separately. Through a careful implementation we achieve almost automatic parallelisation of the linear algebra.

The efficiency of the approach is illustrated on several problems arising in the financial planning, namely in the asset and liability management. The problems are modelled as multistage decision processes and by nature lead to nested block-structured problems. By taking the variance of the random variables into account the problems become non-separable quadratic programs. A reformulation of the problem is proposed which reduces density of matrices involved and by these means significantly simplifies its solution by an interior point method. The object-oriented parallel solver achieves high efficiency by careful exploitation of the block sparsity of these problems. As a result a problem with over 50 million decision variables is solved in just over 2 hours on a parallel computer with 16 processors. The approach is by nature scalable and the parallel implementation achieves nearly perfect speed-ups on a range of problems.

1 Introduction

We are concerned in this paper with a parallel structure exploiting interior-point solver to tackle convex quadratic programs (QPs). An advantage of interior point methods (IPMs) is a consistent small number of iterations needed to reach the optimal solution of the problem. Indeed, modern IPMs rarely need more than 20-30 iterations to solve a QP, and this number does not increase significantly even for problems with millions of variables. However, a single iteration of interior-point method may be costly because it involves solving large and potentially difficult systems of linear equations. Indeed, the efficiency of interior point methods for quadratic programming critically depends on the implementation of the linear algebra operations.

We deal with the primal-dual interior-point method in this paper. It is widely accepted (Andersen, Gondzio, Mészáros and Xu 1996, Wright 1997) that the primal-dual algorithm is usually faster and more reliable than the pure primal or pure dual method. The primal-dual method is applied to the primal-dual formulation of the quadratic program

Primal		Dual	
min	$c^T x + \frac{1}{2} x^T Q x$	max	$b^T y - \frac{1}{2} x^T Q x$
s.t.	$Ax = b,$	s.t.	$A^T y + s - Qx = c,$
	$x \geq 0;$		y free, $x, s \geq 0,$

where $A \in \mathcal{R}^{m \times n}, Q \in \mathcal{R}^{n \times n}$, $x, s, c \in \mathcal{R}^n$ and $y, b \in \mathcal{R}^m$. The main computational effort of this algorithm consists in the computation of the primal-dual Newton direction. Applying standard transformations (Wright 1997) leads to the following linear system to be solved at each iteration

$$\begin{bmatrix} -Q - \Theta^{-1} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r \\ h \end{bmatrix}, \tag{1}$$

where $\Theta = XS^{-1}$ and X and S are diagonal matrices in $\mathcal{R}^{n \times n}$ with elements of vectors x and s spread across the diagonal, respectively.

In this paper we are interested in the case where A and Q are large, block-sparse matrices; indeed we assume them to be of a nested block-sparse structure. We note that an augmented system matrix such as in (1) whose components A, Q are nested block-structured matrices can be naturally reordered by symmetric row and column permutations (provided the structures of A and Q are compatible), such that the resulting matrix is again a nested block-structured matrix whose elementary blocks are of augmented system type. Thus the structures of A and Q imply in a natural way a structure of the augmented system matrix.

The philosophy pursued in our solver is to replicate the nested block-structure of the augmented system matrix in the linear algebra routines. Nested block-structured matrices can be represented using a tree: the whole matrix is the root of the tree, its block components are the nodes immediately below the root, whereas the final elementary blocks are the leaves of the tree. We associate which each node the structure of the sub-matrix that it represents. Thus we have different types of nodes for each structure that is supported by the linear algebra.

Matrix structures that are supported by our solver are derived types such as block-diagonal, block-bordered diagonal (which includes primal and dual block-angular structures) and rank-corrector (matrices of the form $A + VV^T$ where V has only few columns) as well as elementary (non-structured) types such as dense, sparse, network and projection matrices.

All linear algebra routines are executed following this tree-structure: each node of the tree has associated with it a complete set of linear algebra routines that are appropriate to exploit the structure present at this node of the tree. This is achieved by using object-oriented program design techniques: a node of the tree is represented by an abstract matrix class. Any particular structure represented by the node corresponds to a particular implementation of the abstract matrix class. Whenever a linear algebra routine at a particular node needs to access results from its child-nodes it does so by referring to the abstract matrix class. At a particular node we only need to access the results from computations at a lower level in the tree: their implementation is hidden from the calling node.

Due to this self-referential property of the abstract matrix class we only need an implementation for each *type* of node-structure supported, not for each possible combination of node-structures. This results in a huge gain of flexibility, since virtually any nested block-structured problem can be easily rebuilt as a tree of matrix-classes automatically leading to efficient structure exploitation at each level of the tree.

Block-matrix operations are natural candidates for parallelization. We have therefore implemented all higher-level matrix classes in parallel. The parallelization is coarse-grained and so is well suited to large scale applications.

Further our implementation is effectively providing a linear algebra library for IPMs for QP. It is therefore possible to completely separate the linear algebra part and the IPM logic part of the algorithm.

There are several other object-oriented implementations of IPMs and more general optimization algorithms reported in the literature: OOQP(Gertz and Wright 2003), TAO(Benson, McInnes and Moré 2001), OPT++(Meza 1994) to name but a few. The recent report of Gertz and Wright (Gertz and Wright 2003) provides a good summary of the various ongoing efforts. However we believe that our philosophy is unique. All other approaches use object-oriented concepts to separate the logic of the interior point method from the used data-types and linear algebra implementation. Our approach is targeted at a different level, in that we use object-oriented design to exploit the particular (nested) block-structure of the matrices in question. Our objects are the blocks of the problem matrices. This way we can easily exploit virtually any practically occurring nested block-structure in the linear algebra *without* any more coding needed. Other approaches offer similar advantages by separating the linear algebra from the logic of the IPM, however a complete new linear algebra implementation would need to be supplied for every new type of matrix. It is possible and would be worthwhile to combine the two approaches (i.e. ours with (Gertz and Wright 2003)) to obtain a complete IPM framework exploiting object-oriented concept on every level. However such an integrated work is not the subject of this report.

To illustrate the potential of the new solver, we apply it to solve multistage stochastic quadratic programs arising in financial planning problems, namely in asset liability management. Such problems have drawn a lot of attention in the literature and have often been modelled as multistage stochastic linear programs (Bradley and Crane 1972, Cariño, Kent, Myers, Stacy, Sylvanus, Turner, Watanabe and Ziemba 1994, Consigli and Dempster 1998, Kusy and Ziemba 1986, Mulvey and Vladimirou 1992, Zenios 1995, Ziemba and Mulvey 1998). In this paper we consider an extension in which the risk associated with the later stages is taken into account in the optimization problem. This is achieved by adding the variance term to the original objective of the problem and leads to a QP formulation.

Multistage stochastic linear programs can be solved by the specialized decomposition algorithms (Birge 1985, Gassmann 1990, Gondzio and Kouwenberg 2001, Linderoth and Wright 2003, Ruszczyński 1985) or by the specialized interior point implementations (Birge and Qi 1988, Jessup, Yang and Zenios 1994, Vladimirou and Zenios 1999, Hegland, Osborne and Sun 2002). Recently a number of interesting specialized approaches have been proposed for multistage quadratic programs. Steinbach (Steinbach 2000) exploits the tree-structure of such problems inherited from the scenario tree of the multistage problem to implicitly reorder the computations in the context of interior point method. Blomvall and Lindberg (Blomvall and Lindberg 2002a, Blomvall and Lindberg 2002b) interpret the problem as an optimal control one and derive the special order in which computations in IPMs should be executed. This approach originates from differential equations and the authors call it the Riccati type method. Parpas and Rustem (Parpas and Rustem 2003) extend the nested Benders decomposition to handle stochastic quadratic programming problems and use an IPM-based solver to solve all subproblems. However all these approaches assume a special problem structure that is then exploited in the computations. Extending the problem formulation will be difficult or even impossible. Our approach does not rely on any particular problem structure: we can solve a wide variety of stochastic programming formulations. Further multistage stochastic QPs are only one class of problems that can be efficiently dealt with by the suggested approach.

The approach presented in this paper is an extension of that implemented in OOPS[1] (Gondzio and Sarkissian 2003, Gondzio and Grothey 2003). The main difference is that when linear problems were considered in the earlier version of OOPS the augmented system of linear equations (1) did not contain matrix Q and was always reduced to normal equations

$$A\Theta A^T \Delta y = A\Theta r + h. \tag{2}$$

This was a viable reduction step because matrix Q was not present in the augmented system and Θ was a diagonal matrix. In the context of quadratic programming such a reduction would lead to almost completely dense normal equations $A(Q + \Theta^{-1})^{-1}A^T$ (see (Duff, Erisman and Reid 1987)) and would be likely to incur prohibitively expensive computations (unless Q is a diagonal matrix). A comparison with CPLEX (which uses normal equations) in section 5 will illustrate this point. Hence in this paper and more generally in the QP extension of OOPS we always deal with the augmented system (1).

The extension of OOPS to QPs is possible but not straightforward. For QP one has to deal with two matrices A and Q each of which might display some nested block-structure. It is not clear a-priori how to exploit both these structures. As we will show the augmented system needs to be re-ordered before any exploitable structure becomes apparent.

Another contribution of our paper is the analysis of the influence of the QP problem formulation on the efficiency of the solution process. We propose a nonconvex reformulation of the problem which reduces density of matrices involved and is better suited to the use of IPMs. To the best of our knowledge studying this reformulation from the point-of-view of solver efficiency is a novelty.

The paper is organized as follows. In Section 2 we briefly discuss how nested block-structures can be exploited in the linear algebraic operations required to implement the interior-point

[1] The acronym OOPS stands for Object-Oriented Parallel Solver
`http://www.maths.ed.ac.uk/~gondzio/parallel/solver.html`

method for QP. We give examples of the types of structures we wish to exploit and discuss how to exploit them. In Section 3 we illustrate the use of object-oriented program design to exploit nested structure in an efficient and flexible way. Section 4 reviews the Asset and Liability Management problem, which we use as an example to demonstrate the efficiency of the code. Two different formulations are given, one convex and dense, the other nonconvex but sparse. Section 5 states the actual computational results achieved with our code both in serial and in parallel, while in the final Section 6 we draw our conclusions.

2 Exploiting Nested Block-Structure in the Linear Algebra

2.1 Nested Block-Structured Matrices

The linear algebra module in OOPS is designed to exploit nested block-structure of matrices A and Q. Examples of block-structured matrices are block-diagonal, primal or dual block-angular (block-diagonal with dense row or column block), bordered block-diagonal (3) as well as rank-corrector matrices (of the form $A + VV^T$, where V has a small number of columns). Nested block-structured matrices are block-structured matrices where each of the component blocks itself can be block-structured. There is virtually no limit on the levels of structures exploitable by the solver except the available computer memory.

Nested block-structured matrices occur frequently in applications. An important example is multistage stochastic programming, where every modelled stage corresponds to one level of nesting in the resulting system matrix (cp. section 4). Other examples are various network problems solved in telecommunications (Gondzio and Sarkissian 2003). Some formulations of Support Vector Machines have system matrices of rank-corrector structure (Cristianini and Shawe-Taylor 2000). Rank-corrector structure also occurs when the Q matrix is not known explicitly but estimated by a quasi-Newton scheme.

We believe that nested block-structure is a general property of very large optimization problems. One can argue that truly large optimization problems are usually generated by some regular process which by its nature will introduce some nested block-structure into the matrices. Typical processes leading to nested structures are discretisations over space or time, uncertainty or dynamics.

We assume that the nested structure of the problem is known to the solver. One could attempt to extend of the approach in (Ferris and Horn 1998) to identify block-angular matrix structure to nested structures, however this is not part of our current research.

Nested block-structured matrices are represented in the solver by a tree. The root of the tree corresponds to the whole matrix and every block of a particular sub-matrix is a child node of the node representing this sub-matrix. Leaf nodes correspond to the elementary sub-matrices that can no longer be divided into blocks. With every node of the tree we associate its children as well as information about what type of structure this node represents. The matrix A in Figure 2 for example is represented by the tree in Figure 1. For quadratic programming both matrices A and Q in (1) can exhibit block-structure. Since IPMs are working with the augmented system, the structures of A and Q clearly have to be related, in order for the resulting augmented system to have exploitable structure.

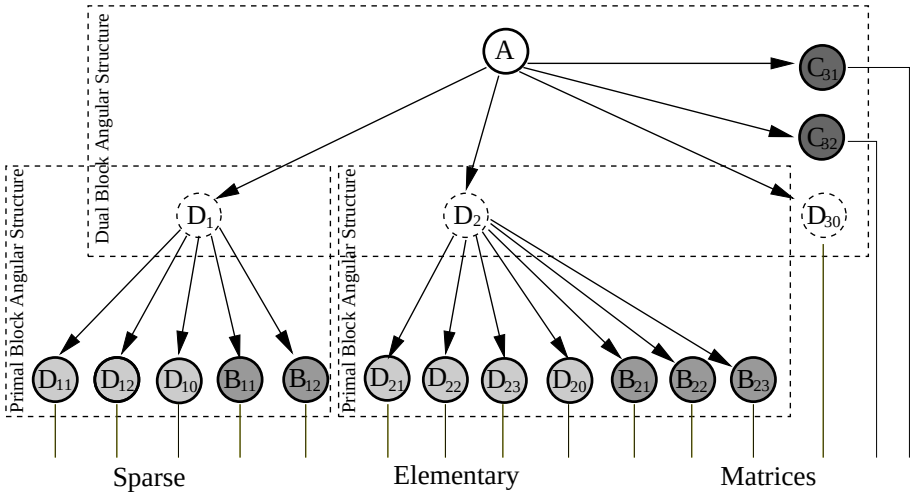

Figure 1: Tree Representation of Blocks.

Assume that matrices A and Q are of the structure displayed in Figures 2 and 3, that is a combination of primal-dual block angular, border diagonal and diagonal structures. Then the matrix $\Phi = \begin{bmatrix} -Q - \Theta^{-1} & A^T \\ A & 0 \end{bmatrix}$ can be reordered into the form displayed in Figure 4, i.e. a nested bordered diagonal form (the tilde in $\tilde{C}_{31}$ indicates that this is a part of the original C_{31} matrix). Since Θ is a diagonal matrix and it does not change the sparsity pattern, we do not display it in Figure 4.

Two matrices A and Q are compatible in the sense that the augmented system matrix formed by them can be reordered into an exploitable nested block-structure if the two trees representing them are compatible. This means that both trees have to agree on the number of children representing diagonal blocks as well as the column-dimensions of these blocks. They do not have to agree on the type of structure represented by each node.

This seems to be a fairly strong restriction, however it can always be satisfied by forcing onto Q a nested block structure compatible with A. Remember that A and Q are not arbitrary matrices, but the Jacobian of the constraints and Hessian of the objective of the same QP problem. Therefore their column dimensions must agree. Clearly the sparsity of such a forced Q matrix might not be efficiently exploitable, at worst Q might have no zero sub-blocks at all. However this can usually be rectified by the introduction of new variables and further linear constraint in the model. Section 4 will give an example for such a remodelling in the case of an Asset and Liability Management problem. Note that the need for such a rewriting of the model is not just a peculiar requirement of our solver, but is a sensible step in order to improve sparsity of Q for any solution approach.

Once it is established that the nested block-structures for A and Q are compatible, the reordering of the augmented system matrix can be done in a generic way - i.e. it is the same for all matrices of a particular structure.

It is worth noting that since we use a tree structure to represent our matrices, we need no further memory to store the reordered augmented system matrix Φ. Its leaf node matrices are identical

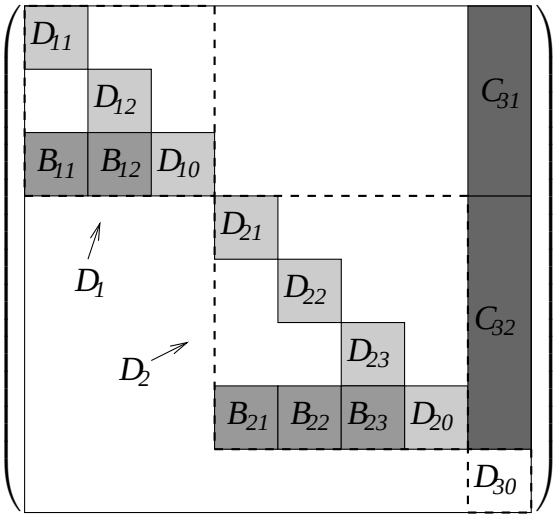

Figure 2: Example of Block-Structured Matrix A.

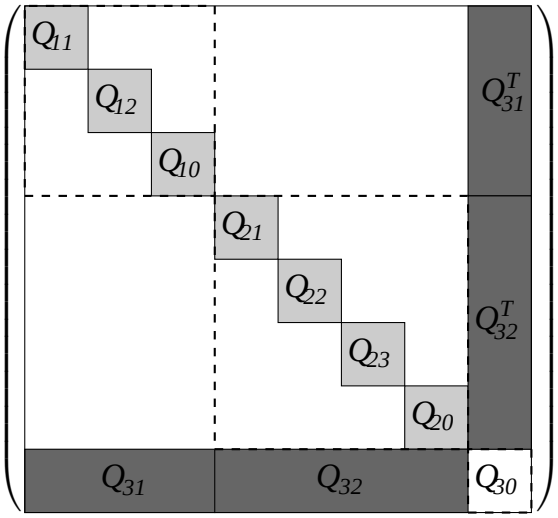

Figure 3: Example of Block-Structured Matrix Q.

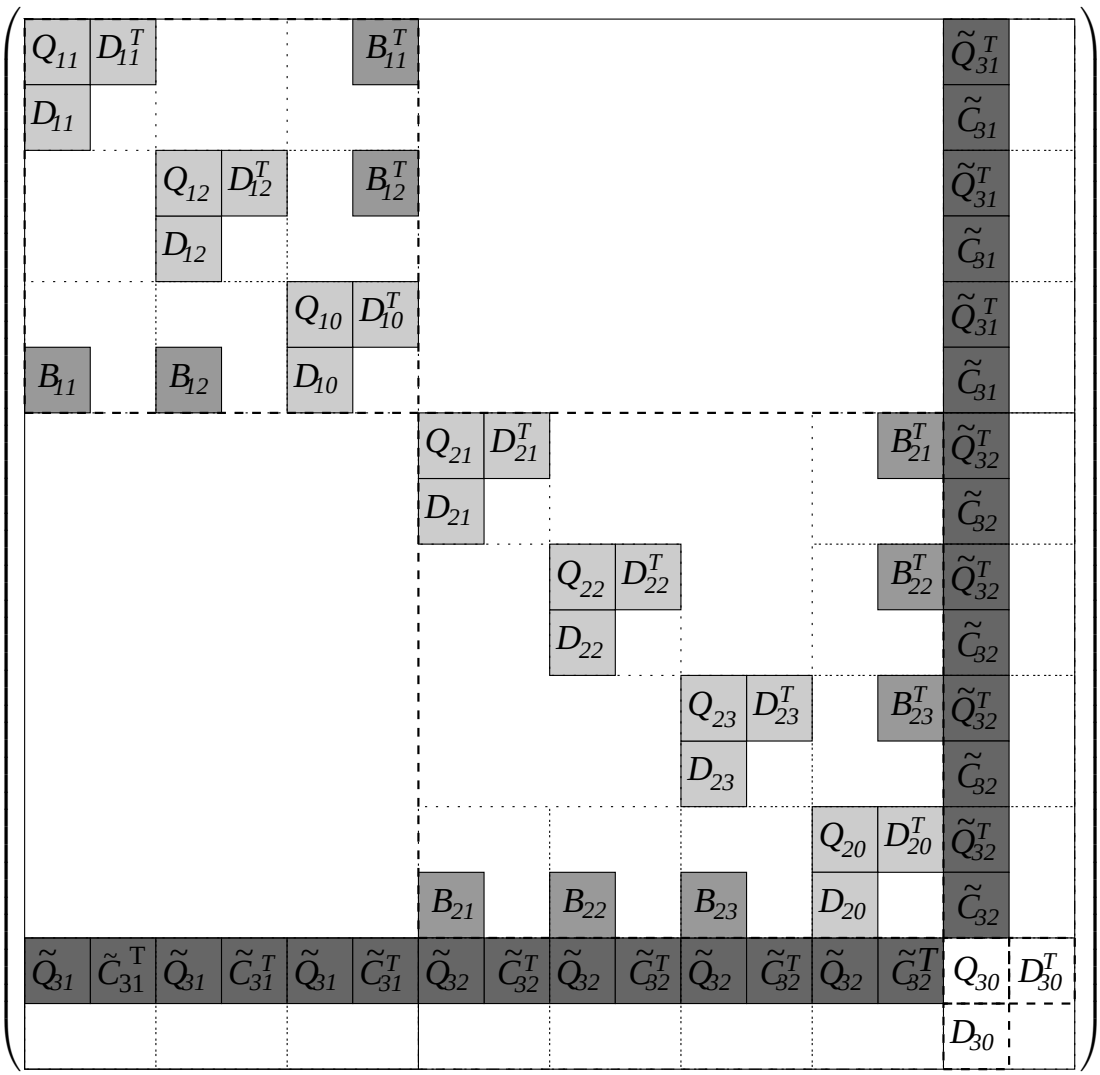

Figure 4: Example of Block-Structured Augmented System Φ.

to those already present in A and Q. Hence the reordered augmented system is represented by a new tree, reusing the same leaf node matrices that make up A and Q. Therefore no physical reordering of memory entries has to be done to obtain the reordered augmented system matrix.

2.2 Exploiting Simple Block-Structures

We will now give an example of how a particular block-structure can be exploited in the linear algebra of the IPM. Suppose a block of the augmented system matrix has the symmetric bordered block-diagonal structure.

$$\Phi = \begin{pmatrix} \Phi_1 & & & & B_1^T \\ & \Phi_2 & & & B_2^T \\ & & \ddots & & \vdots \\ & & & \Phi_n & B_n^T \\ B_1 & B_2 & \cdots & B_n & \Phi_0 \end{pmatrix}, \tag{3}$$

where $\Phi_i \in \mathcal{R}^{n_i \times n_i}, i = 0, ..., n$ and $B_i \in \mathcal{R}^{n_0 \times n_i}, i = 1, ..., n$. Matrix Φ has then $N = \sum_{i=0}^{n} n_i$ rows and columns. Such blocks will appear as seen in Figure 4 as diagonal blocks or root blocks of the augmented system matrix if its component A and Q blocks are of block-diagonal and/or block-angular structure.

With

$$L = \begin{pmatrix} L_1 & & & & \\ & L_2 & & & \\ & & \ddots & & \\ & & & L_n & \\ L_{n,1} & L_{n,2} & \cdots & L_{n,n} & L_c \end{pmatrix}, \quad D = \begin{pmatrix} D_1 & & & & \\ & D_2 & & & \\ & & \ddots & & \\ & & & D_n & \\ & & & & D_c \end{pmatrix}$$

and

$$\begin{aligned} \Phi_i &= L_i D_i L_i^T && \text{(4a)} \\ L_{n,i} &= B_i L_i^{-T} D_i^{-1} && \text{(4b)} \\ C &= \Phi_0 - \sum_{i=1}^{n} B_i \Phi_i^{-1} B_i^T && \text{(4c)} \\ &= L_c D_c L_c^T && \text{(4d)} \end{aligned}$$

matrix Φ can be decomposed as

$$\Phi = L D L^T. \tag{5}$$

Because of this (5) can be seen as a (generalized) Cholesky factorization of Φ.

We can use this to compute the solution to the system

$$\Phi x = b,$$

where $x = (x_1, \ldots, x_n, x_0)^T$, $b = (b_1, \ldots, b_n, b_0)^T$ by the following operations:

$$\begin{aligned} z_i &= L_i^{-1} b_i, \quad i = 1, \ldots, && \text{(6a)} \\ z_0 &= L_c^{-1}(b_0 - \sum_{i=1}^{n} L_{n,i} z_i) && \text{(6b)} \\ y_i &= D_i^{-1} z_i, \quad i = 0, \ldots, n && \text{(6c)} \\ x_0 &= L_c^{-T} y_0 && \text{(6d)} \\ x_i &= L_i^{-T}(y_i - L_{n,i}^T x_0), \quad i = 1, \ldots, n && \text{(6e)} \end{aligned}$$

It is worth noting that the order in which matrix multiplications are performed in (4) and (6) may have a significant influence on the overall efficiency of the solution of equation $\Phi x = b$. The sum to compute C in (4c) is best calculated from terms $(L_i^{-1} B_i^T)^T D_i^{-1} (L_i^{-1} B_i^T)$, which in turn is best calculated as sparse outer products of the sparse rows of $L_i^{-1} B_i^T$. Also the matrices $L_{n,i}$ are best not calculated explicitly. Instead $L_{n,i} z_i$ in (6b) is most efficiently calculated as $B_i(L_i^{-T}(D_i^{-1} z_i))$ and similarly for $L_{n,i}^T x_0$ in (6e). For these reasons we refer to complex factorizations such as (4/6) as *implicit* factorizations.

We have concentrated in this section on exploiting block bordered-diagonal structure as an example of what we mean by structure exploitation. Other structures can be exploited along similar lines. A rank-corrector structure for example would be exploited by using the Sherman-Morrison-Woodbury formula.

A common feature of all structure exploiting approaches is that operations (factorizations, back-solves) for the whole system can be reduced to operations performed on the sub-blocks. The routine working on the top-level system only needs to access the results of the lower-level routine, no details about their particular implementation are assumed. This feature is the base of our object oriented approach to nested structure exploitation.

2.3 Linear Algebra for Elementary Sparse Matrices

The main computational effort of any primal-dual IPM consists in the computation of the primal-dual Newton direction. This requires solving the following linear system

$$\begin{bmatrix} -Q - X^{-1}S & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \xi_d - X^{-1}\xi_\mu \\ \xi_p \end{bmatrix}, \tag{7}$$

where

$$\begin{aligned} \xi_p &= b - Ax, \\ \xi_d &= c - A^T y - s + Qx, \\ \xi_\mu &= \mu e - XSe. \end{aligned}$$

In our tree based linear algebra implementation this system has to be solved at each of the tree nodes corresponding to elementary sparse matrices. The matrix involved in (7) is symmetric but indefinite (even for convex problems when Q is positive definite). For the sake of accuracy and efficiency, the matrix in the reduced Newton system is regularized with diagonal terms R_p and R_d

$$H = \begin{bmatrix} -Q - X^{-1}S & A^T \\ A & 0 \end{bmatrix} + \begin{bmatrix} -R_p & 0 \\ 0 & R_d \end{bmatrix} \tag{8}$$

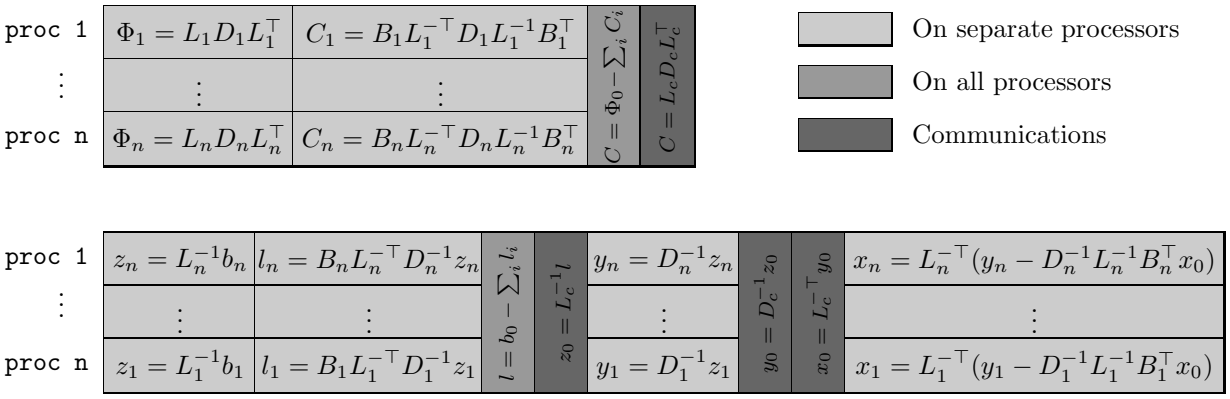

Figure 5: Split of Computations Between Processors.

to obtain a quasi-definite matrix (Vanderbei 1995) and allow more flexibility in the pivot ordering. The use of primal-dual regularization (8) guarantees the existence of a Cholesky-like LDL^T factorization in which the diagonal D contains both positive and negative elements. It avoids the need of using the 2×2 pivots required otherwise to decompose an indefinite matrix (Bunch and Parlett 1971, Duff et al. 1987).

The matrix in the regularized augmented system (8) is symmetric and quasi-definite (with n negative pivots, corresponding to the Q part of the matrix and m positive pivots, corresponding to the A part). In our implementation of interior point method system (7) is regularized and then solved in two steps: (i) factorization of the form LDL^T and (ii) backsolve to compute the directions $(\Delta x, \Delta y)$. Since the triangular decomposition exists for any symmetric row and column permutation of the quasi-definite matrix (Vanderbei 1995), the symbolic phase of the factorization (reordering for sparsity and preparing data structures for sparse triangular factor) can be done before the optimization starts.

2.4 Parallelisation

The scheme indicated in section 2.2 lends itself naturally to parallelization as indicated in Figure 5. Steps (4a) and (4b) and the calculation of $B_i \Phi_i^{-1} B_i^T$ can be distributed among the processors. The only part that remains to be done on all processors is steps (4c)(4d): adding the individual processors contributions to C and factorizing C. Typically the linking block is much smaller in size than the other blocks, and therefore the time spent on factorizing C is small in comparison. Similarly for the backsolves all steps involving matrices $L_i, L_{n,i}, D_i$ and vectors x_i, y_i, z_i and b_i can be distributed among processors, while the steps involving x_0, y_0, z_0, b_0, L_c and D_0 have to be performed on all processors. Thus each processor will need access to its local blocks (subscript $_i$) as well as the complicating blocks (subscripts $_{0,c}$). These blocks will be stored on each processor. Our linear algebra implementation is carefully designed to exploit parallelism along the lines indicated above when possible.

The solver supports coarse grain parallelism, that is the component blocks of the root matrix are distributed among processors. Further down the tree structure is exploited but all computations are done on the same processor. The solver uses an a-priori heuristic to assign component blocks to processors based on the dimension and density of the component matrices. Therefore even

unbalanced matrices can be distributed in an efficient manner.

Summing up, important savings can be achieved in the storage requirements as well as in the efficiency of computations if the linear algebraic operations exploit the block structure of matrix Φ. Last but not least, exploiting block structure allows an almost straightforward parallelization of many of the computational steps when computing the decomposition of Φ and when solving equations with its factorization.

3 Object-Oriented Design

So far we have argued that any particular (nested) structure of the problem should be taken into account when implementing efficient linear algebra routines for IPM. This could be done in a traditional implementation and indeed many such approaches have been presented over time, specialised to exploit some specific structure. However the drawback of this approach is that the resulting code can only handle one particular structure. Any change to the problem might result in the necessity of a significant coding effort.

On the other hand our object-oriented approach, following the ideas outlined in (Gondzio and Sarkissian 2003) gives us the flexibility to model *any* supported nested block-structure without the need to modify the code.

3.1 Matrix Interface for Augmented System Computations

As pointed out earlier we represent a nested block-structured matrix by a tree. Each node corresponds to a sub-block of the matrix and with each node we associate the type of structure present at this node as well as the sub-blocks making up this node.

Observe that the *type* of the node determines how the linear algebraic operations for the corresponding block of the matrix should be executed. Therefore we also associate a set of linear algebra routines with each node: those which exploit the structure present at the node in question.

The representation of the tree storing the nested matrix structure and the linear algebra routines associated with each node of the tree is done using object-oriented principles. We introduce a `Matrix` class that is defined as an *interface* to a selection of linear algebra routines. The class itself does not provide any implementation of the particular methods, it merely defines how they should be called: the class is *abstract*. For a more thorough discussion of this `abstract Matrix class` we refer the reader to (Gondzio and Sarkissian 2003).

For each supported type of block structure we have an implementation of this abstract matrix class: i.e. a module that implements each of the specified routines in an appropriate way for the block-structure in question. When references to the component blocks of a structure are needed this is done by a call to the appropriate linear algebra routine from the implementation of the abstract matrix class living on the child node associated with this component. Similarly the building of the re-ordered augmented system matrix from its components A and Q is done by a re-ordering method in the interface that has a different implementation for each supported

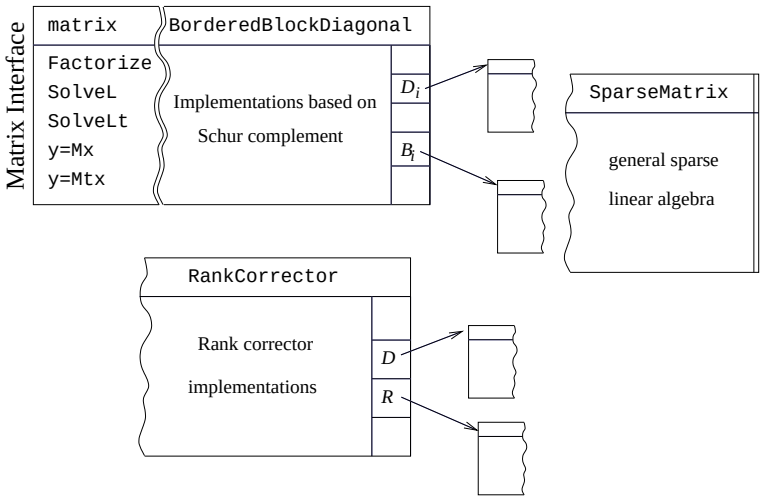

Figure 6: The `Matrix` Interface and Several Implementations of it: Building Blocks for the Tree-Structure.

structure.

The abstract matrix class and its implementations for all types of supported block structure are the building blocks from which the trees representing the matrices A, Q and the re-ordered augmented system are build (Figure 6).

3.2 Implementation Details

The linear algebra code is written in C. We decided to use C rather than C++ or Java due to the more direct control offered by C. While language support for object-oriented design as offered in C++ or Java would undoubtedly make the coding easier, these language features incur some overhead that might decrease the overall efficiency of the solver. It is well possible to use C for object-oriented program design at the cost of a slight increase in coding effort when compared to an object-oriented language. The bottom level routines that implement the elementary sparse matrix factorization and backsolves are written in Fortran, again for efficiency reasons.

The parallel implementation of OOPS uses MPI. This choice offers a great flexibility concerning the choice of platform and the code should run just as well on a network of PCs as on a dedicated parallel machine. The numerical results reported in this paper were obtained on a SunFire 15000 offering 24 processors running at 900MHz. Comparisons as to the efficiency of OOPS on different parallel environments have been reported in (Gondzio and Sarkissian 2003) and we refer the reader there.

The interior point algorithm used to drive the linear algebra library is the infeasible primal-dual method with multiple centrality correctors (Gondzio 1996). Other choices are possible and could be easily implemented without having to touch the linear algebra kernel itself.

4 Asset and Liability Management Problems

There are numerous sources of structured linear programs. Although we have developed a general structure exploiting IPM solver, we have tested it in the first instance only on Asset and Liability Management (ALM) problems. This section will briefly summarize the structure of these problems. We will follow the problem description of (Steinbach 2000, Steinbach 2001) and we refer the reader to (Ziemba and Mulvey 1998) and the references therein for a detailed discussion of these problems.

Assume we are interested in finding the optimal way of investing into assets $j = 1, \ldots, J$. The returns on assets are uncertain. An initial amount of cash b is invested at $t = 0$ and the portfolio may be rebalanced at discrete times $t = 1, \ldots, T$. The objective is to maximize the final value of the portfolio at time $T + 1$ and minimize the associated risk measured with the variance of the final wealth. To model the uncertainty in the process we use discrete random events ω_t observed at times $t = 0, \ldots, T$; each of the ω_t has only a finite number of possible outcomes. For each sequence of observed events $(\omega_0, \ldots, \omega_t)$, we expect one of only finitely many possible outcomes for the next observation ω_{t+1}. This branching process creates a *scenario tree* rooted at the initial event ω_0. Let L_t be the level set of nodes representing past observations $(\omega_0, \ldots, \omega_t)$ at time t, L_T the set of final nodes (leaves) and $L = \bigcup_t L_t$ the complete node set. In what follows a variable $i \in L$ will denote nodes, with $i = 0$ corresponding to the root and $\pi(i)$ denoting the predecessor (parent) of node i. Further p_i is the total probability of reaching node i, i.e. on each level set the probabilities sum up to one. We draw the reader's attention to the important issue of scenario tree generation (Høyland, Kaut and Wallace 2003).

Let v_j be the value of asset j, and c_t the transaction cost. It is assumed that the value of the assets will not change throughout time and a unit of asset j can always be bought for $(1 + c_t)v_j$ or sold for $(1 - c_t)v_j$. Instead a unit of asset j held in node i (coming from node $\pi(i)$) will generate extra return $r_{i,j}$. Denote by $x^h_{i,j}$ the units of asset j held at node i and by $x^b_{i,j}, x^s_{i,j}$ the transaction volume (buying, selling) of this asset at this node. We assume that we start with zero holding of all assets but with funds b to invest. Further we assume that one of the assets represents cash, i.e. the available funds are always fully invested.

We will now present two slightly different versions of the model resulting from these assumptions. One of them is convex but has a dense Q matrix. The other which is obtained by introducing an extra variable and constraint has a sparse Q matrix at the expense of a nonconvex problem formulation. However the problem is still convex in the space of the constraints, which is why we use this second formulation due to its sparsity benefits. The non-convexity of the model does not seem to adversely affect the performance of the interior point algorithm.

The objective of the ALM problem is to maximize final wealth while minimizing risk. Final wealth y is simply expressed as the expected value of the final portfolio converted into cash

$$y = I\!E((1 - c_t) \sum_{j=1}^{J} v_j x^h_{T,j}) = (1 - c_t) \sum_{i \in L_T} p_i \sum_{j=1}^{J} v_j x^h_{i,j},$$

while risk is expressed as its variance

$$\begin{aligned}
\mathrm{Var}((1-c_t)\sum_{j=1}^{J} v_j x^h_{T,j}) &= \sum_{i\in L_T} p_i[(1-c_t)\sum_j v_j x^h_{i,j} - y]^2 \\
&= \sum_{i\in L_T} p_i(1-c_t)^2[\sum_j v_j x^h_{i,j}]^2 - 2y\underbrace{(1-c_t)\sum_{i\in L_T} p_i \sum_{j=1}^{J} v_j x^h_{i,j}}_{=y} + y^2 \underbrace{\sum_{i\in L_T} p_i}_{=1} \\
&= \sum_{i\in L_T} p_i(1-c_t)^2[\sum_j v_j x^h_{i,j}]^2 - y^2.
\end{aligned}$$

The actual objective function of the problem is a linear combination of these two terms as in the Markowitz model (Steinbach 2001). The ALM problem can then be expressed as

$$\begin{aligned}
\max_{x,y\geq 0} \quad & y - \rho[\sum_{i\in L_T} p_i[(1-c_t)\sum_j v_j x^h_{i,j}]^2 - y^2] \\
\text{s.t.} \quad & (1-c_t)\textstyle\sum_{i\in L_T} p_i \sum_j v_j x^h_{i,j} = y \\
& (1+r_{i,j})x^h_{\pi(i),j} = x^h_{i,j} - x^b_{i,j} + x^s_{i,j}, \quad \forall i \neq 0, j \\
& \textstyle\sum_j (1+c_t) v_j x^b_{i,j} = \sum_j (1-c_t) v_j x^s_{i,j}, \quad \forall i \neq 0 \\
& \textstyle\sum_j (1+c_t) v_j x^b_{0,j} = b.
\end{aligned} \tag{9}$$

The risk aversity factor ρ has been set to 1.0 in our computations. To illustrate the structure of this problem we introduce $x_i = (x^s_{i,1}, x^b_{i,1}, x^h_{i,1}, \ldots, x^s_{i,J}, x^b_{i,J}, x^h_{i,J})$, and define matrices

$$G = \begin{pmatrix} 1 & -1 & 1 & & & & \\ & & & \ddots & & & \\ & & & & 1 & -1 & 1 \\ -c^s_1 & c^b_1 & 0 & \cdots & -c^s_J & c^b_J & 0 \end{pmatrix}, \quad B_i = \begin{pmatrix} 0 & 0 & 1+r_{i,1} & & & & \\ & & & \ddots & & & \\ & & & & 0 & 0 & 1+r_{i,J} \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{pmatrix}$$

and

$$\begin{aligned}
Q_i \in I\!R^{3J\times 3J} \quad &: \quad \begin{cases} (Q_i)_{3j,3k} = p_i(1-c_t)^2 v_j v_k, \ j,k = 1,\ldots,J, & i \in L_T \\ Q_i = 0, & i \notin L_T \end{cases} \\
d_i \in R^{1\times 3J} \quad &: \quad (d_i)_{3j} = (1-c_t)p_i v_j,
\end{aligned}$$

where $c^b_j = (1+c_t)v_j, c^s_j = (1-c_t)v_j$. The Q_i are lattice matrices (▦) that have entries only in elements with a row and column index divisible by three (corresponding to the x^h variables). We can now rewrite problem (9) as

$$\begin{aligned}
\max_{x,y\geq 0} \quad & y - \rho[\sum_{i\in L_T} x_i^T Q_i x_i - y^2] \quad \text{s.t.} \quad \textstyle\sum_{i\in L_T} d_i^T x_i = y \\
& B_{\pi(i)} x_{\pi(i)} = G x_i \quad \forall i \neq 0 \\
& G x_0 = b e_{J+1}.
\end{aligned} \tag{10}$$

If we further assemble the vectors $x_i, i \in L$ and y into a big vector $x = (y, x_{\sigma(0)}, x_{\sigma(1)}, \ldots, x_{\sigma(|L|-1)})$ where σ is a permutation of the nodes $0,\ldots,|L|-1$ in a depth-first order, the problem can be written as

$$\max_x \ c^T x - \rho x^T Q\, x \quad \text{subj. to} \ \ Ax = b$$

with matrices A and Q of the form

$$A = \left(\begin{array}{c|ccccc|ccccc|c|ccccc}
-1 & & 0 & d_i & \cdots & d_i & \cdots & 0 & d_i & \cdots & d_i & & & & & \\
 & G & & & & & & & & & & & & & & \\
 & B_i & G & & & & & & & & & & & & & \\
 & 0 & B_i & G & & & & & & & & & & & & \\
 & \vdots & \vdots & & \ddots & & & & & & & & & & & \\
 & 0 & B_i & & & G & & & & & & & & & & \\
 & \vdots & & & & & \ddots & & & & & & & & & \\
 & B_i & & & & & & G & & & & & & & & \\
 & 0 & & & & & & B_i & G & & & & & & & \\
 & \vdots & & & & & & \vdots & & \ddots & & & & & & \\
 & 0 & & & & & & B_i & & & G & & & & &
\end{array}\right) \tag{11}$$

$$Q = \left(\begin{array}{c|ccc|ccc|c|ccc}
-1 & & & & & & & & & & \\
 & \mathbf{0} & & & & & & & & & \\
 & & \mathbf{0} & & & & & & & & \\
 & & & \ddots & & & & & & & \\
 & & & & \blacksquare & & & & & & \\
 & & & & & \ddots & & & & & \\
 & & & & & & \blacksquare & & & & \\
 & & & & & & & \ddots & & & \\
 & & & & & & & & \mathbf{0} & & \\
 & & & & & & & & & \ddots & \\
 & & & & & & & & & & \blacksquare
\end{array}\right) \tag{12}$$

which is of nested primal/dual block-angular form. However due to the -1 term in the diagonal of Q (while all other Q_i — being variance-covariance matrices — are positive semi-definite) this formulation of the problem is not convex.

We can however substitute for the y term in this formulation, using

$$\begin{aligned}
y^2 &= [\sum_{i \in L_T} (1 - c_t) p_i \sum_{j=1}^{J} v_j x_{i,j}^h]^2 \\
&= (1 - c_t)^2 \sum_{i \in L_T} \sum_{l \in L_T} \sum_{j=1}^{J} \sum_{k=1}^{J} p_i p_l v_j v_k x_{i,j}^h x_{l,k}^h
\end{aligned}$$

resulting in a block-dense Q matrix with $|L| \times |L|$ blocks of the form

$$Q_{i,l} : \begin{cases}
(Q_{i,l})_{3j,3k} = -(1 - c_t)^2 p_i p_l v_j v_k, \quad j,k = 1,\ldots,J, & i,l \in L_T, i \neq l \\
(Q_{i,i})_{3j,3k} = [p_i (1 - c_t)^2 - (1 - c_t)^2 p_i^2] v_j v_k, \quad j,k = 1,\ldots,J, & i \in L_T \\
Q_{i,l} = 0, & i \notin L_T \text{ or } l \notin L_T.
\end{cases}$$

This formulation of the problem is convex (Q now being a variance-covariance matrix) but dense:

$$Q = \left(\begin{array}{c|ccc|c|ccc|} \mathbf{0} & & & & & & & \\ \hline & \mathbf{0} & & & & \mathbf{0} & & \\ & \blacksquare & \cdots & \blacksquare & & \blacksquare & \cdots & \blacksquare \\ & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ & \blacksquare & \cdots & \blacksquare & & \blacksquare & \cdots & \blacksquare \\ \hline & \multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c|}{\vdots} \\ \hline & \mathbf{0} & & & & \mathbf{0} & & \\ & \blacksquare & \cdots & \blacksquare & & \blacksquare & \cdots & \blacksquare \\ & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ & \blacksquare & \cdots & \blacksquare & & \blacksquare & \cdots & \blacksquare \\ \hline \end{array}\right)$$

Because of high density of matrix Q there is no point in solving such a formulation. Instead, we have used the first formulation in our numerical tests.

5 Numerical Results

We will now present computational results that illustrate the efficiency of our structure exploiting interior point code for quadratic programming, enabling it to solve QP problems with millions of variables and constraints. Further we demonstrate that the code parallelizes well obtaining almost perfect speed-ups on 2-8 processors.

All computations were done on the Sunfire 15000 machine at Edinburgh Parallel Computing Centre (EPCC). This machine features 24 900MHz UltraSparc-III processors and a total of 48GB of memory.

In Table 1 we summarize problem statistics of our test-problems. All problems are randomly generated Asset and Liability Management problems using the non-convex formulation of section 4. 'Stages' and 'Total Nodes' refer to the geometry of the scenario trees for these problems. 'Blocks' is the number of first stage scenarios. Problems ALM1-ALM5 use a symmetric scenario tree while problem UNS1 uses an unsymmetric scenario tree. In this case the sub-blocks are distributed among processors by a greedy algorithm (assigning largest sub-blocks first). Table 2 gives the performance of OOPS on the six smaller problems ALM1-ALM5 and UNS1 for 1-8 processors. Blank entries indicate that the problem was not run on the given number of processors. Problems ALM1-ALM5 have only been run for processor numbers that are a divisor of the number of blocks. For all test we have used a stopping tolerance of $\texttt{rel\_gap} < 10^{-8}$. We report the solution time in seconds (t), the number of interior point iterations (it) and the parallel efficiency (pe) in percent:

$$100 \times \frac{\texttt{serial time}}{\texttt{number procs} \times \texttt{parallel time}}$$

All problems can be solved in a reasonable time and with a reasonable amount of interior point iterations. The speed-up of the parallel implementation for the symmetric problems is near perfect, due to the exact balance of the blocks treated by each processor and by far the

Problem	Stages	Blocks	Assets	Total Nodes	Constraints	Variables
	S	B	J	$\|L\|=\sum_{i=0}^{S-1} B^i$	$(J+1)\|L\|+1$	$3J\|L\|+1$
ALM1	5	10	5	11111	66,667	166.666
ALM2	6	10	5	111111	666,667	1,666,666
ALM3	6	10	10	111111	1,222,222	3,333,331
ALM4	5	24	5	346201	2,077,207	5,193,016
ALM5	4	64	12	266305	3,461,966	9,586,981
UNS1	5	35	5	360152	2,160,919	5,402,296
ALM6	4	120	5	1742521	10,455,127	26,137,816
ALM7	4	120	10	1742521	19,167,732	52,275,631

Table 1: Asset and Liability Management Problems: Problem Statistics.

Problem	1 proc		2 procs		4 procs		5 procs		6 procs		8 procs	
	t	it	t	pe	t	pe	t	pe	t	pe	t	pe
ALM1	86	14	45	95.5			18	95.5				
ALM2	1933	22	968	99.8			387	99.9				
ALM3	8172	29	4102	99.6			1638	99.8				
ALM4	6435	33	3221	99.9	1642	98.0			1092	98.2	856	94.0
ALM5	8958	18	4566	98.0	2286	98.0					1195	93.7
UNS1	6119	27	3260	93.8	1633	93.7	1312	93.3	1092	93.4	872	87.7

Table 2: Results of OOPS on Smaller Problems.

major computational task being distributed amongst the processors. Problem ALM4 has also been run on 12 and 24 processors with a parallel efficiency of 93.9% and 90.9% respectively, indicating that the approach scales well to a higher number of processors. Also the speed-ups for the unsymmetric problem are satisfactory showing that our approach does not heavily rely on symmetry of the scenario tree.

Table 3 shows the performance of OOPS on the two huge problems ALM6/ALM7. These problems have only been solved in parallel on 16 processors. As can be seen, we are able to solve problem ALM7 with more than 50 million variables in just over 2 hours.

5.1 Comparison with CPLEX

In order to evaluate the competitiveness of our solver we have compared its performance with that of CPLEX 7.0 QP barrier solver. This version of CPLEX can only solve QP problems with a positive semi-definite Hessian. We have therefore done our comparisons on a convexified version of the sparse nonconvex ALM formulation (10). Although this problem has no longer

Problem	time	iter
ALM6	1470	18
ALM7	8465	19

Table 3: Solving Large Problems on 16 Processors.

Problem	Stages	Blocks	Assets	Total Nodes	Constraints	Variables
	S	B	J	$\|L\|=\sum_{i=0}^{S-1} B^i$	$(J+1)\|L\|+1$	$3J\|L\|+1$
cALM1	5	10	5	11111	66.667	166.666
cALM2	6	10	5	111111	666.667	1.666.666
cALM8a	4	10	50	1111	56.662	166.651
cALM8b	3	33	50	1123	57.274	168.451
cALM8c	3	50	50	2552	130.153	382.801
cALM8d	3	70	50	4971	253.522	745.651

Table 4: Convexified Problems Used for Comparison with CPLEX.

Problem	CPLEX 7.0			OOPS		
	time (s)	iter	mem (Mb)	time (s)	iter	mem (Mb)
cALM1	215	36	202	231	14	91
cALM2	3107	51	1859	4570	26	922
cALM8a	1317	29	452	1196	14	258
cALM8b	2838	31	637	368	16	142
cALM8c	10910	29	1590	860	16	319
cALM8d	(51000)	(30)	Out of Mem	1723	17	678

Table 5: Results of Comparing CPLEX with OOPS.

an interpretation in terms of Asset and Liability optimization, its numerical behaviour should be similar to that of the unmodified problem. Indeed, the convexification does not change the sparsity structure of the problem hence the cost of a single iteration should not change. The problem does change significantly though and this may affect the number of iterations to reach optimality. This is confirmed by comparing the performance of OOPS on both problems.

Since we do not have a CPLEX licence on the SunFire 15000 used for the results reported above, the following tests have been run on a slower 400MHz Ultrasparc-II processor with 2GB of memory. The sizes of these problem instances are thus smaller than those reported above.

CPLEX also uses a solution tolerance of `rel_gap` $< 10^{-8}$. For the test problems CPLEX consistently uses between 2-5 times as much memory as OOPS does. Actually, CPLEX ran out of memory for problem cALM8d. The reported solution time is an extrapolation based on the number of flops needed per iteration reported by CPLEX and assuming that 30 interior point iterations will be needed. The analysis of the results collected in Table 5 indicate that OOPS needs consistently fewer interior point iterations. This is an effect of the multiple centrality correctors (Gondzio 1996) employed by our implementation. The solution times are similar for cALM1/cALM2 but OOPS performs vastly superior for the cALM8 problems, especially the last three of these problems which feature large elementary matrices and a high number of first stage scenarios.

Based on our experience, the problem size staying the same, the memory requirements and the solution time of OOPS would decrease as the number of blocks increases. This is clearly seen when comparing the solution statistics for problems cALM8a and cALM8b. Although they have very close dimensions (Constraints and Variables), cALM8a is a four-stage problem with fewer

blocks in the first stage. Indeed, cALM8a has 10 larger first stage blocks while cALM8b has 33 smaller blocks. OOPS on cALM8b needs less memory and is solved faster. For CPLEX both solution time and memory requirements increase from cALMa to cALMb. It is here that the power of our approach is expressed best.

6 Conclusion

We have presented a structure exploiting implementation of an interior point method for quadratic programming. Unlike other such implementations which are geared towards one particular type of structure displayed by one particular type of problem, our approach is general enough to exploit *any* block structure and indeed any nested structure. We achieve this flexibility by an object-oriented layout of the linear algebra library used by the method. An additional advantage of this layout is that it lends itself naturally to efficient parallelization of the algorithm.

We have given computational results of our algorithm on a selection of randomly generated Asset and Liability Management problems. These problems can be formulated as nonconvex sparse quadratic programs and can be solved very efficiently and with near-perfect speed up in the parallel implementation, enabling us to solve a problem of more than 50 million variables in just over two hours on 16 processors.

Acknowledgements
We are grateful to the anonymous referees for constructive comments, resulting in an improved presentation.

References

Andersen, E. D., Gondzio, J., Mészáros, C. and Xu, X.: 1996, Implementation of interior point methods for large scale linear programming, *in* T. Terlaky (ed.), *Interior Point Methods in Mathematical Programming*, Kluwer Academic Publishers, pp. 189–252.

Benson, S., McInnes, L. C. and Moré, J. J.: 2001, TAO users manual, *Technical Report ANL/MCS-TM-249*, Argonne National Laboratory.

Birge, J. R.: 1985, Decomposition and partitioning methods for multistage stochastic linear programs, *Operations Research* **33**, 989–1007.

Birge, J. R. and Qi, L.: 1988, Computing block-angular Karmarkar projections with applications to stochastic programming, *Management Science* **34**(12), 1472–1479.

Blomvall, J. and Lindberg, P. O.: 2002a, A Riccati-based primal interior point solver for multistage stochastic programming, *European Journal of Operational Research* **143**, 452–461.

Blomvall, J. and Lindberg, P. O.: 2002b, A Riccati-based primal interior point solver for multistage stochastic programming - extensions, *Optimization Methods and Software* **17**(3), 383–407.

Bradley, S. and Crane, D.: 1972, A dynamic model for bond portfolio management, *Management Science* **19**, 139–151.

Bunch, J. R. and Parlett, B. N.: 1971, Direct methods for solving symemtric indefinite systems of linear equations, *SIAM Journal on Numerical Analysis* **8**, 639–655.

Cariño, D., Kent, T., Myers, D., Stacy, C., Sylvanus, M., Turner, A., Watanabe, K. and Ziemba, W.: 1994, The Russel-Yasuda Kasai model: an asset/liability model for Japanese insurance company using multistage stochastic programming, *Interfaces* **24**(1), 29–49.

Consigli, G. and Dempster, M.: 1998, Dynamic stochastic programming for asset-liability management, *Annals of Operations Research* **81**, 131–162.

Cristianini, N. and Shawe-Taylor, J.: 2000, *An Introduction to Support Vector Machines and Other Kernel Based Learning Methods*, Cambridge University Press.

Duff, I. S., Erisman, A. M. and Reid, J. K.: 1987, *Direct methods for sparse matrices*, Oxford University Press, New York.

Ferris, M. C. and Horn, D. J.: 1998, Partitioning mathematical programs for parallel solution, *Mathematical Programming* **80**, 35–62.

Gassmann, H. I.: 1990, MSLiP: A computer code for the multistage stochastic linear programming problems, *Mathematical Programming* **47**, 407–423.

Gertz, E. M. and Wright, S. J.: 2003, Object-oriented software for quadratic programming, *ACM Transactions on Mathematical Software* **29**(1), 58–81.

Gondzio, J.: 1996, Multiple centrality corrections in a primal-dual method for linear programming, *Computational Optimization and Applications* **6**, 137–156.

Gondzio, J. and Grothey, A.: 2003, Reoptimization with the primal-dual interior point method, *SIAM Journal on Optimization* **13**(3), 842–864.

Gondzio, J. and Kouwenberg, R.: 2001, High performance computing for asset liability management, *Operations Research* **49**(6), 879–891.

Gondzio, J. and Sarkissian, R.: 2003, Parallel interior point solver for structured linear programs, *Mathematical Programming* **96**(3), 561–584.

Hegland, M., Osborne, M. R. and Sun, J.: 2002, Parallel interior point schemes for solving multistage convex programming, *Annals of Operations Research* **108**, 75–85.

Høyland, K., Kaut, M. and Wallace, S. W.: 2003, A heuristic for moment-matching scenario generation, *Computational Optimization and Applications* **24**(2/3), 169–186.

Jessup, E. R., Yang, D. and Zenios, S. A.: 1994, Parallel factorization of structured matrices arising in stochastic programming, *SIAM Journal on Optimization* **4**(4), 833–846.

Kusy, M. and Ziemba, W.: 1986, A bank asset and liability model, *Operations Research* **34**, 356–376.

Linderoth, J. and Wright, S. J.: 2003, Decomposition algorithms for stochastic programming on a computational grid, *Computational Optimization and Applications* **24**(2/3), 207–250.

Meza, J. C.: 1994, OPT++: An object-oriented class library for nonlinear optimization, *Technical Report SAND94-8225*, Sandia National Laboratories.

Mulvey, J. and Vladimirou, H.: 1992, Stochastic network programming for financial planning problems, *Management Science* **38**, 1643–1664.

Parpas, P. and Rustem, B.: 2003, Decomposition of multistage stochastic quadratic problems in financial engineering, *Technical report*, Department of Computing, Imperial College. Presented at the International Workshop on *Comutational Management Science, Economics, Finance and Engineering*, Cyprus, 28-30 March, 2003.

Ruszczyński, A.: 1985, A regularized decomposition method for minimizing a sum of polyhedral functions, *Mathematical Programming* **33**, 309–333.

Steinbach, M.: 2000, Hierarchical sparsity in multistage convex stochastic programs, *in* S. Uryasev and P. M. Pardalos (eds), *Stochastic Optimization: Algorithms and Applications*, Kluwer Academic Publishers, pp. 363–388.

Steinbach, M.: 2001, Markowitz revisited: Mean variance models in financial portfolio analysis, *SIAM Review* **43**(1), 31–85.

Vanderbei, R. J.: 1995, Symmetric quasidefinite matrices, *SIAM Journal on Optimization* **5**, 100–113.

Vladimirou, H. and Zenios, S. A.: 1999, Scalable parallel computations for large-scale stochastic programming, *Annals of Operations Research* **90**, 87–129.

Wright, S. J.: 1997, *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia.

Zenios, S.: 1995, Asset/liability management under uncertainty for fixed-income securities, *Annals of Operations Research* **59**, 77–97.

Ziemba, W. T. and Mulvey, J. M.: 1998, *Worldwide Asset and Liability Modeling*, Publications of the Newton Institute, Cambridge University Press, Cambridge.